Automatische Bestimmung von fixierten Objekten bei verzerrter Projektion auf eine zylindrische Leinwand

Studienarbeit im Fach Bioinformatik

Sommersemester 2005
Universität Tübingen

vorgelegt von
Christian Rothländer

21. November 2006

Zusammenfassung

Im Zuge der Doktorarbeit "Kompensatorische Augen- und Kopfbewegungen bei Patienten mit hemianopen Gesichtsfeldausfällen" von Gregor Hardieß am Lehrstuhl der kognitiven Neurowissenschaften von Prof. H. A. Mallot entstand eine für Versuche mit Probanden geeignete Projektionsanlage. Dieses System besteht aus einer gekrümmten Leinwand, auf die von zwei digitalen Projektoren eine dreidimensionale Landschaft projiziert wird, in der sich die Versuchsperson mit Hilfe eines Joysticks bewegen kann. Dabei werden mittels Eye-[1] und Headtracker[2] die Blickwinkel auf die Leinwand bestimmt.

Das Ziel dieser Arbeit war es, einen Algorithmus zu entwickeln, der aus den am Probanden gemessenen und berechneten[3] Blickwinkeln das auf der Leinwand fixierte Objekt im 3D-Raum bestimmt. Dabei konnte teilweise auf Ideen aus der Studienarbeit von David Hrabal zurückgegriffen werden, der ein ähnliches Problem (in diesem Fall 2D auf 3D-Mapping) in seiner Arbeit gelöst hatte. Bislang erfolgt diese Auswertung manuell mittels eines Videofilms, was einen deutlich erhöhten Aufwand bei der Auswertung der Versuche mit sich bringt.

[1]ASL Mobile 501 eye tracking system

[2]ARTtrack Dtrack

[3]siehe Studienarbeit von Stefanie Mayer am selben Lehrstuhl

Inhaltsverzeichnis

1 Einleitung **4**

1.1 Intuition der Versuchsentwicklung 4

1.2 Aufbau des Versuchsraums 5

1.2.1 Hardware 5

1.2.2 Bilderstellung in der Software 6

1.2.3 Datenerhebung während des Versuchs 7

1.3 Versuchsdurchführung 7

1.4 Ziel der Studienarbeit 8

2 Realisierung **9**

2.1 Grundlegendes 9

2.2 Formate 9

2.2.1 Format der Eingabedaten 9

2.2.2 interne Datenformate 11

2.2.3 Ausgabeformat 12

2.3 Funktionen 12

2.3.1 int size() 12

2.3.2 void init(String Dateiname) 13

2.3.3 private void makecoordinates() 13

2.3.4 void nextjoypos() 13

2.3.5 void nextimage() 14

2.3.6 private void naechsterblick() 14

2.3.7 private pfSeg naechsterblick(double α, double ε) 14

2.3.8 vector⟨intersection_data⟩ rundumblick(double α, double ε) . 17

2.3.9 private void isecTest(pfSeg line) 18

2.3.10 void auswertung(int Fenstergroesse, float Radius) 19

2.3.11 void writeoutput(String Dateiname) 20

3 Bedienung/ Einbindung in eigene Programme **20**

3.1 Parameter 20

3.2 Eingliederung der Klasse in eigene Programme 21

4 Überprüfung des Algorithmus 22

A shareddata.h 24

B nameit.h 25

C nameit.cpp 26

D ausgabe.txt 32

E testmain.txt 33

Index 35

1 Einleitung

1.1 Intuition der Versuchsentwicklung

Im Rahmen der Erforschung von Gesichtsfeldsausfällen werden am Patienten verschiedene Tests ausgeführt, um zu ermitteln, in wie weit das Sehvermögen eingeschränkt ist. Diese Tests hatten bislang das Ziel, die Größe des betroffenen Netzhautgebietes zu ermitteln. Es zeigte sich allerdings, dass die physikalische Größe des Gebietsausfalls nicht viel über tatsächliche Beeinflussung auf das tägliche Leben aussagt. So lassen sich einige dieser Ausfälle durchaus durch vermehrte Kopfbewegungen auffangen. Um diese Kompensationen zu erforschen, wurden verschiedene Experimente durchgeführt, die neben interaktiver Computergrafik eine Projektionsanlage nutzen, die einen großen Teil des Gesichtsfelds der Versuchsperson ausfüllt. Ziel dieser Arbeit ist die Entwicklung eines Programms, welches ein automatisiertes Auswerten der Blickpunkte ermöglicht.

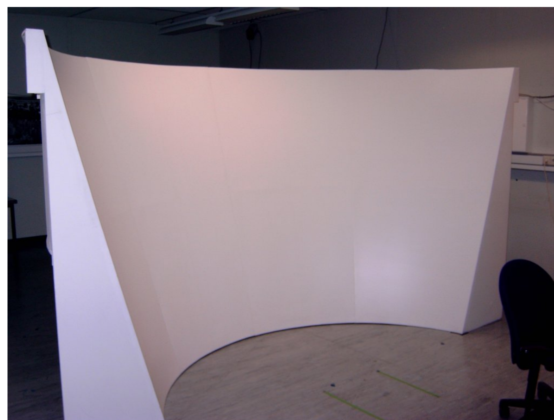

Abbildung 1: Leinwand

1.2 Aufbau des Versuchsraums

1.2.1 Hardware

Im Labor befindet sich eine Projektionsleinwand (Abb. 1), ein Eyetrackersystem (Abb. 2) und ein Headtrackersystem. Der Eyetracker ist am Kopf der Versuchsperson montiert und zeichnet mittels Infrarotabtastung die Augenbewegungen auf. An diesem System ist ein Kugelgeweih befestigt, das vom Trackingsystem durch 6 Kameras im Raum geortet wird. Dieses System bestimmt mit hoher Genauigkeit die Position des Kopfes im Raum und ermöglicht die Berechnung der Sichtwinkel.

Die Projektionsleinwand wird von zwei digitalen Projektoren (Beamern) (Abb. 3) ausgeleuchtet, die je nach Versuch verschiedene Umgebungen zeigen. Darüberhinaus kann der Proband mit Hilfe eines Joysticks durch den virtuellen Raum navigieren.

Abbildung 2: Eyetracker

Abbildung 3: Beamer

1.2.2 Bilderstellung in der Software

Zur Visualisierung wird der SGI OpenGL Performer verwendet, welcher die Möglichkeit bietet, eine große Anzahl von Datenformaten für 3D-Szenen einzulesen. Nach dem Einlesen der Szene erfolgt eine von Sabine Gillner und Hansjürgen Dahrmen entwickelte Bildverzerrung über verschiedene Algorithmen (Abb. 4). Dieser Schritt kompensiert die von der kegelschnittförmigen Leinwand physikalisch herbeigeführte Verzerrung des Bildes. Hierzu werden von einer vorher definierten Anzahl virtueller Kameras[4] Bilder der dreidimensionalen Landschaft berechnet, die im Backbuffer der OpenGL-Pipeline abgelegt werden. Im Anschluss werden diese Grafiken als Textur ausgelesen und über Winkelberechnungen so verzerrt, dass das Bild dem vor der Leinwand sitzenden Betrachter wieder unverzerrt und realistisch erscheint.

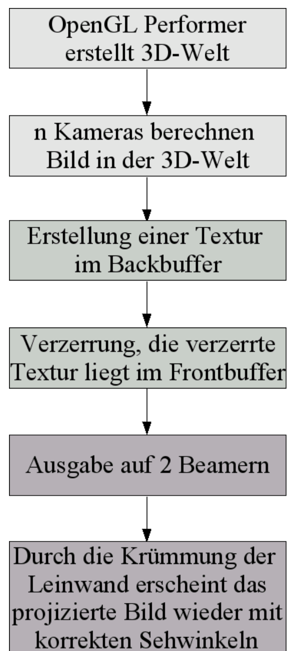

Abbildung 4: Bilderstellung: Der Weg vom 3D-Szenengraphen zum Bild auf der Leinwand

[4] Parameter "CAM"

1.2.3 Datenerhebung während des Versuchs

Während eines Versuches werden die von den Trackern ermittelten Bewegungen des Kopfes der Versuchsperson aufgezeichnet. Zusätzlich werden vom Hauptprogramm die durch den Joystick eingegebenen Bewegungstrajektorien gespeichert. Im Anschluss an den Versuch werden diese Datensätze mittels verschiedener Matlab-Scripte zusammengeführt und in einer Datei gespeichert, die es ermöglicht das Experiment ohne Probanden zu replizieren. Bei dieser Wiederholung des Versuches ist es auch möglich rechenintensive Erweiterungen und Analysen auszuführen, die ihre Daten nicht in Echtzeit (das System läuft während der Versuchsaufzeichnung mit mit 60Hz) berechnen können. Diese Trennung von Versuch und Analyse verringert die Fehleranfälligkeit der Software während des Versuchs und ermöglicht eine vielfältigere Analyse im nachhinein. Auch wird es so möglich in Zukunft entstehende Analysemethoden mit alten Datensätzen zu verwenden, ohne die den Probanden nochmals zu rekrutieren. Die Szene selber wird aus statischen Dateien eingelesen, mobile Objekte werden im Code des Hauptprogramms eingefügt und von diesem zur Laufzeit in der Landschaft verschoben. (Abb. 5)

