Eberhard Karls Universität Tübingen
Mathematisch- Naturwissenschaftliche Fakultät
Fachbereich Informatik

Diplomarbeit Bioinformatik

Entwicklung und Validierung eines spiegelstereoskopischen Virtual-Reality Aufbaus

Matthias David Hannig
Tübingen, Mai 2012

Gutachter

Prof. Dr. Hanspeter A. Mallot
(Biologie)
Lehrstuhl für Kognitive Neurowissenschaft
Universität Tübingen

Prof. Dr. Wolfgang Rosenstiel
(Bioinformatik)
Fachbereich Informatik (Wilhelm-Schickard-Institut)
Universität Tübingen

Betreuer

Dr. Gregor Hardieß

Erklärung:

Hiermit erkläre ich, dass ich diese Arbeit selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind wurden durch Angaben von Quellen als Entlehnung kenntlich gemacht.
Diese Diplomarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Tübingen, den 30. Mai 2012

Danksagung

Ich möchte mich zunächst für die Überlassung des Themas, die Erstellung des Erstgutachtens und die Betreuung bei Prof. Dr. Hanspeter A. Mallot bedanken.

Ich danke auch Prof. Dr. Wolfgang Rosenstiel für die Erstellung des Zweitgutachtens.

Besonderer Dank geht an Dr. Gregor Hardieß für die Betreuung meiner Diplomarbeit. Vielen Dank!

Desweiteren bedanke ich mich herzlich bei allen Teilnehmern am Versuch und der gesamten Arbeitsgruppe.

Außerdem geht mein Dank an meine Familie. Danke für Eure Unterstützung!

Zu guter Letzt gebührt einem Menschen besonderer Dank: Danke für das unermüdliche Korrekturlesen. Danke für den nicht abreißenden Nachschub an Kaffee und Mate. Danke, dass du immer für mich da warst. Danke Leonie.

Inhalt

Abbildungen

1. Einleitung

Die Fähigkeit, Orte zu erkennen und uns an diese zu erinnern, ermöglicht es uns, diese immer wieder aufzusuchen und uns in unserer Umgebung zu orientieren. Dabei ist es anerkannt, dass wir dazu Informationen, die wir über unsere Wahrnehmung erhalten, nutzen. Wir erinnern uns dabei an Informationen, die wir über diesen Ort gesammelt haben. Im Allgemeinen werden diese Informationen als Landmarken bezeichnet. Dieser Begriff ist dabei in seiner Bedeutung weit gefasst und nicht immer eindeutig definiert. Wir betrachten die Definition der Landmarke als die der lokalen Positionsinfomation (Mallot, 2005). Darunter fallen alle Sinneseindrücke, die an einem Ort und in einer Blickrichtung aufgenommen werden. Diese Landmarken müssen dabei nicht zwangsläufig als Objekte in einem Bild identifiziert werden. Ein Beispiel dafür sind sogenannte Snapshots, bei denen die reine, nur wenig verarbeitete Bildinformation zur Navigation genutzt werden kann. Dieses Verhalten wurde in der Literatur bereits für verschiedene Insektenarten wie Grabwespen (Tinbergen und Kruyt, 1938), Bienen (Cartwright und Collett, 1983) und Waldameisen (Durier et al., 2003) gezeigt. Diese mit einem bestimmten Ort assoziierten Snapshots sind dabei die Erinnerungen an den sensorischen Input und nicht das Erinnern an bestimmte, identifizierte Objekte, die sich an dieser Stelle befunden haben. Zu den sensorischen Inputs gehören beispielsweise Farbwert- und Intensitätsverteilungen.

Von Gillner et al. (2008) wurde bereits gezeigt, dass Menschen in der Lage sind, einen Ort in einem Raum ausschließlich anhand von Intensitätsinformationen zu identifizeren und einen direkten Weg dorthin zurückzulegen.

Neben den hauptsächlich monokularen Bildeigenschaften können auch binokulare Bildeigeschaften wie die Tiefeninformationen als lokale Positionsinformation verwendet werden. Diese Tiefensignaturen, wie beispielsweise der Abstand zu den Wänden in einem Raum, können ebenfalls zur Erkennung eines Ortes, und damit zur Navigation, genutzt werden. In der Literatur wurde die Nutzung solcher geometrischer Informationen in Versuchen mit Ratten gezeigt (Cheng, 1986).

Derzeit wird der Einfluss einzelner Wahrnehmungen, welche wir zur Extraktion von Rauminformationen nutzen, im Rahmen der Kognitionsforschung untersucht. In dem im Rahmen der vorliegenden Arbeit durchgeführten Experiment sollte nun die Wahrnehmung ausschließlich auf Tiefensignaturen beschränkt werden. Dabei sollten monokulare Bildinformationen wie beispielsweise Verdeckung, Perspektive, Gradienten der visuellen Verarbeitung vorenthalten und ausschließlich die Stereopsis zur Extraktion von Rauminformationen verwendet werden. Welche Informationen wir wirklich zur Orientierung

und Navigation nutzen ist noch unbekannt.

1.1. Aufbau des Stereoskops und Konzept des Experiments

Zur Untersuchung des Einflusses von Tiefensignaturen auf die Zielnavigation wurde ein „Random-Dot-Limited-Lifetime“-Stimulus verwendet, um alle Bildinformationen außer der Tiefeninformation der visuellen Verarbeitung unzugänglich zu machen. In dem Experiment zur Zielnavigation wurde den Probanden der „Random-Dot-Limited-Lifetime“-Stimulus auf einem spiegelstereoskopischen Virtual-Reality-Aufbau präsentiert. Der Aufbau orientiert sich an dem von Wheatstone erfundenen Stereoskop (Abb. 1) und setzt die von Kollin und Hollander (2007) beschriebene Ein-Spiegel-Variante um.

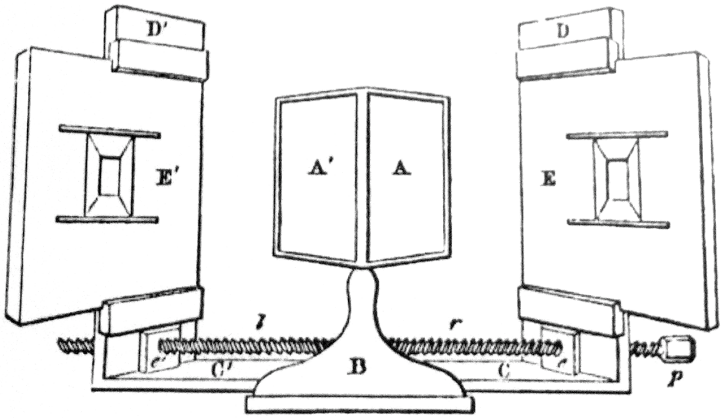


Abb. 1: Zeichnung des Wheatstone-Stereoskops aus Wheatstone (1852)

Das Wheatstone-Stereoskop bietet dabei den Vorteil eines einfachen Versuchsaufbaus mit einem qualitativ hochwertigen 3D-Erlebnis ohne die Nachteile anderer Stereoskopieverfahren: Eine volle Farbwiedergabe ist im Gegensatz zu Anaglyph-Verfahren möglich. Es gibt kein Übersprechen von Bildern (Ghosting), wie bei farb- und polarisationsfilterbasierenden Stereoskopen. Es ist ohne Modifikation des Aufbaus für Brillenträger geeignet. Die von den Monitoren unterstützte Auflösung ist höher als bei Head-Mounted-Displays und die Bildwiederholfrequenz wird ebenfalls nicht reduziert, wie es bei Shutter-Brillen der Fall ist.

Man kann den Materialaufwand weiter reduzieren, indem man einen der beiden Monitore vor dem Betrachter an der Stelle positioniert, an der das virtuelle Bild beim klassischen Aufbau entstehen würde. Der Vorteil der Ein-Spiegel-Variante (Abb. 2) ist die leichtere

Justierbarkeit, da nur noch ein Spiegel und ein Monitor ausgerichtet werden müssen. Außerdem lässt sich der Abstand zum Monitor, der in die Erzeugung der Projektionsmatrix einfließt, leichter bestimmen.

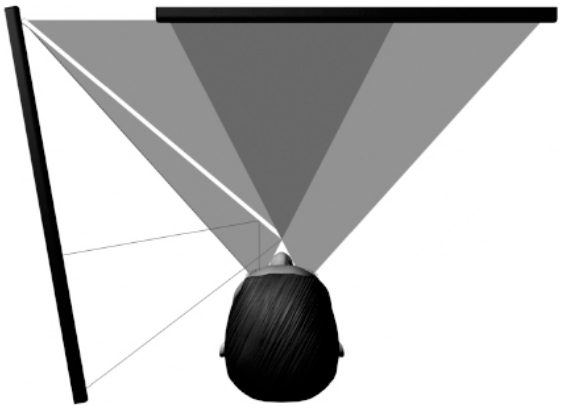


Abb. 2: Stereoskopischer Ein-Spiegel-Aufbau aus Kollin und Hollander (2007)

Zur Untersuchung des Einflusses der Stereopsis wurde in dieser Arbeit der einfachst Fall der Navigation betrachet: Die Navigation zu einem vorgegebenen Ziel, das sogenannte „Homing“. Dazu wurde ein „Walk-To-Cued-Place“-Experiment durchgeführt: Der Proband musste von einem gewählten Ausgangspunkt in der virtuellen Umgebung zu einem vorher erkundeten Ziel navigieren. Die Abweichung vom gewählten Ziel und die Trajektorie wurde dabei aufgezeichnet und ausgewertet.

1.2. Erzeugung von Stereopaaren

Das visuelle System verarbeitet eine Reihe von Tiefeninformationen. Darunter fallen die binokularen Informationen wie die Querdisparität und vertikale Disparität, die Akkomodation und die Vergenz der Augen (Mallot, 2000). Die Kenntnis über den Vergenzwinkel kann dabei propriozeptorisch aber auch als Efferenzkopie in die Verarbeitung der sensorischen Information eingehen (Mallot, 2000).

Um diese Tiefeninformationen den am Experiment teilnehmenden Probanden unter Verwendung des stereoskopischen Aufbaus zu präsentieren, ist die Erzeugung von Stereopaaren ein zentraler Bestandteil der verwendeten Software. Die OpenSceneGraph-Bibliothek beinhaltet bereits die Fähigkeit zur Erzeugung stereographischer Bildpaare für verschiedene Ausgabegeräte. Im Folgenden soll das dabei zum Einsatz kommende Verfahren kurz erläutert werden.

Die Tiefeninformation soll dem Betrachter dabei über die horizontale Disparität des Stereopaares vermittelt werden. Dazu sollen zwei getrennte Bilder so erzeugt werden, dass die durch die Parallaxe bedingte Disparität die Position eines virtuellen Punktes im Raum gesteuert werden kann. Ein Punktpaar mit positiver Parallaxe bedeutet dabei, dass die durch die Projektion entstehenden Punkte auf der selben Seite liegen wie das zugehörige Auge. Der Punkt erscheint dem Betrachter, nach der Fusion des Punktpaares, hinter der Bildschirmebene. Bei einer negativen Parallaxe liegen die Punktpaare dem zugehörigen Auge gegenüber, wodurch der virtuelle Punkt vor der Bildschirmebene erscheint. Bei Punkten, die auf der Bildschirmebene liegen sollen, darf keine horizontale Disparität auftreten (Abb. 3).

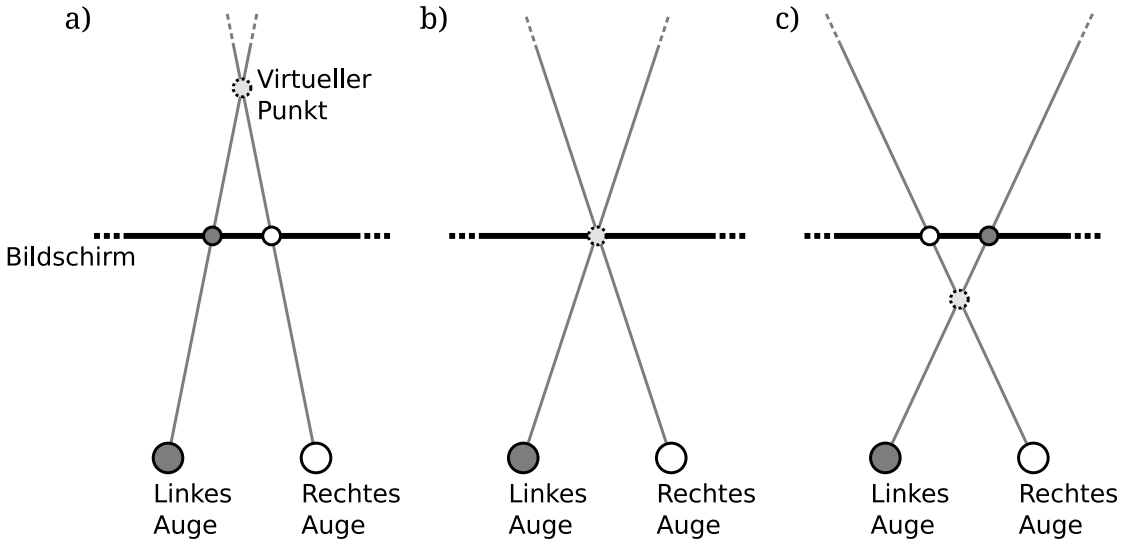


Abb. 3: Querdisparität von Punkten und ihre virtuelle Position relativ zum Bildschirm - a) Ungekreuzte Querdisparität - der virtuelle Punkt erscheint hinter dem Bildschirm b) keine Querdisparität - der Punkt erscheint auf der Fokusebene c) Gekreuzte Querdisparität - der virtuelle Punkt erscheint vor dem Bildschirm

Um die parallaktisch bedingte Disparität der auf dem Stereoskop gezeigten Stereopaare zu erzeugen, reicht es nicht aus, die Kamera um einen Fokuspunkt zu drehen. Diese sogenannte „toe-in“-Methode führt zu einer vom Zentrum der Projektionsebene aus wachsenden vertikalen Parallaxe, die zu einem unangenehmen 3D-Eindruck für den Betrachter führt (Bourke, 1999). Die dadurch entstehende vertikale Disparität erschwert es, zwei korrespondierende Punkte zu fusionieren. Bourke (1999) verwendet zur Lösung des Problems eine „parallel axis asymmetric frustum perspective projection“. Dabei wird

durch eine Scherung der Projektionsmatrix ein asymmetrisches Frustum so erzeugt, dass die Projektionsebene parallel zur Bildschirmebene liegt. Um dabei zu erreichen, dass die horizontale Disparität auf der Bildschirmebene null wird, wird die Projektionsmatrix so verzerrt, dass die Scherung im Abstand des Betrachters zum Monitor (sd) dem halben Augenabstand (ed) entspricht. Die Projektionsmatrix zur Erzeugung des linken Bildes des Stereopaars muss dabei nach rechts, die des rechten Bildes nach links geschert werden (Abb. 4). Die Projektionsmatrix für die Erzeugung des linken Bildes muss daher um eine Scherung erweitert werden, die durch die folgende Gleichung beschrieben wird:

$$x' = x + \frac{0,5 \cdot ed}{sd} \cdot z \tag{1}$$

Die neue Projektionsmatrix erhalten wir durch die Multiplikation der Projektionsmatrix mit einer Scherungsmatrix, die dieser Gleichung entspricht.

Die Projektionsmatrix zur Erzeugung des rechten Bildes gleicht der zur Erzeugung des Bildes für das linke Auge. Einzig das Vorzeichen der Scherung muss geändert werden.

Die Koordinaten der Vertices müssen vor der Überführung in den Clipping-Raum der Scherung nachgeführt werden. Dies wird durch die Multiplikation der View-Matrix mit einer Translationsmatrix erreicht. Die Punkte werden dabei für das linke Bild um den halben Augenabstand nach rechts, für das rechte um denselben Wert nach links, verschoben (Abb. 4).

