Untersuchung zur Reduzierung von Motion Sickness in virtuellen Umgebungen durch den Einsatz von Kalman-Filtern

DIPLOMARBEIT

zur Erlangung des akademischen Grades
Diplom-Informatiker
an der Fachhochschule für Technik und Wirtschaft Berlin

Fachbereich Wirtschaftswissenschaften II
Studiengang Angewandte Informatik

erstellt bei der Firma
Weißhuhn und Weißhuhn Kommunikationsmanagement GmbH

1. Betreuer: Prof. Dr. Thomas Jung
2. Betreuer: Arndt Weisshuhn, MSc

Eingereicht von Daniel Grabert
12. Mai 2003

Inhaltsverzeichnis

1. Einleitung

1.1. Problemstellung

Eines der Hauptziele der *Virtual Reality* ist es, dem Benutzer eine computergenerierte Umgebung bereitzustellen, die in Bezug auf die optische Erscheinung, das Verhalten und die Interaktionsmöglichkeiten der realen Welt entspricht. Die erstmals im Jahre 1968 [Sut68] vorgestellten *Head-Mounted-Displays* [HMDs] erweiterten die Nutzungsmöglichkeiten der virtuellen Realität um eine neue Dimension. Mit Hilfe von HMDs kann der Benutzer in eine virtuelle Welt “eintauchen”. Es ist ihm möglich, sich in einer computererzeugten dreidimensionalen Umgebung frei zu bewegen und begrenzt in ihr zu interagieren. Leider sind heutige Computersysteme und HMDs nicht in der Lage eine überzeugende *Virtual Reality* zu erschaffen. Der Grund dafür liegt in einer Vielzahl von ungelösten oder unzureichend gelösten Problemen auf diesem Gebiet.

Zum einen sind die Displays, wie sie in den HMDs verwendet werden, aufgrund von geringem Auslösungsvermögen und niedriger Bildwiederholfrequenz nicht dazu geeignet, ein Bild wiederzugeben, wie man es aus der realen Welt gewöhnt wäre. Auch das auf Dauer nicht unerhebliche Gewicht der Geräte trägt dazu bei, dass heutige HMD-basierte Systeme von einem vollimmersiven Eindruck für den Benutzer weit entfernt sind. Hinzu kommt, dass auch die Möglichkeiten zur Erzeugung der visuellen Eindrücke in Echtzeit nicht mit der Realität zu vergleichen sind. Zwar wurden in den letzten Jahren enorme Forschritte im Bereich der Grafikprozessoren gemacht, trotzdem sind auch diese Systeme nicht in der Lage, eine überzeugende virtuelle Umgebung darzustellen, und wir müssen uns deshalb mit Abstraktionen der Realität begnügen.

Der letzte große und für die Arbeit zentrale Problemkreis bei der Verwendung von Head-Mounted-Displays liegt in der Interaktion mit dem Benutzer. Damit ein Benutzer eine virtuelle Umgebung sinnvoll nutzen kann, muss das System ständig die Ausrichtung seines Kopfes im Raum und seine Position kennen. Üblicherweise werden sogenannte *Tracker* verwendet, um diese Daten periodisch zu bestimmen. Leider arbeiten auch diese Systeme nicht in Echtzeit, wodurch schon im *Tracker* eine Latenz zwischen dem Auslesen der aktuellen Orientierung und dem Senden der Daten an den steuernden Rechner entsteht. Hinzu kommen Verzögerungen durch Datenübertragungen und das Errechnen der dreidimensionalen Darstellungen.

Die Summe aller Verzögerungen wird als *end-to-end system delay* bezeichnet und kann negative Auswirkungen auf die Nutzbarkeit des Systems haben. So empfindet ein Mensch eine Verzögerung von nur 5 ms als störend. Diese minimalen Latenzen können bei längerer

Nutzung von HMDs Symptome von Unbehagen über Kopfschmerz bis hin zu Übelkeit und Erbrechen hervorrufen. Die Auswirkungen auf den Körper sind mit einer Seekrankheit zu vergleichen und haben auch dieselbe Ursache: Die optische Wahrnehmung stimmt nicht mit der Orientierung des Kopfes im Raum überein, so wie sie vom Innenohr wahrgenommen wird.

Da nicht abzusehen ist, dass diese Probleme einzig durch die Verbesserung der Hardware zu lösen sind, ist es eine gute Möglichkeit diesen Effekten entgegenzuwirken, indem die Bewegungen des Kopfes vorausberechnet werden. Wenn das System in der Lage wäre, die Orientierung des Kopfes zum Zeitpunkt der Darstellung im voraus zu bestimmen, könnte es diese Daten statt der verzögerten Messdaten verwenden, um die gewünschte Ansicht zu erzeugen. Eine perfekte Voraussage würde es möglich machen, das *end-to-end system delay* vollständig zu kompensieren.

1.2. Zielstellung

In dieser Arbeit soll ein praktischer Einblick in die Möglichkeit zur Reduzierung des *end-to-end system delay* durch die zeitliche Vorausberechnung der Kopforientierung mittels eines auf dem Kalman-Filter basierenden Vorhersagesystems gegeben werden.

Das Kernstück stellt die prototypische Implementierung eines Vorhersagesystem basierend auf Kalman-Filtern dar, die es ermöglichen soll, aus den Messwerten des am HMD befestigten *Trackers* die zukünftige Orientierung des Kopfes vorherzusagen. Um objektive Kriterien zur Bewertung der Güte der Vorhersagen möglich zu machen, werden die Daten von mehreren Benutzern über einen repränsentativen Zeitraum aufgezeichnet. Der zu implementierende Vorhersagemechanismus, insbesondere jedoch seine Parametrisierung, soll dann off-line mit Benutzerprofilen getestet werden, um eine Vorhersage zu erreichen. Zur Anwendung kommt die Bewegungsvorhersage dann um die Benutzbarkeit der virtuellen Umgebung zu verbessern und extreme Effekte wie etwa *Motion Sickness* zu vermeiden. Das Ziel ist die prototypische Implementierung einer virtuellen Umgebung zur Ansicht von beliebigen 3D-Studio-MAX-Dateien. Die Darstellung soll stereoskopisch auf einem HMD vom Typ Cybermind hi-Res900^{TM}3D erfolgen, dass freundlicherweise von der FHTW Berlin für diese Arbeit zur Verfügung gestellt wurde. Die gesamte Arbeit wird gezielt für diese Hardware entwickelt, da ihre Spezifikation großen Einfluss auf die Auswahl von Technologien und Algorithmen hat - besonders bei der Wahl des Mechanismus´ zur Bewegungsvorhersage (Kapitel 3).

1.3. Aufbau der Arbeit

Die Arbeit gliedert sich in fünf Teile:

Im ersten Teil werden die Grundlagen der Systeme und Systemkomponenten beschrieben, die im Zusammenhang mit dem Problemverständnis wichtig sind. Hierbei sollen die Komponenten und ihre Funktionen nur abstrakt beschrieben werden, da es für das Verfahren nicht von Belang ist, ob beispielsweise ein optisches, mechanisches oder magnetisches Tracking-System benutzt wird und wie diese Systeme genau funktionieren. Um Begriffe zu erläutern, wird das Feld der *Virtual Reality*, der *Head-Mounted-Displays* und insbesondere die Problematik des *Tracking* beleuchet. Dies schließt auch einige mathematische Grundlagen über die Repäsentation von Rotationen ein, die wichtig für das Verständis einiger Entscheidungen während der Softwareerstellung sind. Darüber hinaus werden Grundlagen erläutert, die zur Erzeugung einer korrekten Ansicht der virtuellen Umgebung mit Tiefenwirkung wichtig sind.

Im zweiten Teil wird die generelle Funktionsweise des zum Einsatz kommenden Schätzverfahrens, des *Kalman-Filters*, eingeführt und das Verfahren von Liang [LSG91] zur Problematik der Kopfbewegungsvorhersage gezeigt.

Der dritte Teil beschreibt die zur erfolgreichen Implementierung des *Kalman-Filters* nötigen Vorbereitungen, wie das Erstellen von verschiedenen Benutzerprofilen und die Bestimmung des zu kompensierenden *end-to-end system delays* aus den *statischen und dynamischen Fehlern* (Kapitel 2.4).

Im vierten Abschnitt werden die vorhergehenden theoretischen Ausführungen in eine Implementierung übertragen. Hierzu werden die einzelnen Zyklen der Softwareentwicklung für den eigentlichen Kalman-Filter und der 3D-Anwendung durchlaufen und beschrieben. Als Abschluss soll eine prototypische Anwendung entstehen, in der der Vorhersagemechanismus des Kalman-Filters zur Anwendung gelangt. Die Anwendung soll es ermöglichen, beliebige 3D-Studio-Szenen stereoskopisch zu betrachten und sich in der Szene zu bewegen.

Teil fünf fasst die durch die Tests entstandenen Erkenntnisse in einem Fazit zusammen und gibt einen Ausblick auf mögliche Verbesserungen.

2. Grundlagen

In diesem Kapitel werden einige Grundlagen aus den verschiedenen Problemkreisen vermittelt, die zur Erreichung der Zielstellung notwendig sind.

2.1. Virtuelle Realitäten

2.1.1. Virtual Reality

Begriffe wie *"Virtual Reality"* und *"Cyberspace"* waren zu Beginn der 90er Jahre noch weitgehend unbekannt. Dank des immer stärker werdenden Medieninteresses gehören diese Begriffe heute jedoch zum üblichen Vokabular. *Virtual Reality* (VR) ermöglicht es dem Menschen, die immer komplexer werdenden Datenmengen zu visualisieren, zu manipulieren und mit der Darstellung zu interagieren. VR stellt einen großen Schritt in den Möglichkeiten der Interaktion mit dem Computer dar. In einem VR-System betrachtet man die Daten nicht von außen wie auf einem Bildschirm, sondern kann in die visuellen Datenrepräsentationen eintauchen (*Immersion*). Die VR-Technologie ermöglicht es dem Benutzer, durch komplexe n-dimensionale Datenwelten zu fliegen und dort mit den Informationen zu interagieren. Die Systeme versorgen die Sinne des Hörens, Sehens und auch des Tastens, um dem Benutzer in eine n-dimensionale Welt einzuhüllen. Das Erlebnis der *Virtual Reality* kann in aufeinander aufbauende Ebenen unterteilt werden.

Die passive Ebene ist mit vielen alltäglichen Dingen vergleichbar. Egal ob wir fernsehen, radiohören oder lesen, es ist uns nicht möglich, in die uns präsentierte Erlebniswelt einzugreifen. Analog dazu existieren *Virtual Reality-Systeme*, die nur eine passive Wahrnehmung erzeugen, in der sich die künstliche Welt um den Nutzer bewegt, er sie aber nicht beeinflussen kann.

Die aktive Ebene stellt eine Art zweite Stufe der *Virtual Reality* dar. Hier ist dem Benutzer möglich, die ihm präsentierte Umgebung selbstständig zu erkunden. Die Möglichkeit der gesteuerten Bewegung in künstlichen Welten stellt einen großen Gewinn für die Funkionalität der VR dar. Man kann nicht nur eine Ansicht betrachten, sondern sich auch auf Elemente der Welt zubewegen, um sie zu untersuchen. Anwendungen dieser Ebene ermöglichen Bewegungen beliebiger Natur und verstärken dadurch den immersiven Eindruck für den Benutzer.

Die interaktive Ebene stellt die bislang höchste Stufe der Immersion dar. Hier ist es nicht nur möglich passiv wahrzunehmen, sondern man ist in die Lage versetzt, virtuelle Objekte

auch zu bewegen oder zu modifizieren. Die Interaktion mit der *Virtual Reality* setzt jedoch eine Vielzahl von Hard- und Softwarekomponenten voraus.

2.1.2. Augmented Reality

Neben vollständig computergenerierten Umgebungen existieren auch Systeme, die die Wirklichkeit um Virtuelles ergänzen. Bei dieser Ausprägung der *Virtual Reality* spricht man von *Augmented Reality*. In der *Augmented Reality* werden virtuelle Objekte und Informationen über den realen Eindruck der Umgebung projeziert. Die Wahrnehmung der realen Umwelt wird bei den verschiedenen Systemen unterschiedlich gelöst (Kapitel 2.2.1).

2.2. Immersive Systeme

Die Immersion, also das Gefühl sich wirklich in einer anderen Umgebung zu befinden, wird maßgeblich vom optischen Eindruck bestimmt. Es ist also vor allem wichig, die Umgebung so glaubwürdig wie möglich zu gestalten, indem dem Auge vom Aussehen und Verhalten so realistische Bilder wie möglich präsentiert werden. *Virtual Reality* kann mittels verschiedener Technologien erlebt werden. Man unterscheidet dabei zwischen *voll-* und *halbimmersiven* Ausgabesystemen. So gehören alle Geräte, die zum Einsatz in der *Augmented Reality* genutzt werden können, zu den halbimmersiven Systemen, da der Kontakt zur realen Welt nicht unterbrochen wird. Neben den Head-Mounted-Displays, die genauer im nächsten Abschnitt erläutert werden, existiert eine Vielzahl von Ansätzen zur Visualisierung virtueller Umgebungen.

Die einfachste Möglichkeit zur Realisierung einer dreidimensionalen Wahrnehmung sind die Rot-Grün Filterbrillen und ihre elektronischen Entsprechungen, die Shutter-Brillen. Sie erlauben es, einen dreidimensionalen Eindruck auf einer zweidimensionalen Projektionsfläche zu erzeugen, indem die Augen mit verschiedenen Bildern versorgt werden. Dies geschieht bei den einfachen Filterbrillen durch das Filtern von Farben und bei den Shutterbrillen durch das periodische verdunkeln der Brillen vor dem linken und rechten Auge. Diese Brillen dienen jedoch nur der einfachen Darstellung und sind keinesfalls als immersiv zu bezeichnen. Sie bilden jedoch die Grundlage für den *Responsive Workbench* und den *CAVE*.

Der Responsive Workbench aus einer Kooperation des früheren GMD[1]-Instituts und der Stanford Universität. Dabei werden vom Computer erzeugte, stereoskopische Bilder auf eine horizontale Fläche mittels eines Systems aus Projektoren und Spiegeln abgebildet. Um einen dreidimensionalen Effekt zu erhalten, wird am Responsive Workbench mit Shutterbrillen gearbeitet. Die Interaktion mit der virtuellen Umgebung wird dabei über eigens für

[1] Das GMD fusionierte 2001 mit dem Fraunhofer Institut.

das System entwickelte virtuelle Werkzeuge realisiert (Abb. 1). Die weiteren Möglichkeiten des Systems, wie z.B. das gleichzeitige Arbeiten von zwei Benutzern und die Entwicklung weiterer Interaktionsgeräte wird an der Stanford-Universität vorangetrieben. Unter http://graphics.stanford.edu/projects/RWB sind weitere Informationen zu diesem System zu finden. Das wohl aufwendigste und eindrucksvollste System ist das vom *Electronic*

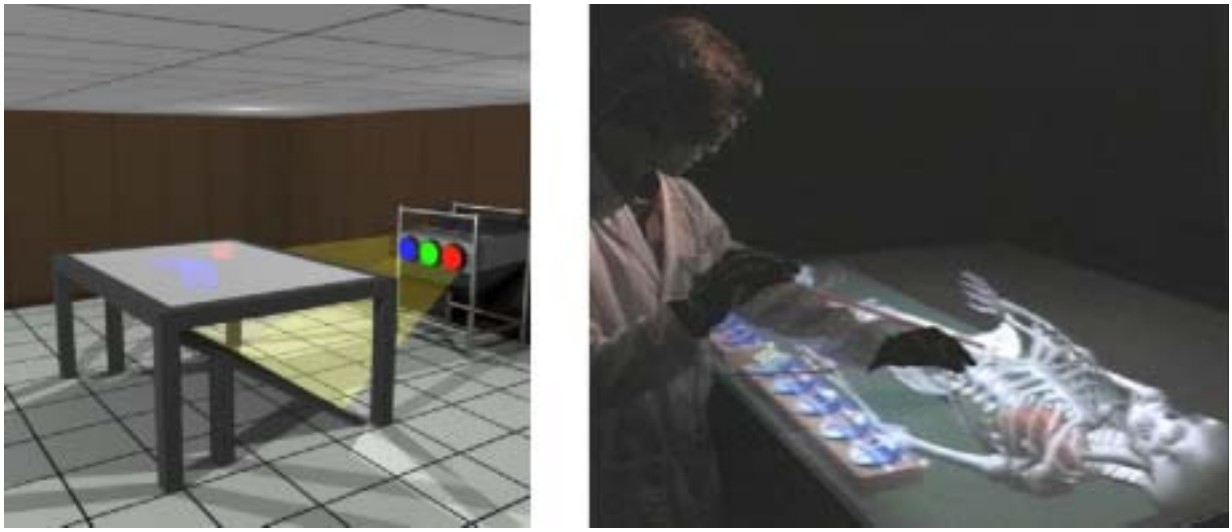

Abbildung 1: Der Responsive Workbench

Visualization Laboratory an der *Universität von Illinois* (http://cave.ncsa.uiuc.edu) entwickelte CAVE (Abb. 2). CAVE steht für *CAVE Automatic Virtual Environment* und bezeichnet einen meist würfelförmigen Raum, dessen Wände Projektionsflächen bilden. Diese Flächen werden durch Projektoren beleuchtet. Um im CAVE einen stereoskopischen Effekt zu erzielen, trägt der Benutzer eine Shutterbrille. Die Position und Orientierung des Benuters wird auch im CAVE mittels eines Trackers bestimmt. Die Trackerinformationen werden benutzt, um die korrekt stereoskopische Darstellung zu erzeugen. Darüber hinaus besteht die Möglichkeit, dass der Benutzer einen zweiten Sensor für die Interaktion mit der Umgebung trägt. Die Anzahl der verwendeten Projektionsflächen variiert bei verschiedenen CAVE-Installationen. Einige benutzen nur die vier Wände. Andere nutzen auch den Boden und die Decke des CAVE zur Darstellung der virtuellen Umgebung, was das Gefühl der Immersion verstärkt, aber auch mehr Rechenleistung voraussetzt. Der CAVE ermöglicht eine hohe Darstellungsqualität und zeichnet sich durch die Möglichkeit aus, auch mehrere Benutzer an der Umgebung teilhaben zu lassen. Die Projektion wird dabei jedoch nur an einen, den Tracker tragenden Benutzer angepasst. Trotz aller Möglichkeiten und Vorteile des CAVE ist dieses System aufgrund seines enormen Preises und der Anforderungen an Rechenleistung und Platz, nicht sehr weit verbreitet.

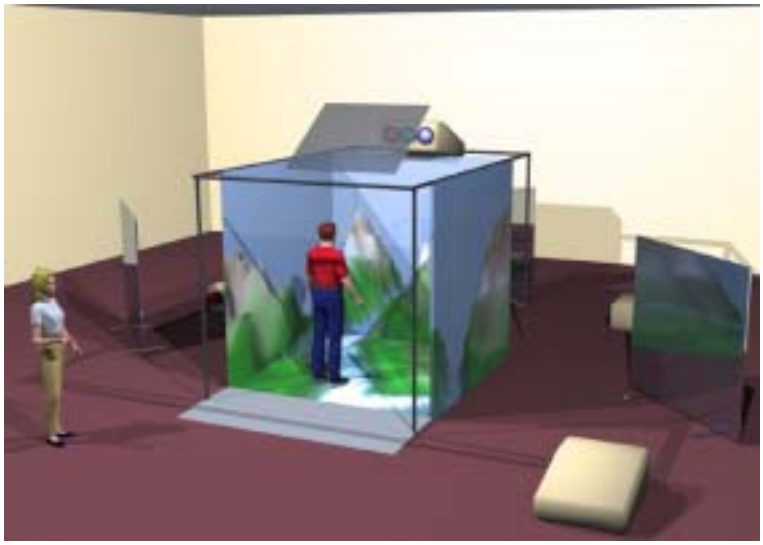

Abbildung 2: Darstellung eines vierseitigen CAVE Automatic Virtual Environment

2.2.1. Head-Mounted-Displays

Die Head-Mounted-Displays (HMD) bieten einen ähnlichen Grad der Immersion wie das CAVE, jedoch bei einem deutlich geringeren Preis und Aufwand an technischer Ausstattung. Das HMD ist im Prinzip ein Helm, an dessen Vorderseite zwei LC-Displays angebracht sind (Abb. 3).

Abbildung 3: Cybermind hi-Res900^{TM}3D

Damit der Mensch auf den Displays in unmittelbarer Augennähe scharf sehen kann (Kapitel 2.6.1), sind die HMDs zusätzlich mit einer Optik ausgestattet, die den Eindruck simuliert, dass der Benutzer auf eine große Projektionsfläche in gewisser Entfernung blickt. Im Fall

des für die Arbeit verwendeten Cybermind hi-Res900^{TM}3D wird durch die Optik ein 45 Zoll Display in zwei Metern Entfernung suggeriert.

Neben den optionalen Kopfhörern sind die HMDs überlichweise mit einem Ortungssystem, dem *Tracker*, ausgestattet. Der Tracker liefert die Orientierungsdaten des Kopfes, die dazu benutzt werden, die jeweilige Ansicht, die der Blickrichtung des Anwenders entspricht, zu errechnen und darzustellen. Die beiden Displays im HMD erlauben es, mit Hilfe einer stereoskopischen Darstellung der virtuellen Umgebung, eine räumliche visuelle Wahrnehmung zu erzeugen. Dazu werden für das rechte und linke Auge jeweils eigene Bilder erzeugt, die vom Gehirn zu einer dreidimensionalen Wahrnehmung verschmolzen werden. Die Funktionsweise dieses Verfahrens wird im Kapitel 2.6 im Detail erläutert.

Man unterscheidet grundsätzlich mehrere Arten von HMDs:

Display-HMDs sind der erste Typus, zu dem auch das von der FHTW bereitgestellte Cybermind hi-Res900^{TM}3D gehört. Sie zeichnen sich durch einen geschlossenen Helm aus und ermöglichen keine Kombination der virtuellen Umgebung mit der realen Welt. Aus diesem Grund gehören HMDs dieses Typs zu den vollimmersiven Systemen.

See-through-HMDs mischen die reale und virtuelle Welt. Dabei wird eine weitere Unterteilung in Abhängigkeit von der Art der Kombination von realer und virtueller Welt vorgenommen.

Optical-see-through-HMDs mischen Realität und computergenerierte Umgebung, indem halbtransparente Spiegel vor den Augen des Benutzers befestigt werden. Durch die teilweise Transparenz der Spiegel kann der Anwender direkt auf die reale Umgebung blicken, sogar wenn das HMD ausgeschaltet ist. Die virtuellen Objekte werden bei Optical-see-through-HMDs auf die halbtransparente Spiegel projeziert, die das Bild auf das Auge reflektieren.

Video-see-through-HMDs benutzen im Gegensatz dazu zwei am Helm angebrachte Kameras. Diese Kameras steuern die Benutzeransicht der realen Welt zur Gesamtansicht bei. Das Videobild der Kameras wird durch virtuelle Objekte ergänzt und kann auf den beiden LC-Displays vor den Augen des Benutzers dargestellt werden. Der Benutzer hat dabei keine direkte Sicht auf die reale Umgebung und ist faktisch blind, wenn das HMD deaktiviert ist. See-through-HMDs finden ihre Anwendung im Bereich der Augmented Reality und sind halbimmersiv.

2.3. Tracking von Kopfbewegungen

Die Lage- und Richtungsbestimmung ist eine der wichtigsten Voraussetzungen zur Erzeugung einer interaktiven *Virtual Reality*. Ohne die Möglichkeit, die Bewegungen des Benutzers messen zu können, wäre eine virtuelle Realität, die ähnlich der realen funktionieren soll, nicht denkbar. Das System muss zu jeder Zeit bestimmen können, wohin der Benutzer schaut und gegebenenfalls auch seine Position kennen, um mit Hilfe dieser Informationen, die gewünschte Darstellung der virtuellen Umgebung zu erzeugen. Das hier verwendete Cybermind hi-Res900^{TM}3D ist mit einem Tracker der Firma InterSense vom Typ InterTrax^2 ausgestattet. Dieser Tracker besitzt drei Freiheitsgrade (Degree of Freedom), d.h. er misst nur die Orientierung des Kopfes anhand von drei Winkeln, die die Rotation um die Raumachsen angeben. Diese Systeme werden als 3-DOF bezeichnet. Es exisieren auch Tracker mit sechs Freiheitsgraden (6-DOF). Sie messen zusätzlich die Position im Raum an den drei Achsen.

Da der InterTrax^2 eben keine solche Sensoren für die Bewegung im Raum besitzt, beziehen sich alle folgenden Betrachtungen nur auf die Orientierung des Kopfes.