1.3 Versuchsdurchführung

Der Proband wird auf einem mit einer Hebevorrichtung modifizierten Autositz vor der Leinwand platziert. Durch diese Hebevorrichtung ist es möglich, die Augenhöhe auf 1,20 m einzustellen. Der Augenabstand zur Leinwand beträgt in diesem Fall bei horizontaler Blickrichtung 1,62 m. Im Anschluss wird der Versuchsperson der Eyetracker aufgesetzt, an dem mit einer Magnetschnappvorrichtung ein "Geweih" aus reflektiven Kugeln zur Positionsbestimmung montiert ist. Nach der Kalibrierung von Eyetracker und Geweih im Raum, ist es möglich den Sehstrahl der Versuchsperson zu ermitteln. Der Proband selber wird durch die Trackervorrichtungen nur

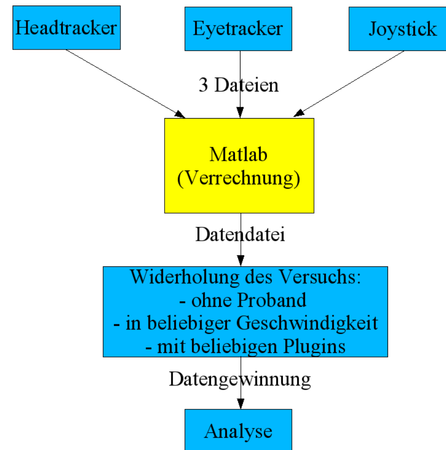

Abbildung 5: Verarbeitungsschritte während der Datenverarbeitung: Die Berechnung von Head- und Eyetrackerdaten erfolgt durch Algorithmen, die in Stefanie Mayers Studienarbeit näher erläutert werden, das Zusammenführen aller Daten erfolgt mittels eines von Gregor Hardieß entwickelten Konverters.

unwesentlich in seiner Sicht- und Bewegungsfreiheit eingeschränkt, so dass eine weitestgehend realitätsnahe Aufzeichnung der Seh- und Kopfbewegungen möglich ist.

1.4 Ziel der Studienarbeit

Das Ziel meiner Arbeit ist die Entwicklung einer automatische Datenauswertung sowie die Erweiterung des Grundprogramms. Bislang muss sich die auswertende Person die Videoaufzeichnung des Versuches anschauen und Bild für Bild feststellen, welches Objekt in der 3D-Szene vom Sichtkreuz des Eyetrackers markiert wird. Dieser Vorgang soll automatisiert werden. Hierzu wurde im Rahmen dieser Arbeit einen Algorithmus entwickelt, der die Verzerrung rückgängig macht, so dass man aus den Blickwinkeln das beobachtete Objekt in der dreidimensionalen Computergraphik berechnen kann.

2 Realisierung

2.1 Grundlegendes

Der erste Arbeitsschritt war die Erweiterung des vorhandenen Programms, um Analysemodule zu ermöglichen. Hierzu wurden die Rahmendefinitionen des rein prozeduralen C-Codes in eine Headerdatei ausgelagert. Dies ist nötig, da der SGI-Performer auf Multithreading optimiert ist, und nur so ein gemeinsamer Datenzugriff mehrerer Klassen auf die Landschafts- und Systemdaten möglich wird. Darüberhinaus werden in dieser Headerdatei (shareddata.h) die globalen Variablen (Leinwandgröße, Winkel, Kameraanzahl, Texturgröße, u.a) vereinbart. Dies bietet auch den großen Vorteil, dass bei Veränderungen der Leinwand nur in einer Datei Veränderungen nötig sind, um das System an die neuen Gegebenheiten anzupassen (Anhang A).

2.2 Formate

2.2.1 Format der Eingabedaten

Es wurde ein Datenformat der Eingabedaten für das Plugin entworfen. Hierfür wurde ein einfaches Textformat gewählt. Ein Binärformat würde zwar den benötigten Speicherplatz reduzieren, allerdings würde dies gleichzeitig eine aufwendige Modifikation der Matlabscripte bedeuten, und gleichzeitig die Kommentierbarkeit durch den Experimentleiter einschränken. Die Datei enthält aufeinander folgende Double-Werte in der Reihenfolge:

α, ε, **X, Y, Z, Heading, Pitch, Roll, Kommentar**

α und ε sind die Azimuth und Elevationswinkel ausgehend von einem 0-Vektor,

der in die Mitte der Leinwand zeigt. Positive Winkel zeigen in diesem Fall nach rechts bzw. oben (siehe Abb. 6). So spannen die durch Augen- und Kopfbewegungen determinierten Winkel α und ε ein Korrdinatensystem auf der Leinwand auf. X, Y und Z sind die aktuellen Koordinaten im 3D-Modell, Heading, Pitch und Roll die Neigung der aktuellen Position im 3D-Modell. Diese 6 Joystickwerte werden während des Versuchs direkt aus dem SharedData-Objekt ausgelesen. Der Kommentar und alle Zeichen dahinter bis zum Zeilenumbruch werden vom Parser ignoriert, was bei eventuellen Erweiterungen die Rückwärtskompatibilität der Dateien sichert, solange sich die Bedeutung der ersten 8 Werte jeder Zeile nicht verändert.

Es wird darüberhinaus vorrausgesetzt, dass für jedes Frame genau eine Zeile in dieser Datei zu finden ist. Somit wird ein zusätzlicher Zeitindex überflüssig, da bei den im System benutzen 60 Hz der Frameindex der aktuellen Zeilenzahl entspricht.

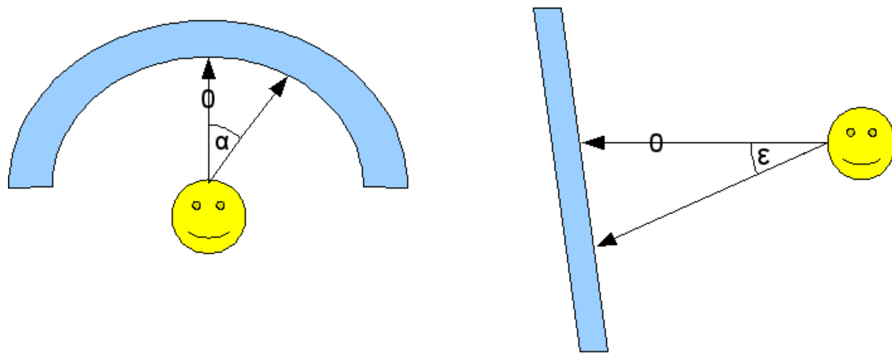

Abbildung 6: Erklärung der verwendeten Winkel: links die horizontale Ebene (Blick von oben), rechts die vertikale Ebene (Blick von der Seite)

2.2.2 interne Datenformate

Neben dem Performer-spezifischem Shared-Objekt, das die 3D-Szene und alle Parameter enthält, werden intern noch weitere Datenblöcke benutzt. Hierzu gehört der Vector of Vector of Double **inputs**, der die Informationen aus der Eingabedatei beinhaltet. Der innere Vector enthält jeweils α und ε, dahinter kommen die 6 Joystickwerte.

Für die Speicherung einer Intersection (siehe Kap. 2.3.8) verwendet man den struct intersection_data. Dieser enthält den Namen des getroffenen Objekts, die Koordinaten des Treffers sowie einen Marker, ob eine Fixierung vorliegt oder nicht.

Des Weiteren findet sich ein Vector of intersection_data namens **outputs**, der die Treffer des genauen Sehstrahls enthält

Ein weiterer Vector of Vector of intersection_data **auswertungen** enthält alle im Umfeld des genauen Sehstrahls getroffenen Objekte, einschließlich des Treffers des genauen Sehstrahls.

Zusätzlich dazu gibt es ein Vector of Vector of Double **coordinates**. Dieser enthält, sofern die Funktion benutzt wurde, zu jedem Winkelpaar die Koordinaten auf der realen Leinwand (x,y,z).

Für die spätere Auswertung finden sich darüber hinaus die beiden Datenblöcke fixierungsstatistik (Vector of Map of ⟨String,int⟩) sowie der Vector of String outputfixationen, welcher nach erfolgter Auswertung zu jedem Frame die möglichen Fixierungen enthält. Näheres hierzu findet sich im entsprechenden Kapitel dieser Arbeit.

Alle internen Datenformate laufen indexsynchron, das heißt zu einem beliebigen Index bekommt man aus allen 3 Datenblöcken die zur entsprechenden Framenummer gehörenden Werte. Die Zählung erfolgt von 0 ab aufwärts.

2.2.3 Ausgabeformat

Das verwendete Ausgabeformat (siehe Anhang D) ist dem Eingabeformat ähnlich. Es ist wiederum eine einfache Textdatei mit aufeinander folgenden Werten. Um die Lesbarkeit für Tabellenkalkulationen zu verbessern, sind hier die Werte durch Tabulatoren getrennt. Das Format baut sich wie folgt auf:

Zu Beginn jeder Datei findet sich eine dreizeilige Erklärung des Aufbaus der Datei.

Im Anschluss hieran befindet sich ein dreizeiliger Parameter-Block: Die erste Zeile besteht aus der Kennung "PARAMETER", die zweite Zeile enthält die Parameter, in denen das Strahlenbündel geschossen wurde (Öffnungswinkel und Anzahl der Strahlen), die dritte Zeile enthält die Parameter, die zur Bestimmung einer Fixierung benutzt wurden (Fenstergröße und maximaler Winkel).