Das Verfahren zur Erzeugung von Stereopaaren mittels asymmetrischer Frustra wird auch von der OpenSceneGraph-Bibliothek verwendet und musste lediglich durch eine Spiegelung des erzeugten Bildes zur Betrachtung durch das Wheatstone-Stereoskop erweitert werden.

In dieser Arbeit wurde ein Verfahren zur Extraktion von Punktwolken aus einer Raumgeometrie entwickelt. Dieses nutzt die Symmetrieeigenschaft der OpenGL-Rendering-Pipeline, indem es die im z-Buffer gespeicherte Tiefeninformationen zur Rekonstruktion von Punkten auf der Geometrie nutzt.

Dieses Verfahren wurde im Rahmen dieser Arbeit bei der Implementierung des „Walk-To-Cued-Place“-Experiments verwendet. Dabei sollte die Fragestellung untersucht werden, inwiefern eine Beschränkung auf die Stereopsis als Möglichkeit zur Extraktion von Raumfomationen die Leistung bei der Navigation zu einem vorgegebenen Ziel beeinflusst. Die Ablaufsteuerung des Experimentes wurde mittels einer Statemachine realisiert.

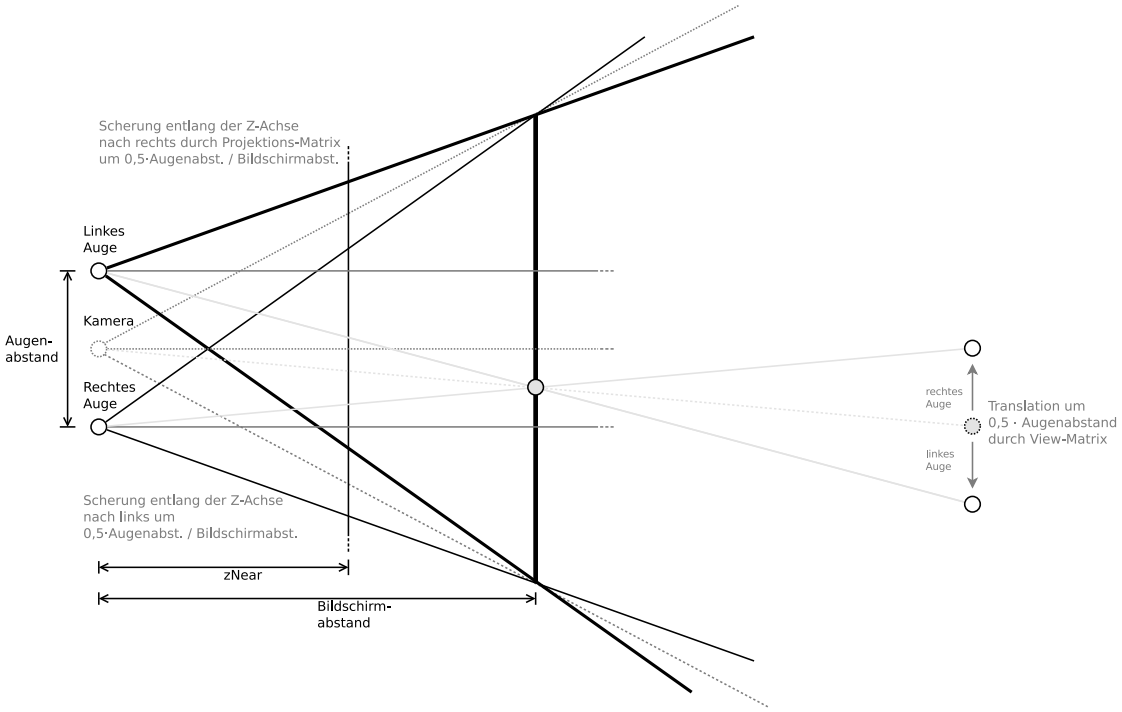


Abb. 4: Konstruktion des asymmeterischen Kamera-Frustums

Zur Abschätzung der Performance des Verfahrens wurde in dieser Arbeit eine eingehende Analyse der Eigenschaften des z-Buffers durchgeführt.

Desweiteren wurde die in der verwendeten OpenSceneGraph-Bibliothek vorhandene Unterstützung zur Erzeugung von Stereo-Graphiken zur Verwendung mit dem stereoskopischen Aufbau erweitert.

Schließlich wurde noch ein Toolset zur Visualisierung der im Experiment erhobenen Daten entwickelt. Dies beinhaltet eine Bibliothek zur Erzeugung von SVG-Graphiken in C++ und zwei Applikationen zur Erzeugung von Trajektorie- und Streudiagrammen, die zur Auswertung des Experimentes verwendet wurden.

Neben der Software zur Duchführung des geplanten Experimentes wurde auch das Stereoskop selbst aufgebaut. Dazu wurde die nötige Hardware angekauft und spezielle Monitorhalterungen handgefertigt.

2. Material und Methoden

Zur Durchführung des Versuches wurde ein Verfahren zur Extraktion von Punkten aus einer Raumgeometrie durch inverse Projektion entwickelt. Dieses Verfahren wird bei der Erzeugung des „Random-Dot-Limited-Lifetime“-Stimulus verwende, da es Punkte im Weltkoordinatensystem erzeugt, aus denen Stereopaare zur Darstellung auf der Stereoskop-Hardware errechnet werden.

2.1. Hard- und Software

Es wurden zwei Samsung SyncMaster S27A850D Monitore mit einer Auflösung von jeweils 2560×1440 px bei 27 Zoll Bildschirmdiagonale verwendet. Betrieben wurden diese an einem durch die Firma Transtec für diesen Zweck angefertigten PC mit einer Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz Quad-Core CPU und 4 GB Arbeitsspeicher. Bei der Auswahl der Hardware wurde besonderen Wert auf eine Graphikkarte mit zwei Dual Link DVI Ausgängen gelegt, um beide hochauflösende Monitore treiben zu können. Die Wahl fiel dabei auf eine NVIDIA® GeForce® GTX 570 HD Graphikkarte.

Darüberhinaus wurde ein Oberflächenspiegel des Herstellers gertenbach.info e. K. verwendet.

Als Graphikbibliothek wurde OpenSceneGraph 3.0.1 eingesetzt. Die Software wurde in C++ entwickelt.

Als Betriebssystem wurde die auf Ubuntu basierende GNU/Linux-Distribution Mint 12 (Lisa) verwendet.

2.2. Aufbau des Stereoskops

Da der vom Hersteller mitgelieferte Monitorfuß sich aufgrund von Instabilität und unzureichenden Einstellmöglichkeiten als ungeeingnet erwiesen hatte, musste ein neuer Monitorhalter entworfen und gefertigt werden (Abb. 5).

Dieser bestand aus 30×30 mm dicken Aluminiumprofilen, die auf einem fünf Millimeter dicken Aluminiumblech montiert wurden. Die Profile bilden dabei ein Gelenk mit zwei Freiheitsgraden, Rotation und Neigung, was eine Feinjustierung des Aufbaus ermöglicht. Abgeschlossen wird die Halterung durch eine ebenfalls fünf Millimeter starke Alumi-

niumplatte, welche den Monitor über die dafür vorgesehene VESA Wandhalterungsverschraubung mit dem neuen Fuß verbindet.

Abb. 5: Monitorhalter aus Aluminiumprofilen; mittig ist das Gelenk mit zwei Freiheitsgraden, zur Einstellung der Neigung und Rotation des Monitors zu sehen. Dieser ist über die VESA Wandhalterungverschaubung mit dem Monitorhalter verbunden.

Für den Oberflächenspiegel wurde ebenfalls eine Halterung konstruiert. Diese ermöglicht eine Neigungs- und Höhenverstellung des Spiegels (Abb. 6). Daraus ergeben sich wiederum zwei Freiheitsgrade für den Spiegel. Dieser soll vor dem Probanden positioniert werden und dient der Umlenkung des Strahlenganges zwecks Positionierung des zweiten Monitors links vom Betrachter. Die Neigungsverstellung dient dem Ausgleich von geringen Höhenunterschieden zwischen beiden Monitoren.

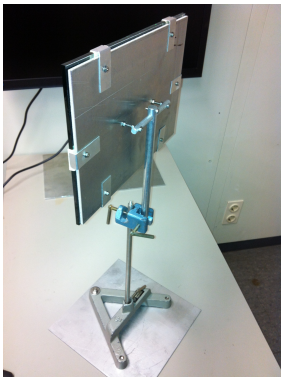


Abb. 6: Spiegelhalterung mit der Möglichkeit zur Neigung des Oberflächenspiegels zum Feinausgleich von Höhenunterschieden zwischen beiden Monitoren.

Das Stereoskop wurde als Ein-Spiegel System nach Kollin und Hollander (2007) aufgebaut (Abb. 2).

2.3. Ausrichtung des Stereoskops

Um einen optimales visuelles Ergebnis zu erhalten, müssen die Bilder beider Monitore durch Ausrichtung der Monitore und des Spiegels zur Deckung gebracht werden (Abb. 7). Um den Vorgang zu unterstützen und die Justierung des Stereoskop-Aufbaus zu kontrollieren Testbild, bestehend aus gut sichtbaren Gitterlinien und Diagonalen Linien, erstellt. Dieses wird auf dem linken Monitor gespiegelt dargestellt. Sind Spiegel und Monitore optimal ausgerichtet, so ist der Spiegel vor dem Frontalmonitor nicht mehr zu sehen.

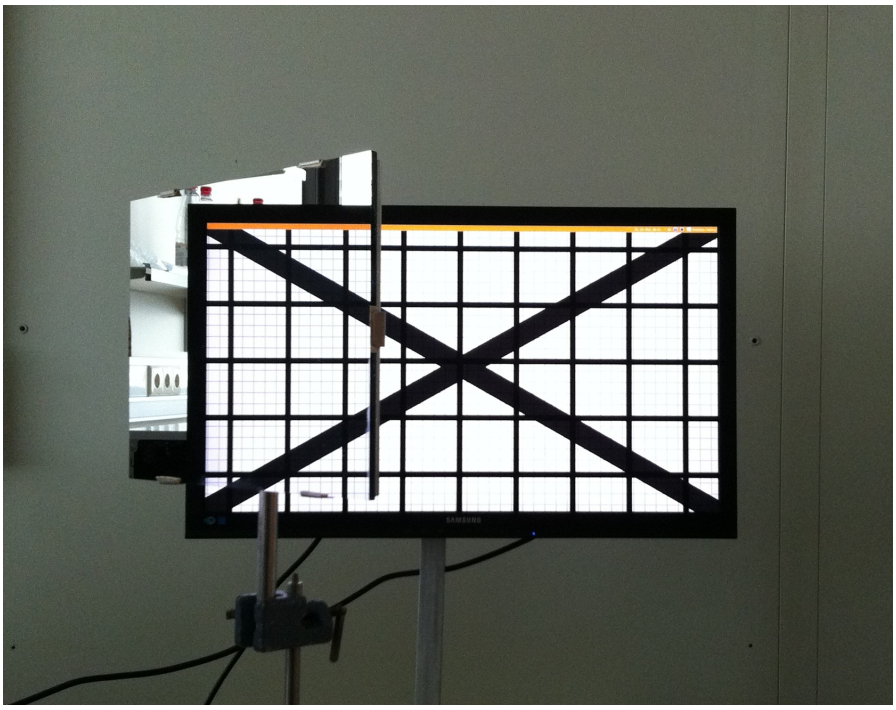


Abb. 7: Ausgerichtetes Stereoskop mit Testbild - das Bild des linken Monitors ist mit dem des rechten zur Deckung gebracht. Der linke Monitor ist dabei auf dem Bild nicht zu sehen, das von ihm dargestellte Bild wird durch den Spiegel so umgelenkt, dass bei einer guten guten Ausrichtung des Stereoskops der Spiegel fast nicht sichtbar ist.

Der für das rechte Auge zuständige Monitor wird dabei zunächst frontal vor dem Probanden positioniert. Anschließend wir der Spiegel so positioniert, dass der Abstand zum Auge möglichst gering ist und der Proband sich nicht über die Tischkante beugen muss. Der Winkel des Spiegels soll dabei so gewählt werden, dass sich der linke Monitor möglichst weit in der Peripherie des Betrachers befindet. Dabei darf der Winkel nicht zu groß gewählt werden, da sonst das Gesicht des Probanden gespiegelt und das horizontale Ge-

sichtsfeld eingeschrenkt würde. Schließlich wird der linke Monitor ausgerichtet. Dieser wird so lange verschoben bis die linke und rechte Kante der Monitore übereinstimmen. Anschließend wird über die Neigungsverstellung von Monitor und Spiegel die obere und untere Kante zur Deckung gebracht.

Es wurden folgende Abstände und Winkel im Versuch verwendet:

- Abstand des Front-Monitors zum Betrachter: 0,85 m
- Abstand des Spiegel zum Front-Monitor: 0,8 m
- Abstand des linken Monitors zum Spiegel: 0,8 m
- Winkel des Spiegels: 49°

Eine einmalige Ausrichtung des Aufbaus vor dem Versuchsbeginn ist für alle Teilnehmer des Versuches ausreichend. Vor dem Beginn und nach dem Abschluss des Versuchs wird die Ausrichtung kontrolliert.

Der Abstand des Monitors zum Betracher sowie der Augenabstand wird der OpenSceneGraph-Bibliothek als Umgebungsvariable im Start-Script (A.2) übergeben.

2.4. Grundlagen der Render-Implementierung

Es wurde eine Software zur Validierung des Stereoskops entwickelt. Zurückgegriffen wurde dabei auf einen Random-Dot-Limited-Lifetime Stimulus: Eine Szene wird dabei ausschließlich mit lebenszeitbeschränkten Punkten dargestellt. Die Punkte liegen dabei auf den in der Szenerie befindlichen Objekten und werden nach Ablauf ihrer Lebensdauer erneut erzeugt. Die Szene und deren Objekte sind dadurch so lange nicht mehr erkennbar, bis der Probant sich in der Szene zu bewegen beginnt und durch den optischen Fluss und Parallax-Effekte Konturen wieder wahrnehmbar sind.

Dabei müssen die Punkte bei einer Bewegung des Probanden in der Szene korrekt nachgeführt werden. Die Schwierigkeit liegt in der Bestimmung der darzustellenden Punkte in der Szene.

Da aufgrund der Echtzeitanforderung nicht auf Raytracing zurückgegriffen werden konnte wurde ein Render-Algorithmus implementiert, der die Symmetrie-Eigenschaft des Geometriepfades der OpenGL Rendering Pipeline (Abb. 8) ausnutzt.

Anschließend wurde die stereographische Darstellung für die Stereoskop-Hardware implementiert.

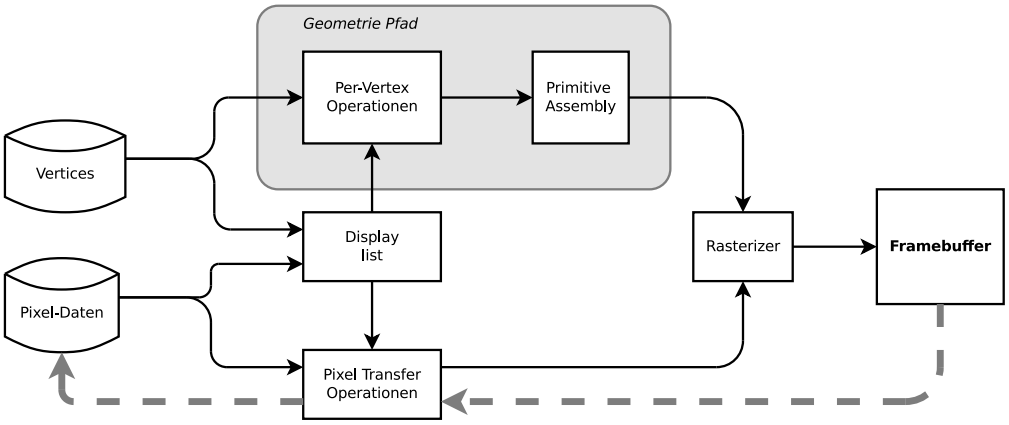


Abb. 8: Vereinfachte Darstellung der OpenGL Rendering Pipeline nach Shreiner et al. (2008)

2.4.1. OpenGL Per-Vertex Operation

In der OpenGL-Pipeline durchlaufen die Vertices der Szene eine Reihe von Transformationen, bis diese dem Rasterizer übergeben werden (Shreiner et al., 2008, S. 103ff). Diese werden für jeden Vertex durchgeführt und überführen ihn vom Weltkoordinatensystem in das Bildschirmkoordinatensystem (Abb. 9).