2.3.1. Systeme

Head-Mounted-Displays besitzen in der Regel einen fest installierten Tracker, um die Orientierungsdaten des Kopfes zu bestimmen. Es existieren verschiedene Technologien, die zur Bestimmung benutzt werden:

Elektromagnetische Tracking-Systeme benutzen einen Sender, um elektromagnetische Felder entlang der orthogonalen Achsen zu erzeugen. Sensoren sind in der Lage, ihre Position relativ zum Sender zu bestimmen. Der Tracker ist frei beweglich und ermöglicht Messungen in allen Dimensionen. Darüber hinaus zeichnen sich diese Systeme durch hohe Auflösung bei der Orientierungsbestimmung aus. Ihre Winkelgenauigkeit liegt in der Regel bei unter 0.1°. Problematisch beim Einsatz dieser Tracker ist zum einen ihre Latenz von etwa zwei bis zehn Millisekunden, zum anderen ihre Anfälligkeit gegenüber metallischen Objekten in der Nähe des Trackers. Die Objekte können die Messfelder stören und Fehler erzeugen. Der InterTrax^2 ist ein solcher elektromagnetischer Tracker mit einer vom Hersteller angegebenen Latenz von vier Millisekunden und einer Winkelauflösung von 0.02°. Die weiteren technischen Details des Trackers sind im Kapitel 4.1.3 zu finden.

Ultraschall Tracking-Systeme nutzen ebensfalls einen Sender. Dieser strahlt Ultraschallwellen aus, die von einem oder mehreren Mikrophonen registriert werden. Die Position des

Anwenders wird durch Laufzeit- und Phasendifferenzmessungen ermittelt. Eine übliche Konfiguration ist es, drei Mikrophone in einem Dreieck anzuordnen und aus den Laufzeitdifferenzen die Position des Senders zu errechnen. Ultraschall-Systeme haben ähnliche Verzögerungen wie die elektromagnetischen Tracker, die durch die Laufzeit des Signals und die Berechnung der Position entstehen. Im Gegensatz zu den elektromagnetischen Trackern sind Ultraschall-Tracker zwar unempfindlich gegenüber magnetischen Störungen, aber ihre Funktionsweise kann durch Gegenstände, wie z.B. Körperteile, zwischen Sender und Mikrophonen beeinträchtigt werden.

Optische Tracking-Systeme benutzen eine Kombination aus LEDs, Videokameras und Methoden der Bildverarbeitung, um die Orientierung zu bestimmen. Dazu werden LEDs auf dem zu verfolgenden Objekt angebracht. Das Tracking wird dann über die Aufnahme der Szene mittels der Kamera und der Errechnung der Orientierung anhand der Abbildungen der LEDs auf der Aufnahme realisiert. Durch die Errechnung der Orientierung aus den Positionen der LEDs wird auch bei diesen Systemen eine Latenz erzeugt und sie sind, genauso wie die Ultraschall Tracker, anfällig für Störungen, die durch Objekte, die zwischen Kamera und LEDs liegen, ausgelöst werden.

Mechanische Tracking-Systeme sind eigentlich ein Verbindungsarm zwischen Helm und einem Gerät an der Zimmerdecke. Sobald sich der Kopf bewegt, überträgt der Arm diese Bewegung an ein Potentiometer, der die Bewegung in ein elektisches Signal umwandelt. Mechanische Systeme sind extrem genau und haben einen hohen Grad an Aktualisierung, schränken jedoch die Bewegungsfreiheit des Benutzers ein. Neben ihrer Genauigkeit zeichnen sich diese Systeme auch durch eine sehr geringe Latenz aus.

So unterschiedlich die Herangehensweise der einzelnen Systeme an das Problem des *Tracking* auch ist, haben doch alle die Gemeinsamkeit, dass keines von ihnen in der Lage ist, die Messdaten ohne Zeitverzögerung zu erfassen.

2.4. Fehlerquellen

Das Tracking von Kopfbewegungen und die Darstellung virtueller Welten mittels eines HMDs hält eine Vielzahl von Problemen und Fehlerquellen bereit, die für eine erfolgreiche Vorausberechnung durch den *Kalman-Filter* zu analysieren und zu bewerten sind. Man unterscheidet dabei zwischen statischen (*static errors*) und dynamischen (*dynamic errors*) Fehlern.
Als *static errors* werden alle Fehler bezeichnet, die *nicht* durch die Bewegung des Benutzers ausgelöst werden, also auch auftreten, wenn der Benutzer sich nicht bewegt. *Dynamic errors* treten auf, wenn der Benutzer seine Orientierung verändert. Sie haben den größten

Anteil am Gesamtfehler und werden durch Latenzen verursacht, die mittels des Vorhersagemechanismus´ des Kalman-Filters soweit wie möglich reduziert werden sollen.

2.4.1. Statische Fehler

Statische Fehler lassen sich in einige Hauptkategorien unterteilen:

- Fehler in den LC-Displays
- Schlechte Justierung des HMDs
- Fehler im Tracking-System
- Fehlerhafte Anzeigeparameter (FOV, Pupillenabstand, falsche Projektion etc.)

Die ersten drei Kategorien der statischen Fehler sind von der Software nicht zu beeinflussen und es bleibt deshalb dem Benutzer überlassen, diese gegebenenfalls zu bemerken und zu korrigieren.

Die letzte Kategorie setzt die Beachtung einiger Hardwareparameter (Kapitel 4.1) und die korrekte Implementierung des Grafiksystems voraus, insbesondere bei stereoskopischer Anzeige (Kapitel 2.6). Leider existieren in diesem Bereich auch Parameter, wie etwa der Augenabstand, die sowohl beim Benutzer, als auch beim HMD unveränderlich sind und bei nicht optimaler Abstimmung schon zu statischen Fehlern führen können. Aus diesem Grund ist bei der Benutzung eines HMDs unbedingt darauf zu achten, jede gebotene Einstellmöglichkeit zu nutzen, um unnötige Wahrnehmungsfehler und damit verbundene Probleme zu vermeiden.

2.4.2. Dynamische Fehler

Dynamische Fehler entstehen durch Latenzen des Systems. Dabei bezeichnet man als *end-to-end system delay* den Zeitabstand zwischen dem Moment, in dem das Tracking-System die Orientierung des Kopfes misst und dem Zeitpunkt, in dem die zur Position entsprechend errechneten Bilder im HMD erscheinen. Die Verzögerungen entstehen dadurch, dass jede am System beteiligte Komponente Zeit benötigt, um ihre Arbeit zu verrichten. Die Latenzen im Tracking-System, die Kommunikationsverzögerungen, die Zeit, die vom Grafiksystem benötigt wird, um die beiden stereoskopischen Ansichten in den Framebuffer zu zeichnen und zuletzt die Zeit, die benötigt wird, um die Bilder aus dem Framebuffer in die Displays zu kopieren, tragen zum end-to-end system delay bei.

Der Betrag des end-to-end system delay hängt von einer Vielzahl von Faktoren ab. Viele der

Einzellatenzen sind zudem nicht konstant und müssen vom System während der Ausführung kontinuierlich neu bestimmt werden. Pauschal kann man sagen, dass auf aktuellen Systemen diese Zeitverzögerung einen Betrag von mindestens 20 Millisekunden hat. Die Latenzzeiten können jedoch je nach Systemkonfiguration stark variieren. So ist es durchaus denkbar, dass der end-to-end system delay auf ausgelasteten Rechnern oder wenn beispielweise der Tracker über ein Netzwerk verbunden ist, auf 250 Millisekunden oder mehr ansteigt. Ebenso verheerend wirkt sich eine sehr komplexe virtuelle Umgebung auf den end-to-end system delay aus, da auch die grafische Darstellung einen Teil der Latenz darstellt.

Die Abfolge der Ereignisse, die zur Darstellung einer virtuellen Umgebung notwendig sind, lassen sich in einer Tabelle darstellen:

Zeitpunkt	Ereignis
t_0	Tracker misst die Orientierung
t_1	Tracker liefert Orientierungsdaten
t_2	Applikation empfängt Orientierungsdaten
t_3	stereoskopische Bilder sind im Framebuffer fertig gezeichnet
t_4	Framebuffer in den Grafikspeicher kopiert
t_5	Bilder werden auf den LC-Displays des HMD vollständig angezeigt

Tabelle 1: Ereignisabfolge in einem Head-Mounted-Display System

Der end-to-end system delay erzeugt nur Fehler in der Wahrnehmung, wenn der Benutzer seinen Kopf bewegt. Angenommen der Benutzer würde seinen Kopf vollkommen ruhig halten, dann würde die natürlich zu jeder Zeit vorhandene Verzögerung keinen Wahrnehmungsfehler produzieren, da ja unabhängig von der Länge des end-to-end system delay der Benutzer zwischen Messung der Kopforientierung und der Darstellung den Kopf nicht bewegt hätte.

Im gegensätzlichen Fall, also wenn der Benutzer eine kontinuierliche Bewegung ausführt, würde der Tracker die Orientierung des Kopfes zum Zeitpunkt t_0 messen. Die entsprechenden Bilder würden nach Ablauf der Verzögerung zum Zeitpunkt t_5 in den Displays des HMD erscheinen. In der Zwischenzeit verbleibt der Kopf aber in Bewegung und hat sich schon einige Grad weitergedreht, bevor die errechneten Bilder der Orientierung vom Zeitpunkt t_0 erscheinen. Das führt zu falschen Bildern und die Bilder scheinen zu schwimmen (*swimming*) oder hinterherzuziehen (*lag*). Besonders auffällig ist dieser Effekt bei Anwendungen der Augmented Reality, da durch die Mischung von virtuellen und realen Objekten eine rea-

le Referenzbewegung wahrzunehmen ist, die eben nicht mit der Bewegung der virtuellen Objekte übereinstimmt.

2.5. Mathematische Grundlagen

Dieses Kapitel gibt einen Überblick über die mathematrischen Grundlagen die zur Realisierung der Zielstellung notwendig sind. Dazu wird zunächst auf die verschiedenen Darstellungsformen von Orientierungen im Raum und ihre Konvertierungen untereinandern eingegangen.

2.5.1. Darstellung von Orientierungen

Die Tracker nutzen nicht nur verschiedene Möglichkeiten zur Messung der Orientierung, sondern auch zu deren Darstellung. Der verwendete InterTrax^2 bietet die Möglichkeit der Wahl zwischen Euler-Winkeln und Quaternionen. Diese Darstellungsmöglichkeiten werden in diesem Kapitel mit ihren Vor- und Nachteilen vorgestellt.

Rotationsmatrizen Als *Caley* 1857 die Matrizen einführte, dienten sie zur kompakten Darstellung von Gleichungssystemen. Heute gehören Matrizen zum Basiswissen aller Natur- und Ingenieurwissenschaftler. Im Bereich der Computergrafik sind Matrizen für die Manipulation von grafischen Objekten, zur Realisierung von Projektionen und für das Handling verschiedenener Koordinatensysteme unverzichtbar. So basiert die Funktionalität der populären Grafikschnittstellen OpenGL und DirectX auf Matrizen- und Vektorrechnung. Hier soll nur ein kleines Segment der Möglichkeiten der Matrizen, die homogenen Rotationsmatrizen im dreidimensionalen Raum, betrachtet werden, um als Grundlage für die Darstellungsform der *Euler-Winkel* zu dienen. Für jede der einzelnen Koordinatenachsen kann jeweils eine Matrix gebildet werden:

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_y = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_z = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Soll ein Vektor mittels der Matizen im dreidimensionalen Raum rotiert werden, so multipliziert man ihn mit der entsprechenden Matrix. Wenn z.B. der Einheitsvektor entlang der x-Achse $[1001]^T$ um 90° um die z-Achse (s. Matrix R_z) gedreht wird, so entsteht der Einheitsvektor entlang der y-Achse:

$$\begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

Eine Rotation um mehrere Achsen ist durch die Multiplikation der einzelnen Rotationsmatrizen realisierbar. Dabei ist jedoch unbedingt darauf zu achten, dass die Reihenfolge der Mutliplikationen immer gleich bleibt, da die Matrixmultiplikation nicht kommutativ ist. Die homogene Transformationsmatrix für die kombinierte Rotation um X, Y und Z (in dieser Reihenfolge) mit den Winkeln ϕ_x, ϕ_y und ϕ_z ist in Gleichung 1 zu sehen. Dabei steht aus Platzgründen c für $\cos$ und s für $\sin$.

$$R_{xyz} = \begin{pmatrix} c(\phi_y)c(\phi_z) & c(\phi_y)s(\phi_z) & -s(\phi_y) & 0 \\ s(\phi_x)s(\phi_y)c(\phi_z) - c(\phi_x)s(\phi_z) & s(\phi_x)s(\phi_y)s(\phi_z) + c(\phi_x)c(\phi_z) & s(\phi_x)c(\phi_y) & 0 \\ c(\phi_x)s(\phi_y)c(\phi_z) + s(\phi_x)s(\phi_z) & c(\phi_x)s(\phi_y)s(\phi_z) - s(\phi_x)c(\phi_z) & c(\phi_x)c(\phi_y) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (1)$$

Die Rotationsmatrizen sind die essentielle Grundlage aller Transformationen in OpenGL. Die lassen sich durch Multiplikation einfach untereinander kombinieren (z.B. Rotation und Translation). Darüber hinaus eignen sie sich um große Anzahlen von Punkten effizient zu transformieren. Leider ist es jedoch recht schwierig, sich anhand einer Matrix eine Rotation vorzustellen. Ein weiterer Nachteil ist, dass sie sich nicht interpolieren lassen.

Euler-Winkel sind wohl die gebräuchliste Form der Repräsentation einer Orientierung. Bei dieser Darstellung wird die Orientierung im Raum durch drei Winkel beschrieben, die als *Yaw*, *Pitch* und *Roll* bezeichnet werden. Diese urprünglich aus der Navigation von Schiffen

und Flugzeugen stammenden Bezeichnungen werden auch für die Orientierung von HMDs benutzt. Aufrecht sitzend, wird der Winkel der Rotation um eine gedachte Achse aus Hals und Wirbelsäule als Yaw bezeichnet. Pitch bezieht sich auf den Winkel beim Heben und Senken des Kinns. Roll ist der Winkel der Bewegung des Legens des Kopfes auf die eine oder andere Schulter (Abb. 4).

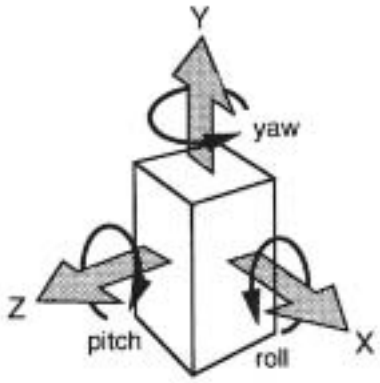

Abbildung 4: Definition von Yaw, Pitch und Roll im Koordinatensystem des Trackers

Die Orientierung wird dabei durch die Hintereinanderausführung der Drehungen um die x-, y- und z-Achse erzeugt. Diese Notation hat den Vorteil, dass sie recht einfach vorstellbar und deshalb intuitiv benutzbar ist. Besonders gute Eignung zur Ausrichtung einer virtuellen Kamera[2] machen die Euler-Winkel zu einer häufig benutzen Darstellungsform. Bei genauerer Betrachtung bergen die *Euler-Winkel* jedoch auch Probleme. Ein einfach zu lösendes Problem bezieht sich auf die Reihenfolge der sequentiellen Rotationen. Da die Reihenfolge der Rotationen das Ergebnis bestimmt, muss sie nur innerhalb einer Anwendung gleich definiert sein, um konsistente Ergebnisse zu erhalten. Bei einer XYZ-Reihenfolge würde also wieder Gleichung 1 gelten.

Schwieriger zu bewältigen ist jedoch ein Phänomen, dass als *Gimbal Lock* bezeichnet wird. Bei der Rotation einer Achse um 90° wird eine andere überlagert. Wenn nun zusätzlich eine Rotation um die überlagerte Achse durchgeführt wird, wird nicht mehr um die überlagerte Achse rotiert, sondern um die bereits gedrehte Achse. Dies führt in speziellen Situationen zu fehlerhaften Ergebnissen bei der Gesamtrotation. Man spricht auch vom Verlust eines Freiheitsgrades, da die Möglichkeit der Rotation um eine Achse durch den Gimbal Lock entfällt und man effektiv nur noch zwei Rotationsachsen zur Verfügung hat. Gleichung 2 zeigt die Auswirkungen des Gimbal Lock am Beispiel der Rotation bei $\phi_y = \frac{\pi}{2}$ um die Y-Achse.

[2] Inbesondere in Bezug auf OpenGL, da Camera-Up Vektor = Pitch

$$R_{xyz} = \begin{pmatrix} 0 & 0 & -1 & 0 \\ s(\phi_x - \phi_z) & c(\phi_x - \phi_z) & \mathbf{0} & 0 \\ c(\phi_x - \phi_z) & -s(\phi_x - \phi_z) & \mathbf{0} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2}$$

In Spalte drei kann man sehen, dass in den Zeilen zwei und drei Nullen entstehen, da $\cos(\frac{\pi}{2}) = 0$. Diese Null-Werte können dann nicht mehr beseitigt werden, da beliebiges Multiplizieren dieser Werte wieder Null-Werte erzeugt. Dadurch wird verhindert, dass die gewünschten folgenden Rotationen durchgeführt werden können.

Ein weiteres Problem, dass allerdings nur die Interpolation zwischen zwei *Euler-Winkeln* betrifft und deshalb hier nur kurz Erwähnung findet, ist ihre Mehrdeutigkeit. So beschreiben $R_{\pi,0,0}$ und $R_{\pi,\pi,0}$ zwar die gleiche Orientierung, aber die Interpolationspfade zwischen diesen Orientierungen und beispielsweise $R_{0,0,0}$ sind absolut unterschiedlich.

Quaternionen schaffen Abhilfe für viele Probleme der *Euler-Winkel*. *Quaternionen* wurden erstmal 1853 von William Rowan Hamilton beschrieben. Quaternionen, nach dem lateinischen *quattor* benannt, können als Erweiterung der komplexen Zahlen verstanden werden. Schon mit “normalen” komplexen Zahlen kann man eine zweidimensionale Rotation um den Winkel ϕ durch eine einfache Multipikation mit $cos\phi + isin\phi$ darstellen. Ein Quaternion hat eine reale und im Gegensatz zu komplexen Zahlen drei, statt zwei imaginäre Komponenten. Sei s die reale Komponente und x, y, z die drei imaginären, so gilt die Schreibweise

$$q := s + xi + yj + zk = s \cdot 1 + x \cdot i + y \cdot j + z \cdot k \tag{3}$$

mit den imaginären Einheiten i,j,k.
In Äquivalenz zu Einheitsvektoren in einem Vektorsystem existiert auch eine spezielle Klasse der *Einheitsquaternionen*. Sie beinhaltet alle *Quaternionen*, die folgenden Regeln entsprechen:

$$i^2 = j^2 = k^2 = ijk = -1 \quad ij = k \quad ji = -k \tag{4}$$

Eine Addition zweier Quaternionen enspricht prinizipiell der Addition von zwei Vektoren.

$$\begin{aligned} q_1 + q_2 = \quad & (s_1 + s_2, x_1 + x_2, y_1 + y_2, z_1 + z_2) \\ & (s_1 + s_2) \cdot 1, (x_1 + x_2) \cdot i, (y_1 + y_2) \cdot j, (z_1 + z_2) \cdot k) \end{aligned} \tag{5}$$

Für Quaternionen ist auch eine Multiplikation definiert:

$$\begin{aligned} q_1 \cdot q_2 &= (s_1 \cdot s_2 - x_1 \cdot x_2 - y_1 \cdot y_2 - z_1 \cdot z_2) \\ &+ (x_1 \cdot s_2 - s_1 \cdot x_2 - z_1 \cdot y_2 - y_1 \cdot z_2) \cdot i \\ &+ (y_1 \cdot s_2 - z_1 \cdot x_2 - s_1 \cdot y_2 - x_1 \cdot z_2) \cdot j \\ &+ (z_1 \cdot s_2 - y_1 \cdot x_2 - x_1 \cdot y_2 - s_1 \cdot z_2) \cdot k \end{aligned} \tag{6}$$

Bei der Multiplikation ist allerdings zu beachten, dass sie nicht kommutativ ist, d.h.

$$q_1 \cdot q_2 \neq q_2 \cdot q_1$$

Analog zu den Vektoren haben auch die Quaternionen einen Betrag $|q| = \sqrt{w^2 + x^2 + y^2 + z^2}$. Ist der Betrag eines Quaterions gleich 1, so spicht man von einem *Einheitsquaternion*. Alle Einheitsquaternionen kann man sich, wenn man zum Kreis der Auserwählten gehört die sich vierdimensionale Dinge vorstellen können, auch als einen Punkt auf einer vierdimensionalen Einheitskugel, d.h mit dem Durchmesser = 1 vorstellen. Ein Versuch der Darstellung einer solchen Kugel ist in Abbildung 5 zu sehen.

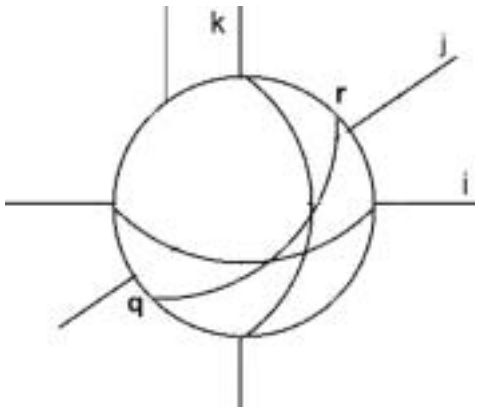

Abbildung 5: Einheitskugel mit zwei Quaternionen und deren Interpolationspfad

Um einen Punkt mittels eines Quaternions im dreidimensionalen Raum zu rotieren, ist auch die *Konjunktion* von nöten. Das zu einem Quaternion $q = (w, xi, yj, zk)$ konjunkte Quaternion ist definiert als $q' = (w, -xi, -yj, -zk)$.

Mit Hilfe des Betrags lässt sich das für ein Quaternion multiplikativ inverse Quaternion $q^{-1} = q'/|q|^2$ bilden. Handelt es sich um ein Einheitquaternion so gilt $q' = q^{-1}$.

Eine Rotation eines Punktes (x, y, z) um den Einheitsvektor (x_0, y_0, z_0) und den Winkel ϕ zu beschreiben bedient man sich der Multiplikation. Zunächst wird aus dem Punkt ein Quaternion erzeugt:

$$q_p = 0 + xi + yj + zk$$

Das Quaternion der Rotation ergibt sich als:

$$q_r = \cos\frac{\phi}{2} + \sin\frac{\phi}{2}x_0 i + \sin\frac{\phi}{2}y_0 j + \sin\frac{\phi}{2}z_0 k$$

Unter Zuhilfenahme des konjukten Quaternions von q_r ergibt sich der rotierte Punkt als:

$$q'_p = q \cdot p \cdot q'_r$$

In der Computergrafik werden Quaternionen häufig zur Steuerung von virtuellen Kameras und bei der *Keyframe-Animation* eingesetzt. Der Grund dafür ist, dass sie sich weitaus besser als Euler-Winkel und Matrizen interpolieren lassen. Die Interpolation von Quaternionen entspricht der Suche nach einem Großkreis auf der Einheitskugel. Abb. 5 zeigt den Interpolationspfad zwischen den Quaternionen q und r. Es existieren verschiedene Verfahren zum Interpolieren von Quaterionen. Die *Spherical Linear Interpolation (SLERP)*, die *Spherical Cubic Interpolation (Squad)* und die Spline Interpolation. Da die Interpolationen von Quaternionen in dieser Arbeit keine Anwendung findet, wird nicht näher darauf eingegangen. Mehr zum Thema der Quaternioneninterpolation ist in [Sho85] zu finden.

2.5.2. Konvertierung von Quaternionen

Für diese Arbeit sind Quaternionen nur in soweit nützlich, als dass sie vom Tracker ausgelesen und mittels des Kalman-Filters vorhergesagt werden. Um die Quaternionendarstellung für die Steuerung der OpenGL-Kamera zu nutzen, ist es notwendig, sie in eine Rotationsmatrix konvertieren zu können.

Die Konvertierung eines Einheitsquaternions in eine Rotationsmatrix ist in Gleichung 7 zu sehen.

$$R_{xyz} = \begin{pmatrix} 1-2(y^2+z^2) & 2xy-2sz & 2xz+2sy & 0 \\ 2xy+2sz & 1-2(x^2+z^2) & 2yz-2sx & 0 \\ 2xz-2sy & 2sx-2yz & 1-2(x^2+y^2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7}$$

Man kann auch aus einer Rotationsmatrix ein Quaternion gewinnen. Eine 4×4 Matrix (siehe Gleichung 8), wie sie in OpenGL gebräuchlich ist, muss dabei die Voraussetzung erfüllen,

dass $M_{03} = M_{13} = M_{23} = M_{30} = M_{31} = M_{32} = 0$ und $M_{33} = 1$.

$$R_{xyz} = \begin{pmatrix} M_{00} & M_{01} & M_{02} & M_{03} \\ M_{10} & M_{11} & M_{12} & M_{13} \\ M_{20} & M_{21} & M_{22} & M_{23} \\ M_{30} & M_{31} & M_{32} & M_{33} \end{pmatrix} \tag{8}$$

Sind diese Voraussetzungen erfüllt, so ergibt sich das Quaterion, das der gleichen Rotation entspricht wie die Matrix M als

$$q = (\pm\sqrt{M_{00} + M_{11} + M_{22} + M_{33}}, \frac{M_{21} - M_{12}}{4s}i, \frac{M_{02} - M_{20}}{4s}j, \frac{M_{10} - M_{01}}{4s})k$$

2.6. Stereoskopische Projektion

Die räumlich visuelle Tiefenwahrnehmung des Menschen beruht auf verschiedenen Faktoren, die eine unterschiedlich starke Tiefenwirkung erzeugen [ST95]. Die einzelnen Faktoren können dahingehend charakterisiert werden, ob sie über ein Auge (*Monokulare Tiefenkriterien*) oder über beide Augen (*Binokulare Tiefenkriterien*) wahrgenommen werden. Die binokularen Tiefenkriterien haben zwar im für HMD-Anwendungen entscheidenden Nah- und Mittelbereich die stärkste Wirkung, dennoch spielt die Beachtung aller Tiefenkriterien eine Rolle bei der Konstruktion von HMDs und bei der Erstellung von Software zur Stereoprojektion.