Hierauf folgen die Datenblöcke. Diese bestehen immer aus einer Kennung "DATEN", gefolgt von einer Zeile, in der die Eingabewinkel (α, ε) angegeben sind. Die nächsten Zeilen, deren Anzahl von den Parametern der Strahlenbündel abhängt, geben die Ergebnisse der einzelnen Strahlen wieder. Hier findet sich die Framenummer, gefolgt vom genauen α und ε, gefolgt von der Position in der ein Objekt getroffen wurde (X,Y,Z), gefolgt vom Namen des getroffenen Objekts.

Abgeschlossen wird jeder Datenblock durch eine Zeile, in der die von der Auswertungsfunktion ermittelten möglichen Fixierungen vermerkt sind.

2.3 Funktionen

2.3.1 int size()

Diese Funktion gibt die Anzahl der eingelesenen Datensätze zurück.

2.3.2 void init(String Dateiname)

In der init-Funktion ist der Eingabedateiparser implementiert. Mit einem gültigen Dateinamen aufgerufen, schreibt sie die Dateiinhalte in das interne Dateiformat. Sie kann auf Wunsch über das Define "KOORDINATENBERECHNUNG" auch die Berechnung der Durchstoßpunkte der Sichtstrahlen durch die Leinwand in Weltkoordinaten starten. Die Funktion sollte am Anfang des Main-Blocks des Hauptprogramms gestartet werden.

2.3.3 private void makecoordinates()

Die Funktion makecoordinates() berechnet aus den Winkeln im Eingabevektor die Weltkoordinate, an der der Sichtstrahl auf die Leinwand trifft. Die Funktion wird vom Hauptalgorithmus nicht benutzt und verbleibt aus Gründen der Erweiterbarkeit im Code. Intern führt diese Funktion eine abgewandelte Form der Leinwandverzerrung aus.

2.3.4 void nextjoypos()

Nextjoypos() kopiert die gespeicherten Joystickkoordinaten aus dem internen Datenformat in das Hauptprogramm. So wird es möglich, die beim Experiment mit der Versuchsperson gewonnenen Trajektorien nachzufahren. Die Funktion sollte in der Grafikschleife nach pfSync() und vor dem Auslesen der Sichtmatrizen aufgerufen werden, da veränderte Joystikkoordinaten die Sichtmatrizen beeinflussen. **Achtung:** der Aufruf von nextjoypos() ohne nextimage() liefert immer die selben Joystickdaten, da nur in nextimage() ein Index weitergezählt wird, der von beiden Funktionen ausgewertet wird.

2.3.5 void nextimage()

Nextimage() veranlasst das Berechnen des nächsten Sichtstrahls oder Sichtstrahlbündels und zählt im Anschluss daran den Index hoch. Die Funktion sollte in der Grafikschleife nach dem Laden der aktuellen Sichtmatrizen und vor dem Erstellen des aktuellen Bildes durch pfFrame() aufgerufen werden.

2.3.6 private void naechsterblick()

Diese Funktion führt nacheinander die beiden folgenden Funktionen mit den aktuellen Winkeln (α, ε) aus. Sie wird standardmäßig ausgeführt und kapselt die eigentlichen Aufrufe.

2.3.7 private pfSeg naechsterblick(double α, double ε)

Um den Blickstrahl zu berechnen werden folgende Schritte abgearbeitet, die Beispiele beziehen sich auf die aktuell gewählten Parameter:

1. **Umrechnen der Winkel α und ε durch Verschieben des Koordinatenursprungs an die linke, untere Leinwandkante:** Dieser Schritt ist notwendig, da wir in den weiteren Berechnungen mit Verhältnissen arbeiten. Die Formeln hierfür sind: $\alpha_{neu} = \alpha + 75$, $\varepsilon_{neu} = \varepsilon + 42,74$. Falls sich die Leinwand oder andere Abstände ändern sollte sind hieran Veränderungen vorzunehmen. *Bsp: aus $\alpha = 15°$ und $\varepsilon = -10°$ werden $\alpha_{neu} = 90°$ und* $\varepsilon_{neu} = 32,74°$

2. **Berechnen der Verhältnisse:** Es wird berechnet, wieviel Prozent der aktuelle Winkel (α_{neu} im Verhältnis zur gesamten Leinwand einnimmt. Dabei wird berücksichtigt, dass der horizontale Öffnungswinkel der Leinwand $150°$ und der vertikale Öffnungswinkel $66,3°$ sind. (siehe Abb. 7) *Bsp: ein α_{neu} von $75°$ entspricht einem horizontalen Verhältnis (ϕ) von 50%*

3. **Ermitteln der für die aktuelle Textur zuständigen internen Kamera:** Das horizontale Verhältnis (ϕ) multipliziert mit der internen Kameraanzahl (entspricht n in der Abb.4) liefert uns den Index der Kamera, die für die aktuelle Texturkachel zuständig ist. Die Übergangskanten zwischen den Einzeltexturen werden immer der nächsthöheren Kamera zugeordnet. *Bsp: 50% wäre bei 6 internen Kamera in der Textur 3 zu finden*

4. **Berechnen des horizontalen Verhältnisses der in 3. ermittelten Kamera:** Abhängig von der Anzahl der verwendeten Texturkameras ergibt sich die Breite des Field of View (Leinwandöffnungswinkel/ Kameraanzahl). Analog zu Punkt 2 wird nun das Verhältnis zur aktuellen Texturkachel berechnet *Bsp: Da in den aktuellen Standardeinstellungen jeder interne Kamera ein Field of View von* $25°$ *hat:* $\alpha = 80°$*:* $80° - 3 \cdot 25° = 5°$, $\frac{5°}{25°}$ *= 20% =* φ

5. Mit den in Punkt 2 und 4 berechneten Verhältnissen (φ und ϕ) lassen sich nun die Durchstoßpunkte des Sehstrahls durch die Near- bzw. Farplane ermitteln. Hierzu lädt man zuerst die Viewmatrix der aktuellen internen Kamera. Die Durchstoßpunkte lassen sich nun über einfache Winkelberechnung ermitteln. Es wird hierbei für beide Achsen ein dreidimensionaler Vektor erstellt (pnear/pfar). Diese Vektoren werden mit dem Produkt aus Near- bzw. Farplaneabstand und einem Offset gefüllt. Die Offsets berechnen sich folgendermaßen (Abb. 8):

$$(dx) = 2 \cdot tan(\alpha) \cdot (\varphi - 0.5) \tag{1}$$

$$(dy) = tan(\varepsilon) \tag{2}$$

   Die beiden Vektoren werden daraufhin über eine interne Funktion mit der Viewmatrix verarbeitet und als pfSeg zurückgegeben.

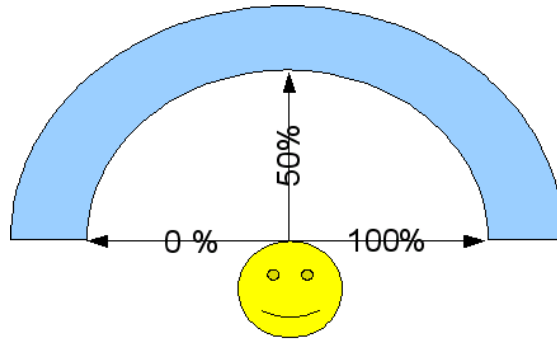

Abbildung 7: Umrechnung von Winkelgrad in Prozentwerte. Am horizontalen Beispiel ist die Umrechnung linear möglich, da der Abstand zwischen Auge und Leinwand konstant ist, in der vertikalen ist eine weitere Umrechnung nötig, die sich aus Sinus- und Kosinussatz herleiten lässt.

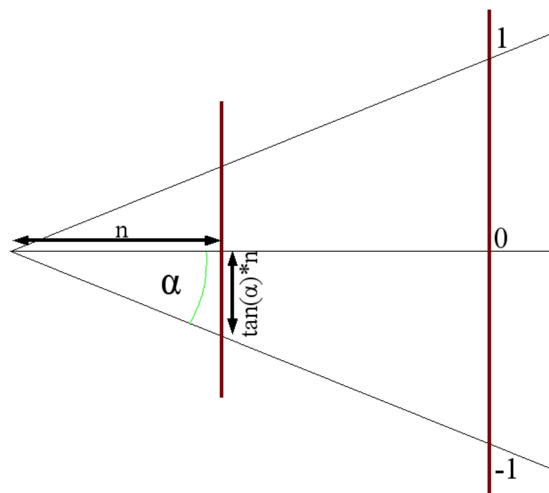

Abbildung 8: Texturberechnung in x-Ausrichtung. Das Offset für Near- und Farplane ergibt sich aus dem Tangens. Diese Berechnung erfolgt analog zur Arbeit von David Hrabal.

2.3.8 vector⟨intersection_data⟩ rundumblick(double α, double ε)

Da das menschliche Auge beim Betrachten eines Objekts nie genau trifft, sondern die direkte Umgebung des fixierten Punktes zur Objekterkennung miteinbezieht, ist es wichtig, diesen Faktor in die Analysefunktion einzubauen. Die hierzu verwendete Funktion berechnet in Abhängigkeit der beiden globalen Defines AUSW_RADIUS und AUSW_SCHRITTE um den genauen Sichtstrahl liegende benachbarte Sichtstrahlen. AUSW_RADIUS bestimmt hierbei den Radius des hierfür verwendeten Kreises, AUSW_SCHRITTE die Anzahl der Schritte, die der Algorithmus in diesem Kreis benutzt. Durch diese beiden Variablen lässt sich die Auswertung auf ein realitätsnahes Niveau bringen, so dass ein Sichtstrahlbündel statt ein einzelner Sichtstrahl benutzt wird. Hierbei ist zu beachten das die Schrittzahl nicht zu hoch gewählt wird, da der Aufwand exponentiell ansteigt. Bei der Standardeinstellung von Radius $= 1,5°$ und 3 Schritten wird pro $0,5°$ ein Sichtstrahl geschossen. Diese Einstellung schießt also pro gemessenem Sehstrahl 25 Sehstrahlen in die Szene. (Abb. 9)

Die so gewonnenen intersection_data werden in einem Vector gebündelt zurückgegeben.