Ob ein Vertex der Szene darstellbar ist wird im Clipping-Raum getestet. Die Überführung in den Clipping-Raum erfolgt durch die Multiplikation des Vertex mit der ModelView-Projection Matrix (2).

$$(x, y, z, w)^t_{clip} = MP \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}_{world} \tag{2}$$

Die ModelView Matrix ist das Ergebnis der Multiplikation der Model-Matrix mit der Kamera-Matrix (View-Matrix). Die Multiplikation eines Vertex mit der View-Matrix überführt diesen in das Kamerakoordinatensystem. Die darauf folgende Multiplikation mit der Projektionsmatrix überführt diesen in den Clipping-Raum. Da diese Operation für alle Vertices der Szene durchgeführt wird, nutzt OpenGL die ModelViewProjection-Matrix als das Produkt der genannten Matrizen.

Die Projektionsmatrix bestimmt wie die Koordinaten projeziert werden. Beispielsweise sind hier perspektivische oder orthonormale Projektion möglich.

Nach der Transformation in den Clipping-Raum erfolgt der Clipping Test: Liegt die x, y oder z Komponente des Vertex außerhalb $[-w, w]$, so wird dieser verworfen.

Die homogenen Koordinaten der verbleibenden Vertices werden perspektivisch projeziert (3).

$$(x, y, z)^t_{device} = \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)_{clip} \tag{3}$$

Wir erhalten normalisierte Geräte-Koordinaten ($x, y, z \in [-1, 1]$).

Durch die Viewporttransformation erhalten wir die Bildschirmkoordinaten. Der Viewport wird definiert durch seinen Ursprung $(x_0, y_0)^t$ sowie seine Dimensionen w (Breite) und h (Höhe). Die Viewporttransformation bildet x und y linear auf das Viewport-Rechteck z auf $[0, 1]$ ab.

Es ergeben sich für x, y, z die linearen Abbildungen (4), (5) und (6).

$$x_{screen} = x_0 + \frac{w}{2} + x\frac{w}{2} \tag{4}$$

$$y_{screen} = y_0 + \frac{h}{2} + y\frac{h}{2} \tag{5}$$

$$z_{screen} = \frac{1}{2}(z + 1) \tag{6}$$

Es erfolgt anschließend die Primitiv Assembly und die Rasterisierung (Abb. 8). Die z-Werte innerhalb des zu zeichnenden Primitivs werden dabei linear interpoliert und für jedes Pixel im z-Buffer gespeichert.

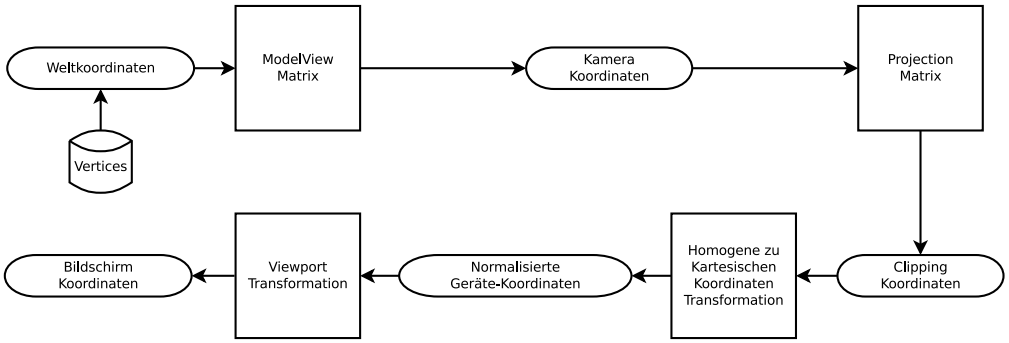


Abb. 9: Per-Vertex Transformation nach Shreiner et al. (2008)

2.4.2. z-Buffer Genauigkeit

Das z-Buffer-Verfahren wurde ursprünglich als Per-Pixel-Sichtbarkeits-Test beschrieben und wird von jeder modernen Graphikhardware unterstützt. Die Speicherung des z-Wertes für jedes Pixel wird ausgenutzt, um die in 2.4.3 beschriebene inverse Projektion durchzuführen zu können.

Die Transformation der z-Koordinate aus dem Kamera-Koordinatensystem in den Clipping-Raum ist nicht linear und ist abhängig von $zNear$ und $zFar$.

Betrachten wir die perspektivische Projektionsmatrix, wie sie mittels `glFrustum` erzeugt wird:

$$M_{proj} = \begin{pmatrix} \frac{2zNear}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2zNear}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{zFar+zNear}{zFar-zNear} & -\frac{2zFar \cdot zNear}{zFar-zNear} \\ 0 & 0 & -1 & 0 \end{pmatrix} \tag{7}$$

Aus (7) folgt direkt

$$z_{clip} = -z_{cam}\frac{zFar + zNear}{zFar - zNear} - w_{cam}\frac{2 \cdot zFar \cdot zNear}{zFar - zNear} \tag{8}$$

$$w_{clip} = -z_{cam} \tag{9}$$

Es kann o. B. d. A. $w_{cam} = 1$ angenommen werden, da die Model-View Matrix nur aus affinen Transformationen besteht, bei denen w unangetastet bleibt. Die normalisierte Gerätekoordinate z_{device} ist also $\frac{z_{clip}}{-z_{cam}}$. Daraus folgt:

$$z_{device} = \frac{zFar + zNear}{zFar - zNear} + \frac{1}{z_{cam}}\left(\frac{-2 \cdot zFar \cdot zNear}{zFar - zNear}\right) \tag{10}$$

Nehmen wir für $zNear = 1$ und $zFar = 10$ an, so können wir die Verteilung der z-Werte berechnen (Abb. 10).

Die Graphikhardware verwendet eine Festkomma-Repräsenatation mit 16, 24 oder 32 Bit. Dafür wird der z-Wert in der Viewport-Transformation in $[0, 1]$ verschoben (6) und mit $s = 2^n - 1$ skaliert, wobei n die Anzahl der zur Verfügung stehenden Bits ist.

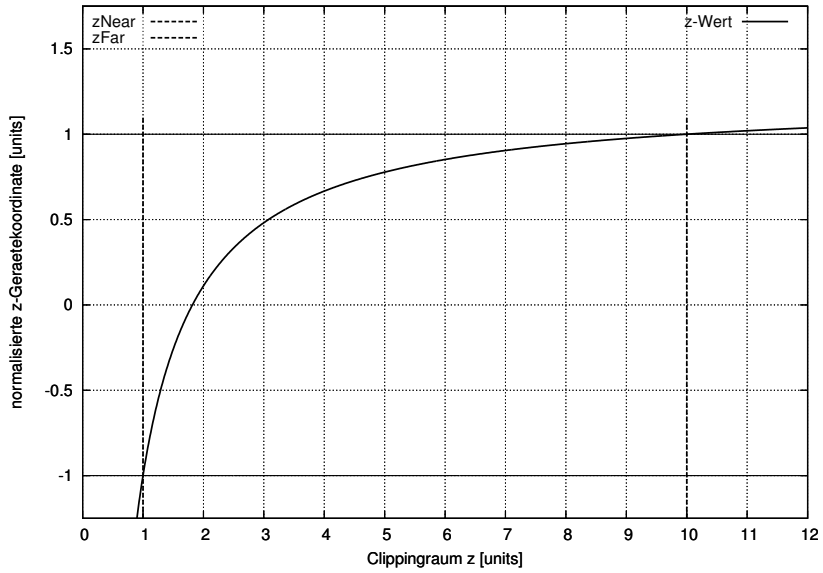


Abb. 10: Durch die Projektion bedingte nichtlineare z-Wert-Verteilung zwischen $zNear$ und $zFar$

Die Genauigkeit mit der Punkte bestimmt werden können ist daher begrenzt durch die Wortbreite des z-Buffers.

Wir erhalten $z_{screen} \in [0, 1]$ durch Einsetzen von (10) in die Viewport-Transformation (6):

$$z_{screen} = \frac{1}{2} \cdot \frac{zFar + zNear}{zFar - zNear} + \frac{1}{z_{cam}} \cdot \frac{zFar \cdot zNear}{zFar - zNear} + \frac{1}{2} \tag{11}$$

Durch Einsetzen von z_{screen} in $\lfloor (2^n - 1)\, z_{screen} \rfloor$ erhalten wir die Festkomma-Repräsentation.

$$z_{fix} = \left\lfloor (2^n - 1) \cdot \frac{1}{2} \cdot \frac{zFar + zNear}{zFar - zNear} + \frac{1}{z_{cam}} \cdot \frac{zFar \cdot zNear}{zFar - zNear} + \frac{1}{2} \right\rfloor \tag{12}$$

Da die Anzahl der zur Verfügung stehenden Bits im z-Buffer begrenzt ist, kommt es zu Rundungsfehlern. Nach OpenGL.org (2012) kommt zu einem Verlust der Genauigkeit von rund $\log_{10} \frac{zFar}{zNear}$ Bits.

Um den kleinst möglichen Abstand zweier Vertices im Kamera-Koordinatensystem zu bestimmen, benötigen wir zunächst die inverse Projektion der Festkomma-Repräsenta-

tion in Kamera-Koordinaten. Diese erhalten wir durch Auflösen von (12) nach z_{cam}

$$z_{cam} = \frac{(2^n - 1) \cdot zFar \cdot zNear}{(2^n - 1) \cdot zFar - z_{fix} \cdot zFar + z_{fix} \cdot zNear} \tag{13}$$

Um den kleinsten z Abstand in Kamera-Koordinaten zu bestimmen bilden wir die Differenz der z-Werte von s und $s - 1$.

$$z_c(s) - z_c(s-1) = zFar - \frac{(2^n - 1) \cdot zFar \cdot zNear}{(2^n - 1) \cdot zFar - (2^n - 2) \cdot zFar + (2^n - 2) \cdot zNear} \tag{14}$$

Ein zu großer Fehler führt zu Aliasing-Artefakten in der Punktwolke. Daher sollte z-Near so groß wie möglich gewählt werden. OpenSceneGraph kann diesen Wert dynamisch automatisch aus den Bounding-Spheres der im Szenengraphen befindlichen Knoten bestimmen und für jeden Frame anpassen. Wir überlassen daher die Bestimmung des Wertes für z-Near der OpenSceneGraph Bibliothek.

Die Konvertierung zwischen der Fest- und Fließkomma-Repräsentation übernimmt der OpenGL Treiber, sodass über ein Framebuffer-Objekt auf den z-Wert nach der Viewport-Transformation zugegriffen werden kann.

2.4.3. Punktauswahl und Erzeugung

Um eine homogene Verteilung der Limited-Lifetime-Dots auf dem Bildschirm zu erreichen werden diese im Bildschirmkoordinatensystem ausgewählt. Dabei werden x_s und y_s zufällig gewählt. Der Inhalt des z-Buffers an dieser Stelle ist z_s.

Alle der in 2.4.1 beschriebenen Vertex-Transformation sind invertierbar. Dies ermöglicht eine eindeutige Bestimmung eines Punktes $(x_w, y_w, z_w)^t$ im Weltkoordinatensystem aus $(x_s, y_s, z_s)^t$.

Um $(x_w, y_w, z_w)^t$ zu erhalten, müssen zunächst normalisierte Geräte-Koordinaten erzeugt werden.

Aus (4) erhalten wir:

$$-x_{device}\frac{w}{2} = x_0 + \frac{w}{2} - x_s$$

Die normalisierte Geräte-Koordinate durch:

$$x_{device} = \frac{2}{w}(x_s - x_0) - 1 \tag{15}$$

Analog zu (15) erhalten wir y_{device}. Durch die Abbildung $f : [0,1] \to [-1,1]$ mit $f(z_s) = 2z_s - 1$ wird z_s verschoben.

Um die homogenen Weltkoordinaten zu erhalten muss der erweiterte Vector $(x_d, y_d, z_d, 1)^t$ mit der inversen Modelview-Projection-Matrix multipliziert werden.

Wir erhalten die inverse Vertex-Transformation.

$$(x_w, y_w, z_w, w)^t = (MP)^{-1} \begin{pmatrix} \frac{2}{w}(x_s - x_0) - 1 \\ \frac{2}{h}(y_s - y_0) - 1 \\ 2z_s - 1 \\ 1 \end{pmatrix} \tag{16}$$

Durch perspektivische Projektion erhalten wir die Weltkoordinaten $(x_w, y_w, z_w)^t$.

$$(x_w, y_w, z_w)^t = \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)^t \tag{17}$$

2.5. Rendering Implementierung

Die Rendering-Software wurde in C++ unter Verwendung der OpenSceneGraph-Bibliothek geschrieben. Das Experiment wurde mit Statecharts modelliert. Implementiert wurde diese mittels der `boost::statechart` Bibliothek. Es wurde dabei wert auf Portierbarkeit gelegt. Die Software ist ohne Anpassung lauffähig unter den Betriebssystemen Linux und Mac OS X.

Es wurde ein First-Person-Kameramanipulator mit Kollisionsdetektion und Trajektorie-Aufzeichnung entwickelt.

Gesteuert werden die einzelnen Trials des Experimentes über eine flache Konfigurations-Datei. Hier wurde ein Parser implementiert.

Es wurde ein Patch für die OpenSceneGraph-Bibliothek entwickelt, der es ermöglicht beliebige bereits bestehende Projekte ohne neues Übersetzen auf dem Steroskop darzustellen (A.1). Dabei handelt es sich um eine Modifikation der `osgUtil`-Bibliothek. Um

diese zu nutzen kann unter Windows die `osgUtil.dll` ausgetauscht bzw. unter Linux mittels `LD_PRELOAD` das modifizierte Shared-Objekt `osgUtil.so` geladen werden (A.2).

2.5.1. Vorraussetzungen

Die Graphik-Hardware muss OpenGL ≥ 2.0 unterstützen. Framebuffer Objects müssen vom Treiber entweder nativ oder als `EXT_framebuffer_object`-Erweiterung (Juliano und Sandmel, 2008) unterstützt werden. Außerdem ist die Unterstützung von Vertex-Buffer Objects entweder nativ oder als `ARB_vertex_buffer_object`-Erweiterung (Hammerstone et al., 2010) erforderlich. Es wird die Unterstützung der `ARB_pixel_buffer_object`-Erweiterung empfohlen. Fehlt diese wird auf `glReadPixel` zurückgefallen. Dies kann sich negativ auf die Performance auswirken. Darüberhinaus wird OpenSceneGraph $\geq 3.0.1$ vorrausgesetzt.

2.5.2. Random-Dot-Limited-Lifetime Renderer

Zunächst wird ein Framebuffer Objekt (FBO) erzeugt und an den Tiefenpuffer (z-Buffer) gebunden. Außerdem wird ein Vertexbuffer Objekt (VBO) zur späteren Darstellung der extrahierten Punkte erzeugt. Mittels `glMapBuffer` wird der Speicherbereich des VBO in den Adressraum des Programms abgebildet. Der Zugriff auf das FBO sollte über ein PixelBufferObject (PBO) erfolgen. Dies ermöglicht sowohl ein schnelles Auslesen der Tiefenpuffer-Daten, als auch die dynamische Anpassung der Punkt-Koordinaten. Die OpenSceneGraph-Bibliothek kapselt diesen Vorgang, sodass die Implementierung dem Entwickler abgenommen wird. Über die `OSG_ASSIGN_PBO_TO_IMAGES`-Umgebungsvariable kann die Verwendung von PBOs erzwungen werden (A.2). Die Implementierung des Limited-Lifetime-Dot-Rendererers ist unabahängig von der Experiment-Implementierung und bezieht sich auf folgende Klassen:

`RandomDotRenderer` Hier wird die Pre-Rendering-Kamera (`osg::Camera`) instanziert, deren z-Buffer an ein FBO gebunden und auf ein `osg::Image` zwecks abgebildet um den Render-To-Texture-Vorgang durchzuführen. Die `DotRenderGeode` wird instanziert und ebenso wie die Pre-Render-Kamera als Kind in diesen Knoten des Szenen-Graphs eingehangen (Abb. 11).