Um eine realistische virtuelle Umgebung zu erschaffen, ist das Wissen über die Tiefenwahrnehmung der Menschen unabdingbar. Es müssen die wahrnehmungsphysiologischen und -psychologischen Vorgänge der Tiefenwahrnehmung berücksichtigt werden, da ansonsten das Gehirn nicht mehr in der Lage ist, die für die beiden Augen generierten Einzelbilder zu einer räumlichen Szene zu verschmelzen. Der Benutzer nimmt in diesen Fällen Doppelbilder, die sogenannten *Geisterbilder* wahr.

2.6.1. Tiefenwahrnehmung

Das Auge stellt den Ausgangspunkt unserer visuellen Wahrnehmung dar, die durch adäquate Reize ausgelöst werden. Adäquat sind Reize für das menschliche Auge bei Wellenlängen zwischen 360 und 760 nm des elektromagnetischen Spektrums. Dieser Bereich des Spektrums wird auch sichtbares Licht genannt und beinhaltet die bekannten Spektralfarben rot, orange, gelb, grün, blau und violett.

Beim Einfall von sichbarem Licht durch die transparente Hornhaut (*Cornea*) ins Auge, werden die Lichtstrahlen durch die *Linse* hinter der *Pupille* gebrochen und auf die Netzhaut

(*Retina*) projeziert. Auf der Retina entsteht dadurch eine um 180° gedrehte Projektion der Außenwelt. Der von den Augen fixierte Punkt (*Blickpunkt*) wird auf die am besten auflösende Stelle der Retina, die Sehgrube (*Fovea*), abgebildet [ST95]. In der Retina befinden sich zwei Arten von lichtempfindlichen Sehzellen.

Die etwa 120 Millionen *Stäbchen* und 6 Millionen *Zapfen* wandeln das Licht unterschiedlich in elektrische Nervenimpulse um, die dann im Gehirn ein Bild entstehen lassen. Die Stäbchen besitzen eine höhere Lichempfindlichkeit als die Zapfen und sind für das Sehen bei geringer Helligkeit, wie etwa in der Dämmerung, zuständig. Sie nehmen allerdings nur verschiedene Grautöne, schemenhafte Abbildungen und Bewegungseindrücke wahr. Im Gegensatz dazu sind die Zapfen in der Lage, ab einer gewissen Helligkeit Farben zu sehen. Zudem haben sie mit 50 Hz eine höhere zeitliche Auflösung als die Stäbchen mit 20 Hz und befinden sich vornehmlich im Bereich der optischen Achse (*Fovea*). Dort werden die Lichtstrahlen aufgrund der hohen Dichte der Zapfen am genauesten abgebildet. In der Sehgrube gibt es jedoch keine Stäbchen. Deren Konzentration nimmt erst in der Netzhautperipherie zu [ST95].

Die von den Sehzellen erzeugten Impulse werden über den Sehnerv, der an der lichtunempfindlichsten Stelle - dem *blinden Fleck* - in das Auge mündet, an das Gehirn geleitet. Der Weg zum Gehirn führt über die partielle Sehbahnkreuzung (*Chiasma opticum*). Dort werden die Bildinformationen über die linken Hälften des *Gesichtsfeldes* beider Augen in die rechte Gehirnhälfte geleitet und umgekehrt [ST95]. Das Gesichtsfeld umfasst dabei all diejenigen Objekte der Außenwelt, die bei ruhendem Auge sowie fixiertem Kopf und Körper überblickt werden können.

Im Gehirn angekommen, erzeugen die Nervenimpulse ein Abbild des Netzhautbildes im Sehzentrum (*visuelle Cortex; Aria striata*). Durch die Überlappung der beiden Gesichtsfelder der Augen werden die meisten Objekte des Gesichtsfeldes in beiden Augen abgebildet. Die Nervenfasern dieser Stellen der Netzhäute, auf denen die gleichen Informationen eines Objektes in der Fixierungsentfernung in beiden Augen abgebildet sind, sind mit derselben Stelle des visuellen Cortex verbunden. Diese Stellen werden *korrespondierende Netzhautstellen* bezeichnet.

Die Korrenspondenz der Netzhautstellen muss für jede Fixierungsentfernung neu eingestellt werden. Das menschliche Auge bedient sich dazu zweier Mechanismen:

Die Akkomodation beschreibt die Nah- bzw. Ferneinstellung des Auges. Der *Ciliarmuskel* bewirkt durch Kontraktion oder Entspannung eine unterschiedliche Krümmung der Linse. Die dadurch erreichte dynamische Brennweiteneinstellung der Linse erlaubt es dem menschlichen Auge, Objekte in einer Entfernung zwischen 10 cm und 5-6 m scharfzustellen. Diese

Akkomodationsfähigkeit ist jedoch stark altersabhängig, da die Spannkraft des Ciliarmuskel im Alter nachlässt.

Die Konvergenz der Bildachsen muss sich beim Betrachten von Objekten in unterschiedlichen Distanzen anpassen, damit sich die optischen Achsen der beiden Augen in dem gewünschten Fixpunkt schneiden. Die Korrespondenz der Netzhautstellen kann jedoch nur für einen Punkt in der Tiefe durch Konvergenz erreicht werden. Dieser angeblickte Punkt wird als *Fixierpunkt* bezeichnet.

Akkomodation und Konvergenz werden auch als Okulomotorische Tiefenkriterien bezeichnet, da die Muskelbewegung dem Gehirn auch Informationen über die betrachtete Szenen liefert. Diese Kriterien liefern dem Körper jedoch nur schwache Hinweise und das auch nur bis zu einer Entfernung von bis zu 3 Metern zum fokussierten Objekt.

Die dreidimensional visuelle Wahrnehmung ist jedoch mehr als das Fixieren von Objekten im Raum. Tiefeneindrücke werden im Gehirn aufgrund von Kriterien erzeugt, die zum einen erlernt oder angeboren sein können und die zum anderen auf Informationen eines oder beider Augen beruhen.

Monokulare Tiefenkriterien sind für das visuelle Erleben von drei Dimensionen unter Mithilfe nur eines Auges verantwortlich. Über diese Fähigkeit zur Bewertung der monokularen Tiefenkriterien verfügt der Mensch nicht von Geburt an, sondern muss sie erst im Laufe des Lebens durch Erfahrungen entwickeln. Zu den monokulare Tiefenkriterien gehören die *Perspektive*, *Größenkonstanz*, *Verdeckung*, *Luftperspektive*, *Bewegungsparalaxe* und das Zusammenspiel von *Licht und Schatten*.

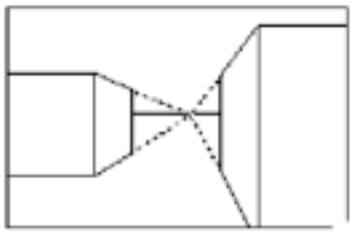

Abbildung 6: Perspektive

Die Größe der Projektion eines Objektes auf der Netzhaut nimmt durch die Akkomodation mit zunehmender Entfernung ab. Dadurch scheinen Linien, die parallel in die Ferne verlaufen, sich in einem Punkt zu schneiden. Nähergelegene Teile eines Objektes erscheinen größer und weiter auseinanderliegend als weiter entfernte. Sind dem Betrachter die Größe und Art der Objekte bekannt, so kann er aufgrund seiner Erfahrungen eine Abschätzung der absoluten und relativen Entfernungen vornehmen. Diese Erfahrungen ermöglichen es dem Sehsystem, die Phänomene der *Perspektive* als Darstellung der Größenverhältnisse in einer Ebene und der *Größenkonstanz* richtig zu deuten. Objekte, die sich weiter hinten in einem Bild befinden,

werden nicht als kleiner, sondern als weiter entfernt wahrgenommen.

Die *Verdeckung* eines Objektes im Gesichtsfeld durch ein anderes, löst den Hinweisreiz aus, dass sich das verdeckte Objekt weiter vom Betrachter entfernt befinden muss, als das andere. Diese Information ist allerdings nur relativ, da sie die Abschätzung absoluter Distanzen nicht zulässt. Das Verdeckungsphänomen gewinnt in Situationen an Gewicht, in denen keine anderen Informationen als Grundlage einer Schätzung vorhanden oder diese nicht eindeutig sind.

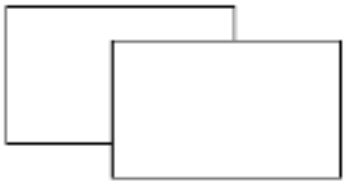

Abbildung 7: Verdeckung

Die *Luftperspektive* entsteht durch kleine Partikel wie Staub, Wassertröpfchen und Verschutzungen aller Art innerhalb der Atmosphäre. Sie brechen und schwächen das Licht beim Betrachten von entfernten Objekten. Durch die Partikel hindurch betrachtet, erscheinen Objekte unscharf und etwas bläulich. Diese Luftstreuung erzeugt die Luftperspektive, bei der weiter entfernte Objekte farblich entsättigt werden. Bei einer zweidimensionalen Darstellung erhält man durch die Anwendung der Luftperspektive eine starke Tiefenwirkung.

Abbildung 8: Luftperspektive

Bewegt sich der Betrachter, scheinen die Netzhautbilder von Objekten in verschiedenen Distanzen mit verschiedenen Geschwindigkeiten vorbeizuziehen. Diese *Bewegungsparalaxe* ist zum Bespiel beim Zugfahren alltäglich, wenn man ein Waldstück passiert und aus dem Fenster blickt. Dabei bewegen sich Objekte, die weiter entfernt sind als der Fixierpunkt, in dieselbe Richtung wie das Auge. Dagegen bewegen sich näherliegende Objekte in die Gegenrichtung. Dadurch entsteht eine unterschiedliche Verdeckung der Objekte oder der Bäume in Abhändigkeit von der Position des Beobachters. Das Phänomen der Bewegungsparalaxe liefert einen starken Tiefenhinweis. Die Bewegungsparalaxe wird auch als *Bewegungsindiziertes Tiefenkriterium* bezeichnet. Dazu kann auch die Verdeckung gezählt werden, wenn sie sich durch die Bewegung der Objekte verändert.

Abbildung 9: Bewegungsparalaxe

Das Zusammenspiel von *Licht und Schatten* wird durch Erfahrung ebenfalls zu einem wichigem Tiefenkriteritum. Es gibt Auskunft über die dreidimensionale Form von Objekten und die Lage der Lichtquelle. Der Schatten visualisiert sozusagen die Verdeckung des Lichts durch ein Objekt. Selbst in der einfachen Abbildung auf der rechten Seite, wird dies deutlich. Die beiden Kugeln haben die gleiche projizierte Größe. Dennoch lässt ihr unterschiedlicher Schattenwurf sie in unterschiedlichen Entfernungen erscheinen.

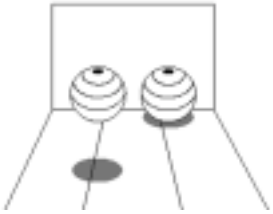

Abbildung 10: Licht und Schatten

Neben den detailliert erläuterten Kriterien existieren noch weitere. So kann das Gehirn auch aus der wahrgenommenen Größe bekannter Objekte (Auto, Haus etc.) Tiefeninformationen gewinnen. Auch die Wahrnehmung von Strukturen, der *Texturgradient*, dient ebenfalls als Tiefenhinweis, da gewöhnlich die Dichte der Struktur mit zunehmender Entfernung ebenfalls größer wird.

Binokulares Tiefenkriterium - Die Querdisparation ist wohl das wichtigste Kriterium zur Wahrnehmung von räumlicher Tiefe und basiert auf dem aus dem Augenabstand resultierenden, unterschiedlichen Blickwinkeln der beiden Augen. Beim Betrachten eines Raumpunktes richten wir unsere Augen mit Hilfe der Fähigkeiten zur Akkomodation und Konvergenz so aus, dass der Fixierpunkt von beiden Augen auf der Fovea abgebildet wird. Der unterschiedliche Blickwinkel bewirkt, dass einander entsprechende Bildpunkte einen Abstand, *Querdisparation* genannt, auf der Retina erzeugen. Dieser Abstand liefert dem Betrachter Informationen über die räumliche Tiefe, da ihr Betrag von der Entfernung der betrachteten Objekte abhängig ist. Da sich die Querdisparation in den Augen nicht direkt messen lässt, verwendet man stattdessen die *Stereoskopische Parallaxe*. Die Querdisparation wird bis zu einer Objektentfernung als dominanter Tiefenhinweis angesehen und kann bis zu einer Distanz von 450 Metern wirken.

Stereoskopische Parallaxe beschreibt den Abstand korrespondierender Bildpunkte im Bildraum. Eine stereoskopische Anzeige unterscheidet sich von einer herkömmlichen nur in der Möglichkeit, Bildpunkte mit Parallaxwerten darstellen zu können. Diese Parallaxe erzeugt Querdisparation in den Augen und schafft damit die Möglichkeit für räumliche Wahrnehmung. Parallaxe und Querdisparation sind also

zwei zusammenhängende Einheiten, da die Parallaxe den
Abstand zweier Punkte auf dem Bildschirm und die Querdisparation zweier Punkte auf der Netzhaut beschreibt. Die Parallaxe erzeugt also gewissermaßen die Querdisparation, welche eigentlich für die Entstehung des räumlichen Bildes im Gehirn sorgt. Da sie also sehr wichtig ist, wird ihren unterschiedlichen Ausprägungen noch mehr Aufmerksamkeit gewidmet. Man unterscheidet vier Arten von Parallaxen, die hier illustriert werden sollen:

Die erste Ausprägung wird *null-Parallaxe* genannt und entsteht, wenn die Bildpunkte beider Halbbilder genau übereinander liegen. Man könnte auch sagen, dass sich die optischen Achsen der beiden Augen in einem Punkt auf der Ebene des Bildschirmes schneiden, wie in Abbildung 11 dargestellt. Diese Art der Parallaxe wird später benutzt, um Textinformationen in der stereoskopisch dargestellten Umgebung einzublenden.

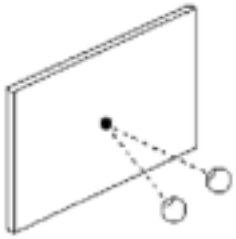

Abbildung 11:
Null-Parallaxe

Die Abbildung 12 links zeigt eine positive Parallaxe. Man spricht von positiver Paralaxe, wenn der Abstand der optischen Achsen auf der Bildschirmebene mindestens dem Augenabstand entspricht. Für einen speziellen Fall von positiver Parallaxe sind die optischen Achsen der beiden Augen parallel. Dies geschieht in der Natur bei der Betrachtung von Objekten in großer Ferne. Verwendet man jedoch bei stereoskopischen Darstellungen Parallaxwerte nahe oder gleich dem Augabstand, so erzeugt dies beim Betrachter Unbehagen.

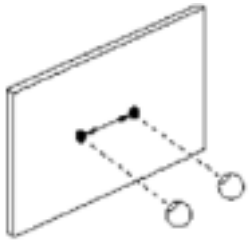

Abbildung 12:
Positiv-Parallaxe

Eine weitere Ausprägung positiver Parallaxe ist die Verwendung von Werten, die größer als der Augabstand sind (Abb. 13). Diese lässt die optischen Achsen der Augen divergieren und zwingt die Augmuskulatur zu anstrengenden, ungewöhnlichen Bewegungen bei der Fokussierung solcher Bilder, da diese Sichtweise in der Natur nicht vorkommt. Deshalb gibt es auch keinen Grund, diese Art der Bilderzeugung zu verwenden.

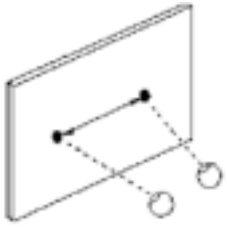

Abbildung 13:
Divergenz-Parallaxe

Sämtliche Objekte mit ungekreuzten, also positiven Parallax-Werten zwischen der null-Parallaxe und dem Augabstand, erzeugen Bilder, die als “innerhalb bzw. hinter dem Bildschirm” wahrgenommen werden. Sie sind also im Bildschirm-Raum.

Kommt es hingegen zu einer Kreuzung der optischen Achsen der Augen, wie in Abbildung 14 zu sehen, so spricht man von gekreuzter oder negativer Parallaxe. Objekte mit negativer Parallaxe erscheinen näher als die Ebene des Bildschirms, also zwischen Betrachter und Bildschirm. Man spricht hierbei vom *Benutzer-Raum*.

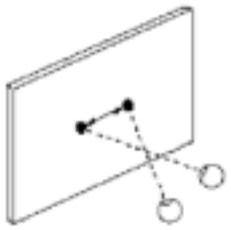

Abbildung 14:
Negativ-Parallaxe

Besonders wichtig ist es, die Kongruenz der beiden Halbbilder zu erhalten. Beide Bilder müssen also bis auf die Abweichung der horizontalen Parallaxe absolut identisch sein, da die Betrachtung für den Benutzer sonst belastend ist. Ein nicht zu unterschätzender Faktor ist das Verhältnis zwischen Fokussierung und Konvergenz der Augen. Beim Betrachen von Bildern der realen Welt sind die Augen auf jenen Punkt fokussiert, in dem auch ihre optischen Achsen konvergieren. Dieses Verhältnis aus Fokus und Konvergenz ist antrainiert und die Anpassung geschieht automatisch beim Betrachten unterschiedlicher Objekte.

Stereoskopische Bilder bilden hierbei eine Ausnahme, denn bei ihnen konvergieren die Augachsen, als ob Bildteile in unterschiedlichen Entfernungen liegen würden, wohingegen die Pupillen ständig auf die Bildschirmebene fokussiert bleiben müssen. Da die Kontrolle der Muskulatur für die Fokussierung bzw. die Ausrichtung der optischen Achsen in diesen Fällen vom angelernten Verhalten abweicht, kann es für manche Personen unangenehm sein, solche stereoskopischen Bilder zu betrachten, besonders wenn sie hohe Parallaxwerte aufweisen. Es hat sich gezeigt, dass es besser ist, die kleinstmöglichen Werte für Parallaxe zur Erreichung des Tiefeneffektes zu wählen und dabei eher die Perspektive zu strapazieren, um die Anpassung der Augen mit geringer Anstrengung zu ermöglichen und trotzdem gute Tiefeneffekte zu erhalten.

2.6.2. Folgerungen für die Computersimulation

Unsere visuelle Wahrnehmung erfolgt über unsere beiden Augen. Dabei schaut das linke Auge etwas von links, das rechte etwas von rechts auf den jeweils fokussierten Punkt. Durch diese minimale Verschiebung entstehen perspektivische Unterschiede in den beiden Bildern der Augen. Unser Gehirn besitzt die faszinierende Fähigkeit diese beiden Bilder zu einem räumlichen Bild mit Tiefenwirkung zusammenzusetzen. Da 3D-Applikationen, auch wenn die

Ausgabe auf einem zweidimensionalem Medium erfolgt, in der Regel perspektivische Projektionen anwenden, ist es nicht besonders schwierig, eine Stereodarstellung zu erreichen. Die Problematik liegt in der Art und Weise, wie die Perspektive für das rechte und linke Auge errechnet wird.

2.6.3. Stereoskopische Projektion

Die Erzeugung eines stereoskopischen Bildes benötigt also zwei monokulare Ansichten, die aus unterschiedlichen Positionen erzeugt werden müssen. Es existiert eine Reihe von Ansätzen, um eine stereoskopische Projektion zu realisieren. Eine Möglichkeit wird *toe-in* genannt. Dabei werden zwei symmetrische Frusta benutzt und die Kameras der beiden Augen fokussieren einen gemeinsamen Punkt (Abb. 15). Beim toe-in entsteht zwar auch ein stereoskopischer Effekt, dennoch ist von der Verwendung dieses Verfahrens abzuraten, da sie durch perspektivische Verzerrungen des Objekts auch eine vertikale Parallaxe erzeugt, welche nichts zum Tiefeneffekt beiträgt, aber zu einer unangenehmen Belastung der Augenmuskulatur führt.

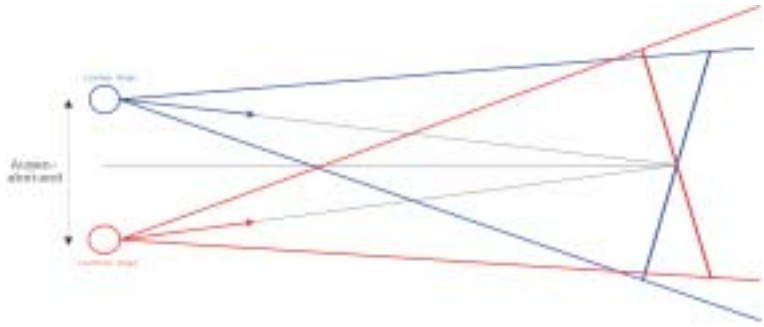

Abbildung 15: Strereoprojektion durch Achsendrehung

Die korrekt Methode wird als *off-axis* bezeichnet. Bei dieser Methode entsteht keine unerwünschte vertikale Parallaxe und somit ist die Betrachtung solcher Bilder weniger anstrengend. Dabei werden die Projektionszentren horizontal verschoben. Die optischen Achsen bleiben jedoch parallel, was zwei asymmetrischen Frusta gleichkommt. Nur diese Methode liefert korrekte Stereo-Paarbilder.
Der Grad des stereoskopischen Effektes hängt von der Distanz der Kamera zur Projektionsfläche und vom Augenabstand ab. Ein zu großer Abstand kann zu Geisterbildern führen und wird auch als *hyperstereo* bezeichnet. Allgemein ist ein zwanzigstel des Abstandes zur Projektionsfläche das Maximum an Augenabstand. Außerdem sollte darauf geachtet werden, dass die negative Parallaxe für keinen Punkt größer wird als der Augenabstand.

Gewöhnlich wird der Parallaxwinkel Θ errechnet durch $\Theta = 2 \arctan \frac{DX}{2d}$. DX beschreibt den

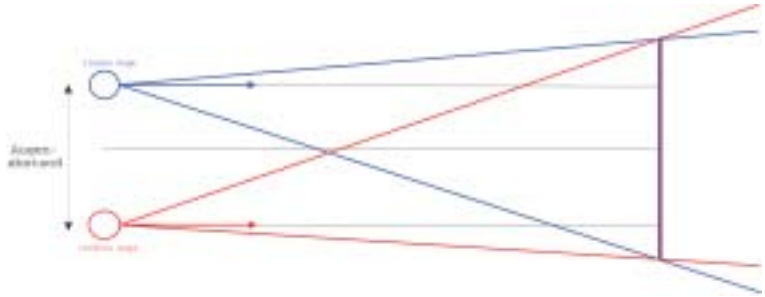

Abbildung 16: Strereoprojektion nach der off-axis Methode

Abstand der Abbildungen eines Punktes in den beiden Halbbildern und d ist der Abstand der Augen von der Projektionsfläche (Abb. 17). Der Wert von Θ sollte jedoch für alle Punkte der dargestellten Szene nie größer werden als 1,5 Grad. Allen Restriktionen zum Trotz sollte der Benutzer die Möglichkeit haben, den Parallaxwert und den Augenabstand während der Betrachtung einstellen zu können.

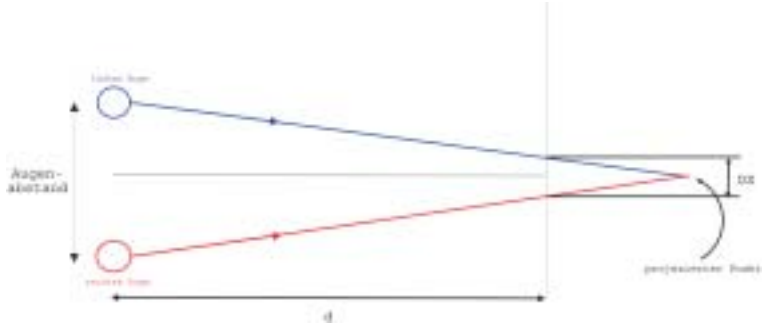

Abbildung 17: Errechnung der Parallaxe

Da sich der Benutzer in der Szene bewegen kann, ist es ohnehin schwierig, die Werte im voraus genau zu bestimmen. Außerdem lassen sich Probleme bei der Projektion aus dem gleichen Grund nicht vollständig eliminieren, da sich der Benutzer einem Objekt auch extrem nähern kann und dabei nicht auszuschließen ist, dass die Projektion unangenehm wird, da dadurch eine sehr starke negative Parallaxe entsteht.