Darüberhinaus wird hier die Fixierungsstatistik (siehe 2.2.2) für die spätere Auswertung angelegt. Hierzu wird für jedes getroffene Objekt im Sichtstrahlbündel ein Element bestehend aus dem Namen und einer Zählvariable erstellt. Falls das Objekt im vorherigen Bild auch getroffen wurde, wird die Zählvariable des entsprechendes Objekts für das aktuelle Frame um eins erhöht, falls nicht, wird mit der Zählung bei eins begonnen. Anhand der Zählvariablen wird es so in der Auswertungsfunktion möglich die Fixierungen herauszulesen.

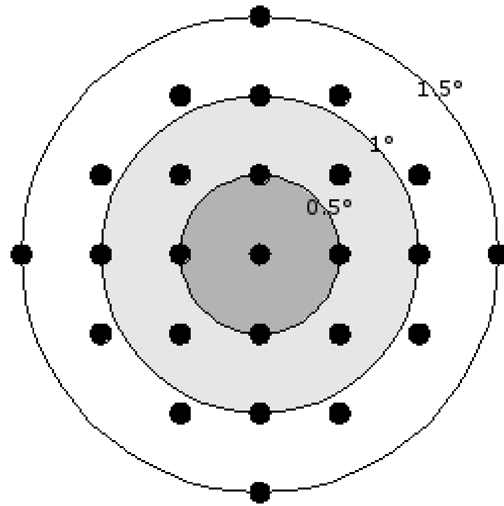

Abbildung 9: Das verwendete Muster in der Standardeinstellung von 3° Öffnungswinkel sowie einer Schrittzahl von 3. In diesem Muster werden die Sichtstrahlen von der Funktion rundumblick() in die Szene gelegt, um fixierte Objekte zu erkennen.

2.3.9 private void isecTest(pfSeg line)

Diese Funktion wurde aus dem Studienarbeitsprojekt von David Hrabal übernommen und angepasst. Die Funktion legt den Sehstrahl, definiert durch (dx) und (dy) in die Szene und sucht nach Intersections mit den in der Szene vorhandenen Objekten. Sie baut aus der Linie ein "SegmentationSet" auf und führt den im SGI Performer implementierten Schnittest durch. Aus dem zurückgegebenen pfHit lassen sich über die query()-Funktion die Eigenschaften des getroffenen Objektes auslesen. In diesem Fall wird die Node-Information benutzt um den Namen herauszufinden. Findet sich kein Name in der aktuellen Node, wird in der Parent-Node nachgesehen ob diese einen Namen besitzt, gegebenenfalls wird bis zur Urgroßelternnode nachgesehen. Hat auch diese keinen Namen (was in Tests nicht vorkam) wird in der Ausgabedatei vermerkt, das kein Name gefunden wurde.

Aus dem bestehenden Code wurde die Schnittpunktanzeige auf der Konsole übernommen, die dem Benutzer die Schnittpunkte im 3D-Weltkoordinatensystem präsentiert. Über den Define "BLOCKSETZTEN" ist es zudem auch möglich, kleine Blöcke in der Landschaft zu platzieren, die jeweils für sich markieren, an welchen Stellen eine Fixierung stattgefunden hat. Bei Aktivierung dieser Option ist zu beachten, das in der Ausgabedatei "CUBE", die Bezeichnung dieser Blöcke, auftaucht, sobald der Proband widerholt auf die gleiche Stelle schaut. Die Funktion ist daher nur für Testzwecke bzw. zur Visualisierung von Blickpunkten einsetzbar und darf bei der eigentlichen Auswertung nicht mitlaufen, da anderenfalls nach einigen Frames die Antwort was gesehen worde immer "CUBE" lautet.

2.3.10 void auswertung(int Fenstergroesse, float Radius)

Diese Funktion dient zur automatischen Auswertung der Fixierungen. Sie muss im Anschluss an die obigen Berechnungen aufgerufen werden und ermittelt aus den berechneten Daten, ob der Proband in ⟨Fenstergröße⟩ Fällen hintereinander im ⟨Radius⟩ Radius ein Objekt angesehen hat, so dass man von einer Fixierung sprechen kann. Das Problem hierbei ist, dass man nur von einer Fixierung im eigentlichen Sinne sprechen kann, wenn der Proband einen Gegenstand über mehrere Frames in seinem Sichtkegel hatte. Wird das Objekt nur in ein oder zwei aufeinanderfolgenden Frames getroffen, kann nicht davon ausgegangen werden, dass der Proband das Objekt fixiert hat. Treffen hingegen mehrere aufeinanderfolgende Sichtstrahlen das Objekt (man geht für die benötigte Zeit einer Fixierung von ca. 100ms = 6 Frames aus), kann das Objekt als fixiert erkannt werden. Anderenfalls muss man davon ausgehen, dass die Messung der Blickbewegung während einer Sakkade (einer schnellen Augenbewegung) erfolgte. Während dieser Zeit ist das Auge "blind" und nimmt keine Informationen auf.

Hierzu läuft eine Schleife über die in der Rundumblick-Funktion erstellten Da-

ten der Fixierungsstatistik. Abhängig vom dort für jedes Frame hinterlegten Wertes der getroffenen Objekte wird der Datenblock outputfixationen gefüllt. Ist die Zählvariable für ein Objekt größer oder gleich der Fenstergröße wird der Abstand der Mittelpunkte der geschossenen Strahlenbündel ermittelt. Sind alle so gewonnenen Abstände kleiner dem gegebenen Radius wird der Name des Objekts in den Datenblock outputfixationen geschrieben.

2.3.11 void writeoutput(String Dateiname)

Diese Funktion schreibt die berechneten Daten in die übergebene Datei. Sie kann entweder beim Aufrufen des ExitFlag, oder aber nach dem Berechnen aller Messwerte aufgerufen werden. Das Format der Ausgabedatei wurde in Abschnitt 2.2.3 erklärt.

3 Bedienung/ Einbindung in eigene Programme

3.1 Parameter

Die Parameter des Programms befinden sich in der Datei shareddata.h. Hier finden sich neben den von Gregor Hardieß eingeführeten Einstellungen für Leinwand, den Abständen sowie dem aktuellen Performer-Szenenobjekt vier neue Optionen:

- **AUSW_RADIUS**: der Radius, in dem das Strahlenbündel geschossen werden soll
- **AUSW_SCHRITTE**: wieviele Strahlen pro Richtung in diesem Radius geschossen werden sollen. (Bsp: bei einem Radius von 1,5 und 3 Schritten wird pro $0,5^\circ$ ein Sichtstrahl in die Szene geschossen)

- **AUSW_FENSTER**: Anzahl von Frames, in denen ein Objekt vom Sichtstrahl werden muss, um als Fixierung zu gelten

- **AUSW_FIXRADIUS**: Radius eines Kreises, in dem alle zentralen Sichtstrahlen liegen müssen, damit eine Fixierung gewertet wird

3.2 Eingliederung der Klasse in eigene Programme

Um das Modul in einem anderes Programm zu nutzen, muss dieses zuerst mit der Datei shareddata.h gelinkt werden. Hier müssen je nach Aufgabe auch Änderungen am SharedData-Objekt vorgenommen werden. Im Anschluss an diese Modifikation wird die Dateien shareddata.h inkludiert. In der Header-Datei des neuen Programmes muss zusätzlich noch ein neues Objekt für die Analyse erstellt werden:

```
nameit* getnames;
```

Für die korrekte Funktion des Moduls muss zuerst ein Analyseobjekt erstellt, und diesem die Eingabedatei sowie das SharedData-Objekt übergeben werden. Dies geschieht z.B. durch:

```
getnames = new nameit(Shared);
getnames -> init("test.txt");
std::cout << getnames -> size() << " Werte eingelesen" << std::endl;
```

Des weiteren bedarf er Ergänzungen in der Grafikschleife: Vor dem Auslesen der Sichtmatrix muss

```
getnames -> nextjoypos();
```

aufgerufen werden, im Anschluss an das Auslesen der Sichtmatrix muss

```
getnames -> nextimage();
```

eingefügt werden.

Im Anschluss an die Grafikschleife kann die Auswertung sowie das Schreiben der Ausgabedatei erfolgen:

```
std::cout << "Datenanalyse erfolgt " << std::endl;
getnames -> auswertung();
std::cout << "Outputfile wird geschrieben" << std::endl;
getnames -> writeoutput("output.txt");
```

Das Einfügen der gerade genannten fünf Codestellen erlaubt die automatische Auswertung der Fixierungen.

4 Überprüfung des Algorithmus

Das Programm wurde auf der Projektionsleinwand getestet. Zur Visualisierung der Sichtstrahlen wurden kleine Blöcke benutzt, die in der 3D-Welt jeweils am Schnittpunkt des Sichtstrahls mit dem getroffenen Objekt eingeblendet werden. So wurde eine einfache visuelle Kontrolle möglich.