Die Szenenknoten wird zusätzlich zwischen Pre-Render- und Punkt-Geometrie-Knoten eingefügt. Die Szenengeometrie wird dadurch in den z-Buffer der Haupt-Kamera gezeichnet, damit Punkte, deren Lebenszeit noch nicht abgelaufen ist, aber

durch die Verschiebung des Betrachters in der Szene durch ein anderes Objekt verdeckt werden, nicht mehr sichtbar sind.

Der Speicher für die Punkt-Lebenszeiten wird hier alloziert. Der `LifetimeUpdateThread` wird instanziert und gestartet. Eine Instanz des `DotExtractor` wird an die Pre-Rendering-Kamera als `PostDrawCallback` übergeben.

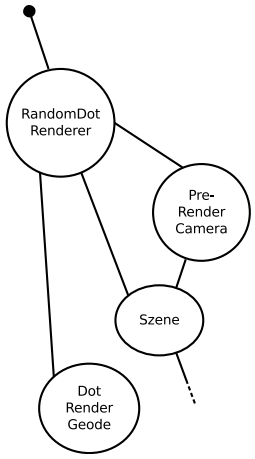


Abb. 11: Szenen-Graph mit Pre-Render- und Punkt-Render-Knoten.

`DotExtractor` Die Punkt-Extraktion ist auf einen aktuellen Zustand des z-Buffers angewiesen, der wiederum nur die z-Werte der Szene enthält. Die neue Erzeugung eines Punktes erfolgt erst wenn dessen Lebenszeit abgelaufen ist. Die Lebenszeit eines Punktes wird bei dessen Erzeugung zufällig ausgewählt und im Punkt-Lifetime-Array abgelegt. Die minimale und maximale Lebenszeit kann über die Konfigurationsvariablen `dot_lifetime_min` und `dot_lifetime_max` in Millisekunden festgelegt werden.

Diese Klasse wird von `osg::Camera::DrawCallback` abgeleitet, der ()-Operator wird implementiert. Ein Objekt dieser Klasse wird der Pre-Render-Kamera als `PostDrawCallback` übergeben. Dem `DotExtractor` ist dabei die Speicheradresse des Vertex-Array der `DotRenderGeode` bekannt, sowie die Adresse des Lifetime-Arrays (Abb. 12).

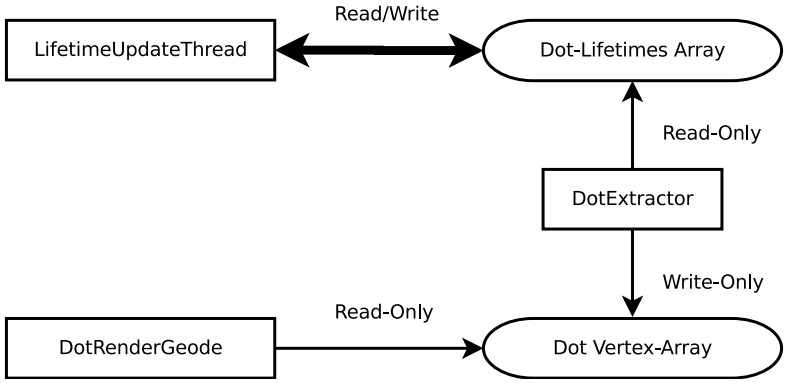


Abb. 12: Zugriff und Abhängigkeitsdiagramm zwischen Extraktor und Renderer.

Die Klasse implementiert den nachfolgenden Algorithmus der Punkt-Extraktion, wie sie in 2.4.3 beschrieben wurde.

procedure EXTRACT_DOTS
  for all $v \in \{v \in dots : lifetime(v) \leq 0\}$ **do**
    $x \leftarrow rand(0, width)$
    $y \leftarrow rand(0, height)$
    $z \leftarrow read\_zbuffer(x, y)$
    $v_t \leftarrow (MP)^{-1} \begin{pmatrix} \frac{2}{w}(x - x_0) - 1 \\ \frac{2}{h}(y - y_0) - 1 \\ 2z - 1 \\ 1 \end{pmatrix}$
    $v \leftarrow \left(\frac{x_{vt}}{w_{vt}}, \frac{y_{vt}}{w_{vt}}, \frac{z_{vt}}{w_{vt}}\right)$
    $lifetime(v) \leftarrow rand(minL, maxL)$
  end for
end procedure

`DotRenderGeode` Die von `osg::Geode`, einer Klasse zum rendern beliebiger Geometrie, abgeleitete Klasse verwaltet das Vertex-Array in welches die Vertices der extrahierten Punkte geschrieben werden. Über ein VBO wird dieses in den Speicherbereich der Graphik-Hardware abgebildet. Die Vertices werden als `GL_POINT`-Primitiv gezeichnet. Die Punkt-Größe kann ebenfalls festgelegt werden.

Da die Punkte coplanar zur Szenengeometrie liegen, verhindert ein Vertex-Shader z-Fighting-Artefakte, indem die Punkte im Kamera-Koordinatensystem um ein geringes z-Offset von $0,005$ Einheiten in Richtung des Betrachters verschoben werden. Dieser Wert wurde experimentell bestimmt. Alternativ kann dies auch mit

`glPolygonOffset` erreicht werden.

`LifetimeUpdateThread` Dies ist ein unabhängig von der Applikation laufender Thread, der bei jeder Ausführung im Punkt-Lifetime-Array den Wert jedes Punktes decrementiert. Dies geschieht einmal pro Millisekunde, sodass für jeden Punkt die Lebenszeit direkt in Millisekunden angegeben werden kann.

Zur Punktextraktion wird ein Multi-Pass-Rendering der Szene durchgeführt:

1. Pass Es erfolgt zunächst das Rendering der Geometrie der Szene. Licht, Shading und Texturierung sind deaktiviert. Die Projektionsmatrix ist unverzerrt. Die Model-Viewmatrix ist unverschoben. Dadurch wird die Szene wie in einem Ein-Kamera System abgebildet.

Es erfolgt nun der Render-To-Texture *(RTT)* Durchlauf.

Die Dimensionen des *Viewport*-Rechteck entsprechen dabei denen, der Zieltextur mit dem Ursprung $(x_0, y_0)^t$ sowie der Breite w und Höhe h der Ziel-Textur.

Der z-Buffer enthält nun die projezierten und normalisierten *z-Koordinaten* aller möglichen Punkte.

Punktauswahl Es werden nun im Bildschirmkoordinatensystem die Punkte ausgewählt, die dem Probanden für die Dauer ihrer Lebenszeit dargestellt werden. Die Auswahl erfolgt im Bildschirmkoordinatensystem um eine gleichmäßige Verteilung aller sichtbaren Punkte zu gewährleisten.

Der zugehörige z-Wert eines Punktes wird direkt aus dem z-Buffer FBO ausgelesen. Man erhält also das Tripel $(x, y, z_s)^t$ mit $0 \leq x \leq hres, 0 \leq y \leq vres$ und $z_s \in [0, 1]$.

Punkterzeugung Es erfolgt nun eine inverse Projektion wie sie in 2.4.3 beschrieben wurde. Die Punkte im Weltkoordinatensystem werden in das Vertex-Array geschrieben, welches an das VBO gebunden wurde (Abb. 12).

2.6. Erweiterung der OpenSceneGraph-Bibliothek

Die OpenSceneGraph-Bibliothek bringt bereits die Fähigkeit, Stereographiken zu erzeugen mit. Sie unterstützt dabei eine Breite Palette von Bildschirmgeräten. Diese reicht von Head-Mounted-Displays über 3D-fähige Projektoren bis hin zu Rot-Cyan-Anaglyph Brillen.

Ein weit verbreitetes Verfahren zur Übertragung von 3D-Inhalten ist das Side-by-Side-Format (SbS), wie es auch im DVB 3D-TV Standard spezifiziert ist (ETSI, 2012). Zu Erzeugung eines Bildes in diesem Format verfügt die OpenSceneGraph-Bibliothek über die Möglichkeit eine horizontale Teilung des Viewports durchzuführen. Auf beiden Viewport-Hälften wird nun das Bild für das jeweilige Auge erzeugt. Um das Bild für das linke respektive rechte Auge auf dem dafür vorgesehenen Monitor darzustellen, spannen wir den Graphik-Context über beide Monitore. Die horizontale Teilung befindet sich dadurch zwischen beiden Monitoren.

Da der linke Monitor in unserem Aufbau durch einen Spiegel betrachtet wird, muss das erzeugte Bild ebenfalls gespiegelt werden:

OpenSceneGraph erzeugt für beide Viewport-Hälften eine ihr jeweils zugeordnete Projektions- und View-Matrix. Dabei wird in der View-Matrix die Welt um jeweils $\pm 0,5 \cdot eyeDistance$ in der Horizontalen verschoben. Um ein asymmetrisches View-Frustum zu erzeugen, wird die Projektionsmatrix mit der in Abschnitt 1 beschriebenen Scherungsmatrix multipliziert.

Schließlich wird das Frustum skaliert. Hier kann im Quelltext die Spiegelung eingebaut werden. Dies wird erreicht indem in der Datei „`src/osgUtil/SceneView.cpp`“ der OpenSceneGraph-Bibliothek in der Methode `computeLeftEyeProjectionImplementation` im Aufruf der Konstruktion der Skalierungs-Matrix der `scale_x`-Parameter durch `-scale_x` ersetzt wird (A.1).

2.7. Design des Experiments

Zur Validierung soll ein „walk-to-cued-place“-Experiment durchgeführt werden. Dabei wird der Proband zunächst an einen zufällig gewählten Ort im Raum, dem Ziel, platziert. Es ist ausschließlich möglich sich umzusehen. Eine Translations im Raum ist nicht möglich.

Anschließend wird eine zufällige Position im Raum als Startposition gewählt und der Probant an diese Stelle transferiert. Die Bewegung im Raum ist jetzt freigegeben. Die Aufgabe des Probanden besteht nun dahin zurück zur ersten Position zu navigieren. Die Trajektorie, die Blickrichtung und der Abstand zum Ziel wurde aufgezeichnet.

Das „walk-to-cued-place“-Paradigma wurde auch von Gillner et al. (2008) verwendet und ist gut etabliert.

Als Umgebung wurde ein drachenförmiger Raum gewählt (Abb. 13), da hier jeder Punkt eindeutig anhand von Seitenlänge und Winkel identifizierbar ist. Dieser Raum ist implementiert in der Klasse `DeltoidRoom` und ist vollständig parametrisiert.

Im Raum wurden fünf Punkte erzeugt die alle sowohl als Start als auch Ziel verwendet wurden. Jedes dieser Ziele sollte jeweils einmal von jedem anderen Punkt aus erreicht werden. Insgesammt ergaben sich daraus 20 Übergänge zwischen den Start- und Ziel-Punkten.

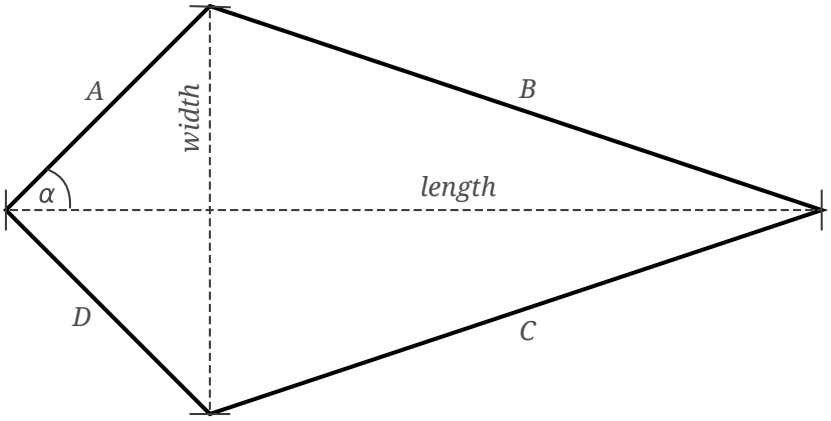


Abb. 13: Parametrisierter drachenförmiger Raum als Experimentalumgebung.

Die Klasse ist abgeleitet von `osg::Geode` und kann somit direkt als Szene im Renderer verwendet werden.

Aufgrund der Flexibilität des Renderers besteht noch die Möglichkeit, die Szene mit einem 3D-Modellierungprogramm (z.B. Blender, http://www.blender.org) zu gestalten, im `3ds`-Format zu exportieren und wiederum als Knoten im Szenengraph einzuhängen.

2.7.1. Parametrisierte Konstruktion eines Drachenvierecks

Bei der Konstruktion des Drachenvierecks, wie sie die Klasse `DeltoidRoom` implementiert, wird die Symmetrie dieser Geometrie ausgenutzt und die einzelnen Seitenlängen berechnet. Bei der Instanzierung ist es möglich, Länge (*length*), Breite (*width*) und halben Öffnungswinkel (α), sowie die Höhe (*height*) des Raumes im Konstruktor zu übergeben.

Betrachten wir das Drachenviereck aus Abb. 13: Sei $\overline{ac}$ die Strecke auf der horizontalen Graden durch den Ursprung mit der Länge *length*. Sei x ein Punkt auf dieser

Strecke. Betrachen wir das rechtwinklige Dreieck, dessen Gegenkathete mit $\frac{width}{2}$ und dessen Öffnungswinkel $\frac{\alpha}{2}$ gegeben ist. Gesucht wird die Ankathete. Diese ist in unserem Drachenviereck die Strecke $\overline{ax}$. Aus $tan(\alpha) = \frac{\text{Gegenkathete}}{\text{Ankathete}}$ folgt direkt

$$\overline{ax} = \frac{0,5 \cdot width}{tan(\frac{\alpha}{2})} \tag{18}$$

Wir konstruieren die Geometrie um den Ursprung. Die jeweils zweite Koordinate ist daher 0. Es werden nun die vier Eckpunkte berechnet. Diese werden außerdem benötigt um die nachfolgende Punktauswahl durchzuführen.

$$a = -\overline{ax} \tag{19}$$
$$b = -\frac{1}{2}width \tag{20}$$
$$c = length - \overline{ax} \tag{21}$$
$$d = \frac{1}{2}width \tag{22}$$

2.7.2. Randomisierte Punktauswahl im Deltoiden

Die Klasse implementiert außerdem noch die Methode `get_random_point()`, mit der sich randomisierte Koordinaten erzeugen lassen, die sich innerhalb des Raumes befinden. Dabei wird wie folgt vorgegagenen:

1. Es wird zufällig eine Kante $e \in \{A, B, C, D\}$ ausgewählt
2. Die zur Kante zugehörigen Vectoren v_1, v_2 werden bestimmt. Sei $e = A$, dann ist $v_1 = (-\overline{ax}, 0)^t$ und $v_2 = \left(0, -\frac{width}{2}\right)^t$.
3. Wähle $s, t \in [0, 1]$ zufällig.
4. Bestimme p durch lineare Interpolation auf e und skaliere p mit s um den Punkt im Raum in Richtung $(0, 0)^t$ zu verschieben.

$$p = s \cdot ((1 - t)v_1 + tv_2) \tag{23}$$

2.7.3. Statemachine

Modelliert wurde das Experiment mittels Statecharts. Es ergeben sich für die State-Machine folgende Zustände:

`Pre-Trial` Es läuft gerade kein Versuchsdurchgang. Es ergeben sich zwei Unterzustände:

- `Waiting` Der Bildschirm ist geschwärzt.
- `Cue` Der Probant bekommt den Hinweis, dass der Durchgang im nächsten Schritt beginnen wird.