2.6.4. Technische Realisierung

Die Grundlagen zur Errechnung der Stereobilder wurden in den letzten Kapiteln erläutert. Um die Einzelbilder auf den des Displays des HMD darzustellen existieren verschiedene Möglichkeiten. Das Cybermind hi-Res900^{TM}3D unterstützt dazu drei Verfahren. Die am

einfachsten zu realisierende ist der Anschluss des HMD an eine sogenannte *Dual-Head-Grafikkarte*. Diese Grafikkarten verfügen über zwei Anschlüsse, so dass das linke und rechte Halbbild getrennt und mit voller Bildwiederholfrequenz übertragen werden kann. Diesen Modus bezeichnet man als *Dual Input Mode* und er ist in jedem Fall zu bevorzugen, vorausgesetzt die entsprechene Grafikhardware steht zur Verfügung.

Die Nutzung des HMD zur stereoskopischen Ausgabe kann jedoch mit nur einem VGA-Anschluss realisiert werden. Dabei müssen jedoch Qualitätsverluste in Kauf genommen werden.

Beim *Interlaced Mode* werden die beiden Bilder in einem kombiniert (Abb. 18). Dazu werden alle Informationen für das rechte Halbbild in die geraden und alle Informationen für das linke in die ungeraden Zeilen des Gesamtbildes kopiert. Man nutzt Framebuffer mit voller Breite, jedoch nur halber Höhe für das linke und rechte Auge. Das HMD filtert das Gesamtbild und zeigt die entsprechenden Zeilen im linken oder rechten seiner Displays an. Bei der Verwendung des Interlaced Mode wird das Bild zwar mit voller Bildwiederholfrquenz dargestellt, jedoch reduziert sich die Anzahl der Zeilen um 50 Prozent, was sich negativ auf die Bildqualität auswirkt. Besonders Helligkeit und Kontrast des Bildes verringern sich. Ein positver Nebeneffekt des Interlaced Mode ist der verringerte Speicherbedarf, da die Bilder nur die halbe Höhe haben.

Abbildung 18: Kombiniertes Stereobild beim Interlaced Mode

Daneben unterstützt das HMD noch den Modus des *Page-Flipping*. Es werden zwei komplette Halbbilder (Abb. 19) erzeugt, die abwechselnd an das HMD übertragen werden. Dabei kommt es zur Reduktion der Bildwiederholfrequenz auf die Hälfte. Bei der Nutzung des Page-Flipping ist unbedingt darauf zu achten, dass die Basis-Bildwiederholfrequenz so hoch wie möglich gewählt wird, da durch die Halbbierung ein starkes Flimmern ausgelöst werden kann, dass vom Benutzer als sehr störend wahrgenommen wird.

Neben den vom Cybermind hi-Res900^{TM}3D unterstützten Modi existieren noch weitere.

Abbildung 19: Stereohalbbilder für linkes und rechtes Auge beim Page-Flipping oder Dual Input Mode

Beim *Above-and-Below* und beim *Side-by-Side* werden die Halbbilder neben- bzw. untereinandern zu einem Bild kombiniert.

Die verschiedenen Modi setzen wie schon erwähnt eine Hardwareunterstützung voraus, da das HMD in der Lage sein muss, die jeweiligen Halbbilder zu trennen und auf den entsprechenden Displays darzustellen. Die Steuerung der Übertragung durch den Grafikkartentreiber in Zusammenarbeit mit dem Grafiksystem (hier OpenGL) übernommen.

Während der Entwicklung dieser Arbeit wurde das Verfahren des Page-Flipping eingesetzt, da es sich sowohl für die Vollbild- als auch für die Fensterdarstellung eignet und ein Monitor für die Programmüberwachung frei bleibt. Nach Abschluss der Entwicklung wird wegen der besseren Qualität jedoch auf den Dual-Input-Mode gewechselt.

2.7. Motion Sickness

Das beim Arbeiten mit einem immersiven System auftretende *end-to-end system delay* beeinträchtigt nicht nur das Gefühl der Immersion, sondern kann auch negative Auswirkungen auf den Körper des Benutzers haben. Ein Gefühl von Unbehagen, Desorientierug, Übelkeit und Schweißausbrüche oder Ermattung können Symptome der sogenannten *Motion Sickness* sein. Die Motion Sickness kann in einer Vielzahl von Situationen auftreten. So können die Auswirkungen bei Flugpassagieren, Fahrzeuginsassen oder Benutzern von immersiven Systemen beobachtet werden. In Bezug auf das Problemfeld der computergenerierten Umgebungen wird auch der Begriff *Simulator Sickness* verwendet. Ein bekannter und allgemein akzeptierten Ansatz, um die Phänomene zu erklären, ist die *Sensor Konflikt Theorie* [VJ00]. Diese Theorie führt die Motion oder Simulator Sickness auf Konflikte zwischen dem *vestibularen* und dem visuellen System des Menschen zurück. Das vestibulare System befindet sich im Innenohr und besteht aus Rezeptoren, die die Orientierung, Position und Neigung des Kopfes an das Gehirn melden. Die Konflikte zwischen der optischen Wahrnehmung der Umgebung und der Informationen des vestibularen Systems führen zu den Ausprägungen

der Motion oder Simulator Sickness.

2.8. Vorhersage von Kopfbewegungen

Bei der Benutzung von Trackerhardware in VR oder AR-Systemen erzeugt der end-to-end system delay Fehler in der Darstellung, wenn der Benutzer seinen Kopf bewegt. Die Verzögerung bewirkt, dass die virtuellen Objekte der Kopfbewegung zu folgen scheinen. Dieses Nachziehen ist abhängig von der Bewegungsgeschwindigkeit. Um eine Bewegungsvorhersage mit dem Kalman-Filter zu implementieren, ist es notwendig, die genaue Vorhersagezeit zu bestimmen, um den end-to-end system delay zu eleminieren oder zu verringern.

Der Zeitabstand vom Zeitpunkt t_0 zum Zeitpunkt t_5 ist abhängig von einer Vielzahl von Faktoren. Dazu zählen die Art des Trackers, die Art des Zugriff auf die Trackerinformationen, die Leistung des Grafiksubsystems und letztlich die Qualität der LC-Displays im HMD. Bewegt der Benutzer seinen Kopf also kontinuierlich, so bekommt er das Bild, dass mit den Orientierungsinformationen zum Zeitpunkt t_0 gerendert wurde, zum Zeitpunkt t_5 zu sehen.

$$\triangle t = t_5 - t_0 \tag{9}$$

Die Aufgabe der Bewegungsvorhersage ist es also, $\triangle t = t_5 - t_0$ zu eleminieren, um unerwünschte Effekte beim Arbeiten mit immersiven Systemen zu vermeiden. Da jeder Schritt (Tab. 1 Seite 12) seine Zeit benötigt, ist nicht davon auszugehen, dass die einfache Beschleunigung der Hardware das Problem löst.

Nachdem alle vermeidbaren Latenzen durch Optimierung der Programmierung oder Verbesserung der Hardware eleminiert wurden, wird die Bewegungsvorhersage dazu benutzt, die unvermeidbaren Latenzen zu kompensieren. Das Ziel ist es, die zukünftige Orientierung des Benutzers aus den vergangenen Bewegungen zu extrapolieren. Die Extrapolation soll die Orientierung des Benutzers zu dem Zeitpunkt, an dem der Benutzer die errechneten Bilder sieht, ergeben. Azuma [Azu95] vergleicht das Problem der Bewegungsvorhersage sehr treffend mit der Situation des Autofahrens, bei der man lediglich in den Rückspiegel schaut um zu steuern. Um das Auto auf der Straße zu halten, muss der Fahrer aus dem, was er im Rückspiegel sieht und aus dem Wissen über Straßen im Allgemeinen, vorhersagen, wohin die Straße führt. Die Schwierigkeit dieser Aufgabe ist abhängig von der Geschwindigkeit, mit der sich das Fahrzeug bewegt und der Form der Straße. Führt die Straße geradeaus, ist die Vorhersage einfach durchzuführen. Windet sich die Straße hin und her und beinhaltet zufällige Kurven, so ist die Vorhersage unmöglich.

Die Bewegungsvorhersage versucht zukünftige Messwerte aus vergangenen zu extrahie-

ren. Prinzipiell versuchen die meisten Methoden die lokalen Ableitungen der Messwerte abzuschätzen, um mittels *Taylor-Reihen* die zukünftigen Werte zu bestimmen. Die Methoden unterscheiden sich meist in der Art und Weise, wie stark die Daten geglättet werden, um die die Ableitungen zu schätzen.

Der einfachste Weg wäre es, eine Gerade durch zwei Messwerte zu ziehen und die Vorhersage zu bestimmen. Dieser Ansatz ist sehr empfindlich gegenüber Ungenauigkeiten in den Messungen. Ein besserer Weg ist es, die gewichteten Kombinationen von mehreren vergangenen Messungen als Grundlage der Schätzung zu verwenden. Diese Herangehensweise reduziert die Fehleranfälligkeit des Algorithmus´ gegenüber dem Messungenauigkeiten, erzeugt aber eine Verzögerung bei der Reaktion auf schnelle Richtungswechsel. Für alle Methoden zur Vorhersage, die auf vergangenen Messungen der Orientierung beruhen, gilt, dass eine Reduktion der Rauschempfindlichkeit immer zu Lasten der Reaktiongeschwindigkeit geht.

Die Genauigkeit der Bewegungsvorhersage der Kopforientierung lässt sich signifikant durch den Einsatz von Trägheitssensoren zur Messung der Ableitungen der Bewegung verbessern. Die Trägheitssensoren bestimmen die Geschwindigkeit und Beschleunigung der Bewegung mit hoher Genauigkeit und diese Werte sind der Differenzation der Position vorzuziehen, da sie weniger fehlerbehaftet und unverzögert[3] sind.

Azuma [Azu95] demonstrierte eine auf dem Kalman-Filter basierende Methode zur Bewegungsvorhersage unter Einsatz von Trägheitssensoren, die spürbar den Effekt des “swimming” in AR-Systemen verringerte. Diese Vorhersagemethode wird mittlerweile im kommerziellen *HiBall™-Tracker* von 3rdTech (`http://www.3rdtech.com/HiBall.htm`) eingesetzt. Leider stehen am Cybermind hi-Res900™3D keine Trägheitssensoren zur Verfügung, so dass diese vielversprechende Methode in diesem Fall keine Anwendung finden kann. Für die Vorhersage von Bewegungsdaten ohne Trägheitssensoren existiert eine Reihe von Methoden. Eine der bewährtesten und robustesten Algorithmen ist die Methode nach Liang [LSG91]. Der Algorithmus setzt vier Kalman-Filter ein, um die einzelnen Quaternionenanteile der Orientierung einzeln abzuschätzen. Dieser Algorithmus soll in dieser Arbeit zur Anwendung kommen und wird im Kapitel 3.2 im Detail erklärt.

[3] Die Beschleunigung und Geschwindigkeit lässt sich mittels Diffenzation erst mit dem jeweils nächsten Messwert errechnen.

3. Vorhersage der Kopforientierung

Um den end-to-end system delay zu kompensieren, wird ein Schätzverfahren zum Einsatz kommen, dass nach seinem Entwickler Rudolph Emil Kalman benannt ist. Der 1960 publizierte Kalman-Filter [Kal60] dient zur Schätzung und Vorhersage des akutellen oder zukünftigen Verhaltens eines Prozesses. Dieser Prozess muss als lineares, dynamisches System modellierbar sein. Der Kalman-Filter berücksichtigt die Historie des Systems und den Grad der Messungenauigkeit. Die Messungenauigkeiten werden als *Rauschen* bezeichnet. Bei jeder Rekursion wird der Zustand des Systems neu errechnet. Dazu werden eine Reihe von Matrizen ständig aktualisiert, die die Historie des Systems sowie die Varianzen und Kovarianzen der Systemvariablen beschreiben.

Unter den mathematischen Algorithmen zur stochastischen Approximation von fehlerbehafteten Messungen ist der Kalman-Filter wohl der am häufigsten verwendete und bekannteste. Der Kalman-Filter ist eine effiziente, rekursive Lösung der “Least-Squares”-Methode und wurde in unzähligen Abwandlungen in den unterschiedlichsten Bereichen eingesetzt. Besonders häufig wurde der *Kalman-Filter* für autonome oder rechnerassistierte Navigation von Schiffen, Flugzeugen, Raketen und sogar der Mondfähre Apollo 11 verwendet. Eine Renaissance erlebt der Kalman-Filter im Bereich der automatischen Bildverarbeitung und der *Virtual Reality*, aber beispielsweise auch der Robotik. Dort findet der Algorithmus vor allem Anwendung beim Tracking von Positions- oder Orientierungsdaten aller Art, wie z.B. zur exakten Positionsbestimmung via GPS.

Insbesondere seine rekursive Natur empfiehlt den Algorithmus für rechnergestütze Anwendungen, da anders als beispielsweise beim *Wiener Filter* [Wie49], der basierend auf der Menge aller Messwerte eine Schätzung vornimmt, die Schätzungen des *Kalman-Filters* auf einer Menge von Matrizen beruhen, die den jeweils aktuellen Zustand des Systems widerspiegeln. Diese ergeben, mit kontinuierlichen Messwerten kombiniert, den Wert der Schätzung. Neben der Möglichkeit den Algorithmus *on-line* benutzen zu können, zeigt der *Kalman-Filter* eine Anzahl von Eigenschaften, die ihn zu einem universal einsetzbaren Schätzverfahren machen. Zum einen ist der Kalman-Filter modellbasiert und lässt sich aus diesem Grund als Schätzverfahren für jede lineare Messreihe verwenden. Zum anderen minimiert das Verfahren den mittleren quadratischen Fehler und wird deshalb auch als *optimaler* Schätzer bezeichnet. Des weiteren gewichtet der *Kalman-Filter* den Einfluss der geschätzten Werte und der Messwerte, um die bestmögliche Vorhersage zu treffen.

Die Einführung in die Funktionsweise des Kalman-Filters beschränkt sich auf lineare, dynamische Systeme. Es existiert zwar auch eine Möglichkeit zur Schätzung von nicht-linearen Systemen - diese sogenannte *Extended Kalman-Filter (EKF)* wird jedoch nicht benutzt, da

eine Echtzeitimplementierung des EKF nicht ohne weiteres möglich ist. Die Bewegung des Kopfes bei einem HMD-Benutzer ist zwar nicht linear, dennoch kann eine Schätzung mittels eines linearen, *diskreten* Kalman-Filters vorgenommen werden, da, wie auch in diesem Fall, nicht-lineare Systeme häufig lokal, durch lineare Systeme approximiert (Kapitel 3.2) werden können.

3.1. Die Theorie des Kalman-Filters

Der Kalman-Filter gehört zur zur Familie der mathematischen Verfahren, die *Schätzer* genannt werden. Schätzer verarbeiten eine Folge von Messwerten und zusätzliche, über das System bekannte Infomationen. Das Verfahren versucht auf dieser Grundlage, Aussagen über den Systemzustand zu machen bzw. ihn zu approximieren.

Die Schätzverfahren werden in drei Typen unterteilt. Die einzelnen Typen unterscheiden sich in dem Zeitpunkt, über den eine Schätzung abgegeben wird. Die sogenannten *Glätter* erzeugen Aussagen über vergangene Zeitpunkte, *Filter* beschreiben den Zustand in der Gegenwart. Die dritte und für das Verfahren zur Kompensation des end-to-end system delay wichtige Gruppe bilden die *Vorhersager*. Sie treffen Aussagen über den zukünftigen Zustand eines Systems.

Um das Verfahren des Kalman-Filters einzuführen, wird zunächst ein eindimensionaler Filter konstruiert (Kapitel 3.1.1). Dieser eindimensionale Filter dient nur zur Einführung und hat kaum praktischen Nutzen, da in der Praxis äußerst selten nur ein einziger Wert geschätzt wird. Auch bei der Vorhersage von Kopfbewegungen nach der gewählten Methode nach Liang [LSG91] benötigt man einen Filter, der mehrdimensionale Zustände verarbeiten kann. Die Beschreibung des mehrdimensionalen Kalman-Filters folgt in Kapitel 3.1.2.

3.1.1. Konstruktion eines eindimensionalen Kalman-Filters

Zur Veranschaulichung wird ein einfaches Beispiel genutzt. Es soll die Entfernung eines Schiffes vom Ufer bestimmt werden. Zunächst wird eine erste Messung zum Zeitpunkt t_1 vorgenommen. Dieser Messwert z_1 wird jedoch nicht durch ein Messgerät festgestellt, sondern schlicht geraten. Wie bereits erwähnt, handelt es sich beim Kalman-Filter um einen rekursiven Algorithmus. Aus diesem Grund ist ein geratenes z_1 in soweit, als das es die Initialisierung der Rekursion darstellt. Da der erste Messwert nur geraten ist, kann man sich seiner Richtigkeit nicht allzu sicher sein. Als sicher kann jedoch angenommen werden, dass sich die tatsächliche Position des Schiffes in einem bestimmten Bereich um z_1 bewegt. Dar-

aus läßt sich wie in Abb. 20 gezeigt eine Wahrscheinlichkeitskurve erzeugen.

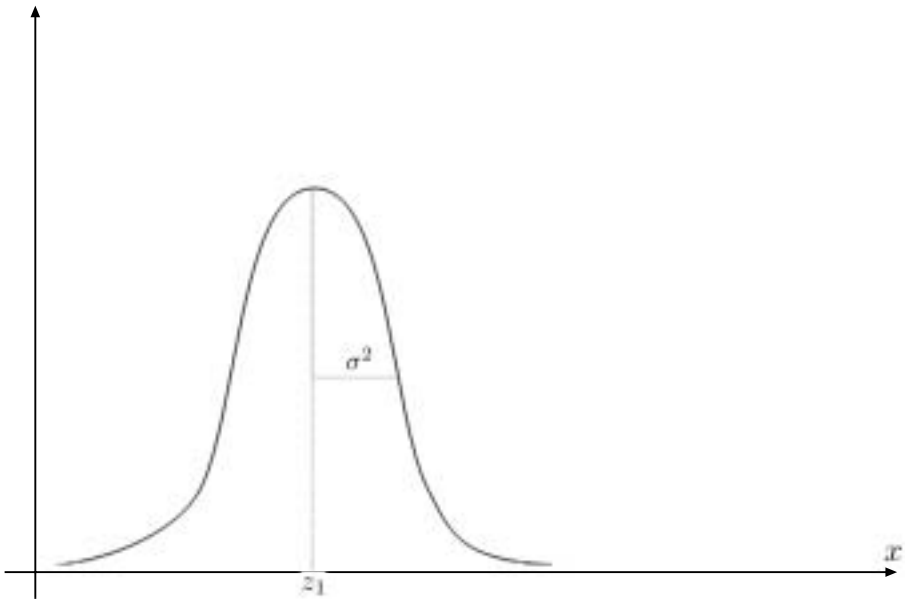

Abbildung 20: Wahrscheinlichkeit für die Richtigkeit von z_1 mit der Varianz σ^2

Die Varianz σ^2 ($= Var(z_1)$) gibt an, wie sehr man dem Messwert vertraut. Mann kann auch sagen, dass er als Maß für die Qualität des Messwertes dient. Je kleiner die Varianz, also je schmaler die Kurve, desto besser ist der Messwert. Ein absolut exakter Wert hätte demzufolge eine Varianz $\sigma^2 = 0$. In diesem Beispiel enspricht z_1 dem Erwartungswert und $\sigma_{z_1}^2$ der Varianz einer Zufallsvariablen. Damit das Verfahren des Kalman-Filters funktioniert, sollte diese Zufallsvariable normalverteilt sein. Der Kalman-Filter funktioniert zwar auch bei Verletzung dieser Normalverteilung, jedoch werden die Ergebnisse dadurch verfälscht. Nach dem zentralen Grenzwertsatz der Wahrscheinlichkeitsrechnung ist glücklicherweise die Summe von einzelnen Zufallsvariablen durch eine einzelne Gauß-normalverteilte Zufallsvariable approximierbar.

Bis zu diesem Zeitpunkt ist die beste Schätzung für die Position des Schiffes:

$$\hat{x}_{t_1|t_1} = z_1 \tag{10}$$

Dabei bezeichnet $\sigma_{t_1|t_1}^2 = \sigma_{z1}^2$ die zum Schätzwert zum Zeitpunkt t_1 gehörende Varianz.

Um zunächst den Kalman-Filter als Filter und später als Vorhersager benutzen zu können, wird eine Notation wie in Gleichung 10 benutzt. Dabei bezeichnet der Wert links vom $|$ den Zeitpunkt, über den eine Aussage gemacht wird und der Wert rechts gibt an, bis zu welchem

Zeitpunkt die Informationen für die Schätzung benutzt werden. $\hat{x}_{a|b}$ ist also ein Schätzwert, der eine Aussage über den Zeitpunkt a enthält und auf Informationen bis zum Zeitpunkt b basiert. Bei einem Vorhersager ist also $a > b$, während sich eine Filter durch $a = b$ auszeichnet.

Um den Algorithmus fortzuführen, wird eine zweite Messung z_2 zum Zeitpunkt t_2 erzeugt. Zunächst wird angenommen, dass $t_1 = t_2$ gilt. Der Messwert z_2 wird im Gegensatz zu z_1 nicht geraten, sondern beispielsweise durch einen Laser-Entfernungsmesser bestimmt. Da davon auszugehen ist, dass die Messung per Laser deutlich zuverlässiger ist als der geratene Wert z_1, ist auch die entsprechende Varianz $\sigma^2_{z_2}$ geringer als σ^2_{z1} - wie in Abb. 21 gezeigt wird. Aus der Messung resultiert nun eine zweite Verteilungskurve:

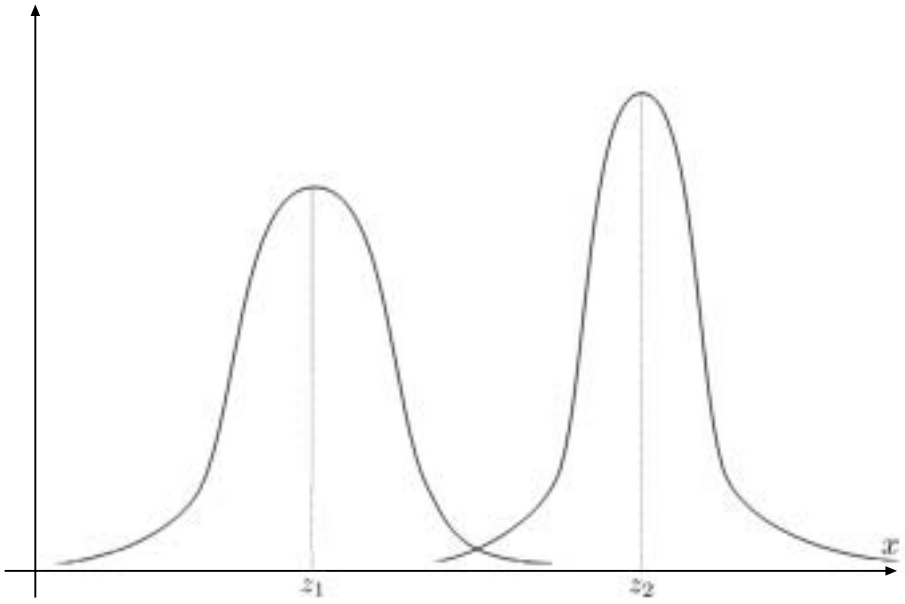

Abbildung 21: Wahrscheinlichkeit für die Richtigkeit von z_1 und z_2

Die Problematik des Schätzers ist es nun, die beiden Messwerte zu einer möglichst optimalen Schätzung für den Zeitpunkt t_2 zu kombinieren. Der Kalman-Filter nutzt dazu die gewichtete Linearkombination beider Messwerte.

$$\hat{x}_{t_2|t_2} = k_1 z_1 + k_2 z_2 \tag{11}$$

Um zu erreichen, dass der Schätzwert $\hat{x}_{t_2|t_2}$ zwischen z_1 und z_2 liegt, wird als Nebenbedingung definiert, dass $k_1 + k_2 = 1(k_1, k_2 \geq 0)$. Das Ziel ist es nun, k_1 und k_2 so zu wählen, dass $\hat{x}_{t_2|t_2}$ optimal ist. Das Maß der Optimalität wird dabei definiert als:

$$\tilde{x} = \hat{x}_{t_2|t_2} - x \tag{12}$$

Das x steht für die wirkliche Position des Schiffes, $\hat{x}_{t_2|t_2}$ bekanntermaßen für den Schätzwert, woraus resultiert, dass $\tilde{x}$ den Fehler der Schätzung beschreibt. k_1 und k_2 sollen also $|\tilde{x}|$ minimieren. Der Kalman-Filter minimiert jedoch nicht $|\tilde{x}|$ direkt, sondern $\tilde{x}^2$, da es sich (stetig) differenzieren lässt. Die wirkliche Position x ist unbekannt und stellt hier nur einen Platzhalter zum besseren Verständnis dar. Diese Variable verschwindet später aus den Formeln.