Für die Verifikation der Algorithmen wurde eine Eingabedatei (Siehe Anhang E) erstellt, welche in 5 vertikalen Linien ($-74°, -35°, 0°, 35°, 74°$) Blöcke mit einem Abstand von 5° zueinander aufträgt.

Beim ersten Test wurde das Kalibrierungsgitter für die Projektoren (ein Raster mit 10° Linienabstand) eingeblendet. Die geschossenen Blöcke erschienen an den richtigen Stellen in Relation zum Gitter.

In einem zweiten Test wurden die Blöcke mit einem Zollstock in Relation zur Leinwand vermessen und anschließend berechnet, ob sie im Vergleich zum mathe-

matischen Leinwandmodell an der gleichen Stelle erscheinen. Auch hier befanden sich die Blöcke an den errechneten Positionen.

Der Algorithmus wird darüber hinaus in einem an diese Studienarbeit anschließendes Projekt mit der Tübinger Augenklinik verwendet.

A shareddata.h

```
1  //Studienarbeit Christian Rothländer, 2005
2  //Christian@rothlaender.net
3  //
4  //
5  // Shared-Data koennen von allen Prozessen (Cull, Application,Draw)
6  // zugegriffen werden
7
8
9  #ifndef shareddataH
10 #define shareddataH
11
12 #include <Performer/pf/pfChannel.h>
13 #include <Performer/pf/pfLightSource.h>
14 #include <Performer/pr/pfLight.h>
15 #include <Performer/pr/pfTexture.h>
16 #include <Performer/pr/pfGeoSet.h>
17 #include <Performer/pr/pfGeoState.h>
18 #include <Performer/pf/pfGeode.h>
19 #include <Performer/pf/pfNode.h>
20 #include <Performer/pf/pfGeode.h>
21 #include <Performer/pr/pfLinMath.h>
22
23 #include <Performer/pfdu.h>
24 #include <Performer/pf/pfDCS.h>
25 #include <Performer/pfutil.h>
26 #include <Performer/pfui.h>
27 #include <Performer/pf.h>
28 #include <Performer/pf/pfText.h>
29 #include <Performer/pr/pfString.h>
30 #include <Performer/pr/pfFont.h>
31
32
33 ///Definitionen die alles betreffen
34
35 #define CAM    6  // Anzahl der Kameras, die auf die Welt gucken
36 #define PROJEKTIONSBREITE 150  // winkelausschnitt, den alle n
37                                // Kameras (CAM) in die scene schauen
38 #define SCHRITTE 10 // Anzahl der Koordinatenzeilen und -spalten
39                     // mit der jede Kamera auf der Projektionsflaeche
40                     // unterteilt wird
41                     // MIN: 2+
42 #define NEIGUNG 15  //Neigung der Leinwand
43
44 #define RADIUS 1.62 //Radius der Leinwand
45
46 #define FOVproCAM 25  // Projektionsbreite geteilt durch cam
47
48
49 #define VERTIKALERSICHTWINKEL 66.3
50 #define VERTIKALERWINKELVOMBODENZURHORIZONTALE 42.74
51 #define WINKELSICHTGERADEAUFUNTERKANTELEINWAND 62.26
52 #define LAENGEVOMAUGPUNKTAUFUNTERKANTELEINWAND 1.768
53 #define LEINWANDHOEHE 2.07
54
55 #define NEARPLANE 0.1
56 #define FARPLANE 1000
57
58 ///Defines für die Auwertung
59
60 #define AUSW_RADIUS 1.5 //wie groß soll der Kreis sein, in dem Strahlen geschossen werden
61 #define AUSW_SCHRITTE 3 //wieviele Schritte sollen gemacht werden (*2+1) (ACHTUNG:EXPONENTIELLER ANSTIEG DES AUFWANDS)
62 #define AUSW_FENSTER 5  //Schiebefenstergröße
63 #define AUSW_FIXRADIUS 3//Radius des Kreises, in dem eine Fixation gewertet werden soll
64
65
66 const int sizeTexX = 512; // Groesse der Textur (Vielfaches von 512)
67 const int sizeTexY = 256;
68
69 const int no_of_hor_ang = 10; //hor_ang und vert_ang sind fr die berechnung der KalibrationsRinge
70 const int no_of_vert_ang = 10;
71
72
73 //der Hauptdatenstruct ...
74
75 struct SharedData
76 {
77
78   pfPipe          *pipe;
79   pfPipeWindow    *pw;
80
81
82   pfChannel       *chan_L;     // guckt auf die Projektionswand
83   pfChannel       *chan_R;     // guckt auf die Projektionswand
84   pfChannel       *chan[CAM];  // gucken auf die Welt
85   pfChannel       *noLogoChan; //eine Cam schaut ausserhalb der welt ins schwarze, Logo Hintergrund wird schwarz
86   pfScene         *scene;
```

```
87    pfCoord           noLogoView;
88    int               exitFlag;
89    int               StartFlag;
90    int               TRACK;
91    int               calibrationTexture;
92    float             h,r,p,x,y,z;
93
94    // in diese Variable wird Framebuffer als Textur eingelesen
95    pfTexture        *envTex;
96    unsigned int     *textureImage;
97
98    //for the movement in front of projection screen
99    float heading_L, roll_L, heading_R, roll_R;
100   float pitch_L, pitch_R;
101   float hoehe_L,X_L,Y_L,hoehe_R,X_R,Y_R;
102   float alpha_L, alpha_R, beta_L, beta_R;
103
104   //Variablen fr Kalibrationsscene "kalibrationCircles"
105   float          upper_elevation;         //upper limit of vis field
106   float          lower_elevation;
107   pfVec3         *circle3DVerts;
108   pfVec3         *line3DVerts;
109   pfVec2         circleVerts[no_of_hor_ang * CAM];
110   pfVec2         lineVerts[8];
111   //pfVec2       lineVerts[Shared->upper_elevation - Shared->lower_elevation)/10 + 1];
112   pfVec4         MeshColor;
113   int            MeshLengths[no_of_hor_ang * CAM];
114
115   // fr die Generierung der Szene
116   pfNode          *readNode[20];
117   pfNode          *root;
118   pfNode          *cube;          //Für Intersectiontest und Blocksetzen
119 };
120
121
122
123 #endif
```

B nameit.h

```
1  //Studienarbeit Christian Rothländer, 2005
2  //Christian@rothlaender.net
3
4  #include <vector>
5  #include <utility>
6  #include <map>
7  #include <string>
8  #include "shareddata.h"
9
10 struct intersection_data{
11         std::string name;
12         double a;
13         double e;
14         double x;
15         double y;
16         double z;
17         bool fix;
18 };
19
20 class nameit{
21
22       private:
23
24         int index;
25         int count;
26         float tanx;
27         float tanyoben;
28         float tanyunten;
29
30         SharedData *Shared;
31
32         std::vector<std::vector<double> > inputs;                       //Inputdatei (Winkel, Joystick)
33         std::vector<intersection_data> outputs;                         //Ausgabe (genau angepeilter Punkt)
34         std::vector<std::vector<intersection_data> > auswertungen;      //Ausgabe (umliegende Punkte)
35         std::vector<std::vector<double> > coordinates;                  //Leinwandkoordinaten der Ausgabe (x,y,z)
36         std::vector<std::map<std::string,int> > fixierungsstatistik;    //Statistik für die Fixierungen
37         std::vector<std::string > outputfixationen;                     //String welche Fixationen pro Block möglich sin
38
39         void makecoordinates();                          //erstellt coordinates
40         void naechsterblick();                           //automatische Sichtgeradenerstellung
41
42         pfSeg naechsterblick(double a,double e);         //manuelle Sichgeradenerstellung
43         std::vector<intersection_data> rundumblick(double a, double e);
44
45         intersection_data isecTest(pfSeg line);          //Test auf Intersection
46
```

```
47          inline double torad(double a){
48                  return a / 180 * M_PI;
49                  };
50
51          inline double todec(double a){
52                  return a * 180 / M_PI;
53                  };
54
55        public:
56
57          nameit(SharedData* sh);
58          ~nameit();
59          void init(std::string infile);
60          void nextimage();
61          void nextjoypos();
62          void writeoutput(std::string outfile);
63          void auswertung();
64          int size() {return count;};
65  };
```