`Trial` Der Versuchsdurchgang läuft. Es ergeben sich wiederum zwei Unterzustände:

- `Learning` Die Lernphase, in der sich der Probant ausschließlich umsehen darf beginnt.
- `Testing` Der Probant darf sich frei bewegen und soll zum Ausgangspunkt navigieren.

Mit dem Übergang `EvAdvanceExperiment` ergibt sich die Statechart in Abb. 14. Diese wurde mittels der `boost::statechart`-Bibliothek implementiert. Die Zustände und Übergänge sind in `Experiment.hpp` definiert.

Das „AdvanceExperiment-Event“ wird durch Drücken der „Enter-Taste“ an die Statemachine gesendet. Eine Steuerung über einen Timer, um beispielsweise die Verweildauer in der Lernphase zu begrenzen, ist aber möglich.

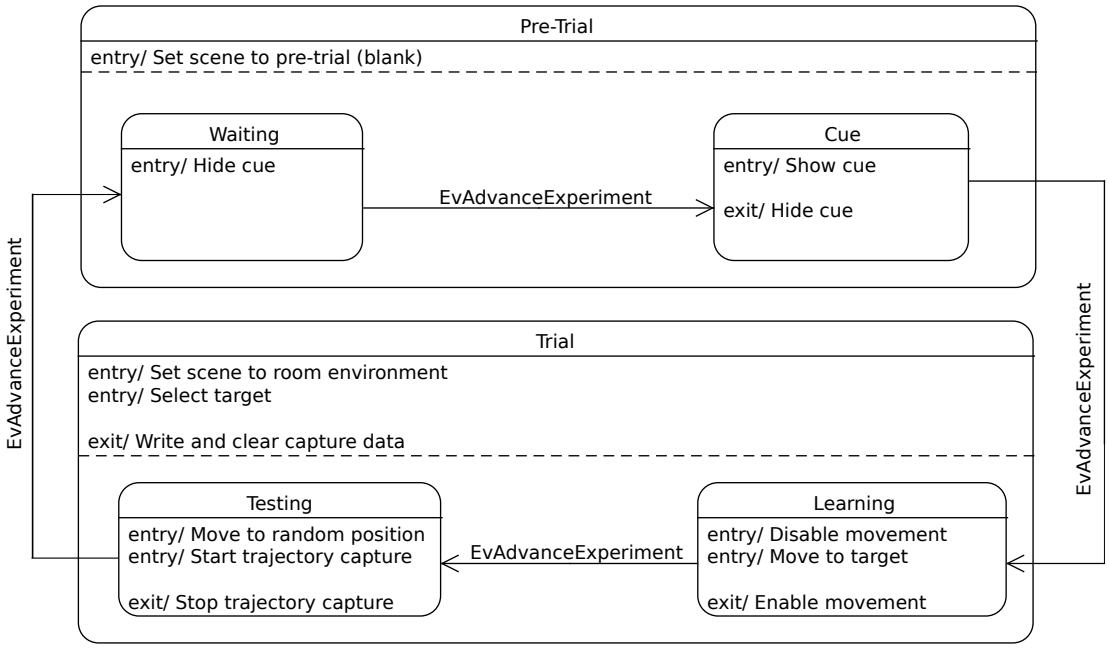


Abb. 14: Statechart des Experiments mit den Zuständen `Waiting` und `Cue` als Zustände innerhalb der „Pre-Trial“-Phase, sowie den Zusänden `Learning` und `Testing` als Zustände innerhalb der „Trial“-Phase; der Zustandsübergang wird durch das `EvAdvanceExperiment` Ereignis ausgelöst.

2.7.4. Ablaufsteuerung und Konfiguration

Im Vorfeld des Experiments werden „Trial-Run“-Konfigurationen erzeugt. Diese enthalten die Parameter zur Erzeugung der Raum-Geometrie, die Koordinaten der Ziele, sowie Start- und Ziel-Koordinaten der einzelnen Versuch-Durchgänge.

Da alle Punkte sowohl als Start und Ziel verwendet werden, kann ein Mindestabstand angegeben werden. Zusätzlich wird eine Vektorgraphik des Raumes mit allen Zielen generiert. Diese kann zur Kontrolle der Lage der Punkte im Raum herangezogen werden.

Die Anzahl der erzeugten Punkte bestimmt die Anzahl der Durchgänge. Insgesamt entstehen $n \cdot (n-1)$ Durchgänge, da jeder Punkt von jedem anderen einmal erreicht werden soll.

Dazu wurde ein flaches Klartext-Konfigurations-Dateiformat mit der zusätzlichen Möglichkeit weitere Dateien einzubinden und ein Parser mit C++-Bindung zur Verwendung im Hauptprogramm des Experiments entwickelt. Das Format hat folgende EBNF:

```
config     = { ( comment | command | assignment ) { newline } } .

comment    = "#" { character | space } .
command    = "@" identifier "(" { argument } ")" .
assignment = identifier { space } "=" { space } value .
identifier = character { character | digit } .
value      = { character | digit | space } .

character  = "\c" .
digit      = "\d" .
space      = "\s" .
newline    = "\n" | "\n\r" .
```

Diese ist in Abb. 15 als „Railroad-Diagramm“ visualisiert.

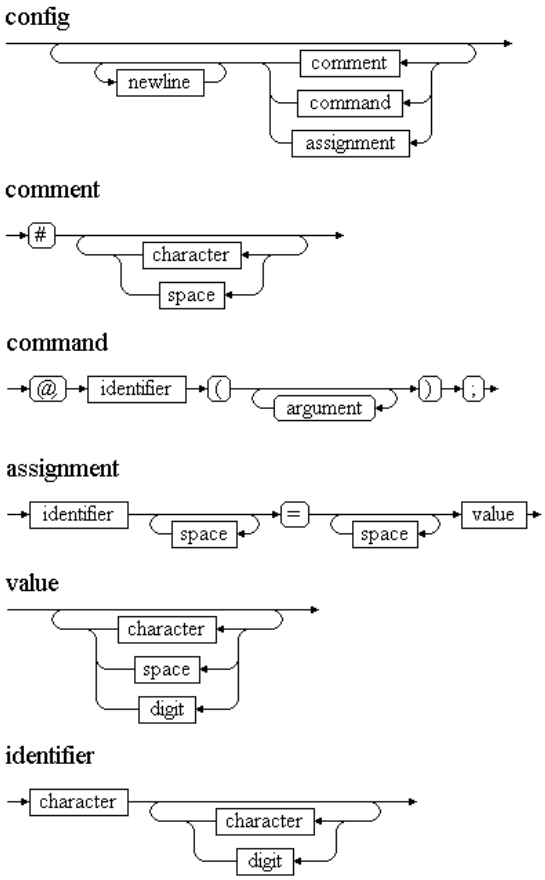


Abb. 15: Railroad-Diagramm der EBNF des Konfigurationsdateiformates

In der vorhandenen Implementation wird einzig der `include`-Befehlt interpretiert. Die referenzierte Datei wird geladen und die Parameterzuweisungen gespeichert. Erneute Zuweisungen in der inkludierenden oder später inkludierten Datei überschreiben den vorherigen Wert. Die Typenkonvertierung erfolgt bei Abfrage des Parameters.

Das Experiment wird über drei Konfigurations-Dateien gesteuert:

`config/default.cfg` Es wird die Basiskonfiguration des Experiments für alle Probanden festgelegt. Dabei sind folgende Parameter entscheidend.

- `dot_lifetime_min` Die minimale Lebenszeit eines Punktes in Millisekunden
- `dot_lifetime_max` Die maximale Lebenszeit eines Punktes in Millisekunden
- `dot_size` Die Größe eines Punkte auf dem Bildschirm in Pixeln.
- `max_visible_dots` Die Anzahl der Punkte, die dem Probanden gleichzeitig präsentiert werden; diese wird konstant auf diesem Wert gehalten.
- `capture_path` Der Pfad unter dem die aufgezeichneten Trajektorien, sowie die Report-Daten der „Trial-Runs“ gespeichert werden

`rooms/trial_N_runs.cfg` Es wird die Konfiguration der Versuchsablaufsteuerung gespeichert. Die gespeicherten Informationen entsprechen den oben beschriebenen. Die Koordinaten der Start- und Zielpunkte werden dabei fortlaufend nummeriert in den Variablen `start_N` und `end_N` gespeichert. Es werden so lange „Trial-Runs“ durchgeführt bis eine Abfrage auf eine Start- oder Zielkoordinate kein Ergebnis mehr liefert. Das Experiment ist damit beendet.

`trials/<name>.cfg` Der name der Konfiguration für jede Versuchsperson ist frei wählbar. Dieser wird als Präfix der Aufzeichnungs- und Report-Dateien verwendet.

2.7.5. Datenaufzeichnung

Die Trajektorien der Versuchspersonen werden für jeden Versuchsdurchlauf aufgezeichnet und gespeichert. Aus dem Dateinamen der „Trial-Konfiguration“ jeder Versuchsperson wird der Präfix der Trajektorieaufzeichnungs- und Reportdateien abgeleitet. Zusätzlich wird ein Zeitstempel im Format `Tag_Monat_Jahr-Stunde_Minute_Sekunde` bezogen auf den Beginn des Experimentes angehängt. Dies verhindert das unbeabsichtigte Überschreiben eines Datensatzes. Für die Trajektorieaufzeichnung wird `_trial_N.cap`, für den Report `_report.csv` angehangen.

Die Bewegung der Versuchsperson im Raum wird zeilenweise im Klartext im Format

```
timestamp x y z azimuth elevation
```

gespeichert.

Die Felder des CSV-Reports sind:

```
run, start_x, start_y, target_x, target_y, pos_x, pos_y, dist
```

Wobei `run` den Versuchsdurchlauf, `pos` die finale Position des Probanden und `dist` den Abstand zum Ziel markieren. Die Datei kann von allen gänigen Tabellenkalkulationsprogrammen verarbeitet werden.

Die Aufzeichnung der Trajektorie erfolgt im Kameramanipulator. Die Bestimmung des Abstandes und die Erzeugung des Reports wird in der Ablaufsteuerung des Experiments durchgeführt (Abb. 14).

2.7.6. Datenaufbereitung

Da die Experiment-Applikation ohne Konfigurations- oder Auswertungs-GUI auskommt, wurde zur Prüfung der Konfiguration und Visualisierung der Versuchsergebnisse ein SVG-Generator in C++ zur Erzeugung von Graphiken entwickelt.

Die Wahl von SVG beruht auf der einfachen Handhabarkeit und dem Vorteil, dass im Druck keine Artefakte oder Unschärfen wie bei Rastergraphikformaten auftreten, da es sich um ein Vektorgraphikformat handelt. Darüberhinaus wird es von einer Vielzahl von Programmen zur Bildbearbeitung unterstützt und kann zur schnellen Sichtung in allen gängigen Webbrowsern dargestellt werden.

Die API ist dabei sehr einfach gehalten und ermöglicht die Erzeugung einer SVG-Graphik in nur wenigen Zeilen Programmcode. Dabei wird zunächst ein `SVG::Document` Objekt als Wurzel des Dokumentbaumes erzeugt. Alle Primitive werden als Kindknoten durch `add( SVG::Node* child )` darin eingehangen. Als Primitive wurden `Text`, `Line`, `Circle` und `Polygon` implementiert. Diese können in einen generischen Transformationsknoten eingehangen werden. Alle Attribute werden als `std::string` gespeichert. Eine Validierung findet nicht statt. Welche Attribute jeweils verwendet werden dürfen, kann unter `http://www.w3.org/TR/SVG/` eingesehen werden. Abb. 16 zeigt das UML-Diagramm der Klassen im SVG-Namensraum.

Der Generator erzeugt einen `std::string`. Die Speicherung oder Ausgabe bleibt dem Nutzer der Bibliothek überlassen.

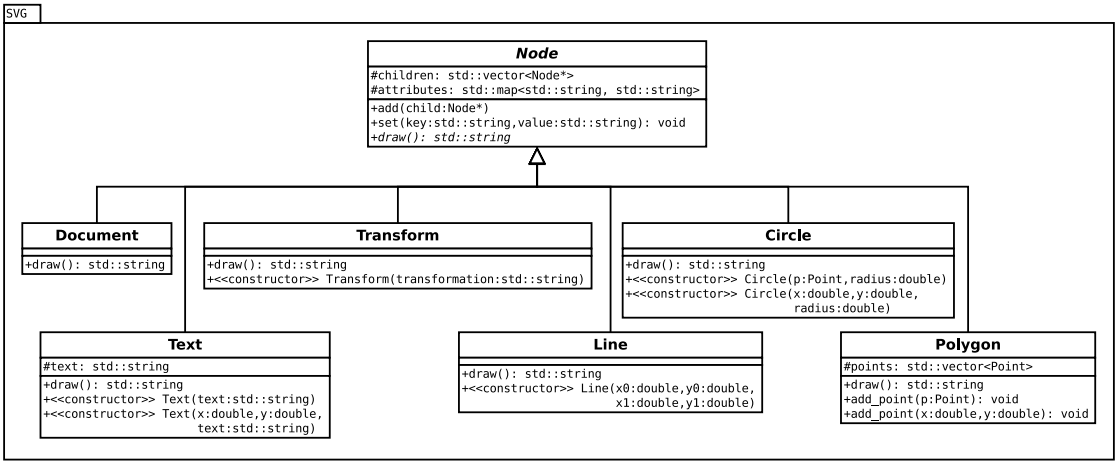


Abb. 16: UML Diagramm des SVG-Generators.

Die Visualisierung der Trajektorie erfolgt durch eine eigens zu diesem Zweck entwickelte Applikation. Diese übernimmt die Parameter der Versuchskonfiguration aus den entsprechenden Konfigurationsdateien und erstellt auf der Basis eines Datensatzes eine SVG-Graphik.

2.8. Durchführung des Experiments

Das Experiment wurde mit vier Teilnehmern durchgeführt. Alle waren mit der Aufgabenstellung und dem theoretischen Hintergrund des Versuches vertraut. Probant vier war bereits erfahren im Umgang mit dem Versuchsaufbau und Experimentablauf und hatte den Versuch mit anderen Parametern der Raumgeometrie schon oft durchgeführt.

Es wurde ein Raum mit einem Drachenviereck als Grundfläche erzeugt. Die Start und Zielpunkte wurden durch die Generator-Applikation zufällig gewählt (Abb. 17). Bei der Erzeugung und Durchführung des Versuches wurden folgende Parameter verwendet:

Raumgeometrie

- Länge: 23 Einheiten
- Breite: 8 Einheiten
- Höhe: 2,50 Einheiten

- Öffnungswinkel: 42°

Starts und Ziele

- Anzahl der Ziele: 5
- Anzahl der Durchgänge: 20
- Minimaler Abstand zwischen Start- und Ziel-Koordinaten: 3,5 Einheiten

Stimulus

- Punktgröße: 3 px
- Minimale Punktlebensdauer: 400 ms
- Maximale Punktlebensdauer: 800 ms
- Maximale Anzahl gleichzeitig sichtbarerer Punkte: 2000

Das Experiment wurde in einem abgedunkelten Raum durchgeführt.

Der Augenabstand des Probanden wurde vor Beginn des Versuchdurchgangs im Startup-Script (A.2) eingetragen.

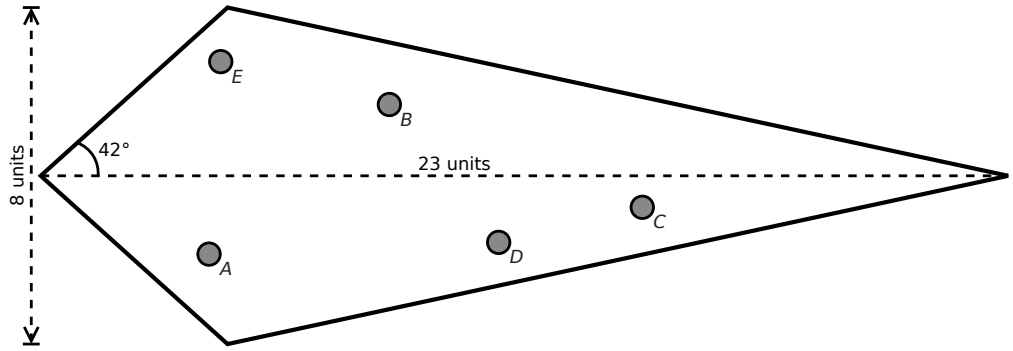


Abb. 17: Automatisch erzeugte graphische Aufbereitung der generierten Start- und Ziel-Koordinaten in der Raumgeometrie.