Das entstandene Maß für Optimalität $\tilde{x}$ ist eine Zufallsvariable, deren Erwartungswert $E(\tilde{x}^2)$ nun minimiert wird. In der Literatur [Kre02] findet sich die folgende Formel um die Varianz $\sigma_{\tilde{x}}^2$ zu beschreiben:

$$\sigma_{\tilde{x}}^2 = E(\tilde{x}^2) - E(\tilde{x})^2 \tag{13}$$

Zusammen mit der zurvor formulierten Nebenbedingung $k_1 + k_2 = 1$ lässt sich zeigen, dass $E(\tilde{x}) = 0$ ist.

$$\begin{aligned} E(\tilde{x}) &= E(\hat{x}_{t_2|t_2} - x) \\ &= E(\hat{x}_{t_2|t_2}) - E(x) \\ &= E(k_1 z_1 + k_2 z_2) - x \\ &= E(k_1 z_1) + E(k_2 z_2) - x \\ &= k_1 E(z_1) + k_2 E(z_2) - x \end{aligned}$$

Um mit der Minimierung des Fehler fortfahren zu können, werden die Messwerte z_1 und z_2 dargestellt als:

$$z_1 = x + r_1$$

$$z_2 = x + r_2$$

Hierbei bezeichnet x immernoch den exakten, aber unbekannten Abstand des Schiffes und r_1 und r_2 repräsentieren das Rauschen der Messungen. Dabei sind r_1 und r_2 Zufallsvariablen mit dem Erwartungswert null.

Durch diese Definition ist es möglich, die obige Rechnung fortzuführen:

$$\begin{aligned} E(\tilde{x}) &= k_1 E(z_1) + k_2 E(z_2) - x \\ &= k_1 E(x + r_1) + k_2 E(x + r_2) - x \\ &= k_1 (E(x) + E(r_1)) + k_2 (E(x) + E(r_2)) - x \\ &= k_1 x + k_2 x - x \\ &= x(k_1 + k_2 - 1) \\ &= 0 \end{aligned}$$

Zusammen mit Gleichung 13 und dem gerade gezeigten $E(\tilde{x}) = 0$ gilt für den Erwartungswert des Quadrates des Schätzfehlers $E(\tilde{x}^2)$:

$$\begin{aligned} E(\tilde{x}^2) &= \sigma_{\tilde{x}^2} = Var(\tilde{x}) \\ &= Var(\hat{x}_{t_2|t_2} - x) \\ &= Var(\hat{x}_{t_2|t_2}) - \underbrace{Var(x)}_{ist\ 0,\ da\ x\ die\ extakte\ Position\ darstellt} \\ &= Var(k_1 z_1 + k_2 z_2) \\ &= Var(k_1 z_1) + Var(k_2 z_2) \\ &= k_1^2 Var(z_1) + k_2^2 Var(z_2) \\ &= k_1^2 \sigma_{z_1}^2 + k_2^2 \sigma_{z_2}^2 \\ &= \underbrace{k_1^2 \sigma_{z_1}^2 + (1 - k_1)^2 \sigma_{z_1}^2}_{unter\ Zuhilfenahme\ von\ k_1 + k_2 = 1} \end{aligned}$$

Diesen Term gilt es nun zu minimieren. Er entspricht der Varianz des Schätzfehlers ebenso wie der Varianz des Schätzers selbst. Durch eine Ableitung nach k_1 und dem Gleichsetzen des Ergebnisses mit null entstehen die Werte k_1 und k_2:

$$k_1 = \frac{\sigma_{z_2}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2} \qquad k_2 = \frac{\sigma_{z_1}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2}$$

Mit dieser Erkenntnis lässt sich nun Gleichung 11 für t_2 wie folgt definieren:

$$\hat{x}_{t_2|t_2} = \frac{\sigma_{z_2}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2} z_1 + \frac{\sigma_{z_1}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2} z_2 \tag{14}$$

Und die Varianz $\sigma_{\hat{x}_{t_2|t_2}}^2$ ergibt sich als:

$$\sigma_{\hat{x}_{t_2|t_2}}^2 = \left(\frac{1}{\sigma_{z_1}^2} + \frac{1}{\sigma_{z_2}^2} \right)^{-1}$$

Hier wird die Möglichkeit zur Gewichtung von Messwerten beim Kalman-Filter deutlich. Setzt man z.B. $\sigma_{z_2}^2 = 0$ so bedeutet es, dass dieser Wert absolut sicher ist. Daraus folgt ebenfalls, dass $\sigma_{z_1}^2 = 0$, da ja die Nebenbedingung $k_1 + k_2 = 1$ gilt. Setzt man diese Werte in die

Gleichung 14 ein, so wie in Gleichung 15 gezeigt, ist es offensichtlich, dass das Ergebnis des Schätzers z_2 ist.

$$\hat{x}_{t_2|t_2} = \frac{0}{1+0} z_1 + \frac{1}{1+0} z_2 \tag{15}$$

Im umgekehrten Fall wäre das Ergebnis analog z_1, und setzt man die Varianzen gleich, so erzeugt der Schätzer den Mittelwert zwischen den beiden Messungen.

Für die weiteren Schritte wird Gleichung 14 erneut umgeformt:

$$\hat{x}_{t_2|t_2} = z_1 + \frac{\sigma_{z_1}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2}[z_2 - z_1] \tag{16}$$

Da die erste Schätzung einfach übernommen wurde, d.h. $\hat{x}_{t_1|t_1} = z_1$ (Gleichung 10), ergibt sich nun:

$$\hat{x}_{t_2|t_2} = \hat{x}_{t_1|t_1} + \frac{\sigma_{z_1}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2}[z_2 - \hat{x}_{t_1|t_1}] \tag{17}$$

Die Gewichtung wird nun als $K_{t_2} = \frac{\sigma_{z_1}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2}$ definiert. Dadurch entsteht:

$$\hat{x}_{t_2|t_2} = \hat{x}_{t_1|t_1} + K_{t_2}[z_2 - \hat{x}_{t_1|t_1}] \tag{18}$$

$$\sigma_{\hat{x}_{t_2|t_2}}^2 = \sigma_{\hat{x}_{t_1|t_1}}^2 - K_{t_2}\sigma_{\hat{x}_{t_1|t_1}}^2 \tag{19}$$

An den Gleichungen 18 und 19 kann man die rekursive Arbeitsweise des Kalman-Filters erkennen. Die Substitution der Gewichtung durch eine neue Variable K_{t_2} hat nicht nur optische Gründe. K_{t_2} repräsentiert einen Korrekturfaktor, der das Gewicht bestimmt, mit dem die Differenz zwischen altem und neuem Schätzwert in die aktuelle Schätzung eingeht. Dieser Faktor wird *Kalman-Gain* genannt. Der Kalman-Gain ist in der Regel größer oder gleich null, woraus sich ableiten lässt (Gleichung 19), dass die Varianz des Schätzers sinkt. Dieses Absinken der Varianz oder anders gesagt die Zunahme an Zuverlässigkeit, kann man durch die steigende Anzahl von Messwerten erklären. So ist eine Schätzung, die auf zwei Messwerten beruht, in der Regel genauer, als eine Schätzung die nur einen Wert als Grundlage hat (Abb. 22).

Bis zu diesem Zeitpunkt fungiert der Kalman-Filter nur als Filter. Um ihn zur Kompensati-

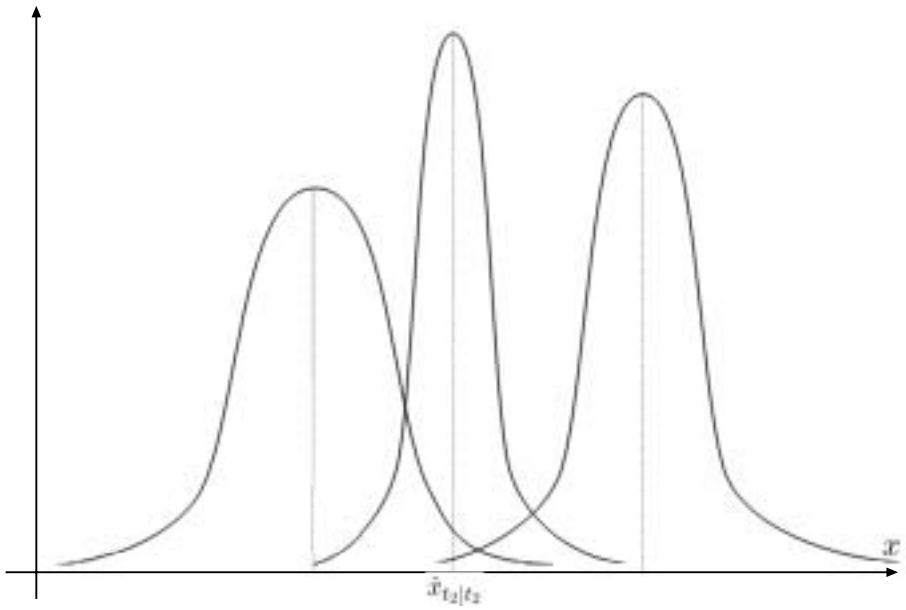

Abbildung 22: Wahrscheinlichkeit für die Richtigkeit von z_1, z_2 und z_3

on des end-to-end system delay zu verwenden, wird jedoch ein Vorhersager benötigt. Aus diesem Grund wird das Beispiel um Dynamik ergänzt. Das Schiff soll sich nun mit einer unbekannten Geschwindigkeit v bewegen. Die Geschwindigkeit wird dabei aus der gemessenen Geschwindigkeit u und dem korresspondierenden Messfehler w zusammengesetzt. Der Messfehler ist eine Zufallsvariable mit dem Erwartungswert null und einer Varianz σ_w^2.

$$v = u + w \tag{20}$$

Unter diesen Annahmen kann der Algorithmus nun als Vorhersager arbeiten. Ein Vorhersage für den nächsten Zeitschritt t_3 sähe dann so aus:

$$\hat{x}_{t_3|t_2} = \hat{x}_{t_2|t_2} + u[t_3 - t_2] \tag{21}$$

Die Varianz der Vorhersage wird ebenfalls errechnet durch:

$$\sigma^2_{\hat{x}_{t_3|t_2}} = \sigma^2_{\hat{x}_{t_2|t_2}} + \sigma_w^2[t_3 - t_2]^2 \tag{22}$$

Nach der Vorhersage wird für den Zeitpunkt t_3 eine Messung durchgeführt. Diese Messung wird als z_3 und ihre Varianz als $\sigma^2_{z_3}$ bezeichnet. Die Verknüpfung der Werte zum Schätzwert wird genauso ausgeführt wie zuvor für z_1 und z_2 (Gleichung 23). Ebenso ergibt sich die Varianz (Gleichung 24).

$$\hat{x}_{t_3|t_3} = \hat{x}_{t_3|t_2} + K_{t_3}[z_3 - \hat{x}_{t_3|t_2}] \tag{23}$$

$$\sigma^2_{\hat{x}_{t_3|t_3}} = \sigma^2_{\hat{x}_{t_3|t_2}} - K_{t_3}\sigma^2_{\hat{x}_{t_3|t_2}} \tag{24}$$

Der Kalman-Gain K_{t_3} wird dazu errechnet durch:

$$K_{t_3} = \frac{\sigma^2_{\hat{x}_{t_3|t_2}}}{\sigma^2_{\hat{x}_{t_3|t_2}} + \sigma^2_{z_{t_3}}} \tag{25}$$

Diese Formel schließt die Herleitung des eindimensionalen Kalman-Filters ab. Der Kalman-Filter besteht im wesentlichen aus den Gleichungen 21, 22, 23 und 24.

Die Anwendung des Algorithmus gestaltet sich recht einfach, worin unter anderem auch die Beliebtheit des Kalman-Filter begründet liegt. Für jeden neuen Messwert wird zunächst eine Vorhersage getroffen (Gleichungen 21, 22). Diese Vorhersage wird dann in den Gleichungen 23, 24 mit dem jeweils neuen Messwert verknüpft und ergibt die korrigierte Schätzung. Dieser Algorithmus bei der Verarbeitung von Messwerten wird als *Kalman-Filter* bezeichnet.

Die Operationen zur Vorhersage eines Wertes mittels der bis dahin bekannten Informationen wird *Time Update* genannt. Der zweite Teilalgorithmus heißt *Measurement Update*.

Im *Time Update* wird der Schätzwert (Gleichung 21) und seine Varianz (Gleichung 22) anhand der Informationen vorhergesagt, die durch die Abfolge der vergangenen Messwerte zur Verfügung stehen. Dieser Schätzwert wird als *a priori*-Schätzwerte bezeichnet.

Beim *Measurement Update* wird der zuletzt gemessene Wert in die Berechnung mit einbezogen. Dadurch werden Schätzwert und Varianz korrigiert. Sie werden als *a posteriori* Schätzwerte bezeichnet. Zusätzlich findet im Measurement Update die Errechnung des Kalman-Gain statt, der die Gewichtung zwischen Vorhersage- und Messwert bei der Bestimmung des a posteriori-Schätzwertes darstellt.

Von den englischen Bezeichnungen für Vorhersage (*Prediction*) und Korrektur (*Correction* wurde für ein solches Verfahren die Bezeichung *predictor-corrector*-Kreislauf abgeleitet (Abb. 23).

Es wird nun nur noch die Variante des Vorhersagers betrachtet und die zur Einführung be-

nutzte Notation der Form $\hat{x}_{a|b}$ ist nicht mehr notwendig. Dies geschieht auch, um eine Konsistenz in der Notation mit nahezu allen Publikationen über den Kalman-Filter zu erreichen [WB02] [Azu95]. Allgemein formuliert, stellt sich der Kalman-Filter für eine Dimension in veränderter Notation nun so dar (vgl. Gleichungen 21, 22, 23 und 24):

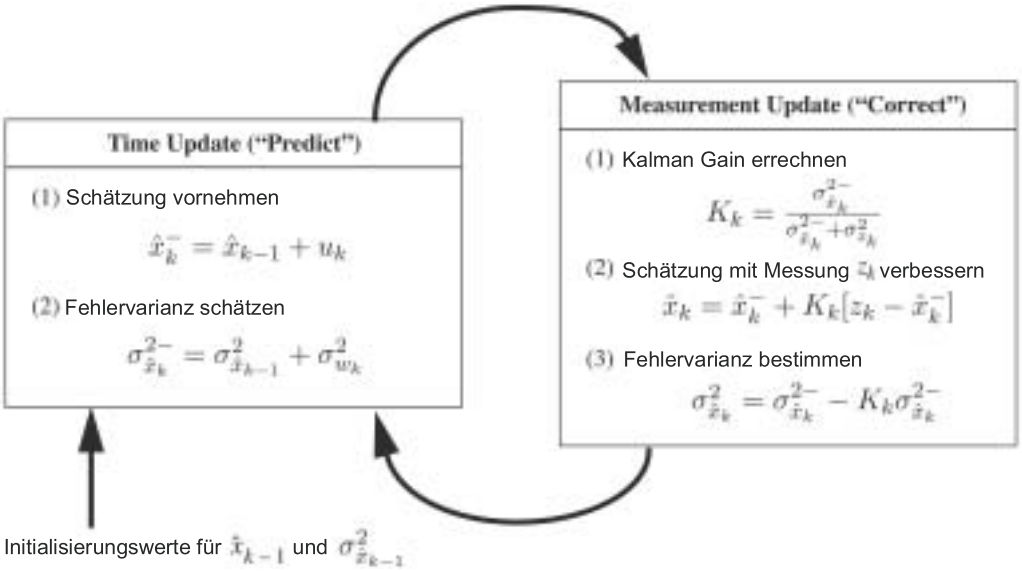

Abbildung 23: Übersichtsbild der Operationen des eindimensionalen Kalman-Filters

Die Notationsänderung bezieht sich nur auf die Änderung der Indizes und die Markierung der a priori-Werte. Nun beschreibt k den aktuellen Schritt, und das *super minus* (z.B. bei $\hat{x}_k^-$) zeigt an, dass es sich um einen a priori-Schätzwert handelt.

3.1.2. Der Multidimensionale Kalman-Filter

Bisher stellt sich der Kalman-Filter nur als einfaches, rekursives Verfahren dar, dass fähig ist, Schätzungen über den Zustand eines Objektes zu liefern. Für die Vorhersage von Kopfbewegungen benötigt man jedoch etwas mehr. Wie in fast allen praxisorientieren Anwendungen des Kalman-Filters und auch in den meisten Einführungen [Kal60] [AWB01] [WB02] wird der Algorithmus mehrdimensional benutzt.

Prinzipell ersetzt man die skalaren Messwerte durch n-dimensionale Vektoren. Aus den Varianzen werden Kovarianzmatrizen und die Bestimmung des Kalman-Gain muss ersetzt werden. Ansonsten funktioniert der mehrdimensionale Kalman-Filter analog zum Eindimensionalen.

Generell kann man nun sagen, dass der Kalman-Filter ein Algorithmus zur Abschätzung des Zustandes $x \in R^n$ eines diskret-zeitabhängigen Systems ist.

Das System bezeichnet dabei im Allgemeinen ein pysikalisches Objekt, dessen Verhalten sich durch Gleichungen beschreiben lässt. Der Zustand des Systems kann z.B. die tatsächliche Entfernung eines Schiffes vom Ufer oder der Winkel einer Kopfbewegung sein. Das lineare, dynamische System muss dabei folgende Eigenschaften erfüllen:

- es hat einen internen, *unbekannten* Zustand
- der Zustand des Systems ist *endlichdimensional*
- Zustandsänderungen und Messungen erfolgen in gewissen Zeitabständen, d.h. in *diskreter* Zeit
- Zustandsänderungen sind abhängig vom aktuellem Zustand des Systems, einer bekannten Eingabe und dem Rauschen
- die Messung kann den Zustand *direkt* oder *indirekt* sowie *vollständig* oder *teilweise* bestimmen
- die Meßwerte sind verrauscht

Das Diagramm (Abb. 24) zeigt ein solches, auf die relevanten Komponenten reduziertes System.

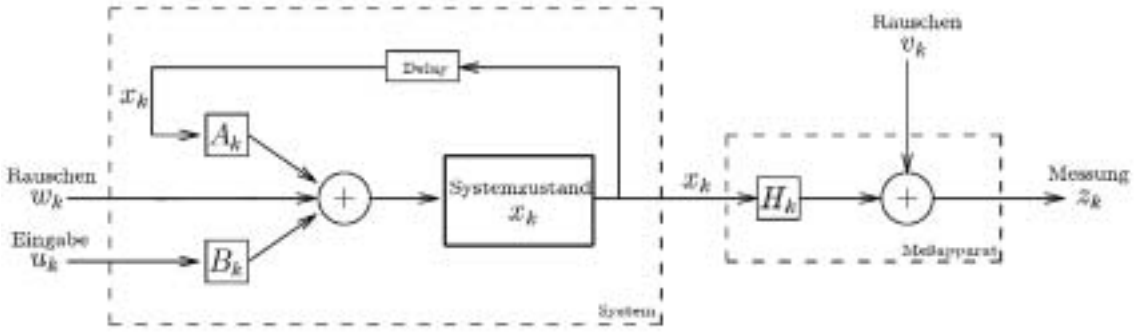

Abbildung 24: Übersichtsbild eines linearen Systems

Der Prozess lässt sich dabei durch eine Gleichung, die *Systemgleichung*, der Form

$$x_k = Ax_{k-1} + Bu_k + w_{k-1} \tag{26}$$

mit einer Messung $z \in R^m$

$$z_k = Hx_k + v_k \tag{27}$$

modellieren.
Im Gegensatz zum eindimensionalen Kalman-Filter (Kapitel 3.1.1) bezeichnen u_k, v_k, x_k, w_k, z_k nun Vektoren und A, B und H Matrizen.
Man bezeichnet die $n \times n$ A als *Zustandsübergangsmatrix* und die $n \times m$ Matrix H als *Messmatrix*. Die Bezeichnungen Mess- und Schätzwert beziehen sich nun auf Vektoren.
Die Zustandsübergangsmatrix A überführt den Zustand des System zum Zeitpunkt $k-1$ in den Zustand zum Zeitpunkt k ohne Beachtung des Prozessrauschens w_k oder des Messrauschens v_k. Die $n \times l$ Matrix B fügt die Eingabe $u \in R^l$ in den Zustand ein. In der Messgleichung 27 gewichtet die Matrix H den vorhergesagten Wert x_k, der zusammen mit dem Messrauschen v_k, den Messwert z_k ergibt.

Dabei sind w_k und v_k das Rauschen im Prozess bzw. bei dessen Messung. Dieses Rauschen folgt immer einer Normalverteilung.

$$p(w) \sim N(0, Q) \tag{28}$$

$$p(v) \sim N(0, R) \tag{29}$$

Genauso wie der eindimensionale Kalman-Filter lässt sich der Algorithmus als zwei Teile darstellen (Abb. 25).

Das *Time Update* ist dafür verantwortlich, den a priori-Schätzwert $\hat{x}^-$ (Gleichung 30) und die Fehlerkovarianzmatrix P_k^- (Gleichung 31) zu errechnen.

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k \tag{30}$$

$$P_k{}^- = AP_{k-1}A^T + Q \tag{31}$$

Beim *Measurement Update* wird der letzte Messwert in die Berechnung einbezogen und der korrigierte *a posteriori*-Schätzwert sowie die Fehlerkovarianzmatrix P_k bestimmt.

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{32}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{33}$$

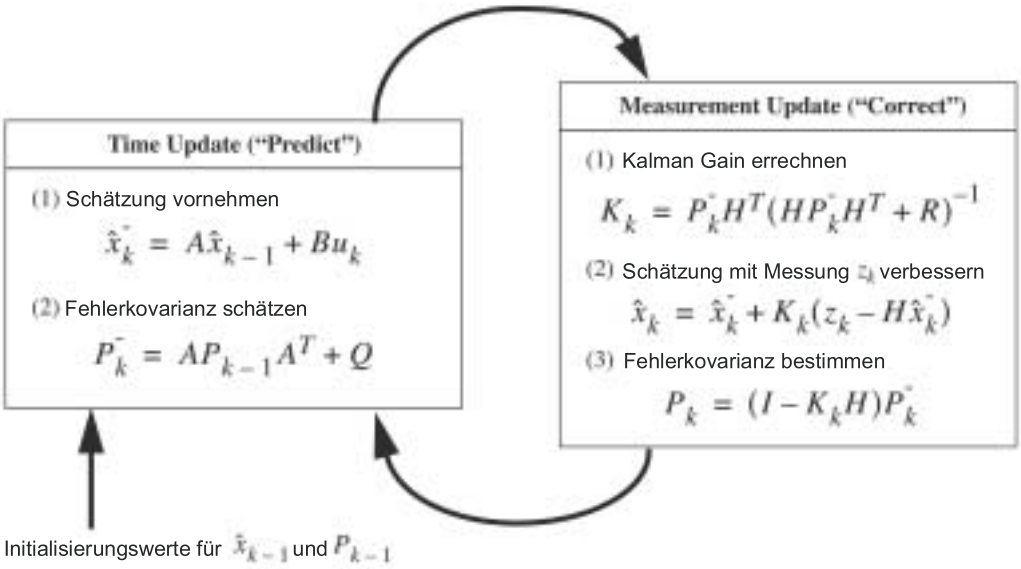

Abbildung 25: Übersichtsbild der Operationen des diskreten Kalman-Filters

$$P_k = (I - K_k H)P_k^- \tag{34}$$

Der *a posteriori*-Schätzwert ergibt sich aus der Linearkombination aus dem *a priori*-Schätzwert und der durch den *Kalman Gain* K gewichteten Differenz von aktuellem Messwert z_k und der Messvorhersage. Man bezeichnet die Diffenenz zwischen aktuellem Messwert und der Messvorhersage $(z_k - H\hat{x}_k^-)$ (aus Gleichung 33) als *Residual*. Ein *Residual* von null bedeutet, dass die Terme übereinstimmen.

Die Lösung des Schätzproblems ist somit das Minimum der *a posteriori*-Fehlerkovarianzmatrix P. Der *Kalman-Gain*, also die $n \times m$ Matrix K aus Gleichung 33 soll den Schätzfehler minimieren. Die Minimierung wurde beim eindimensionalen Kalman-Filter noch hergeleitet. Beim mehrdimensionalen stellt sich diese Herleitung ungleich schwieriger dar. Deshalb wird den Publikationen [WB02] [Azu95] [AWB01] vertraut, aus denen hervorgeht, dass die schon gezeigte Gleichung 32 den Fehler eines mehrdimensionalen Kalman-Filters minimiert.

3.2. Methode nach Liang

Die Methode zur Voraussage der Kopforientierung nach Liang [LSG91] stellt einen recht einfachen Ansatz dar, der trotzdem eine gute Vorhersage ermöglicht. Der Algorithmus benötigt keine zusätzlichen Messungen über Geschwindigkeits- oder Beschleunigungssensoren, wie etwa [Azu95], arbeitet also ausschließlich mit den Orientierungsdaten des Trackers. Der Algorithmus ist eine Anwendung des diskreten Kalman-Filters.