C nameit.cpp

```
1   //Studienarbeit Christian Rothlï¿½der, 2005
2   //Christian@rothlaender.net
3   //
4   //Input: Winkeldatei "alpha epsylon ...."
5   //Output: Im Objektbaum angeschaute Objekte als Liste ber die einzelnen Frames
6   //Optionen:
7
8   #include <iostream>
9   #include <fstream>
10
11  #include "nameit.h"
12
13  #define NO_DEBUG
14  #define NO_DEBUG2
15  #define NO_DEBUGDAVID
16  #define NO_BLOCKSETZEN
17  #define NO_KOORDINATENBERECHNUNG
18  #define NO_DEBUGMULTI
19  #define NO_DEBUGAUSWERTUNG
20
21  nameit::nameit(SharedData* sh){
22          index = 0;
23          count = 0;
24          Shared = sh;
25
26          //Berechnungen fr die spï¿½ere Sichtgeradenberechnung, auseinandergezogen wegen gcc-problematik mit Inline+Defin
27          tanx = FOVproCAM/2;
28          tanx = std::tan(torad(tanx));
29          tanyunten = VERTIKALERWINKELVOMBODENZURHORIZONTALE;
30          tanyunten = std::tan(torad(tanyunten));
31          tanyoben = VERTIKALERSICHTWINKEL - VERTIKALERWINKELVOMBODENZURHORIZONTALE;
32          tanyoben = std::tan(torad(tanyoben));
33
34          //Vordefinieren der Vectorgrößen (Auf 21 Minuten Versuchslänge = 75600 Einzelbilder)
35          inputs.reserve(75600);
36          outputs.reserve(75600);
37          auswertungen.reserve(75600);
38          coordinates.reserve(75600);
39          fixierungsstatistik.reserve(75600);
40          outputfixationen.reserve(75600);
41  }
42
43  nameit::~nameit(){
44  }
45
46  void nameit::init(std::string infile){  /*Liest Eingabedatei aus und wandelt in internes Datenformat*/
47
48          //Eingabedatei lesen
49
50          std::ifstream in(infile.c_str());
51          std::string useless;
52          double a;
53
54          #ifdef DEBUG
55          std::cerr << "Werte fuer (index, alpha, epsylon, joystikkoordinaten(x,y,z,h,p,r): "<< std::endl;
56          #endif
57
58          while (in.good()){
59                  std::vector<double> tmp;
60
61                  count++;
62
```

```
63                  //8 Werte einlesen: alpha, epsylon, dann Joystikkoordinaten: x,y,z, h,p,r
64                  for (int i = 0; i != 8; i++){
65                          in >> a;
66                          tmp.push_back(a);
67                  }
68
69                  inputs.push_back(tmp);
70
71                  getline(in, useless);
72
73                  #ifdef DEBUG
74                  std::cerr << count << "\t" << tmp[0] << "\t"<< tmp[1] << "\t|"<< tmp[2] << "\t"<< tmp[3] <<"\t"<< tmp[4]
75                  #endif
76
77                  in.peek();
78          }
79
80          in.close();
81
82          count--;
83
84          #ifdef KOORDINATENBERECHNUNG
85          makecoordinates();
86          #endif
87          //Eingabedatei lesen ende
88
89  }
90
91  void nameit::makecoordinates(){ /*Funktion erreichnet aus Winkeln die Positionen x,y,z auf der Leinwand wo man hinschaut
92          #ifdef DEBUG
93          std::cerr << "Werte_fuer_(x,y,z):_"<< std::endl;
94          #endif
95
96          double cosng = std::cos(torad(NEIGUNG));
97          double sinng = std::sin(torad(NEIGUNG));
98
99          for (int i = 0; i != count; i++){
100
101                 std::vector<double> tmp;
102
103                 double cosa = std::cos(torad(inputs[i][0]));
104                 double cose = std::cos(torad(inputs[i][1]));
105                 double epsylonplusneigunginrad = torad(inputs[i][1]+NEIGUNG);
106
107                 tmp.push_back(RADIUS * cosng * cose * std::sin(torad(inputs[i][0])) / std::cos(epsylonplusneigunginrad))
108                 tmp.push_back(RADIUS * cosng * cose * cosa / std::cos(epsylonplusneigunginrad));
109                 tmp.push_back(1.2 - RADIUS * sinng * cosng + RADIUS * cosng * cosng * std::tan(epsylonplusneigunginrad))
110                 coordinates.push_back(tmp);
111
112                 #ifdef DEBUG
113                 std::cerr << tmp[0] << "_" << tmp[1] << "_" << tmp[2] << std::endl;
114                 #endif
115         }
116 }
117
118 void nameit::naechsterblick(){
119
120         if (count == index) Shared -> exitFlag = 1;
121         if (count <= index) return;     //Wenn die Datendateien nicht 100% bereinstimmen gehe ich so Segfaults aus dem u
122
123         pfSeg line = naechsterblick(inputs[index][0], inputs[index][1]);
124         intersection_data intersection = isecTest(line);
125         outputs.push_back(intersection);
126
127         auswertungen.push_back(rundumblick(inputs[index][0], inputs[index][1]));
128 }
129
130 std::vector<intersection_data> nameit::rundumblick(double a, double e){
131
132         std::vector<intersection_data> daten;
133
134         float rad = AUSW_RADIUS;
135
136         int schritt = AUSW_SCHRITTE;
137
138         float schrittweite = rad / AUSW_SCHRITTE;
139
140         std::map <std::string,int> mp;
141
142         #ifdef DEBUGMULTI
143         std::cout << "------" << std::endl;
144         std::cout << "multiple_Analyse:" << std::endl;
145         #endif
146
147         for (int i = -schritt; i <= schritt; i++)
148                 for (int j = -schritt; j <= schritt; j++){
149                         if (std::abs(i)+std::abs(j) <= schritt)
150                                 {
151                                 //Wenn valider Rasterpunkt -> Daten ermitteln.
```

```
152                                     pfSeg line = naechsterblick(a+i*schrittweite,e+j*schrittweite);
153                                     intersection_data intersection = isecTest(line);
154                                     intersection.a = a+i*schrittweite;
155                                     intersection.e = e+j*schrittweite;
156                                     daten.push_back(intersection);
157
158                                     #ifdef DEBUGMULTI
159                                     std::cout << i << "\t" << j<< "\t" << intersection.x << "\t" << intersection.y << "\t" <
160                                     #endif
161
162
163                                     ///Hier beginnt die Indexerstellung für die spätere Auswertung:
164                                     /*Falls ein Gegenstand im vorherigen Frame schonmal gefunden wurde wird der dort gefunde
165                                     Wert um eins hochgezählt und als aktueller Wert benutzt*/
166
167                                     int wert = 1;
168                                     if (index!=0)
169                                             {
170                                             std::map<std::string,int>::iterator iter = fixierungsstatistik[index-1].find(int
171                                             if (iter != fixierungsstatistik[index-1].end()){
172                                                     wert = (*iter).second + 1;
173                                                     //if (wert > AUSW_FENSTER) wert=1; //bewirkt Rasterung der Daten
174
175                                                             #ifdef DEBUGAUSWERTUNG
176                                                             std::cout << index << "␣" << (*iter).first << "␣" << (*iter).sec
177                                                             #endif
178                                                     }
179                                             }
180                                     mp.insert(make_pair(intersection.name,wert));
181
182                                     ///Ende Indexerstellung
183                                     }
184                     }
185
186             fixierungsstatistik.push_back(mp);
187             return daten;
188     }
189
190     pfSeg nameit::naechsterblick(double a, double e){
191
192             //Sicherung der Orginalen Variablen, werden evtl. noch benötigt
193             double origa = a;
194             double orige = e;
195
196             //Umrechnen der Winkel in Leinwandgrad (0,0 = links unten)
197             a += (PROJEKTIONSBREITE/2);
198             e +=  VERTIKALERWINKELVOMBODENZURHORIZONTALE;
199
200                     #ifdef DEBUG
201                     std::cout << "\n␣Winkel:␣" << a << "␣" << e << std::endl;
202                     #endif
203
204             //wir ermitteln die Verhï¿½tnisse:
205             ///Ein solches Verhï¿½tnis beschreibt den Abstand in % vom linken/unteren Rand zur Gesamtbreite/-hhe
206
207             double verha = a / PROJEKTIONSBREITE;                                           //ergibt %-Wert
208             float ywinkel = torad(180-WINKELSICHTGERADEAUFUNTERKANTELEINWAND-e);            //temporï¿½e Variable
209             double verhe = LAENGEVOMAUGPUNKTAUFUNTERKANTELEINWAND * std::sin(torad(e)) / std::sin(ywinkel) / LEINWANDHOEHE;
210
211             //Mit diesen Verhï¿½tnissen knnen wir die Kamera ermitteln:
212
213             int cam = (int) (CAM * verha);
214
215             //und den horizontalen offset in dieser Kamera:
216
217             double hoff = a - cam*FOVproCAM;        //temporï¿½e Variable
218             double verhoriz = hoff / FOVproCAM;     //<-- gesuchtes Verhï¿½tnis in X-Richtung
219
220                     #ifdef DEBUG2
221                     std::cout << "Verhätnisse:␣" << verhoriz << "␣" << verhe << std::endl;
222                     #endif
223
224             //Grï¿½ der Textur in Pixeln:
225
226             float tary = sizeTexY;
227             float tarx = sizeTexX / CAM;
228
229             //Unser angeblickter Punkt in dieser Kamera ist der  Folgende (x,y):
230
231             float x = tarx * verhoriz;
232             float y = tary * verhe;
233
234                     #ifdef DEBUG
235                     std::cout <<"Pixel␣der␣Kamera:␣(Cam,␣x,␣y)␣"<< cam << "␣" << x << "␣" << y << std::endl;
236                     #endif
237
238             //aktuelle Viewmatrix holen:
```

```
239
240         pfMatrix mat;
241         Shared -> chan[cam] -> getViewMat(mat);
242
243         //nun berechnen wir analog zu David die Sichtgerade:
244
245         float dx = tanx * 2 * (verhoriz - .5);
246         float dy = std::tan(torad(orige));
247
248                 #ifdef DEBUG
249                 std::cout <<"dx, dy" << dx << " " << dy << std::endl;
250                 #endif
251
252         pfVec3 pnear = pfVec3(dx * NEARPLANE, NEARPLANE, dy * NEARPLANE);
253         pfVec3 pfar = pfVec3(dx * FARPLANE, FARPLANE, dy * FARPLANE);
254
255         pnear.xformPt(pnear, mat);
256         pfar.xformPt(pfar, mat);
257
258         pfSeg line (pnear,pfar);
259
260         return line;
261         //isecTest(line);
262 }
263
264 void nameit::nextimage(){
265         naechsterblick();
266         index++;
267         if (index % 10 == 0) std::cout << "Datensatz " << index << " wird bearbeitet\n";
268 }
269
270 void nameit::nextjoypos(){
271
272         if (index < count){
273                 Shared -> x = inputs[index][2];
274                 Shared -> y = inputs[index][3];
275                 Shared -> z = inputs[index][4];
276                 Shared -> h = inputs[index][5];
277                 Shared -> p = inputs[index][6];
278                 Shared -> r = inputs[index][7];
279         }
280 }
281
282 void nameit::writeoutput(std::string outfile){
283         std::ofstream out(outfile.c_str());
284
285         out << "Zeile 1      :\t" << "Inputwerte" << std::endl;
286         out << "Zeile 2-(n-1):\t"<< "lft.Nr\tAlpha\tEpsylon\tx\ty\tz\tObjektname" << std::endl;
287         out << "Zeile n      :\t" << "Fixationen" << std::endl;
288         out << "PARAMETER" << std::endl;
289         out << "Strahlenbuendelparameter(Schritte,Radius): " << AUSW_SCHRITTE << " " << AUSW_RADIUS << std::endl;
290         out << "Fixationsparameter(Fenstergroesse,Radius): " << AUSW_FENSTER << " " << AUSW_FIXRADIUS << std::endl;
291         out << "DATEN" << std::endl;
292
293         for (int i = 0; i != count; i++){
294
295                 out << "Input:\t" << inputs[i][0] << "\t" << inputs[i][1]<< "\n";
296
297                 for (int j = 0; j != auswertungen[i].size(); j++){
298
299                         out << i << "\t" << auswertungen[i][j].a << "\t" << auswertungen[i][j].e << "\t" << auswertungen
300
301                 }
302
303                 out << "moegliche Fixationen:\t" << outputfixationen[i] << "\n\n";
304         }
305
306
307         out.close();
308 }
309
310 intersection_data nameit::isecTest(pfSeg line){          ///aus Davids Arbeit bernommen und modifiziert
311
312         intersection_data daten;
313
314         pfSegSet segset;
315         pfHit **hits[32];
316         int isect;
317
318         segset.activeMask = 1;
319         segset.isectMask = 0x01;
320         segset.discFunc = NULL;
321         segset.mode = PFTRAV_IS_PRIM;
322
323         segset.segs[0].pos.set(line.pos[0], line.pos[1], line.pos[2]);
324         segset.segs[0].dir.set(line.dir[0], line.dir[1], line.dir[2]);
325         segset.segs[0].length = line.length;
326
327         isect = Shared->scene->isect(&segset, hits);
```

```
328
329                 #ifdef DEBUG
330                 std::cout << "Name␣wird␣ermittelt␣fr␣Objekt␣Nummer␣:" << index << std::endl;
331                 #endif
332
333         //Hier wird die Namensbersetzung ausgefhrt:
334         //Immer im Elternknoten wird nach dem Namen nachgeschlagen
335         pfGeode *node;
336         (*hits[0]) -> query(PFQHIT_NODE, &node);
337
338
339         if (node->getNumParents() == 0 || node->getName() != NULL) //Wir haben keine Elternknoten -> meistens Himmel
340                 {
341                 //std::cout << "a" << std::endl;
342                 if (node->getName() != NULL){
343                         //std::cout << node->getName() << std::endl;
344                         daten.name = (node->getName());
345                         }
346                 else{   daten.name = "Unbenannt";
347                         }
348                 }
349         else    {       //Wir suchen in 3 Ebenen nach einem Namen, wenn dieser nicht gefunden wird -> PECH
350
351                 if (node->getParent(0)->getName() != NULL){
352                         //std::cout << "b" << std::endl;
353                         //std::cout << node->getParent(0)->getName() << std::endl;
354                         daten.name =(node->getParent(0)->getName());
355                         }
356                 else if (node->getParent(0)->getNumParents() != 0 && node->getParent(0)->getParent(0)->getName() != NULL
357                         //std::cout << "c" << std::endl;
358                         //std::cout << node->getParent(0)->getParent(0)->getName() << std::endl;
359                         daten.name =(node->getParent(0)->getParent(0)->getName());
360                         }
361                 else if (node->getParent(0)->getParent(0)->getNumParents() != 0 && node->getParent(0)->getParent(0)->get
362                         //std::cout << "d" << std::endl;
363                         //std::cout << node->getParent(0)->getParent(0)->getParent(0)->getName() << std::endl;
364                         daten.name =(node->getParent(0)->getParent(0)->getParent(0)->getName());
365                         }
366                 else {  daten.name =("Unbenannt");
367                         //std::cout << "Kein Name gefunden" << std::endl;
368                         }
369                 }
370
371
372                 #ifdef DEBUG
373                 std::cout << "␣fertig" << std::endl;
374                 #endif
375
376         //pnt == Punkt der Intersection.. wie komm ich dran mit was die sich schneidet
377         pfVec3 pnt;
378         pfMatrix matt;
379         (*hits[0])->query(PFQHIT_POINT, pnt.vec);
380         (*hits[0])->query(PFQHIT_XFORM, matt.mat);
381         pnt.xformPt(pnt, matt);
382
383         ///Punkte der getroffenen Objekte speichern
384
385         /*std::vector<double> tmp;
386         tmp.push_back(pnt[0]);
387         tmp.push_back(pnt[1]);
388         tmp.push_back(pnt[2]);
389         tmp.push_back(0);
390         auswertungsdaten.push_back(tmp);*/
391
392         daten.x = pnt[0];
393         daten.y = pnt[1];
394         daten.z = pnt[2];
395         daten.fix = 0;
396
397                 #ifdef DEBUGDAVID
398                 std::cout << "intersection␣test:"<< std::endl;
399                 std::cout << "isect␣=␣" << isect << std::endl;
400                 std::cout <<"pnt␣X␣=␣"<< pnt[0] << std::endl;
401                 std::cout <<"pnt␣Y␣=␣"<< pnt[1] << std::endl;
402                 std::cout <<"pnt␣Z␣=␣"<< pnt[2] << std::endl;
403                 #endif
404
405         #ifdef BLOCKSETZEN
406         //Blocksetzen
407         pfDCS *test = new pfDCS();
408         pfGroup *asd = new pfGroup();
409         test->setTrans(pnt[0], pnt[1], pnt[2]);
410         Shared->cube = pfdLoadFile("./nametheseenobject/test.obj");
411         Shared->cube->setName("CUBE");
412         asd->addChild(Shared->cube);
413         asd->setName("CUBE");
414         test->addChild(asd);
415         Shared->scene->addChild(test);
```

```
416         //Blocksetzen Ende
417         #endif
418 
419         return daten;
420 }
421 
422 void nameit::auswertung(){
423 
424         for (int l = 0; l != AUSW_FENSTER ; l++) outputfixationen.push_back("");
425 
426         for (int i = AUSW_FENSTER; i != count; i++){
427 
428                 std::string fix;
429 
430                 for ( std::map<std::string,int>::iterator iter = fixierungsstatistik[i].begin(); iter != fixierungsstati
431 
432                         if ((*iter).second >= AUSW_FENSTER){
433 
434                                 bool valid = true;
435 
436                                 ///Abstandstest der Fixation
437                                 for (int k = 0; k != AUSW_FENSTER; k++)
438                                   for (int l = k; l != AUSW_FENSTER; l++){
439 
440                                         double test = std::sqrt(  std::pow((outputs[i-k].x - outputs[i-l].x),2)
441                                                                 + std::pow((outputs[i-k].y - outputs[i-l].y),2)
442                                                                 + std::pow((outputs[i-k].z - outputs[i-l].z),2));
443 
444                                         if (test > AUSW_FIXRADIUS) {    //falsche Fixation
445                                                         valid = false;
446                                                         std::cout << "Abstand_zu_groß:" << test
<< "_" << i << std::endl;
447                                                         break;}
448                                   }
449 
450                                 if (valid == true){                     //echte Fixation
451                                         fix += (*iter).first;
452                                         fix += "_";
453                                         }
454                         }
455 
456                 }
457 
458                 outputfixationen.push_back(fix);        //gefundene Fixationen zurückspeichern
459         }
460 
461 
462 
463         ///Alter CODE (für nur-Sichtstrahl-Auswertung nach Fixationen, falls es nochmal wer braucht):
464         /*for (int i = 0; i <= count-fenstergroesse; i++){
465                 std::cout << "Teste: " << i << " von " << count-fenstergroesse << std::endl;
466                 //Test auf Namensgleichheit:
467                 bool valid = true;
468                 for (int p = 0; p != fenstergroesse; p++){
469                         if (valid == true && outputs[i].name == outputs[i+p].name) ; else valid = false;
470                         }
471 
472                 //Namensgleichheit gewährleistet, Abstandstest:
473                 if (valid == true){
474                         std::cout << "Basis geschaffen -> weitere Tests. Objektnummer: " << i << std::endl;
475                         for (int k = 0; k != fenstergroesse; k++)
476                            for (int l = k; l != fenstergroesse; l++){
477 
478                                 double test = std::sqrt(  std::pow((outputs[i+k].x - outputs[i+l].x),2)
479                                                         + std::pow((outputs[i+k].y - outputs[i+l].y),2)
480                                                         + std::pow((outputs[i+k].z - outputs[i+l].z),2));
481 
482                                 if (test > radius) {valid = false;
483                                                 std::cout << "aus:" << test << std::endl;
484                                                 break;}
485                            }
486                         }
487 
488                 //Falls valid = true haben wir eine Fixation und setzen den Marker auf 1;
489                 if (valid == true) {
490                         outputs[i+fenstergroesse-1].fix = 1;
491                         std::cout << "Daten Modifiziert" << std::endl;
492                         i += (fenstergroesse-1);
493                         }
494 
495         }*/
496 }
```