3. Ergebnisse

Nach dem Abschluss der Versuchsdurchgänge wurden die erhobenen Daten des Experiments ausgewertet und graphisch aufbereitet. Die Rohdaten wurden mit Hilfe der zu diesem Zweck entwickelten Werkzeuge visualisiert, die Trajektoriegraphiken und die Ergebnisse der einzelnen Durchgänge wurden visuell ausgewertet.

Desweiteren wurden die Eigenschaften des z-Buffers untersucht, da dieser ein zentraler Bestandteil des im Experiment verwendeten Verfahrens zu Punktextraktion ist. Dabei wurde insbesonders auf die durch die Projektion der Vertices entstehende Nichtlinearität in der Verteilung der z-Buffer-Werte eingegangen, da diese die entscheidende Fehlerquelle bei der inversen Projektion ist. Diese Analyse soll dabei der Untersuchung der Leistungsfähigkeit des Verfahrens dienen und die potenzielle Optimierungen unterstützen.

Die fehlenden Daten einzelner Versuchsteilnehmer in Abb. 18 a), c), d), f), k) und t) sowie in der tabellarischen Aufbereitung der Fehler (Tab. 3) in den Zeilen 5, 10, 13, 15, 16 und 17 beruhen auf einem Bedienungsfehler bei der Nutzung der Experimentiersoftware.

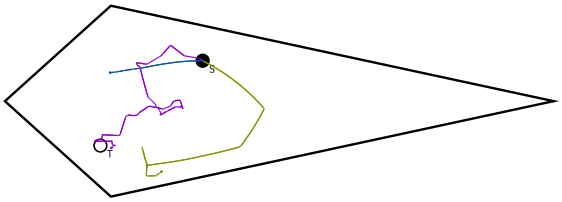
3.1. Ergebnisse der Versuchsdurchführung

Zur Visualisierung der im Experiment aufgezeichneten Trajektorien der Teilnehmer wurde die „ReportPlotter“-Applikation verwendet. Die Trajektorien sind in Abb. 18 a) - t) dargestellt. Sie wurden anhand ihres Zieles sortiert, um durch diese Gruppierung eine leichtere Erkennung von Trends und Taktiken bei der Navigation zu einzelnen Zielen, von unterschiedlichen Startpunkten aus zu ermöglichen.

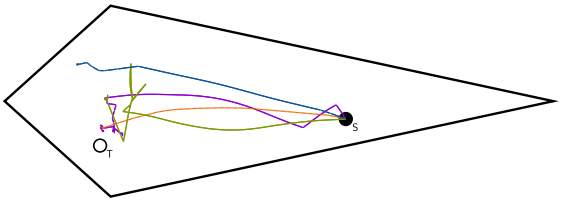
Es ergibt sich dabei eine Gruppierung von jeweils vier Trajektoriegraphiken für einen Zielpunkt. Die Abbildungen 18 a) bis d) zeigen die Trajektorien zum Ziel A, e) bis h) zum Ziel B, i) bis l) zum Ziel C. Die Trajektorie zum Ziel D wird durch Abbildung 18 m) bis p) und zum Ziel E durch q) bis t) dargestellt. Innerhalb einer Gruppe sind die Trajektoriegraphiken von links nach rechts, zeilenweise, chronologisch geordnet.

Bei der Betrachtung der Trajektorien lässt sich erkennen, dass bei jedem Versuchsdurchgang mindestens drei der am Versuch teilnehmenden Probanden in der Lage waren, sich den vorgegebenen Zielen bis auf wenige Einheiten (Units) Entfernung zu nähern. Ausnahmen dazu zeigen die Trajektoriegraphiken in Abb. 18 a) mit zwei von drei, k) mit einer von drei und e) mit einer erfolgreichen Annäherung von vier versuchten Annäherungen zum Ziel.

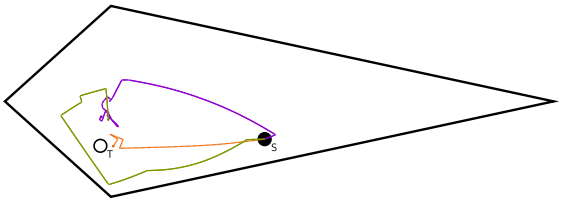
a) $B \rightarrow A$



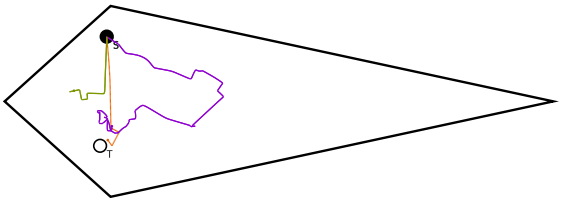
b) $C \rightarrow A$



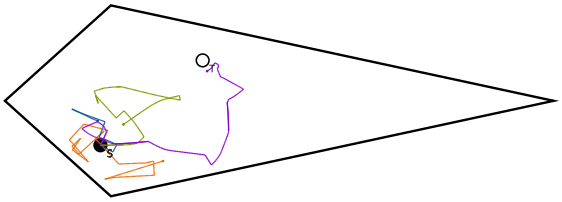
c) $D \rightarrow A$



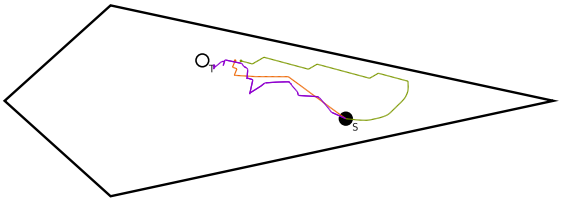
d) $E \rightarrow A$



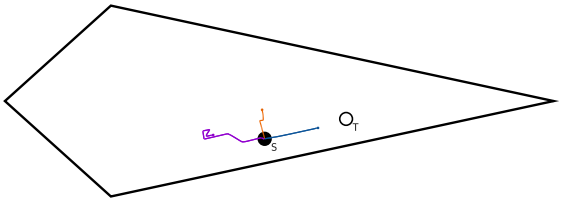
e) $A \rightarrow B$



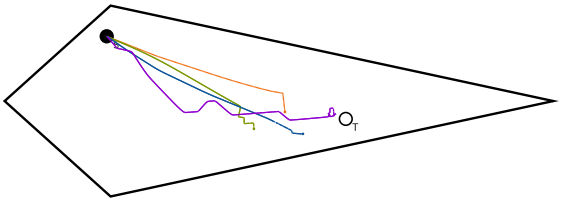
f) $C \rightarrow B$



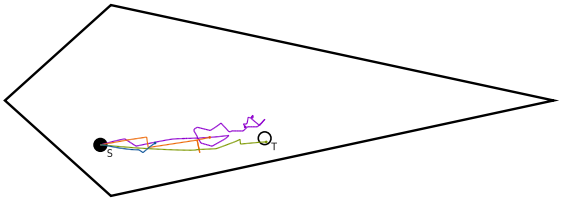
k) $D \rightarrow C$



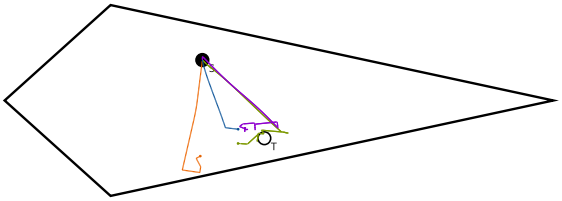
l) $E \rightarrow C$



m) $A \rightarrow D$



n) $B \rightarrow D$



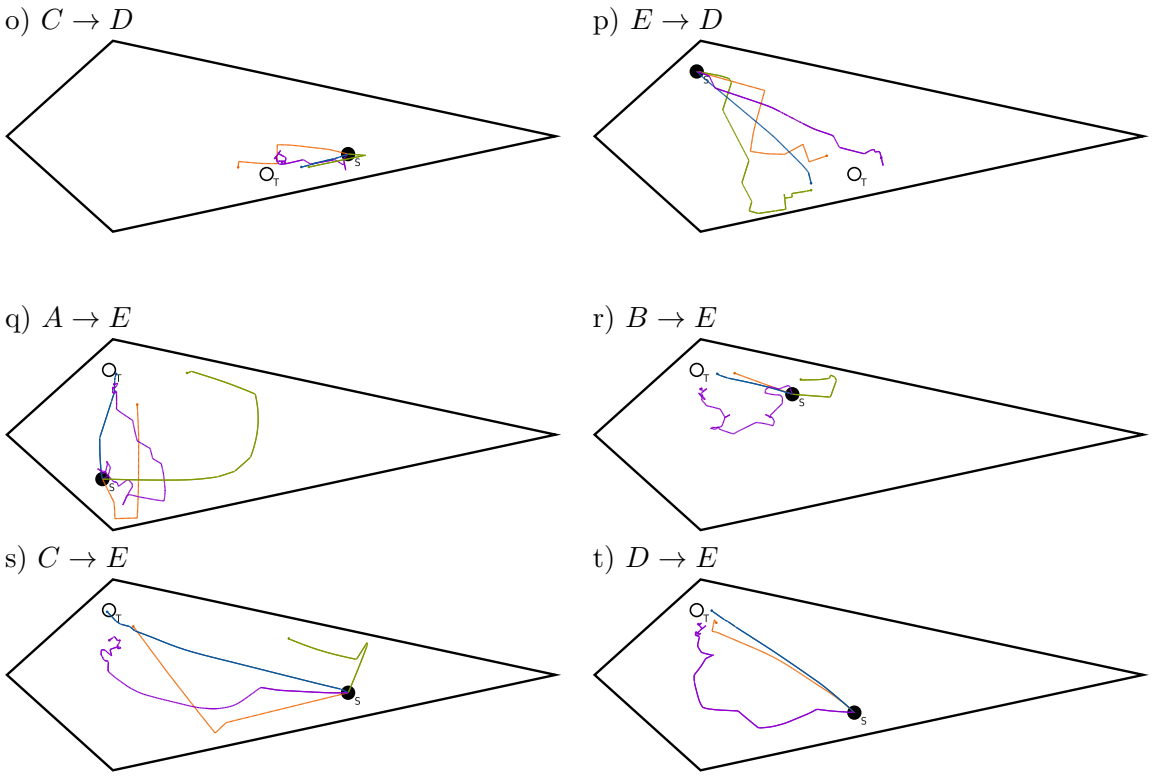


Abb. 18: a) - t) Visualisierung der Trajektorie; Proband 1 - orange, Proband 2 - grün, Proband 3 - blau, Proband 4 - violett; a) Daten von Proband 1 fehlen; k) und t) Daten von Proband 2 fehlen; c), d) und f) Daten von Proband 3 fehlen

Bei der Verfolgung der Trajektorien der Probanden lässt sich erkennen, dass in den meisten Fällen die Annäherung an den Zielpunkt auf direktem Wege vollzogen wurde.

Bei der Betrachtung des Abstandes der Start- und Zielpunkte in Abb. 18 zeichnet sich ab, dass dieser keinen Einfluss auf den Erfolg und die Qualität der Annäherung zum Ziel durch einen Probanden hat. Bei großen Abständen ist ein Trend dahingehend erkennbar, dass der Großteil der Wegestrecke ohne Richtungskorrektur zurückgelegt wurde. Dies zeigt sich im Besonderen in den in Abbildung 18 b), i), l), p), s) und t) dargestellten Trajektorien.

Desweiteren lässt sich bei der Verfolgung der Trajektorien erkennen, dass bei der Wahl der Zielposition der Abstand zum vorgegebenen Ziel tendenziell überschätzt wurde. Die einzige ersichtliche Ausnahme davon sind die Probanden eins und vier in Abbildung 18 h) bei der Navigation zu Ziel B.

Die Qualität der Annäherung lässt sich über den Abstand zwischen dem vom Probanden gewählten Zielpunkt und dem tatsächlichen Ziel in Units feststellen. Dieser Abstand wird im Folgenden als Fehler bezeichnet. Der Fehler der Probanden innerhalb der einzelnen Durchgänge und der Median des Fehlers sind in Tabelle 3 aufgeführt. Die Tabelle ist chronologisch sortiert, was eine Betrachtung der Veränderung des Fehlers der Probanden über den zeitlichen Verlauf des Experimentes ermöglicht.

Bis auf Proband vier starten alle Teilnehmer des Versuches mit einem Fehler im Bereich von 4,5 Einheiten. Dieser nimmt sichtbar im weiteren Verlauf des Versuches ab. Bei Proband eins ist der Fehler bereits nach vier durchgeführten Versuchsdurchläufen deutlich reduziert und dauerhaft niedrig. Ähnliches gilt für die Probanden zwei und drei, wenn auch mit deutlich höheren Schwankungen des Fehlerwertes.

Vergleicht man Proband eins und vier über den zeitlichen Verlauf des Versuchs, so zeichnet sich eine Angleichung der Qualität in der Zielnavigation nach nur fünf durchgeführten Durchläufen ab.

Bei den Probanden eins bis drei liegt der Median des Fehlers im Bereich von 1,4 bis 1,87 Einheiten bei einer Raumgröße von 23 Einheiten in der Länge und 8 Einheiten in der Breite. Bei Probant vier ist der Median des Fehlers kleiner als eine Einheit (Tab. 3).

Abb. 19 veranschaulicht den Median des Fehlers der einzelnen Probanden über alle Versuchsdurchgänge. Der Median des Fehlers ist bei keinem der Probanden größer als 1,87 Einheiten. Relativ zueinander betrachtet ist die Gesamtleistung von Proband vier deutlich besser als die der anderen Versuchsteilnehmer. Die Leistung der Probanden zwei und drei ist annähernd gleich.

Tab. 3: Abstand und Median des Abstandes über alle Versuchsdurchgänge zum Ziel aller teilnehmenden Probanden; fehlende Werte wurden durch einen Bedienfehler verursacht. Die Durchgänge sind chronologisch aufgelistet.

		Abstand zum Ziel			
Durchgang		**Proband 1**	**Proband 2**	**Proband 3**	**Proband 4**
1	$A \to B$	4,534700	4,275290	5,078900	0,486940
2	$A \to C$	2,398310	1,362670	6,070900	1,727430
3	$A \to D$	2,304830	0,166940	4,599090	0,946803
4	$A \to E$	1,872850	3,272970	0,336412	0,640943
5	$B \to A$		2,783570	3,080220	0,475997
6	$B \to C$	0,764668	0,292989	4,785670	0,337807
7	$B \to D$	2,786760	1,125560	1,160280	0,910823
8	$B \to E$	1,594710	4,382160	0,876888	0,822237
9	$C \to A$	0,750016	1,961920	3,533150	0,622753
10	$C \to B$	1,373670	1,618240		0,541128
11	$C \to D$	1,227420	1,797790	1,488540	0,893672
12	$C \to E$	1,225320	7,624060	0,111558	1,265190
13	$D \to A$	0,539845	1,147220		1,225110
14	$D \to B$	0,755730	1,238150	0,782783	1,227390
15	$D \to C$	3,541640		1,232110	5,595990
16	$D \to E$	0,977019		0,629502	0,602322
17	$E \to A$	0,401430	2,565270		0,971061
18	$E \to B$	0,593067	0,138488	2,285320	2,068790
19	$E \to C$	2,567990	3,860300	1,900760	0,704001
20	$E \to D$	1,387810	1,940310	1,857290	1,312380
Median:		**1,373670**	**1,869050**	**1,857290**	**0,902248**

Schließlich soll noch die Qualität der Zielnavigation in Verbindung mit der Verteilung der von den Probanden gewählten Zielpositionen untersucht werden: Trägt man den Median der Koordinaten der von den Probanden gewählten Zielpunkte als Mittelpunkt eines Kreises mit dem Median des Fehlers als Radius in die Graphik der Raumgeometrie ein (Abb. 20), kann man eine deutliche Häufung der gewählten Zielpunkte in der Nähe der vorgegebenen Ziele erkennen.