Das Problem der Bewegungsvorhersage ist leider kein triviales, denn die Bewegungen sind zufällig und die Messungen des Trackers können zusätzlich Ungenauigkeiten enthalten. Glücklicherweise ist der Algorithmus des Kalman-Filters dennoch für dieses Problem benutzbar, denn er bringt sowohl die zufällige Natur des Prozesses der Bewegung, als auch das Rauschen der Messdaten in Einklang. Um den Kalman-Filter für Vorhersage der Kopfbewegung nutzen zu können, müssen zwei Probleme gelöst werden:

1. Die Linearisierung der Messdaten
2. Die korrekte Wahl des Prozessmodells für den Kalman-Filter

Die Linearisierung ist notwendig, da der Kalman-Filter nur lineare Prozesse abschätzen kann und Orientierungsdaten des Trackers in Form von Quaternionen übermittelt werden. Die Quaternionendarstellung birgt zwar weniger Probleme als die Darstellung mit Euler-Winkeln (siehe Kapitel 2.5.1) und vermeidet Komplikationen wie den *gimbal lock*. Leider beschreibt aber ein Einheitsquaternion einen Punkt auf einer vierdimensionalen Kugel und stellt somit keine lineare Repräsentation dar, wie sie für das Schätzverfahren benötigt wird. Um ein Einheitsquaternion der Form $q = (\cos\frac{\theta}{2}, \sin\frac{\theta}{2}x, \sin\frac{\theta}{2}y, \sin\frac{\theta}{2}z)$ zu linearisieren, wird der Einheitsvektor $n = (x, y, z)$ gebildet, indem man θ extrahiert. n stellt nun nur noch die Achse der Rotation dar, da ja der Betrag der Rotation entfernt wurde. Der Annahme folgend, dass die Änderung der Orientierung in einer Samplingperiode nur sehr klein ist, kann die Beschränkung der Einheitslänge gelockert und die Elemente x, y und z unabhängig voneinander gefiltert werden. Die gefilterten x-, y- und z-Werte werden normalisiert und bilden den Einheitsvektor n'. Dieser Einheitsvektor wird mit dem ebenfalls gefilterten θ kombiniert und bildet so das resultierende Quaternion.

Das Prozessmodell des Kalman-Filters bildet sozusagen das Herz des Verfahrens. Die Kopfbewegungen des Benutzers lassen sich durch eine Abfolge von kurzen Phasen schneller Bewegungen und relativ langen Phasen der Bewegungslosigkeit charakterisieren. Aus

dieser Charakteristik lassen sich die Schlüsse ziehen, dass zum einen die Änderung der Kopforientierung infrequent ist und zum anderen die Winkelgeschwindigkeit nur während der nichtperiodischen Änderungen der Orientierung $\neq 0$ ist. Man kann sich also die Kopfbewegung des Benutzers als eine zufällige Variable, die sich in zufälligen Zeitabständen ändert, vorstellen. Die Änderungen können dabei einen bestimmten Höchstwert nicht überschreiten. Liang [LSG91] entwickelte aus diesen Annahmen folgende Systemgleichung:

$$x'' = -\beta x' + \sqrt{2\sigma^2\beta}\omega(t) \tag{35}$$

x' und x'' bezeichnen die erste und zweite Ableitung der Filtervariable. $\omega(t)$ ist die Sequenz des weißen Messrauschens mit der Varianz. Der Zeitfaktor β beschreibt das Maximum an Geschwindigkeit und Beschleunigung der Kopfbewegungen. Durch Anpassung von β und des Varianzfaktors σ^2 lassen sich die Glätte und das Überschwingen der Vorhersage steuern. Je größer β gewählt wird, desto schneller keht der Vorhersagewert der Geschwindigkeit zu null zurück. Diese Erhöhung geht jedoch zu Lasten des Überschwingens der Vorhersage.

Die zweite Zustandsgleichung setzt den gemessenen Wert y und den wirklichen Wert x in folgende Beziehung:

$$y = x + \gamma\vartheta(t) \tag{36}$$

Dabei stellt $\vartheta(t)$ das Rauschen der Messdaten dar, dass ebenso wie $\omega(t)$ normalverteilt ist und eine Varianz von eins hat (weißes Rauschen). $\vartheta(t)$ und $\omega(t)$ sind unkorreliert. Weiterhin stellt γ in Gleichung 36 die Größenordnung des Rauschens dar.

Die von Liang [LSG91] entwickelten Gleichungen 35 und 36 lassen sich nun als Zustandsgleichungen im Kalman-Filter einsetzen. Der Algorthmus teilt sich in zwei Teile.

Im ersten werden die Matrizen des Kalman-Filters durch die Charakteristika der Liang-Gleichungen initialisiert. Dabei spielen die Parameter β, σ^2, γ, δ und τ entscheidende Rollen, da sie sowohl die Funktion beschreiben, als auch für das Einstellen des Filters verwendet werden. Die einzelnen Variablen werden auf einen optimalen Wert eingestellt, der entweder aufgrund aufgenommener Bewegungsprofile (siehe Kapital 4.2) oder durch Berechnungen ermittelt wird. Dieses *Tuning* ist ein wichtiger Aspekt für die korrekte Funktionsweise des Algorithmus´ und erfordert Vorbetrachtungen wie die Messung des end-to-end system delay und den Einsatz verschiedener Analysen der Bewegungsdaten. Dieses Tuning wird aufgrund seines Umfangs im Kapitel 6 gesondert betrachtet.

Im Initialisierungsteil des Kalman-Filters wird zunächst der Zustandsvektor x mit dem ersten

verfügbaren Messwert z_0 initialisiert:

$$x = \begin{pmatrix} z_0 \\ 0 \end{pmatrix} \tag{37}$$

Initialisierung der Messmatrix:

$$H = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{\delta} \end{pmatrix} \tag{38}$$

Die Fehlerkovarianzmatrix wird zunächst auf die Einheitsmatrix gesetzt. P wird in jeder Iteration des Kalman-Filters neu errechnet und erhält hier nur einen Initialisierungswert, um einen Kalman-Gain im ersten Schritt errechnen zu können.

$$P = I \tag{39}$$

Initialisierung der Fehlerkovarianzmatrix R der Messung:

$$R = \gamma^2 H^T H \tag{40}$$

Initialisierung der Fehlerkovarianzmatrix Q des Systems:

$$Q = \begin{pmatrix} \frac{2\sigma^2}{\beta}[\delta - \frac{2}{\beta}(1 - e^{-\beta\delta}) + \frac{1}{2\beta}(1 - e^{-2\beta\delta})] & 2\sigma^2[\frac{1}{\beta}(1 - e^{-\beta\delta}) - \frac{1}{2\beta}(1 - e^{-2\beta\delta})] \\ 2\sigma^2[\frac{1}{\beta}(1 - e^{-\beta\delta}) - \frac{1}{2\beta}(1 - e^{-2\beta\delta})] & \sigma^2(1 - e^{-\beta\delta}) \end{pmatrix} \tag{41}$$

Initialisierung der Zustandsübergangsmatrix Φ. In [LSG91] wird die Φ statt A als Bezeichnung für die Zustandsübergangsmatrix benutzt. Diese Notation wird beiberhalten, obwohl sie in Anbetracht aller anderen Publikationen über die Kopfbewegungsvorhersage mittels Kalman-Filter ungewöhnlich ist.

$$\Phi = \begin{pmatrix} 1 & \frac{1}{\beta}(1 - e^{-\beta\delta}) \\ 0 & e^{-\beta\delta}) \end{pmatrix} \tag{42}$$

Die Matrix Ψ dient zur Berücksichtigung des Rauschen bei der Vorhersage (siehe Gleichung 48):

$$\Psi = \begin{pmatrix} 1 & \frac{1}{\beta}(1 - e^{-\beta\gamma}) \\ 0 & e^{-\beta\gamma}) \end{pmatrix} \tag{43}$$

Damit sind alle Initialisierungen für den Vorhersagemechanismus abgeschlossen. Beim Algorithmus nach Liang [LSG91] bleiben die Matrizen während des gesamten Prozesses der Vorhersage konstant. Nur x, P und die Matrix des Kalman-Gain K verändern sich im Laufe

der Zeit. Der Initialisierungsteil ist somit für alle zu filternden Quaternionenanteile gleich, was letztlich auch die Implementierung vereinfacht.

Die eigentliche Vorhersage läuft gemäß des Verfahrens des Kalman-Filters im Wechsel zwischen dem Time- und Measurement-Update.

Zunächst wird der Kalman-Gain errechnet, der die Gewichtung von Messung und Schätzung darstellt, um den Fehler zu minimieren:

$$K = PH^T(HPH^T + R)^{-1} \tag{44}$$

$$z = \begin{pmatrix} z_i - z_{i-1} \\ z_i \end{pmatrix} \tag{45}$$

$$x = x + K(z - Hx) \tag{46}$$

Die Fehlerkovarianzmatrix P wird angepasst:

$$P = \Phi(I - KH)P\Phi^T + Q \tag{47}$$

Errechnung des Vorhersagewertes:

$$\begin{pmatrix} y_i \\ y_i' \end{pmatrix} = \Psi x \tag{48}$$

Der Zustand x des Systems wird mittels der Zustandsübergangsmatrix Φ in den nächsten Schritt überführt:

$$x = \Phi x \tag{49}$$

Die Gleichungen 44 bis 49 stellen den Predictor-Corrector-Kreislauf des Kalman-Filters dar. Trotz der etwas veränderten Notation sind die Gleichungen leicht als das Time- und Measurement-Update des Kalman-Filters wiederzuerkennen.

Viele der Variablen sind schon aus der Erläuterung des Kalman-Filters bekannt (Kapitel 3.1.2). Darüber hinaus sind die Werte $z_0, z_1, ..., z_n$ die gemessenen Werte. δ beschreibt die Schrittlänge und τ bezeichnet die Länge der Vorhersage. Es gilt $\tau = k\delta$, wobei k die Anzahl der Vorhersageschritte ist. Der Wert y_i enthält den vorhergesagten Wert für $z_0, z_1, ..., z_n$.

Im Initialisierungsteil kann man sehen, dass die Matrizen H, R, Q, Φ, Ψ konstant sind, da sie sich aus den ebenfalls konstanten Parametern $\beta, \sigma^2, \gamma, \delta$ und τ ergeben. Daraus folgt, dass

sich nur die Fehlerkovarianzmatrix P und der Zustandsvektor x in jedem Schritt verändern. Sie spiegeln den Zustand des Systems für jeden Quaternionanteil wieder und müssen auch für jeden Anteil gespeichert werden.
Um die Qualität der Vorhersage einschätzen zu können, sei der Fehler, also die Differenz zwischen vorhergesagtem und gemessenem Wert, wie folgt definiert:

$$D_k(\beta, \sigma^2) = \sum_{j=1}^{n-k} \sum_{l=0}^{3} |p_j^{(l)} - q_{j+k}^{(l)}| \tag{50}$$

Diese Gleichung dient als Grundlage zur Bestimmung der optimalen Parameter des Kalman-Filters. Nach der Implementierung dieses Verfahren ist es notwendig β und σ^2 anzupassen.

4. Vorbereitungen

4.1. Computersystem

Das Cybermind hi-Res900^{TM}3D wird für die Erarbeitung dieser Arbeit an einen handelsüblichen PC angeschlossen. Alle Tests, Berechnung und Messungen werden mit dem unter *Hardware* (Kapitel 4.1.1) beschriebenen System ausgeführt. Da die *dynamischen Fehler* (siehe Kapitel 2.4.2) direkt von der Performance des zugrundeliegenden Systems abhängig sind, werden auch alle Kalibrierungen und Kalman-Filter-Parameter auf dieses Referenzsystem abgestimmt. Eine Erläuterung zur Bestimmung der für den Vorhersagemechanismus relevanten Parameter wird in Kapitel X beschrieben.

4.1.1. Computersystem

Die benutzte Hardware besteht zum einen aus einem handelsüblichen PC mit folgenden Eckdaten:

- AMD Athlon 1GHz
- VIA KT133A Chipsatz
- Promise FastTrack 100 RAID-0 Controller
- 2x 60GB IBM IC35L060 als 120GB RAID-0 Array
- 512 MB Hauptspeicher
- GeForce4 Ti4400 128MB RAM (Dual-Head)

Neben diesem System wird ein weiterer Rechner mit:

- Pentium II 350MHz
- Intel BX Chipset
- 40GB IBM IC35L040
- 384 MB Hauptspeicher
- PCI Grafikkarte

eingesetzt, um die Kompilierung der erstellten Software auf einem Linux-System zu testen.

4.1.2. Cybermind hi-Res900^{TM}3D

Das für die Arbeit benutzte HMD ist das Cybermind hi-Res900^{TM}3D (siehe Abb. 26) der niederländischen Firma Cybermind. Die Daten des HMD werden hier kurz aufgeführt, da sie den evtl. vorhandenen statischen Fehler (Kapitel 2.4.1) mitbestimmen. Insbesondere die Bildwiederholfrequenz und Parameter wie FOV müssen bei der Implementierung beachtet werden.

- Signalsysteme für VGA/SVGA/NTSC/PAL und S-VHS
- Bilderzeugung auf zwei LC-Displays je 0,49 Zoll in 24-Bit Farbe
- Auflösung von 800x600 Pixel bei 60Hz/72Hz und 75Hz Bildwiederholfrequenz
- Optik suggeriert 45 Zoll Display in 2 Meter Entfernung
- 31,2° FOV (Field of View) für jedes Auge
- Einstellmöglichkeiten für Helligkeit, Kontrast, Lautstärke, ...
- externe Anschlüsse für 15-Pin D-Sub, S-VHS, RCA Video und Audio
- 700g Gewicht (inkl. InterTrax^2 Tracker)
- Unterstützung für drei 3D-Modi: Interlaced, Page-Flipping, Dual Input Mode

Abbildung 26: Cybermind hi-Res900^{TM}3D

4.1.3. InterSense Intertrax^2 Tracker

Der im HMD integrierte Tracker ist vom Typ *InterTrax*^{2} der Firma InterSense. Die Spezifikation des Trackers ist eine der wichtigsten Informationen zur Konstruktion des *Kalman-Filters*, da die Trackereigenschaften auch zum *dynamischen Fehler* (Kapitel 2.4.2) beitragen. Hier werden nur die Parameter zur Übersicht aufgeführt, die Integration der relevanten Trackereigenschaften in den Vorhersageprozess wird in Kapitel 6 erläutert.

- 3 Freiheitsgrade (Yaw, Pitch, Roll)
- Winkelbereiche: Pitch $\pm 80^\circ$, Yaw $\pm 180^\circ$ und Roll $\pm 90^\circ$
- 3° pro Sekunde minimale Winkelauflösung
- 256Hz interne Frequenz
- 4 Millisekunden interne Latenz
- $0{,}02^\circ$ minimale Winkelauflösung
- RS232 und USB Interface
- USB HID kompatibles Protokoll

Abbildung 27: InterSense InterTrax^2

4.2. Benutzerprofile

Um *off-line* den zu implementierenden Mechanismus zur Vorhersage der Kopfbewegungen testen und parametrisieren zu können, werden Daten benötigt. Diese Daten wurden von fünf Benutzern gewonnen, die sich in einer generierten Testumgebung frei bewegten. Dazu

wurde eine einfache virtuelle Welt mit vier relativ weit voneinander entfernten Einzelobjekten in die Applikation zur Ansicht von 3DS-Szenen geladen und die Benutzer angewiesen, jedes der Einzelobjekte genau zu betrachten. Die Testwelt wurde durch zwei dynamische Objekte ergänzt, die in zufälligen Abständen und mit variierender Geschwindigkeit die Welt durchfliegen, um den Benutzer zu zufälligen, schnellen Bewegungen zu animieren. Die Bewegungen des Kopfes wurden mit einer Frequenz von 100 Hz vom Tracker gelesen und in eine Datei gespeichert. Die Frequenz wurde so hoch gewählt, damit später Daten in der höchstmöglichen Auslösung für die Tests zur Verfügung stehen, aus denen bei Bedarf auch eine geringere Frequenz extrahiert werden kann. Das hochauflösende Sampling geht bei der Aufnahme der Profile sehr zu Lasten der Darstellungsgeschwindigkeit. Diese Einschränkung wurde jedoch für eine bessere Datengrundlage in Kauf genommen.

Als Format wird eine einfache textbasierte Darstellung verwendet, um die Daten auch in externen Anwendungen wie Microsoft Excel™verwenden zu können. In der ersten Zeile der Profildatei findet sich die Anzahl der Spalten und Zeilen der Daten. In den nächsten Zeilen folgen dann die eigentlichen Informationen. Es werden in den Profilen ausschließlich fünf Spalten benutzt. Die erste Spalte beinhaltet den Zeitstempel der Orientierungsmessung, gefolgt von den Quaternionenanteilen w, x, y und z.

Die Daten der einzelnen Profile wurden auf 3000 Werte, was bei 100 Hz genau 30 Sekunden Aufnahmedauer entspricht, gekürzt. Diese Vereinheitlichung ist notwendig, um den Betrag des Fehlers bei der Parametrisierung (Kapitel 6) vergleichen zu können.

Die ausgelesenen Datenreihen der einzelnen Probanden findet sich auf der beigelegten CD-ROM im Verzeichnis *profile*.

4.3. Analyse des end-to-end system delay

In Kapitel 2.4.2 wurde beschrieben, wie sich die zu eleminierende Latenz zusammensetzt. Um eine sinnvolle Vorhersage der Kopforientierung vornehmen zu können, ist es unabdingbar, diesen end-to-end system delay so genau wie möglich zu bestimmen. Alle nun folgenden Messungen beziehen sich auf das Entwicklungssystem (Kapitel 4.1) und können nicht allgemeingültig sein.

Alle in der Tabelle 1 auf Seite 12 aufgeführten Einzelfehler t_0-t_5 sollen nun so genau wie möglich bestimmt werden. Dazu wurden eine Reihe von kleineren Programmen implementiert, deren Funktion im jeweiligen Schritt erläutert wird.

Als erstes gilt es, die interne Latenz des Trackers zu messen. Leider setzt die Bestimmung

dieser Verzögerung technisches Equipment voraus, dass zur Erstellung dieser Arbeit nicht zur Verfügung stand, so dass die vom Hersteller des Trackers angegebenen 4 ms als richtiger Wert angenommen werden müssen. Eine Möglichkeit, die interne Latenz genau zu bestimmen, ist es, den Tracker an einem Objekt zu befestigen, dass sich in einer bekannten Weise bewegt. Dazu eignet sich z.B. ein Pendel. Die Bewegung wird dann von einem zweiten Tracker gemessen, der folgende Eigenschaften erfüllen muss:

1. Die Latenz des Referenztrackers muss bekannt oder so klein sein, dass sie vernachlässigt werden kann.
2. Der Referenztracker darf keine elekomagnetischen Interferenzen erzeugen.
3. Es dürfen sich keine metallischen Objekte in der Nähe des zu messenden Trackers befinden.

Diesen Voraussetzungen folgend, eignet sich beispielsweise eine Videokamera, um die periodische Bewegung des Pendels zu messen. Durch die Videoaufnahme des Pendels lässt sich die zeitliche Differenz zwischen dem Zeitpunkt, an dem die Videokamera das Pendel in Ruhe abbildet und dem Zeitpunkt, in dem der Tracker die gleiche Position misst, bestimmen.

Der nächste Schritt ist die Bestimmung der Zeit, die benötigt wird, um die Daten vom Tracker zu lesen. Dazu wurde ein kleines Kommandozeilenprogramm implementiert, dass 10 000 Mal das Auslesen der Trackerdaten ausführt. Zunächst stellt das Programm das Ausgabeformat des Trackers auf Quaternionen um, da evtuell bei den unterschiedlichen Formaten auch unterschiedliche Zeiten entstehen und später ebenfalls mit Quaternionen gearbeitet wird. Danach wird in jeder Iteration der Zeitindex und die Orientierung vom Tracker ausgelesen. Es werden keinerlei Daten gespeichert und auch keine Bildschirmausgaben vorgenommen, um möglichst alle anderen Verzögerungen zu vermeiden. Das Ausführen benötigte auf dem Entwicklungssystem (Kapitel 4.1) eine Zeit von 0,04 Millisekunden. Dieser geringe Wert ist auf den direkten Anschluss des Trackers an den Rechner zurückzuführen. Hier können auch deutlich größere Werte entstehen, wenn der Tracker z.B. über den seriellen Anschluss oder gar über eine Netzwerkverbindung abgefragt wird.

Den größten Anteil an der Gesamtlatenz hat das Errechnen und Zeichnen der beiden monokularen Ansichten für die Stereoprojektion im HMD. Da der Kalman-Filter nur in der Lage ist, diskrete Zeitabstände vorherzusagen, muss erreicht werden, dass das Rendern in konstanter Zeit abläuft. Da die Renderzeit allerdings abhängig von der Komplexität der Szene und auch von der Bewegung ist, muss diese Konstanz künstlich erzeugt werden. Dieses Problem lässt sich erst während der Implementierung lösen und wird in Kapitel 5.3 genau erläutert.

Der letzte Zeitabstand wird durch das Kopieren der fertigen Bilder in die HMDs erzeugt.

Um ihn zu messen, wurde ein einfaches OpenGL-Programm entwickelt, dass die GLUT-Teekanne in einer sonst leeren Umgebung darstellt. Die Zeit, die zum Kopieren der Puffer benötigt wird, wurde durch einen hochauflösenden Timer bestimmt. Dabei musste leider festgestellt werden, dass dieser Wert keineswegs konstant (Abb. 28) ist. Aus diesem Grund ist es notwendig, die Zeit, die zum Kopieren der Puffer benötigt wird, während der Programmausführung ständig neu für jeden errechneten Frame zu bestimmen.

Die Gesamtlatenz lässt sich im Voraus nicht bestimmen, da ein Großteil der Latenzen dy-

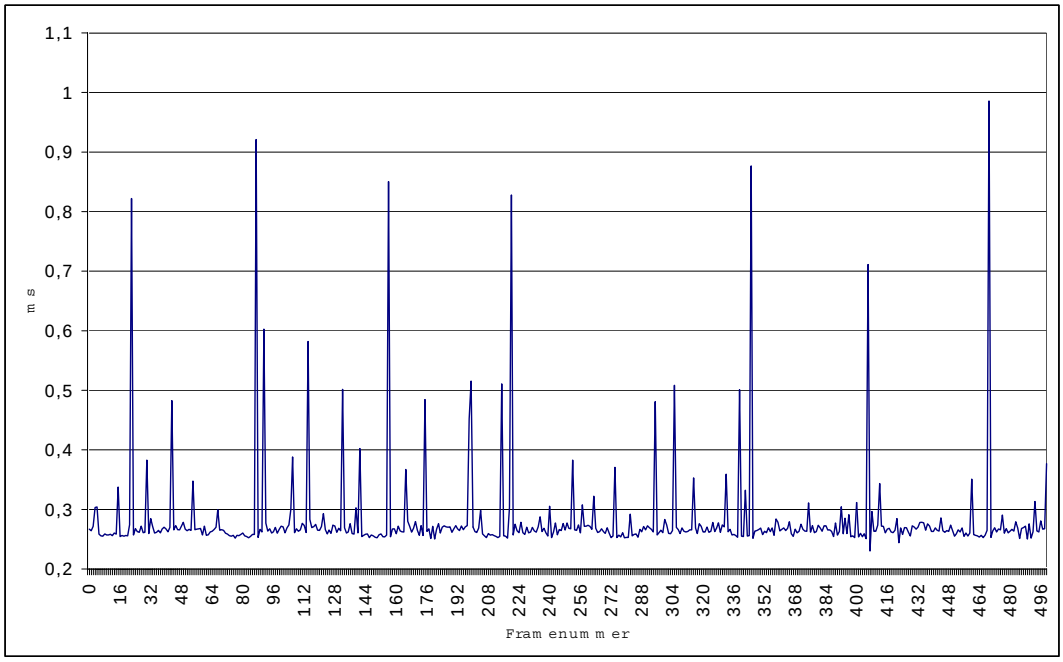

Abbildung 28: Diagramm der Pufferkopierzeit

namisch ist. Die Summe der statischen Latenzen ergibt sich zu diesem Zeitpunkt aus der internen Trackerlatenz von 4 ms und der Verzögerung beim Auslesen der Orientierungsdaten von 0,04 ms. Man kann also sehen, dass sich die Gesamtlatenz aus einer Anzahl von im Vorfeld gemessenen, also statischen Latenzen und den dynamischen Latenzen, die während der Errechnung der Ansichten entstehen, zusammensetzt. Das zu entwickelnde System muss also in der Lage sein, diese Latenzen während der Ausführung zu bestimmen und geeignet darauf zu reagieren, um die Darstellung mit den mittels des Kalman-Filters vorhergesagten Werten in Einklang zu bringen.

5. Entwicklung der Software

In den vorangegangenen Kapiteln wurden die Konzepte und Algorithmen erläutert, um den end-to-end system delay bei immersiven, trackergesteuerten Anwendung durch den Kalman-Filter zu minimieren. Die nun zu implementierende Software teilt sich in drei Teilbereiche. Das Kernstück stellt die Implementierung des Kalman-Filters zur Vorhersage der als Quaternionen dargestellten Kopfbewegungen nach der Methode von Liang [LSG91], dar.
Um den Kalman-Filter überprüfen und parametriesieren zu können, wird zusätzlich eine Testumgebung entwickelt. Die Testumgebung soll die Aufnahme von Kopfbewegungen bieten und eine grafische Analyse der Orientierungsdaten ermöglichen. Zusätzlich soll der Kalman-Filter in dieser Anwendung integriert werden, um die Vorhersage mit verschiedenen Parameterkonfigurationen an den Benutzerprofilen (Kapitel 4.2) testen und visualisieren zu können. Diese Anwendung wird hier im Softwareentwicklungsprozess nicht mit betrachtet. Eine kurze Übersicht über ihre Funktionen ist im Anhang B zu finden.