D ausgabe.txt

```
1  Zeile 1       :  Inputwerte
2  Zeile 2-(n-1):  lft.Nr  Alpha   Epsylon x       y       z       Objektname
3  Zeile n       :  Fixationen
4  PARAMETER
5  Strahlenbuendelparameter(Schritte,Radius): 3 1.5
6  Fixationsparameter(Fenstergroesse,Radius): 5 3
7  DATEN
8  Input:  -74     25
9  0       -75.5   25      -257.109        69.0591 122.215 parkplatz3.flt
10 0       -75     24.5    -258.199        71.6049 120.441 parkplatz3.flt
11 0       -75     25      -257.172        71.3201 122.728 parkplatz3.flt
12 0       -75     25.5    -256.145        71.0353 125.016 parkplatz3.flt
13 0       -74.5   24      -259.299        74.1906 118.655 parkplatz3.flt
14 0       -74.5   24.5    -258.268        73.8955 120.95  parkplatz3.flt
15 0       -74.5   25      -257.236        73.6003 123.245 parkplatz3.flt
16 0       -74.5   25.5    -256.205        73.3052 125.541 parkplatz3.flt
17 0       -74.5   26      -255.173        73.01   127.836 parkplatz3.flt
18 0       -74     23.5    -260.409        76.8171 116.858 parkplatz3.flt
19 0       -74     24      -259.373        76.5113 119.162 parkplatz3.flt
20 0       -74     24.5    -258.337        76.2057 121.465 parkplatz3.flt
21 0       -74     25      -257.301        75.9    123.767 parkplatz3.flt
22 0       -74     25.5    -256.264        75.5944 126.07  parkplatz3.flt
23 0       -74     26      -255.228        75.2886 128.373 parkplatz3.flt
24 0       -74     26.5    -254.191        74.9828 130.678 parkplatz3.flt
25 0       -73.5   24      -259.447        78.8522 119.672 parkplatz3.flt
26 0       -73.5   24.5    -258.406        78.5358 121.983 parkplatz3.flt
27 0       -73.5   25      -257.366        78.2195 124.294 parkplatz3.flt
28 0       -73.5   25.5    -256.325        77.9031 126.604 parkplatz3.flt
29 0       -73.5   26      -255.284        77.5867 128.915 parkplatz3.flt
30 0       -73     24.5    -258.477        80.8861 122.506 parkplatz3.flt
31 0       -73     25      -257.431        80.5588 124.824 parkplatz3.flt
32 0       -73     25.5    -256.386        80.2316 127.143 parkplatz3.flt
33 0       -72.5   25      -257.497        82.9184 125.36  parkplatz3.flt
34 moegliche Fixationen:
```