Die Abstände der Ziele zu Ecken des Raumes sind in drei Kategorien aufgeteilt: Die Ziele A und E haben einen geringen, B einen mittleren, C und D haben den größten Abstand. Anhand der Größe der Fehlerkreise lässt sich erkennen, dass der Fehler bei Zielen, die sich in der Nähe einer Ecke befinden, kleiner ist als bei Zielen, deren nächstgelegene

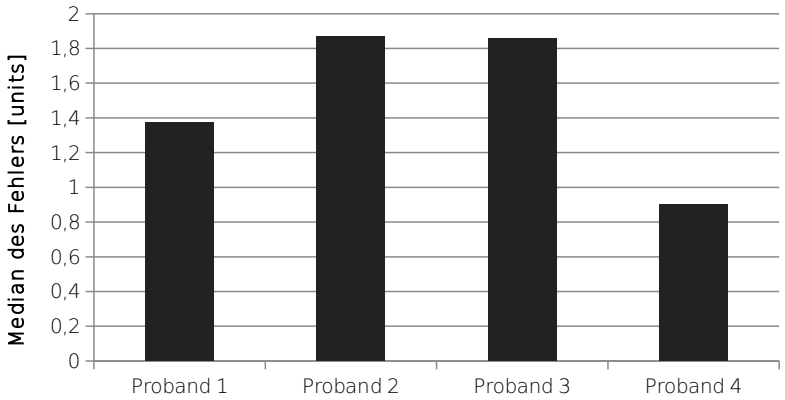


Abb. 19: Median des Abstandes zum Ziel über alle Versuchsdurchgänge für alle Probanden

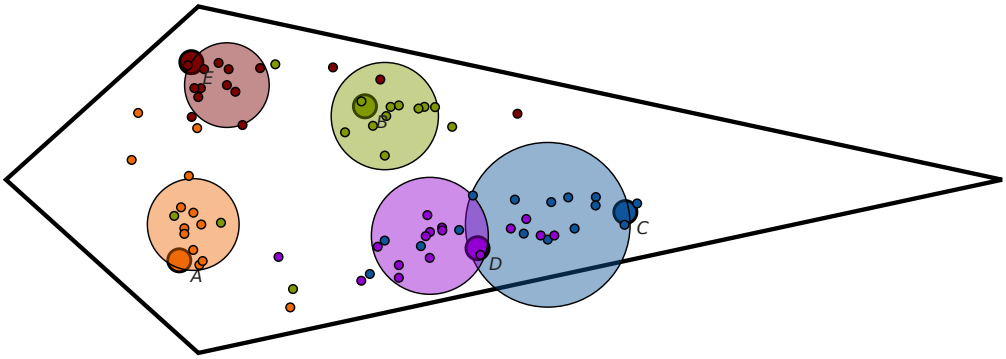


Abb. 20: Ziele A (orange), B (grün), C (blau), D (violett) und E (rot), die von den Probanden während aller Durchgänge gewählten Zielpunkte sind in der gleichen Farbe wie das jeweilige Ziel gehalten. Der Mittelpunkt des gleichfarbigen Kreises ist der Median aus den von den Probanden gewählten Zielpunkten, der Radius ist der Median der Fehler (Abstände zwischen den gewählten Zielpunkten und dem tatsächlichen Ziel)

Ecke weiter entfernt liegt. Es zeichnet sich ein Zusammenhang zwischen der Position eines Zieles im Raum, relativ zu den umgebenden Ecken, und der Größe des gemachten Fehlers ab.

Betrachtet man die Streung der zu Ziel C gewählten Zielpunkte, so deutet sich eine Verteilung auf einer zur nächstgelegenen Wand parallelen Geraden an.

3.2. Auswertung der z-Buffer-Analyse

Um ein gutes Ergebnis bei der inversen Projektion eines Punktes zu erhalten, ist die Wahl der z-Near- und z-Far-Ebene und die Konfiguration des z-Buffers entscheidend. Durch die bei der Projektion entstehende Nichtlinearität in der Verteilung der z-Werte kann es bei einer unzureichenden Genauigkeit, bedingt durch eine zu gering gewählte Wortbreite in Bit, zu Aliasingeffekten kommen. Diese entstehen durch die Abbildung von unterschiedlichen z-Werten auf die gleiche Integerrepräsentation und hat führt zur Bildung von Treppen entlang der z-Achse in der gezeigten Szene.

Im Folgenden werden die Eigenschaften eines 16, 24 und 32-Bit z-Buffer berechnet.

Nehmen wir einen 32-Bit z-Buffer mit $zNear = 1$ und $zFar = 100$ an. Der maximale Fehler bei der in 2.4.3 beschriebenen Punkerzeugung liegt im Bereich von $zFar$. Durch Einsetzen der Werte in (14) erhalten wir einen maximalen Fehler von rund $0,23 \cdot 10^{-5}$ Einheiten auf der z-Achse.

$$100 - \frac{\left(2^{32} - 1\right) \cdot 100 \cdot 1}{(2^{32} - 1) \cdot 100 - (2^{32} - 2) \cdot 100 + (2^{32} - 2) \cdot 1} = \frac{1650}{715827899} \approx 0,23 \cdot 10^{-5} \quad (24)$$

Tabelle 4 zeigt die maximalen Fehler in Abhängigkeit von z-Near, z-Far und den verschiedenen z-Buffer-Bit-Tiefen.

Tab. 4: Maximaler Fehler bei verschiedenen Bit-Tiefen in Abhängigkeit von z-Near und z-Far.

		Maximaler Fehler *(units)*		
z-Near	**z-Far**	**16-Bit**	**24-Bit**	**32-Bit**
$0,2$	$50,0$	$0,18926$	$0,74207 \cdot 10^{-3}$	$0,289 \cdot 10^{-5}$
$2,0$	$50,0$	$0,01830$	$0,07153 \cdot 10^{-3}$	$0,27 \cdot 10^{-6}$
$5,0$	$50,0$	$0,00687$	$0,02682 \cdot 10^{-3}$	$0,9 \cdot 10^{-7}$
$10,0$	$50,0$	$0,00305$	$0,01192 \cdot 10^{-3}$	$0,4 \cdot 10^{-7}$

Trägt man den z-Wert der normalisierten Gerätekoordinaten der z-Werte in Clip-Koordinaten zwischen z-Near und z-Far auf, lässt sich die Nichtlinearität der Projektion erkennen (Abb. 21). Die Nichtlinearität hat einen höheren Dynamikumfang im Bereich der z-Near-Ebene gegenüber der z-Far-Ebene zur Folge.

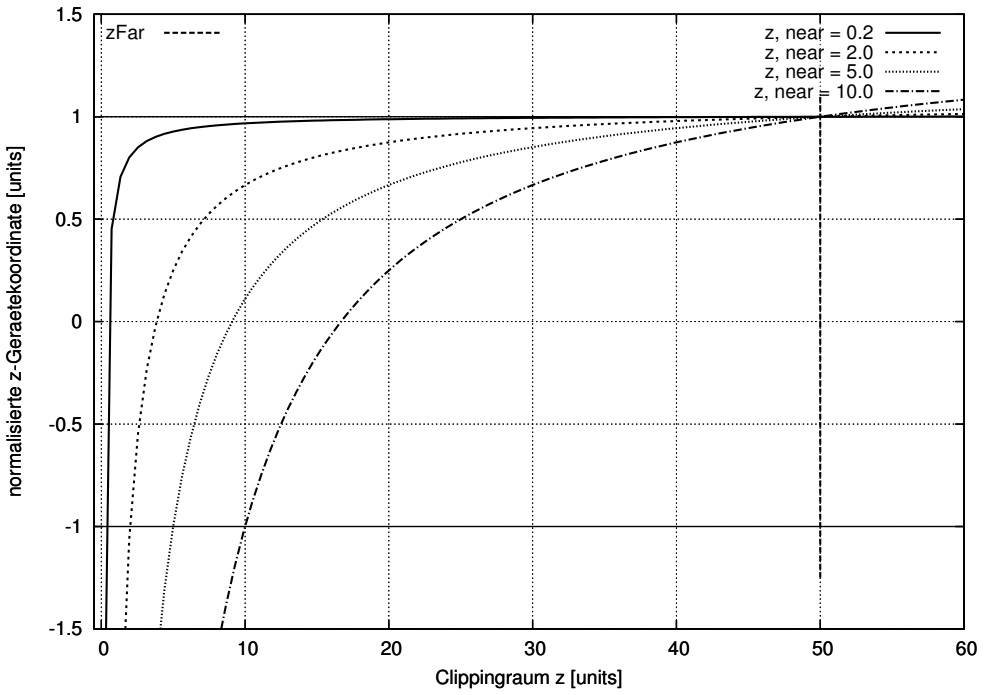


Abb. 21: z-Werte in Abhängigkeit von z-Near (0,2, 2, 5, 10 und festem z-Far = 50)

4. Diskussion

In dieser Arbeit galt es, den Einfluss von Tiefensignaturen auf die Navigation zu einem gegebenen Ziel auf Basis eines „Walk-To-Cued-Place“-Experiments zu untersuchen und den zur Durchführung benötigten Versuchsaufbau mitsamt der dazugehörigen Software zu realisieren. Zur Beschränkung der Wahrnehmung der Probanden auf die Tiefensignaturen wurde ein „Random-Dot-Limited-Lifetime“-Stimulus verwendet. Dazu wurde ein Verfahren zur Erzeugung von Punktwolken auf einer beliebigen Raumgeometrie aus den im z-Buffer gespeicherten Daten entwickelt.

Das durchgeführte Experiment diente dabei der Validierung des Aufbaus und der Implementierung der Experimentiersoftware.

4.1. Psychophysisches Experiment

Anhand der bei der Durchführung des Experiments erhobenen Daten wird deutlich, dass die Probanden in der Lage waren, sich im Raum zu orientieren, das vorgegebene Ziel zu erkennen und dorthin zu navigieren. Dies war bereits während der Entwicklung des dabei zum Einsatz kommenden Verfahrens absehbar, da der Verlauf der Wände und die Eckpunkte des Raumes bei der Betrachtung der Szene durch das Stereoskop für einen Menschen mit normaler Tiefenwahrnehmung gut erkennbar sind.

Die Leistung der Probanden nahm nach bereits wenigen Durchgängen tendenziell zu: Die Probanden eins bis drei hatten im Gegensatz zu Proband vier keine Übung im Umgang mit dem Versuchsaufbau und kein Training der Orientierungsfähigkeit in der virtuellen Umgebung. Sie konnten allerdings ihren Fehler, gemessen am Abstand des gewählten Zielpunktes zum vorgegebenen Ziel, innerhalb der ersten vier Versuchsdurchgänge deutlich reduzieren. Der durchgehend geringe Fehler des vierten Probanden lässt sich durch einen Trainigseffekt erklären. Vergleicht man jedoch den Fehler zwischen Proband eins und vier, so schnitt letzterer bereits nach sechs Durchgängen nicht mehr wesentlich besser ab.

Die Mediane der von den Probanden gemachten Fehler liegen zwischen 0,9 Einheiten bei Proband vier und 1,87 Einheiten bei Proband zwei im Kontext eines 23 Einheiten langen und 8 Einheiten breiten, drachenförmigen Raumes, gemessen jeweils an der längsten, respektive breitesten Stelle. Der insgesamt geringe Fehler aller Probanden beim Aufsuchen des vorgegebenen Zieles lässt darauf schließen, dass die Tiefensignaturen der Ziele

zur Navigation ausreichen. Die Angleichung der Leistung der ungeübten Probanden an die des geübten innerhalb eines kurzen Zeitraums lässt sich dadurch erklären, dass die Probanden zunächst den Umgang mit der Experimentiersoftware erlernen mussten. Die augenscheinliche Orientierungslosigkeit der Probanden eins bis drei in Abb. 18 e) lässt sich darauf zurückführen, dass dies der erste Durchgang war. Es empfielt sich also den Probanden nach der Instruktion zunächst eine virtuelle Umgebung im Rahmen eines Probedurchlaufs zu präsentieren, damit sie sich mit der Benutzung der Versuchsapparatur vetraut machen können.

Betrachtet man weiter die aus den Daten erzeugten Trajektorie-Graphiken, so scheint es, dass die meisten Probanden sich nach dem Versetzen an ihre Start-Position zunächst im Raum orientierten, um dann ohne größere Umwege direkt zu ihrem Ziel zu laufen und ihre Richtung dabei nur mäßig zu korrigieren. Dies spricht dafür, dass das vorgegebene Ziel schon bereits aus der Entfernung heraus erkannt wird.

Betrachtet man die Streung der von den Probanden gewählten Zielpunkte im Kontext des gemachten Fehlers bei der Zielnavigation, lässt sich ein Zusammenhang zwischen der Position des vorgegebenen Zielpunktes relativ zu den Eckpunkten des Raumes beobachten: Je weiter ein Ziel von den Ecken des Raumes entfernt liegt, desto größer wird der Fehler. Dies wird besonders bei den Zielen C und E deutlich. Der Radius der Fehlerkreise in der aus den erhobenen Daten erzeugten Streugraphik ist für das Ziel C am kleinsten und für Ziel E am größten. Der Radius entspricht dabei dem Median des Fehlers bei der Wahl des Zielpunktes in der Navigationsphase. Das Ziel E lag in der erzeugten Raumgeometrie am nächsten zu einer Ecke des Raumes, Ziel C war am weitesten von einer Ecke entfernt. Es ist ersichtlich, dass der Fehler bei Punkt C deutlich höher war als bei allen anderen Punkten. Der Fehler im Abstand zur Wand an diesem Punkt ist dabei deutlich kleiner als der Fehler, der durch die Abschätzung des Abstandes zu den Ecken des Raumes entsteht. Die von den Probanden gewählten Punkte verteilen sich dem Anschein nach auf einer zur Wand parallelen Graden, was darauf hindeutet, dass die Entfernung des Zieles zur Wand durchgehend gut eingeschätzt wurde. Dies spricht dafür, dass die Position eines Zieles innerhalb des Raumes von den Probanden relativ zu einer oder mehreren Ecken abgeschätzt wurde. Anders als bei Gillner et al. (2008) Verteilen sich die von den Probanden gewählten Zielpunkte nicht Kreisförmig um die vorgegebenen Ziele, sondern die der nächstliegenden Ecke abgewandten Seite des Zieles wird von den Probanden stark bevorzugt. Die Ausnahme ist Ziel D, hier verteilen sich die Punkte ebenfalls wie bei C auf einer zur nächsten Wand, im Abstand des Zieles, parallelen Graden - allerdings zu beiden Seiten des Zieles. Daraus kann man schließen,

dass der Abstand zu den Ecken von den Probanden als Landmarke gewählt wurde.

Bei der Zielnavigation lässt sich der Trend erkennen, dass die Nähe zum vorgegebenen Ziel überschätzt wird. Es scheint in der virtuellen Umgebung zu Fehlern bei der Abschätzung von Abständen zu kommen, welche sich in Form einer Unterschätzung von Entfernungen manifestiert. Eine mögliche Erklärung dafür könnte sein, dass durch das Fehlen von Informationen wie Texturierung und die perspektivische Verzerrung der Textur, Beleuchtung und Schattierung der Proband zur Abschätzung der Entfernung allein die Disparität der lebenszeitbeschränkten Punkte heranziehen konnte. Dies könnte die Abschätzung der Entfernungen erschweren. Eine weitere Erklärung dafür könnte die Nichlinearität des z-Buffers sein. Der Fehler der dargestellten Punkte wird dabei ebenfalls mit zunehmendem Abstand zum Betrachter größer. Da zum Zeitpunkt der Durchführung des Experiments nur ein 24-Bit z-Buffer erzeugt werden konnte, liegt der maximale Fehler bei der Punkterzeugung im Bereich von $0,74 \cdot 10^{-3}$ Units auf der z-Achse. Dies, könnte sich ebenfalls negativ auf die Fähigkeit, größerer Entfernungen einzuschätzen, auswirken.