Der fertige Kalman-Filter soll letztlich in einer Anwendung zur stereoskopischen Anzeige von 3D-Szenen eingesetzt werden, um den end-to-end system delay zu minimieren und damit einen angenehmeren Eindruck der virtuellen Umgebung zu erzeugen und Probleme der durch die Latenz ausgelösten Motion- oder Simulatorsickness zu verringern.
Die einzelnen Softwareteile werden im Softwarezyklus gesondert betrachtet.

5.1. Analyse und Systemdefinition

5.1.1. Kalman-Filter

Der zu implementierende Kalman-Filter folgt wie in Kapitel 3.2 gezeigt dem Algorithmus nach Liang [LSG91]. Die Implementierung soll es ermöglichen, die Kopforientierung für einen diskreten Zeitabstand vorherzusagen. Die Darstellung der Orientierung soll als Quaternionen erfolgen. Der Kalman-Filter soll sowohl für off-line als auch für online-Anwendungen benutzbar sein. Besonderes Augenmerk muss dabei auf der Ausführungsgeschwindigkeit liegen, um durch den Einsatz nicht zusätzliche Latenzen zu erzeugen.

5.1.2. Kalman-Umgebung

Diese prototypische Anwendung dient zur visuellen Demonstration des Kalman-Filter unterstützten Tracking. Die Anwendung soll es ermöglichen, 3DS-Szenen stereoskopisch im HMD zu betrachten und sich in der geladenen Umgebung zu bewegen. Die durch den Tracker realisierte Interaktion des Benutzers mit der Umgebung soll durch den Einsatz des Kalman-Filters verbessert werden. Die Anwendung soll die geladene Szene stereoskopisch

korrekt darstellen und den end-to-end system delay eliminieren oder verringern. Voraussetzung für die Implementierung ist die möglichst exakte Messung des end-to-end system delay, was die Implementierung mehrerer kleinerer Messtools, wie in Kapitel 4.3 beschrieben ist, voraussetzt. Das im Anhang A zu findende, aus der Analyse resultierende Pflichtenheft, behandelt sowohl die eigentliche Anwendung als auch die Testtools.

5.2. Entwurf

5.2.1. Kalman-Filter

Um den Kalman-Filter zu implementieren, werden zunächst Klassen für das Rechnen mit Vektoren und Matrizen benötigt. Diese Klassen zur Realisierung der Filtermathematik werden speziell für den Algorithmus angepasst und sollen nicht allgemeingültig sein. So werden beispielsweise die Matrizen auf die Größe 2x2 beschränkt, um ein einfacheres Transponieren und Inverse bilden zu ermöglichen. Die Klassen *Matrix* und *Quaternion* stellen ihre Funktionalität fast ausschließlich durch überladene Operatoren zur Verfügung.

Betrachtet man sich den Algorithmus von Liang (Kapitel 3.2), so kann man erkennen, dass für jeden Quaternionenanteil ein eigener Filter mit mindestens den Zustandsvariablen A und x benötigt wird. Diese Einzelfilter werden als eigene Klasse modelliert. Dieser *SingleKalman* soll das Filtern eines einzelnen normalisierten Quaternionenanteils realisieren. Jede Instanz von SingeKalman hat einen festen Satz von Variablen, die die Parametrisierung des Filters darstellen (Kapitel 3.2). Wie in Abbildung 29 zu sehen ist, besteht die Klasse aus den zur Realisierung des Kalman-Filters benötigten Matrizen und Vektoren. Sie bietet nur einen Konstruktor, dem als Parameter die schon bekannten Werte zum Einstellen des Filteralgorithmus´ übergeben werden. Darüber hinaus liefert die Methode *predict* den jeweils nächsten a posteriori-Schätzwert unter Berücksichtigung des als Parameter übergebenen Messwertes.

Die Klasse *QuadKalman* kombiniert nun vier Instanzen der Klasse SingleKalman, um eine durch ein Quaternion dargestellte Orientierung vorauszusagen. Die Parametriesierung ist, wie in der Abbildung zu sehen, mit der des SingleKalman identisch. Lediglich die Filtervariable wurde durch ein Quaternion ersetzt. QuadKalman linearisiert das Quaternion, filtert jeden Teil mittels einer Instanz von SingleKalman und fügt die Ergebnisse der einzelnen Kalman-Filter zum resultierenden Quaternion zusammen und normalisiert es.

Abbildung 29: UML Diagramme der Klassen des Kalman-Filters

5.2.2. Kalman-Umgebung

Ausgehend vom Pflichtenheft der Anwendung (Anhang A) kann man einige zur Realisierung notwendige Kernkomponenten definieren. Zum Laden von 3DS-Dateien wird die Open-Source Biblithek *lib3ds* [Dev02] benutzt. Die Funktionalität der Bibliothek beschränkt sich auf des Einlesen von 3D-Studio Max™Dateiformat 3DS. Die Klasse *gl3DS* nutzt die Funktionen von lib3ds [Dev02] und ergänzt dieses Laden der Dateien um die Möglichkeit, eine Szene in eine OpenGL-Display Liste [WND97] zu rendern. Dabei werden sowohl die Beleuchtungs-, als auch die Geometrieinformationen der Szene interpretiert.

Zur Realisierung der Ausgabe auf dem HMD benötigt es keinerlei explizite Implementierung, sondern nur eine Grafikkarten- und Treiberkonfiguration, die die OpenGL-Erweiterung des Stereoformats unterstützt. Die Erzeugung der eigentlichen stereoskopischen Ansichten nach der off-axis-Methode (Kapitel 2.6.3) wird in eine Klasse *glCamera* gekapselt. Sie dient der Errechnung und dem Setzen der korrekten Kameraparameter, um die zwei Einzelansichten für das linke und rechte Auge zu erzeugen. Da während der Entwicklung nicht immer stereoskopische Ansichten notwendig sind, bietet die Klasse auch die Möglichkeit einer monokularen Ansicht mittels der Methode *matrix*. Die Methoden für die Positionierung, Orientierung und das Einstellen der Projektion für das linke und rechte Auge werden in den Methoden *leftmatrix* und *rightmatrix* implementiert. Die Positionierung der Kamera wird durch die Methoden

raise, *strafe* und *move* für die einzelnen Achsen realisiert. Die Bewegung soll jedoch nicht an den Koordinatenachsen, sondern an der Orientierung des Kopfes ausgerichtet sein, d.h. eine Vorwärtsbewegung verläuft entlang der Blickrichtung und das Bewegen in Querrichtung (strafe) analog rechtwinklig zum Sichtvektor. Eine Ausnahme stellt die Bewegung nach oben und unten (raise) dar. Diese Bewegung verläuft auf der Y-Achse des Weltkoordinatensystems, da sich bei der Entwicklung herausgestellt hat, dass eine Bewegung in Richtung des Kamera-Up-Vektors zu Verwirrungen führen kann, wenn der Kopf stark geneigt ist.

Die Interaktion mit der Trackerhardware stellt den nächsten Teil des Pflichtenheftes dar.

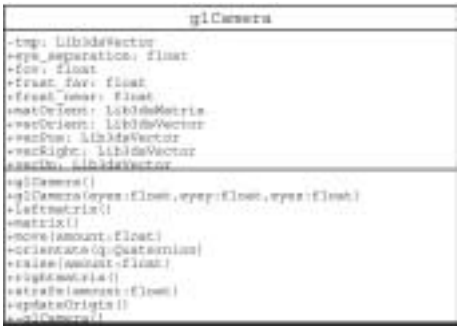

Abbildung 30: UML-Diagramm der Klasse glCamera

Diese Anforderung kann ebenfalls in einer einfachen Klasse abgebildet werden. Die Klasse *glTracker* stellt eigentlich nur eine Kapselung der C-Funktionalität des iSense SDK in einer C++ Klasse dar. Sie dient dem Auffinden und dem Verbinden mit der Trackerhardware. Für diese spezielle Anwendung wird, wie schon erläutert, nur die Darstellungsform der Quaternionen für die Orientierung benutzt. glTracker stellt das Ausgabeformat des Trackers automatisch auf Quaternionen um. Eine Verabeitung von Euler-Winkeln wird nicht implementiert. Neben der Bestimmung der Orientierung und des dazugehörenden Zeitstempels in Methode *readTracker* kann die Klasse auch Informationen über die Trackerhardware liefern, wie z.B. dessen Bezeichnung und die Übertragungsgeschwindigkeit mit der der Tracker arbeitet. Die Klasse soll möglichst wiederverwendbar sein. Das UML-Diagramm der Klasse glTracker ist im Abbildung 31 dargestellt.

Zusammen mit dem Kalman-Filter (Kapitel 5.2.1) steht mit den Klassen gl3DS, glCamera und glTracker ein Großteil der Funktionalität zur Verfügung, die für die Anwendung notwendig ist. Das Hauptproblem bei der Implementierung, die nicht beim Entwurf gelöst werden kann, liegt im sensiblem Timing der Anwendung.

Das Zusammenspiel der Komponenten wird von einem Windowsprogramm gesteuert, dass einzig aus einem OpenGL-Fenster besteht. Die Funktionsweise wird detailliert im Kapitel 5.3 beschrieben.

Abbildung 31: UML-Diagramm der Klasse glTracker

5.3. Implementierung

Das folgende Kapitel soll über die relevanten Eckpunkte der Implementierung informieren. Dazu gehören die Beschreibung der Entwicklungsumgebung und einzelne Details der Realisierung einiger Softwaresegmente, die sich als besondere Schwer- und Problempunkte der Implementierung erwiesen haben.

5.3.1. Verwendete Bibliotheken

Die Verwendung von Bibliotheken bezieht sich nur auf die eigentliche Anwendung, da bei der Implementierung des Kalman-Filters keinerlei Bibliotheken zum Einsatz kamen.

In der Anwendung kommen im Gegensatz dazu eine Reihe von Bibliotheken zum Einsatz. Um auf den Tracker zuzugreifen, wurden InterSense Interface Libraries SDK in der Version 3.45 verwendet, die neben Windows auch Linux und IRIX als Plattform unterstützen. Diese Bibliotheken erlauben den Zugriff auf alle Tracker der Firma InterSense und stellen je nach verwendetem Modell verschiedene Funktionalitäten zur Verfügung.

Daneben wird als Grafikbibliothek OpenGL in der Version 1.4 verwendet, da sie universtell einsetzbar und auf nahezu allen Systemen verfügbar ist sowie eine ausgezeichnete Unterstützung für Stereoanwendungen bereitstellt. Insbesondere die Stereounterstützung bietet einen Vorteil gegenüber Microsofts DirectX. Im Zusammenhang mit OpenGL kommt außerdem das OpenGL Utility Toolkit *GLUT* zum Einsatz, um grundlegende Funktionen wie das Erstellen des OpenGL-Fenster und die Interaktion mit dem Keyboard komfortabler benutzen zu können.

Zur Implementierung der Anzeige von 3DS-Dateien wird zum Parsen des Dateiformates die Open-Source Bibliothek *lib3DS* genutzt. Diese Bibliothek dient nicht dem eigentlichen Rendern der Szenen, sondern bietet nur die Funktionalität, um die Dateien der bekannten Modellierungsapplikation 3D-Studio MAX™zu parsen.

5.3.2. Wahl der Programmiersprache

Das Hauptkriterium bei der Wahl der Programmiersprache ist, wegen der zu lösenden zeitkritischen Problematik, die Performance. Daneben muss die schnelle und effektive Verwendung des HMD und der OpenGL-Schnittstelle möglich sein. Die Software soll unter Microsoft Windows XP entwickelt werden, da eine effektive Implementierung unter Linux leider durch die fehlende Möglichkeit der Benutzung des HMD-Trackers über USB behindert wird. Unter Windows unterstützt das zum HMD gelieferte InterSense SDK die Anbindung mittels USB, das weniger Latenz erzeugt als die Anbindung über einen seriellen Anschluss. Die im Software Development Kit enthaltene Bibliothek in Form einer DLL (Dynamic Link Library) zur Anbindung des Cybermind hi-Res900^{TM}3D macht es zwar möglich, das HMD über API-Calls in nahezu jeder beliebigen Sprache zu benutzen, dennoch ist die Anbindung nur für Microsoft Visual Basic und C ausreichend dokumentiert. Die besten Optimierungsmöglichkeiten und eine perfekte Zusammenarbeit mit OpenGL bietet somit C. Da allerdings funktionales Programmieren die Wiederverwendung von einzelnen Teilen dieser Arbeit verhindern würde, kommt letztlich C++ zum Einsatz.

5.3.3. Kalman-Filter

Durch die Implementierung der Matrizen und Vektoren mittels überladener Operatoren stellt sich die Implementierung des SingleKalman als Abschrift des in Kapitel 3.2 beschriebenen Algorithmus´dar. Die Initialisierung der Matrizen (Kapitel 3.2) wird durch den Konstruktor der Klasse vorgenommen. Der restliche Teil des Algorithmus´ wird in der Methode **predict** realisiert, die einen kompletten predictor-corrector Zyklus (Kapitel 3.1.2) des Kalman-Filters darstellt und den vorhergesagten Wert zurückgibt.

Die Klasse **QuadKalman** realisiert das Linearisieren und Normalisieren der gemessenen bzw. vorhergesagten Quaternionen nach dem Schema, dass in Kapitel 3.2 beschrieben wurde. Sie enthält ebenso wie SingleKalman nur einen Konstruktor und eine Methode **predict**. Die predict-Methode normalisiert das übergebene Quaternion und trifft eine Vorhersage für jeden Anteil mittels eines eigenen SingleKalman-Filters. Das resultierende, gefilterte Quaternion wird normalisert und zurückgegeben.

5.3.4. Anwendung

In Kapitel 5.2.2 wurden die Komponenten entworfen, die für die Applikation notwendig sind. Die Implementierung der Klasse glTracker wird hier nicht weiter untersucht, da die Implementierung nur eine Kapselung der Funktionen des InterSense SDK sind. Ebenso werden die Details der Implementierung der Klasse gl3DS nicht weiter erläutert, da dort lediglich

Funktionen der Bibliothek lib3ds[Dev02] genutzt werden, um Szenen in eine geeignete Datenstrukturen einzulesen. gl3DS rendert diese Szenen dann in OpenGL-Displaylisten. Mehr zum Zeichnen mit OpenGL ist z.B. in [WND97] und [WND99] zu finden.

glCamera muss die Funktionalität bieten, das off-axis Stereoprojektionsverfahren für das HMD zu implementieren. Die theoretische Grundlage dazu wurde bereits in Kapitel 2.6.3 besprochen. Die Methoden **leftmatrix** und **rightmatrix** errechnen die Parameter für glFrustum() für die off-axis-Projektion und setzen `GL_PROJECTION` und `GL_MODELVIEW` entsprechend. Aus diesem Grund kann nach Aufruf der Methoden sofort mit dem Zeichnen der Geometrie begonnen werden. Für die Matrix und Vektoroperationen wurde auf die Funktionalität von lib3DS zurückgegriffen, da sie eine ausgezeichnete Performance bieten und die für den Kalman-Filter entwickelten Matrix-Klasse extra für den Filter angepasste 2×2-Matrizen implementieren. Das UML-Diagramm der Klasse glCamera ist in Abbilung 30 zu sehen.

Das Timing der Anwendung stellt das größte Problem der Realisierung dar. Da der Kalman-Filter nur einen diskreten Zeitabstand verarbeiten kann, ist es notwendig, ein solches konstantes Timing zu erzeugen. Der Ablauf zur Erzeugung eines Frames mit stereoskopischer Projektion und unter Verwendung des Kalman-Filters zur Minimierung der Latenz ist in Abbildung 32 dargestellt:

Trotz der Verwendung von GLUT wird für die Errechnung der Frames nicht das im Toolkit übliche Verfahren des Callback genutzt. Normalerweise würde bei der Initialisierung des Toolkits eine Funktion definiert, die automatisch aufgerufen wird, wenn das Fenster neu zu zeichnen ist. Da diese Methode die Kontrolle über das Timing der Anwendung verkomplizieren würde, wird das Rendern der einzelnen Frames durch eine nicht von GLUT gesteuerte Methode implementiert. GLUT wird nur zur Initialisierung des OpenGL-Fensters benutzt, da dessen Funktionen viel Implementierungsaufwand ersparen.

Im ersten Schritt der Frameerzeugung wird zunächst die Orientierung des Trackers ausgelesen. Es wird angenommen, dass dieser Wert bereits um 4 ms verzögert ist (Kapitel 4.3). Neben der Orientierung wird hier der Zeitstempel erzeugt, der den Zeitpunkt t_0 definiert. An dieser Stelle wird auch die Vorhersage durch den Tracker errechnet. Der Kalman-Filter erzeugt eine Vorhersage für die möglichst hoch gewählte Trackerfrequenz und einen Zeitschritt in die Zukunft, da dabei der Fehler am geringsten ist (Kapitel 6.1). Hier muss bei der praktischen Anwendung, also bei der Darstellung mehr oder weniger komplexer Szenen, ein Kompromiss gefunden werden.

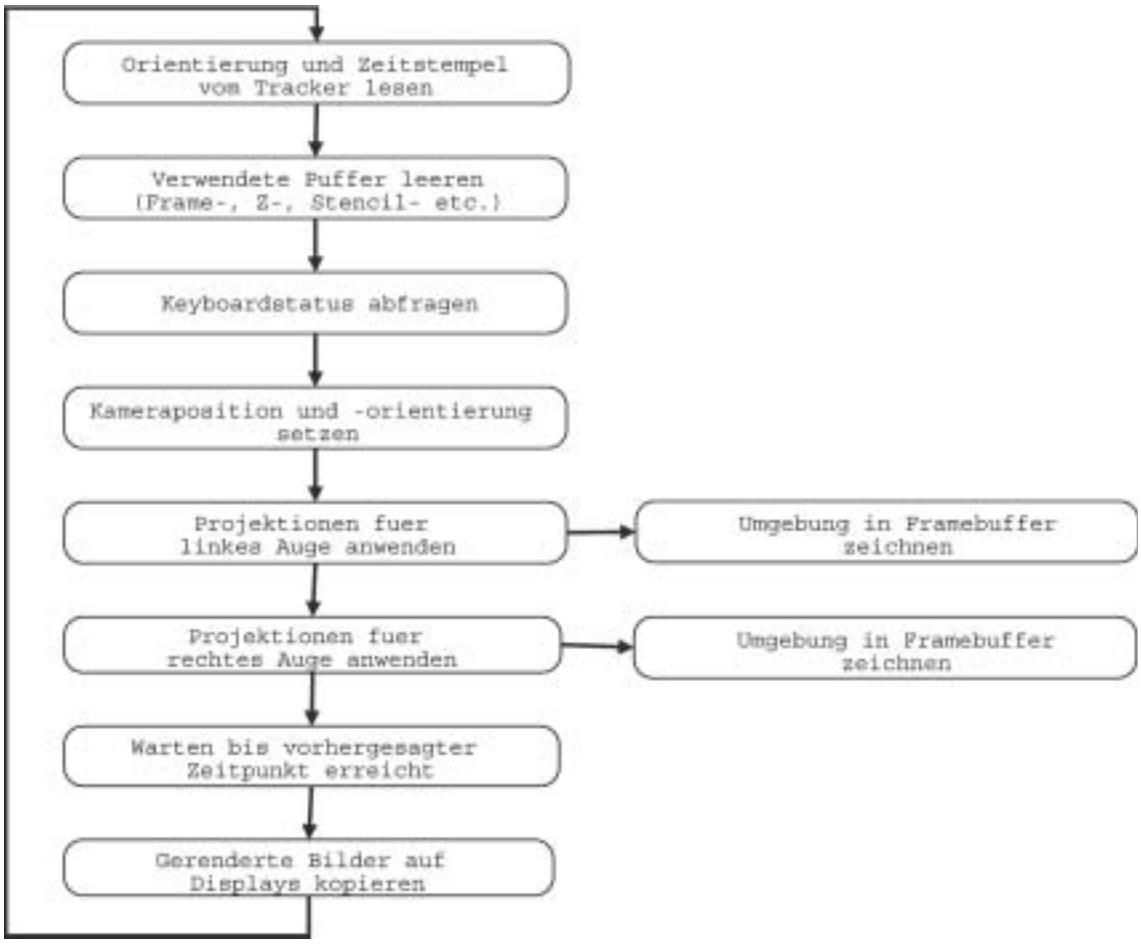

Abbildung 32: Ablauf der Erzeugung eines Frames

Die Trackerfrequenz diktiert durch den Zwang des konstanten Zeitabstandes der Schätzung die Anzahl der zu errechnenden Frames pro Sekunden. Angenommen der Tracker wird mit einer Frequenz von 100 Hz, also einem Intervall von 10 ms, abgefragt. Dann bleibt nach Subtraktion der statischen Latenz des Trackers von 4 ms nur eine Zeit von 6 ms, um alle anderen Aufgaben inklusive des Kopierens der fertigen Bilder zu erfüllen. In den meisten Fällen ist dies aufgrund der Komplexität der Szene nicht möglich und die Trackerfrequenz muss herabgesetzt werden, um ausreichend Zeit für die Errechnung der Ansichten und alle anderen Aufgaben zu haben. Bei der Entwicklung wird zunächst eine sehr einfache Szene verwendet, die nur aus einem Schachbrettmuster besteht (Abb. 33. Selbst bei dieser primitiven Szene reicht die Leistung des Entwicklungssystems nicht aus, um beide Ansichten zur Errechnen. Aus diesem Grund wird eine Frequenz von nur 50 Hz verwendet.

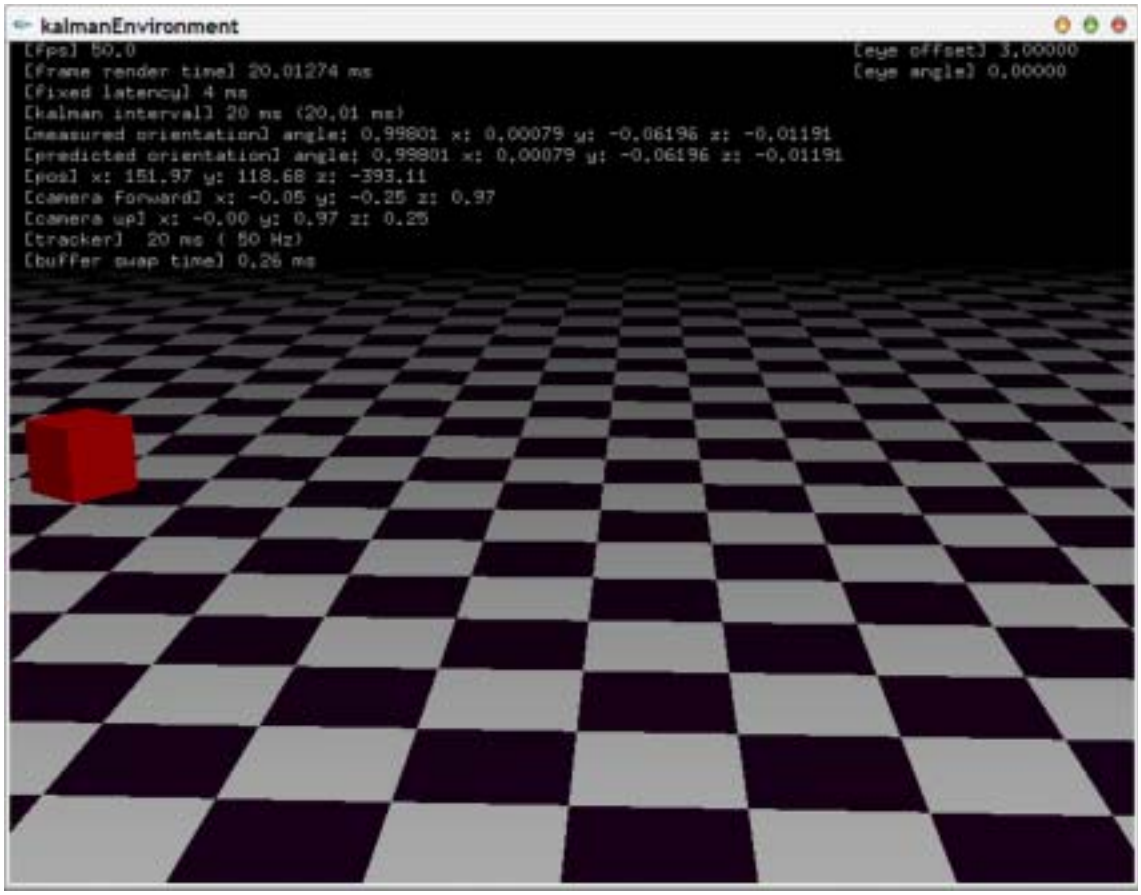

Abbildung 33: OpenGL Fenster mit der Darstellung der Umgebung bei der Entwicklung

Nachdem die Orientierung bestimmt und die Vorhersage errechnet ist, werden die Frame- und Z-Puffer gelöscht. Beim OpenGL-Stereomodus wird für das linke Auge der Puffer `GL_BACK_LEFT` und für das rechte `GL_BACK_RIGHT` benutzt. Das Leeren der Puffer kann

jedoch signifikant beschleunigt werden. Die meisten OpenGL-Implementierungen, darunter auch die hier verwendete von nVidia, leeren beim Löschen des Puffers `GL_BACK` sowohl `GL_BACK_LEFT` als auch `GL_BACK_RIGHT`.