E testmain.txt

```
1   -74 25 0 0 1 0 0 0
2   -74 20 0 0 1 0 0 0
3   -74 15 0 0 1 0 0 0
4   -74 10 0 0 1 0 0 0
5   -74 5 0 0 1 0 0 0
6   -74 0 0 0 1 0 0 0
7   -74 -5 0 0 1 0 0 0
8   -74 -10 0 0 1 0 0 0
9   -74 -15 0 0 1 0 0 0
10  -74 -20 0 0 1 0 0 0
11  -74 -25 0 0 1 0 0 0
12  -74 -30 0 0 1 0 0 0
13  -74 -35 0 0 1 0 0 0
14  -74 -40 0 0 1 0 0 0
15  74 25 0 0 1 0 0 0
16  74 20 0 0 1 0 0 0
17  74 15 0 0 1 0 0 0
18  74 10 0 0 1 0 0 0
19  74 5 0 0 1 0 0 0
20  74 0 0 0 1 0 0 0
21  74 -5 0 0 1 0 0 0
22  74 -10 0 0 1 0 0 0
23  74 -15 0 0 1 0 0 0
24  74 -20 0 0 1 0 0 0
25  74 -25 0 0 1 0 0 0
26  74 -30 0 0 1 0 0 0
27  74 -35 0 0 1 0 0 0
28  74 -40 0 0 1 0 0 0
29  0 25 0 0 1 0 0 0
30  0 20 0 0 1 0 0 0
31  0 15 0 0 1 0 0 0
32  0 10 0 0 1 0 0 0
33  0 5 0 0 1 0 0 0
34  0 0 0 0 1 0 0 0
35  0 -5 0 0 1 0 0 0
36  0 -10 0 0 1 0 0 0
37  0 -15 0 0 1 0 0 0
38  0 -20 0 0 1 0 0 0
39  0 -25 0 0 1 0 0 0
40  0 -30 0 0 1 0 0 0
41  0 -35 0 0 1 0 0 0
42  0 -40 0 0 1 0 0 0
43  35 25 0 0 1 0 0 0
44  35 20 0 0 1 0 0 0
45  35 15 0 0 1 0 0 0
46  35 10 0 0 1 0 0 0
47  35 5 0 0 1 0 0 0
48  35 0 0 0 1 0 0 0
49  35 -5 0 0 1 0 0 0
50  35 -10 0 0 1 0 0 0
51  35 -15 0 0 1 0 0 0
52  35 -20 0 0 1 0 0 0
53  35 -25 0 0 1 0 0 0
54  35 -30 0 0 1 0 0 0
55  35 -35 0 0 1 0 0 0
56  35 -40 0 0 1 0 0 0
57  -35 25 0 0 1 0 0 0
58  -35 20 0 0 1 0 0 0
59  -35 15 0 0 1 0 0 0
60  -35 10 0 0 1 0 0 0
61  -35 5 0 0 1 0 0 0
62  -35 0 0 0 1 0 0 0
63  -35 -5 0 0 1 0 0 0
64  -35 -10 0 0 1 0 0 0
65  -35 -15 0 0 1 0 0 0
66  -35 -20 0 0 1 0 0 0
67  -35 -25 0 0 1 0 0 0
68  -35 -30 0 0 1 0 0 0
69  -35 -35 0 0 1 0 0 0
70  -35 -40 0 0 1 0 0 0
```

Abbildungsverzeichnis

1 Leinwand 4
2 Eyetracker 5
3 Beamer 5
4 Bilderstellung: Der Weg vom 3D-Szenengraphen zum Bild auf der Leinwand 6
5 Verarbeitungsschritte während der Datenverarbeitung: Die Berechnung von Head- und Eyetrackerdaten erfolgt durch Algorithmen, die in Stefanie Mayers Studienarbeit näher erläutert werden, das Zusammenführen aller Daten erfolgt mittels eines von Gregor Hardieß entwickelten Konverters. 8
6 Erklärung der verwendeten Winkel: links die horizontale Ebene (Blick von oben), rechts die vertikale Ebene (Blick von der Seite) 10
7 Umrechnung von Winkelgrad in Prozentwerte. Am horizontalen Beispiel ist die Umrechnung linear möglich, da der Abstand zwischen Auge und Leinwand konstant ist, in der vertikalen ist eine weitere Umrechnung nötig, die sich aus Sinus- und Kosinussatz herleiten lässt. 16
8 Texturberechnung in x-Ausrichtung. Das Offset für Near- und Farplane ergibt sich aus dem Tangens. Diese Berechnung erfolgt analog zur Arbeit von David Hrabal. 16
9 Das verwendete Muster in der Standardeinstellung von 3° Öffnungswinkel sowie einer Schrittzahl von 3. In diesem Muster werden die Sichtstrahlen von der Funktion rundumblick() in die Szene gelegt, um fixierte Objekte zu erkennen. 18

Index

Ausgabeformat, 12
auswertungen, 11
Azimuth, 9

Beamer, 5
Berechnen des nächsten Sichtstrahls, 14

coordinates, 11

Datenformat, 9

Eingabeformat, 9
Elevation, 9
Eyetracker, 5

fixierungsstatistik, 11, 17
Formate, 9

globale Variablen, 9

heading, 9

init(), 13
inputs, 11
interne Datenformate, 11
intersection test, 18
isecTest(), 18

Joystick, 5, 10

Leinwand, 5
Leinwandveränderung, 9

makecoordinates(), 13

naechsterblick(double α, double ε), 14
nextimage(), 14
nextjoypos(), 13

OpenGL-Pipeline, 6
outputfixationen, 11
outputs, 11

Parameter, 12
pitch, 9
Projektionsleinwand, 5

roll, 9
rundumblick(double α, double ε), 17

Schnittberechnung, 18
Shared-Object, 11
shareddata.h, 21

Trackingsystem, 5

Verhältnisberechnung, 14
Versuchsbeschreibung, 4
Verzerrung, 6

Winkelumrechnung, 14
writeoutput(), 20