Diese Schlüsse basieren auf einer visuellen Auswertung der im Experiment gesammelten Daten. Ein Vergleich mit der Untersuchung von Gillner et al. (2008), deren Probanden sich in einem runden, bzw. eckigen Raum mit einem Farbverlauf an den Wänden orientieren sollten, ist schwierig, da bei Gillner et al. (2008) zusätzlich zu den Tiefeninformationen bedingt durch eine farbliche Absetzung von Fußboden und Decke zu den Wänden der Intensitätsgradient und die physische Bewegung der Probanden, die sich mit einem Head-Mounted-Display im Raum bewegen konnten, zur Navigation verwendet werden konnten. Im vorliegenden Versuch wurden die Rauminformationen weiter verringert, dennoch war es den Probanden möglich, sich im Raum zu orientieren und zu navigieren.

Zur Validierung der hier vorgeschlagenen Erklärungen muss allerdings eine statistische Analyse durchgeführt werden. Die vorhandene Datenlage reicht dafür jedoch nicht aus. Der Versuch sollte daher mit einer größeren Probandenanzahl und mehr Versuchsdurchläufen erneut durchgeführt werden.

In späteren Versuchen kann darauf aufbauend untersucht werden, wie sich die Leistung der Probanden bei der Ziel-Navigation ändert, wenn man entweder ausschließlich oder zusätzlich monokulare Tiefeninformation dem visuellen System zur Verarbeitung zugänglich macht.

4.2. Software

Die im Experiment erhobenen Daten haben auch gezeigt, dass die im Rahmen dieser Arbeit entwickelte Software geeignet ist, um die Frage zu untersuchen, ob Menschen ausschließlich auf Basis von Tiefeninformationenen in der Lage sind, gelernte Orte zu identifizieren und wieder aufzusuchen.

Das bei diesem Experiment verwendete Verfahren nutzt dabei den Vorteil, dass moderne Graphikkarten auf die Rasterisierung von Polygonen optimiert sind. Es wird dabei ausgenutzt, dass bei der Rasterisierung einer Szene Tiefeninformationen anfallen und im z-Buffer gespeichert werden. Auf deren Basis kann aufgrund der Symmetrieeigenschaft der „OpenGL-Rendering-Pipeline“ die Szenengeometrie rekonstruiert und eine Punktwolke darauf erzeugt werden. Beschränkt man die Lebenszeit der einzelnen Punkte und randomisiert bei der Auswahl ihre Position im Bildschirmkoordinatensystem, so erhält man einen „Random-Dot-Limited-Lifetime“-Stimulus, der im Experiment zur Unterdrückung von monokularen Bildinformationen verwendet wird.

Da der z-Buffer vor dem Zeichnen der Punktwolke nicht gelöscht wird, wird automatisch ein Sichtbarkeitstest des Punktes durchgeführt. Dies ermöglicht auch virtuelle Umgebungen mit undurchsichtigen Trennwänden oder mit mehreren zu einem Irrgarten verbundenen Räumen. Die Implementierung des Verfahrens wurde deshalb unabhängig zu der des Experiments gestaltet. So kann die `RandomDotRenderer`-Klasse zusammen mit der OpenSceneGraph-Bibliothek zur Darstellung beliebiger Szenen, die auch aus 3D-Modellier-Software exportiert werden können, verwendet werden. Die im Kameramanipulator implementierte Kollisionsdetektion verhindert dabei das Durchschreiten von Wänden.

Die Komplexität des Verfahrens wächst im Gegensatz zu einem auf Raycasting basierenden Algorithmus linear: Die Szenengeometrie wird im beschriebenen Verfahren einmal gezeichnet und die Tiefeninformation durch ein Auslesen einer Speicheradresse durchgeführt. Die Umrechnung der Tiefeninformation bei bekannten Bildschirmkoordinaten ist mit wenigen Rechenoperationen durchführbar. Auch die Invertierung der ModelViewProjection-Matrix muss nur einmal für alle in einem Durchgang erzeugten Punkte durchgeführt werden. Daher sinkt die Performance selbst bei einer hohen Anzahl gleichzeitig sichtbarer Punkte nur unwesentlich. Selbst eine Darstellung von 100000 Punkten läuft auf der bei der Entwicklung hauptsächlich verwendeten vier Jahre älteren Hardware (Intel(R) Core(TM)2 Duo CPU E8500 @ 3.16GHz, nVidia GeForce 9600 GT) ohne merklichen Leistungseinbruch. Die Bewegung im Raum läuft genauso flüssig

wie bei den im Experiment verwendeten 2000 Punkten. Daher können auch komplexe Szenerien, Szenen mit mehreren Räumen und hochpolygonale Modellen als lebenszeitbegrenzte Punktwolke dargestellt werden. Der limitierende Faktor ist die Füllrate der Graphikhardware. Ebenfalls sind dynamische Szenen mit beweglichen Elementen mit diesem Verfahren realisierbar. Die Möglichkeit zur direkten Auswahl der zu erzeugenden Punkte im Bildschirmkoordinatensystem hat den Vorteil, dass sich die Anzahl der gleichzeitig sichtbaren Punkte von selbst im Rahmen von Toleranzschwellen konstant hält.

Es wurden ebenfalls Überlegungen angestellt, einen Raycasting-Ansatz zu verwenden, in dem ein Volumendatensatz der Szenengeometrie erzeugt wird. Dies hat sich allerdings nicht bewährt: Die erzeugte Datenmenge wächst dabei im ungünstigen Fall sehr schnell in Abhängigkeit von der Abtastauflösung und der Anzahl Schnittflächen in der Szene. Die Szenengeometrie muss zusätzlich gerastert werden damit der z-Buffer gefüllt wird und dadurch Verdeckungen und undurchsichtige Objekte ermöglicht werden. Außerdem kann es sonst bei gekrümmten Flächen zur Bildung von Punktclustern kommen, was wiederum zur Bildung von erkennbaren Konturen in der Szene führt. Dynamische Szenen sind ebenfalls nicht möglich. Darüberhinaus muss die Auswahl der sichtbaren Punkte aus diesem Datensatz durch ein View-Frustum-Culling-Verfahren erfolgen, um die Anzahl der sichtbaren Punkte konstant zu halten.

Ein für die Qualität der im implementierten Verfahren verwendeten inversen Projektion entscheidender Faktor ist die Wahl eine geeigneten z-Buffer-Konfiguration. Die Untersuchung der Eigenschaften des z-Buffers hat ergeben, dass bei einer zu niedrig gewählten z-Buffer-Auflösung, bedingt durch die bei der Projektion auftretenden Nichtlinearität Aliasingartefakte auftreten können. Dies ist eine Folge der Abbildung von verschiedenen z-Koordinaten auf die gleiche Integerrepräsentation. Der minimale Punktabstand der invers projezierten Punkte auf der z-Achse durch die im z-Buffer gespeicherten Daten ist daher beschränkt und hängt von der Wahl der zNear- und zFar-Ebene, sowie der Wortbreite des z-Buffers ab. Die Auswertung der der z-Buffer-Analyse ergab, dass dieser minimale Abstand bei einem 24-Bit z-Buffer, wie er in diesem Experiment verwendet wurde, im Bereich von $0,74 \cdot 10^{-3}$ Einheiten auf der z-Achse liegt.

Eine weitere Möglichkeit, das Verfahren zu optimieren, ist die Auslagerung der Umrechnung der z-Buffer-Werte auf die GPU. Wendet man ein Fragmentshader-Programm auf die gesamte Szenengeometrie an, so hat man innerhalb des Shader-Programms Zugriff über die eingebaute Variable `gl_FragCoord` auf den dem Rasterisierer bekannten z-Wert und die Bildschirmkoordinate. Dies ermöglicht eine parallelisierte inverse Projek-

tion vieler Punkte direkt auf der Graphik-Hardware. Zur Ausgabe werden die x, y und z Koordinaten in drei Render-Targets über `gl_FragData[0..2]` geschrieben. Diese Renderbuffer können über wiederum über Pixelbufferobjects ausgelesen werden. Eine weitere Optimierung des Verfahrens kann durch die Linearisierung des z-Buffers mit Hilfe von Shaderprogrammen erreicht werden.

Der in der vorliegenden Arbeit verwendete Drachen-Viereck-Raum besitzt eine geringe Komplexität, die sich mit sechs Vierecken oder 12 Dreiecken beschreiben lässt. Bei dieser geringen Polygonzahl könnte ein Raycastingverfahren auch in Echtzeit durchführbar sein. Gegenüber dem Raycasting bietet das entwickelte Verfahren den Vorteil, dass nicht nur simple, sondern auch komplexe Geometrien ohne Leistungseinbruch dargestellt werden können.

Es wurde in dieser Arbeit gezeigt, dass der entwickelte Algorithmus zur Extraktion von Punkten aus der Raumgeometrie auf Basis der im z-Buffer vorhandenen Daten ausreicht, um eine virtuelle Umgebung zu erschaffen in der sich Probanden ausschließlich anhand von Tiefensignaturen orientieren und zu einem vorgegebenen Ziel navigieren können. Das dabei beschriebene Verfahren ist robust und ermöglicht auch die Darstellung von komplexen virtuellen Umgebungen als „Random-Dot-Limited-Lifetime“-Stimulus.

Bei der Auswertung der beim Experiment erhobenen Rohdaten wurde die im Rahmen der Entwicklung der Experimentiersoftware enstandene Wergzeugsammlung zur Visualisierung von Trajektorien und Erzeugung von Streudiagrammen verwendet. Die dafür entwickelte SVG-Bibliothek wurde auch zur Erzeugung der Ziel-Kontroll-Graphik verwendet.

Die Modifikation des `osgUtil`-Moduls der OpenSceneGraph-Bibliothek erweitert diese um die Möglichkeit der Erzeugung von Stereopaaren für die Betrachtung auf der im Rahmen dieser Arbeit gebauten Ein-Spiegel-Variante des von Wheatstone (1852) erfundenen klassischen Stereoskopaufbaus und ermöglicht dessen Verwendung auch mit anderer Software, welche diese Bibliothek verwendet.

Literatur

P. Bourke. Calculating stereo pairs. 1999. URL http://paulbourke.net/miscellaneous/stereographics/stereorender/. Online, Stand 20.05.2012.

B. A. Cartwright und T. S. Collett. Landmark learning in bees. *Journal of Comparative Physiology A: Neuroethology, Sensory, Neural, and Behavioral Physiology*, 151: 521–543, 1983. ISSN 0340-7594. URL http://dx.doi.org/10.1007/BF00605469. 10.1007/BF00605469.

K. Cheng. A purely geometric module in the rat's spatial representation. *Cognition*, 23 (2):149–178, 1986.

V. Durier, P. Graham, und T. S. Collett. Snapshot memories and landmark guidance in wood ants. *Current biology*, 13:1614–1618, 2003.

ETSI. Digital Video Broadcasting (DVB); Frame Compatible Plano-stereoscopic 3DTV, Januar 2012. URL http://www.etsi.org/deliver/etsi_ts/101500_101599/101547/01.01.01_60/ts_101547v010101p.pdf. ETSI TS 101 547.

S. Gillner, A. M. Weiß, und H. A. Mallot. Visual homing in the absence of feature-based landmark information. *Cognition*, 109(1):105 – 122, 2008. ISSN 0010-0277. doi: 10.1016/j.cognition.2008.07.018. URL http://www.sciencedirect.com/science/article/pii/S001002770800173X.

R. Hammerstone, M. Craighead, K. Akeley, und The Architecture Review Board. Arb_vertex_buffer_object specifications. Online, Stand 05.05.2012, October 2010. URL http://www.opengl.org/registry/specs/ARB/vertex_buffer_object.txt.

J. Juliano und J. Sandmel. Ext_framebuffer_object specifications. Online, Stand 03.05.2012, April 2008. URL http://www.opengl.org/registry/specs/ARB/vertex_buffer_object.txt.

J. Kollin und A. Hollander. Re-engineering the wheatstone stereoscope. *SPIE Newsroom*, 2007. ISSN 18182259. doi: 10.1117/2.1200702.0673. URL http://spie.org/x8365.xml?ArticleID=x8365.

H. A. Mallot. *Sehen und die Verarbeitung visueller Information: Eine Einführung*. Vieweg computational intelligence. Vieweg, 2000. ISBN 3528156597.

H. A. Mallot. *Script Kognitive Neurobiologie, Wintersemester 2005 / 2006*. Lehrstuhl für Kognitive Neurowisenschaft, Universität Tübingen, 2005.

OpenGL.org. Depth buffer precision. Online, Stand 02.05.2012, January 2012. URL http://www.opengl.org/wiki_132/index.php?title=Depth_Buffer_Precision&oldid=4593.

D. Shreiner, M. Woo, J. Neider, T. Davis, und OpenGL Architecture Review Board. *OpenGL Programming Guide*. Addison-Wesley, 6th edition, 2008.

N. Tinbergen und W. Kruyt. Über die Orientierung des Bienenwolfes (Philanthus triangulum Fabr.) III: Die Bevorzugung bestimmter Wegmarken. *Zeitschrift für vergleichende Physiologie*, 25:292–334, 1938.

C. Wheatstone. Constributions to the physiology of vision. on some remarkable, and hitherto unobserved, phenomena of binocular vision. *Philosophical Transactions of the Royal Society of London*, 1852.

A. Anhang

A.1. Geänderte computeLeftEyeProjectionImplementation-Methode

```
osg::Matrixd SceneView::computeLeftEyeProjectionImplementation(const osg::Matrixd& projection) const
{
    double iod = _displaySettings->getEyeSeparation();
    double sd = _displaySettings->getScreenDistance();
    double scale_x = 1.0;
    double scale_y = 1.0;

    if (_displaySettings->getSplitStereoAutoAdjustAspectRatio())
    {
        switch(_displaySettings->getStereoMode())
        {
            case(osg::DisplaySettings::HORIZONTAL_SPLIT):
                scale_x = 2.0;
                break;
            case(osg::DisplaySettings::VERTICAL_SPLIT):
                scale_y = 2.0;
                break;
            default:
                break;
        }
    }

    if (_displaySettings->getDisplayType()==osg::DisplaySettings::HEAD_MOUNTED_DISPLAY)
    {
        // head mounted display has the same projection matrix for left and right eyes.
        return osg::Matrixd::scale(scale_x,scale_y,1.0) *
               projection;
    }
    else
    {
        // all other display types assume working like a projected power wall
        // need to shjear projection matrix to account for asymetric frustum due to eye offset.
        return osg::Matrixd(1.0,0.0,0.0,0.0,
                            0.0,1.0,0.0,0.0,
                            iod/(2.0*sd),0.0,1.0,0.0,
                            0.0,0.0,0.0,1.0) *
               // Mirror left eyes image throgh using -scale
               osg::Matrixd::scale(-scale_x,scale_y,1.0) *
               projection;
    }
}
```

A.2. Startup-Script der Experiment-Applikation

```
#!/bin/sh

#
# stereo configuration
#

# Distance of screen and eye distance
export OSG_SCREEN_DISTANCE=0.79
export OSG_EYE_SEPARATION=0.065

# Screen
export OSG_SCREEN_WIDTH=0.60
export OSG_SCREEN_HEIGHT=0.34

# Enable horizontal split stereo
export OSG_STEREO=ON
export OSG_STEREO_MODE=HORIZONTAL_SPLIT
export OSG_SPLIT_STEREO_HORIZONTAL_SEPARATION=0

# Force use of PBOs
export OSG_ASSIGN_PBO_TO_IMAGES=ON

TRIAL_FILE=$@

if [ ! -f "$TRIAL_FILE" ]; then
 echo "Missing trial configuration file."
 echo "Usage: $0 trials/trial_file.cfg"
 exit
fi

LD_PRELOAD="../../libosgUtil.so" ./experiment --trial $TRIAL_FILE
```