Im nächsten Schritt werden die Tasten zur Steuerung der Kameraposition abgefragt. Wie eingangs schon erwähnt, wird das bei GLUT übliche Verfahren der Callback-Methoden hier nicht benutzt, da eine möglichst gleichmäßige Bewegung erreicht werden soll. Aus diesem Grund kommt ein Verfahren zum Einsatz, das *Time Based Movement* genannt wird. Dabei wird die Kamera bei einem Tastendruck nicht um einen festen Betrag bewegt. Die Kamera bewegt sich stattdessen um einen festen Betrag pro Zeiteinheit, damit die Bewegung auch bei unterschiedlicher Framerate gleichmäßig ist.

Es folgt das Positionieren und Orientieren der Kamera.

Nachdem die Kamera ausgerichtet und positioniert ist, werden die Projektionen für das linke Auge angewendet und die Geometrie der Umgebung in den Puffer `GL_BACK_LEFT` gezeichnet. Analog wird mit dem rechten Auge verfahren, wobei in den Puffer `GL_BACK_RIGHT` gezeichnet wird.

Im nächsten Schritt folgt nun die Synchronisation der Anzeige mit der Trackerfrequenz. Im ersten Schritt wurde eine Vorhersage für einen bestimmten Zeitabstand getroffen. Die Bilder müssen genau zu dem Zeitpunkt in den LC-Displays des HMD erscheinen, für den die Vorhersage erzeugt und die Bilder gerendert wurden. Dieser Zeitpunkt muss natürlich in der Zukunft liegen und außerdem muss noch genügend Zeit für das Kopieren der Framepuffer zur Verfügung stehen. Die Zeit, die für das Kopieren der Puffer notwendig ist, wird in jedem Frame gemessen. Diese Messung dient dazu, eine Schleife zu konstruieren, die genau so lange läuft bis der Zeitpunkt erreicht ist, in dem das Kopieren der Puffer ausgelöst werden muss, damit die Bilder zum Vorhersagezeitpunkt mit der entsprechenden Orientierung in den Displays erscheinen. Danach beginnt der Kreislauf von vorn.

Ein wichtiges Detail bei der Implementierung stellt der verwendete Timer dar. Das GLUT-Toolkit beinhaltet zwar auch eine Möglichkeit zur Nutzung von Timern, jedoch reicht die Auflösung von 1 ms für das hier benötigte Timing nicht aus. Stattdessen wird das Timing mittels der Windows API-Funktionen *QueryPerformanceFrequency* und *QueryPerformanceCounter* realisiert. Diese beiden Funktionen ermöglichen den Zugriff auf einen deutlich höher auflösenden Timer. Auf dem Entwicklungssystem läuft dieser Timer mit einer Frequenz von 3579545 Hz. Leider existiert diese Möglichkeit nur unter Windows und die bis dahin theoretische Plattformunabhängigkeit der Anwendung muss leider aufgegeben werden.

Damit sind alle Implementierungen abgeschlossen und die Umgebung ist in der Lage, eine vorhergesagte Orientierung zum gewünschten Zeitpunkt darzustellen. Voraussetzung dafür ist jedoch, dass das System schnell genug ist, alle Berechnung innerhalb des Trackerintervalls abzüglich der statischen Latenzen auszuführen. Kann dies nicht erreicht werden, so wird der Kalman-Filter deaktiviert, da er in dieser Situation ohnehin nutzlos wäre und die zusätzlich für die Vorhersage benötigte Zeit dann für eine höhere Framerate zur Verfügung steht.

Im Screenshot der Anwendung (Abb. 33) ist eine Vielzahl von Textausgaben zu sehen, die auf die Ebene der Null-Parallaxe projeziert wird. Diese Ausgaben sind für die Beobachtung der Funktion des Systems notwendig. Neben der aktuellen *Framerate*, der gemessenen und vorhergesagten Orientierung und der Position im Raum werden Informationen über den Kalman-Filter angezeigt. Von besonderer Wichtigkeit ist dabei die dritte Zeile, die den Vorhersagezeitraum des Kalman-Filters anzeigt (**[kalman interval]**). Dort sind zwei Werte zu erkennen. Zum einen der *geplante* Intervall und unmittelbar dahinter der durch das "Ausbremsen" tatsächlich erreichte. Damit die Vorhersage funktioniert, müssen diese Werte so genau wie möglich übereinstimmen, da sonst trotz Kalman-Filter der Benutzer Bilder für falsche oder genauer für den Zeitpunkt falsche Orientierung zu Gesicht bekommt, was das Verfahren ad absurdum führt. Dies kann durch eine zu hohe Auslastung des Systems oder eine zu komplexe Szene geschehen.

5.4. Bedingungen beim Programmstart

Um die Applikation ausführen zu können, muss eine Reihe von Voraussetzungen erfüllt sein. Das verwendete System muss ausreichend Leistung besitzen, um das Rendern der Szene innerhalb einer Samplingperiode des Trackers ausführen zu können. Es sollten keinerlei weitere Programme während der Benutzung der Umgebung auf dem System laufen, da durch die Ausführung anderer Prozesse das Timing der Anwendung gestört werden kann. Wenn möglich sollte die Anwendung immer mit höchster Priorität ausgeführt werden. Selbstverständlich wird ein korrekt installiertes OpenGL-System vorausgesetzt, dass auch eine Stereounterstützung bietet. Fehlt diese Unterstützung wird die Anwendung in einem einfachen Fenster ausgeführt. Darüber hinaus müssen die verwendeten Bibliotheken (Kapitel 5.3.1) installiert und das HMD über einen USB-Anschluss mit dem Rechner verbunden sein. Je nach Wahl des Verfahrens zur Ansteuerung des HMD (Kap. 2.6.4) muss das Gerät ensprechend angeschlossen und der Grafikkartentrieber konfiguriert sein.

5.5. Organisation der Quelldateien

Als Entwicklungsumgebung für diese Arbeit wurde Microsoft Visual C++ in der Version 6.0 mit Service Pack 5 verwendet. Alle entwickelten Softwareteile sind Bestandteil eines *Arbeitbereiches*, der auf der CD-R im Verzeichnis *sources* zu finden ist. Der Arbeitsbereich enthält vier Projekte: Die Projeke kalmanGraph, kalmanHill und kalmanEnvironment sind

Projektname	Beschreibung
kalmanFilter	behinhaltet alle Klassen für den eigentlichen Kalman-Filter
kalmanGraph	Visuelle Testumgebung (siehe Anhang B)
kalmanHill	Konsolenanwendung für den Hill-Climing-Algorithmus
kalmanEnvironment	Die filterunterstützte Anwendung

Tabelle 2: Liste der Projekte des Arbeitsbereiches

vom Projekt kalmanFilter abhängig, da sie Klassen aus diesem Projekt verwenden. Um kalmanEnvironment zu kompilieren, ist eine korrekte Installation von lib3DS notwendig. Die Quellen und auch fertig übersetzten Bibliotheken sind auf der Projektseite bei *SourceForge* unter `http://lib3ds.sourceforge.net` zu finden. Außerdem wird eine Installation von OpenGL mindestens in der Version 1.1 und GLUT in der Version 3.7 benötigt.

Die Softwarebestanteile sollten möglichst im Arbeitsbereich kompiliert werden, da verschiedene Kompileroptimierungen für die Projekte definiert wurden. So wird beispielsweise vom Compiler versucht, so viele Methoden wie möglich *inline* zu kompilieren, um zu Lasten der Dateigröße eine bessere Performance zu erreichen.

6. Tests und praktisches Tuning des Kalman-Filter

Die Implementierung des Kalman-Filters und seiner Komponenten stellt sich als, wie in Kapitel 5.3.3 erwähnt aus Entwicklersicht, recht einfach dar. Dennoch erfordert natürlich die Implementierung einerseits Tests, um die korrekte Funktion des Verfahrens testen zu können. Um die Umsetzung der Quaternionen-, Matrizen- und Vektorenoperationen sowie des Filteralgorithmus´ selbst überprüfen zu können, wurde der Algorithmus nocheinmal in *Matlab* implementiert. Die einzelnen Benutzerprofile wurden nun vom C++ und Matlab-Filter mit identischer Parameterkonfiguration vorhergesagt und der Fehler für die Gesamtvorhersage wurde bestimmt. Der Fehler wird, wie in Kapitel 3.2 beschrieben, wie folgt berechnet:

$$D_k(\beta, \sigma^2) = \sum_{j=1}^{n-k} \sum_{l=0}^{3} |p_j^{(l)} - q_{j+k}^{(l)}| \tag{51}$$

Da zu diesem Zeitpunkt noch keine besseren Parameter bestimmt werden konnten, wurden die Benutzerprofile auf 20Hz, also 50ms Schrittweite heruntergerechnet und die Parametrisierung von Liang [LSG91] verwendet. Das Ergebnis des Funktionstests ergab einen identischen Fehler bei beiden Implementierungen über alle Benutzerprofile. Somit kann der Algorithmus als mathematisch richtig angenommen werden.

Die mathematische Richtigkeit stellt natürlich die Hauptvoraussetzung für das Funktionieren der Vorhersage dar. Allerdings ist die korrekte Parametrisierung des Algorithmus entscheidend für die Qualität der Vorhersage. Die schon im Kapitel 3.2 erläuterten Parameter des Verfahrens sind β, σ^2, γ, δ und τ. Dabei bezeichnet γ das Messrauschen. Das Messrauschen beim InterTrax^2 wird vom Hersteller InterSense als nichtexistent (*zero noise*) angegeben. Aus diesem Grund kann der Wert von γ sehr klein gewählt werden. Für das Tuning der restlichen Parameter erhält γ den konstanten Wert von 0.001.

6.1. Off-line Parameteroptimierung

Die Gleichung 51 ist das Maß für die Qualität der Vorhersage in Abhängigkeit von den Parametern β und σ^2. Dabei ist bei der Bestimmung der Parameter zu beachten, dass ihre optimalen Werte für jede Sampling- und Vorhersagedauer (δ und τ) neu bestimmt werden müssen. Für die Optimierung der Parameter verschiedener Konfigurationen von δ und τ wurde ein *Hill-Climbing-Algorithmus* implementiert. Alle Parameteroptimierungen mit Hilfe dieses Verfahrens wurden *off-line*, also durch die Verwendung der Benutzerprofile, anstatt von Echtzeittrackerdaten durchgeführt, da dies die einzige Möglichkeit darstellt, vergleichbare Werte zu erhalten. Das Hill-Climbing funktioniert nach dem einfachen Prinzip: “Wähle den

Weg, der nach oben führt". "Oben" beschreibt im Fall der Parameterbestimmung den kleinsten Fehler. Der Hill-Climbing-Algorithmus lässt sich in seiner Anwendung zur Bestimmung der beiden Parameter β und σ^2 kurz als Pseudocode darstellen:

```
Solange das Minimum nicht gefunden ist {

 bestimme den Fehler

 erhöhe β um einen bestimmten Betrag
 wenn der Fehler sich verringert hat, so übernehme neues β
 ansonsten verringere β um einen bestimmten Betrag
 wenn der Fehler sich verringert hat, so übernehme verringertes β

 erhöhe σ^2 um einen bestimmten Betrag
 wenn der Fehler sich verringert hat, so übernehme neues σ^2
 ansonsten verringere σ^2 um einen bestimmten Betrag
 wenn der Fehler sich verringert hat, so übernehme verringertes σ^2

 wenn keine Verbesserung mehr eintritt, so ist das Minimum gefunden
}
```

Der Algorithmus bricht also ab, wenn in einer Iteration keine Verbesserung mehr gefunden werden kann. Da der Hill-Climbing-Algorithmus immer nur in der Umgebung des letzten Wertes nach einer Optimierung sucht, ergibt sich ein Nachteil, der bei der Parametersuche Beachtung finden muss. Es ist nicht auszuschließen, dass der Algorithmus auf einem lokalen Optimum sozusagen *hängenbleibt* und das globale Optimum der Parameter nicht gefunden wird. Um diesem Effekt entgegenzuwirken, wurden alle Bestimmungen der Parameter mehrfach und mit verschiedenen Ausgangswerten durchgeführt. Für β wurde 1 als untere und 100 als obere Schranke gewählt. Die Werte von σ müssen größer als 0.01 und kleiner als 1 sein. Der Betrag um den β während der Iteration des Hill-Climbing verändert wird, wurde auf 0.001 festgelegt. Bei σ^2 beträgt dieser Wert 0.01. Man könnte auch kleinere Werte benutzen um genauere Bestimmungen der Parameter zu erreichen. Allerdings verlängert sich die Laufzeit des Algorithmus dadurch um ein Vielfaches und da bei der on-line-Benutzung des Kalman-Filters ohnehin mit kleinen Schwankungen im Timing zu rechnen ist, sind Bestimmungen der vierten oder fünften Nachkommastelle nicht sinnvoll.
Die Logdateien des Hill-Climbing sind auf der CD im Verzeichnis *hilllog* zu finden.

Der in den Tabellen dargestellt Fehler wird nach Gleichung 51 errechnet. Um verschiedene Samplingraten betrachten und vergleichen zu können, wurden die Profile heruntergerech-

net, d.h. bei einer Schrittweite von δ=50 ms würde jeder fünfte Wert benutzt. Damit die Fehler vergleichbar sind und vor allem damit eine Einschätzung über die Verwendbarkeit der Schätzung in der virtuellen Umgebung möglich ist, wird jedoch nicht der absolute Fehler angezeigt. Stattdessen zeigt der Hill-Climbing-Algorithmus den durchschnittlichen Fehler pro Quaternionenkomponente an. Durch Probieren konnte herausgefunden werden dass, ein Fehler von mehr als etwa 0,07 pro Quaternionenkomponente die Vorhersage so ungenau macht, dass es besser ist, ohne Kalman-Filter zu agieren.

	β	σ^2	Fehler
Profil 1	9.468594	0.120000	0.00696
Profil 2	9.482599	0.120000	0.00733
Profil 3	9.452587	0.110000	0.00800
Profil 4	9.498606	0.110000	0.00882
Profil 5	9.511611	0.080000	0.00798

Tabelle 3: Durch Hill-Climbing optimierte Werte von β und σ^2 für eine Vorhersagezeit von $\tau = 1\delta$ und δ=10 ms

	β	σ^2	Fehler
Profil 1	9.283519	0.060000	0.00953
Profil 2	9.341542	0.070000	0.00947
Profil 3	9.254507	0.060000	0.01109
Profil 4	9.294523	0.060000	0.01268
Profil 5	9.211490	0.060000	0.01327

Tabelle 4: Durch Hill-Climbing optimierte Werte von β und σ^2 für eine Vorhersagezeit von $\tau = 2\delta$ und δ=10 ms

In den Beträgen der Fehler lassen sich einige Gesetzmäßigkeiten finden, die in der Literatur zur Bewegungsvorhersage [AWB01] [Azu95] [LSG91] und in allgemeinen Beschreibungen des Kalman-Filters [Kal60] nachzulesen sind. So ist erkennbar, dass eine höhere Samplingfrequenz den Gesamtfehler deutlich reduziert (vgl. Fehler in Tabellen 3 und 5). Genauso wirkt sich eine möglichst geringe Vorhersagedauer positiv auf die Qualität des Ergebnisses aus, wie im Vergleich der Fehler in den Tabellen 3 und 4 zusehen ist. Die Samplingfrequenz hat jedoch die größte Auswirkung auf den Filter. Aus diesem Grund ist einer Vorhersage,

	β	σ^2	Fehler
Profil 1	16.127163	0.080000	0.02955
Profil 2	16.090183	0.060000	0.03228
Profil 3	16.206120	0.060000	0.03202
Profil 4	16.319057	0.060000	0.03728
Profil 5	16.214115	0.060000	0.03593

Tabelle 5: Durch Hill-Climbing optimierte Werte von β und σ^2 für eine Vorhersagezeit von $\tau = 1\delta$ und δ=20 ms

basierend auf einer hohen Samplingfrequenz mit größerer Vorhersagedauer, der Vorzug vor einer niedrigeren Samplingfrequenz mit kleiner Vorhersagedauer zu geben.

Es ist also für die Benutzung des Filters anzustreben, dass eine möglichst hohe Samplingfrequenz gewählt wird und möglichst nur ein Schritt ($\tau = \delta$) vorhersagt werden muss. Diese Abhängigkeit lässt sich auch mit bloßem Auge erkennen. Die Abbildungen 34 und 35 zeigen den identischen Abschnitt der y-Komponente von Profil 1 mit einer Schrittweite von 10 und 20 ms und einer Vorhersage von jeweils einem Schritt unter Benutzung der vorher bestimmten optimalen Werte für die Parameter σ^2 und β. Dabei ist zu sehen, dass die Kurve der Vorhersage bei der geringeren Samplingperiode deutlich besser mit der Messung übereinstimmt. Besonders auffällig ist der Unterschied der Vorhersage an Wendepunkten der Graphen. Dies leuchtet in soweit ein, als dass der Kalman-Filter mit einer höheren Frequenz die Vorhersage während der Rekursion schneller korrigieren kann und deshalb der Gesamtfehler geringer bleibt. Auch der Betrag des Vorhersagefehlers wird bei linearen Bewegungen des Kopfes nicht vergrößert, sondern die Wendepunkte der Bewegung werden bei zunehmendem Samplingabstand und größerer Vorhersagedauer zum Ursprung des größten Fehlers.

Aus den Werten in den Tabellen 3, 4 und 5 lässt sich auch erkennen, dass die optimalen Werte für β und σ^2 für verschiedene Profile unterschiedlich sind, auch wenn die gleiche Vorhersagedauer und Frequenz benutzt wird. Um Azumas [Azu95] Analogie der Bewegungsvorhersage mit dem Autofahren (Kapitel 2.8) noch einmal zu benutzen, so kann man sagen, dass jedes Profil eine mehr oder weniger verwundene Straße darstellt, deren optimale Vorhersage jedesmal andere Parameter benötigt. Daraus folgt, dass keine allgemeingültigen optimalen Werte für β und σ^2 existieren. Dennoch bilden diese *off-line* bestimmten Parameter zwangsläufig die Grundlage für die on-line-Benutzung des Filters. Dazu werden die erstellten Profile (Kapitel 4.2) als repräsentativer Querschnitt aller Bewegungen angenommen und das Mittel der optimalen Werte für den on-line-Filter benutzt.

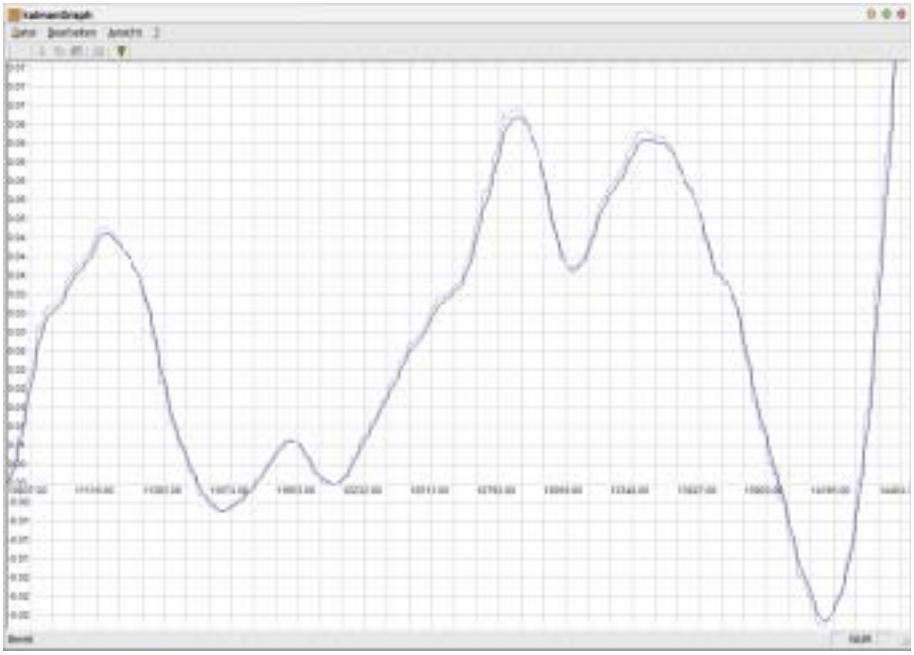

Abbildung 34: Gemessene (schwarz) und Kalman-Kurve (blau) für die y-Komponente des Quaternions bei $\delta = 10ms$ und $\tau = 10ms$

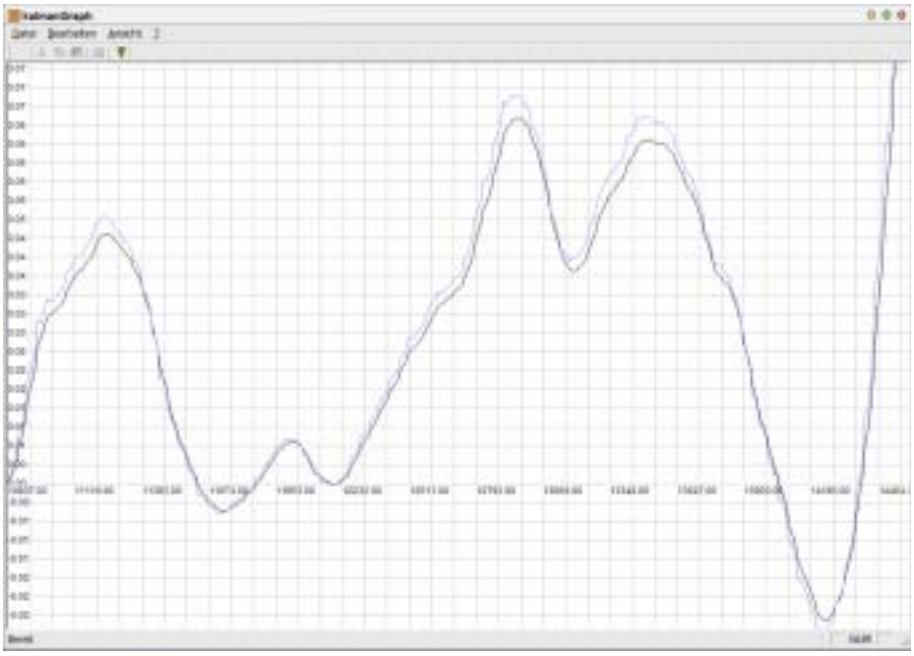

Abbildung 35: Gemessene (schwarz) und Kalman-Kurve (blau) für die y-Komponente des Quaternions bei $\delta = 20ms$ und $\tau = 20ms$

Beim off-line Test arbeitet der Vorhersagemechanismus mit einem annehmbaren Fehler, solange die Samplingrate 20 ms nicht übersteigt und nicht mehr als zwei Schritte vorhergesagt werden müssen. Die Parametrisierung für die 20 ms und einen Schritt Voraussberechnung wurde nun in den on-line Filter der Anwendung übertragen.

6.2. On-line Test

Nachdem nun die Parameter für den Kalman-Filter bestimmt sind, kann der Algorithmus auch on-line benutzt werden. In der Umgebung wird eine Samplingfrequenz von 50 Hz, also 20 Millisekunden Schrittweite benutzt. Da der end-to-end system delay bei der verwendeten Konfiguration und dargestellten Umgebung nicht besonders hoch ausfällt, reicht es aus, dass der Kalman-Filter einen Schritt voraussagt. Eine Bewertung des on-line Filters ist sehr schwierig. Da das verwendete HMD nicht vom Typ see-through (Kapitel 2.2.1) ist, hat man keine Vergleichsobjekte, die als Referenz für die Verbessung der Latenz dienen könnten. Um eine rein subjektive Bewertung durchzuführen, wurden die Probanden, die schon für die Profilerstellung zur Verfügung standen, gebeten, sich erneut in die virtuelle Umgebung zu begeben.

Ihre durchzuführende Aufgabe war identisch mit der zur Profilerstellung (Kapitel 4.2). Es wurden lediglich mehrere Durchläufe ausgeführt, nach denen die jeweilige Person eine Bewertung der Bewegungsinteraktion abgeben musste. Natürlich wurde dabei die Umgebung in einigen Fällen mit und ohne Kalman-Filter ausgeführt. Keine der Personen beschrieb eine substanzielle Verbesserung, wenn der Filter aktiv war. Dennoch wurde in einer Vielzahl von Fällen die Bewegung in der Umgebung als angenehmer beschrieben, wenn der Filter aktiv war.

Als nächstes wurde ein Test mit dem Modell des Athener Parthenon durchgeführt. Das Modell besteht aus 8708 Flächen. Bei dieser detaillierteren Umgebung, erhöhte sich die Berechnungszeit für die Szene auf ein Vielfaches, so dass es notwendig wurde, drei Schritte vorherzusagen. Dabei steigt der Fehler der Vorhersage jedoch so stark an, dass es zu einer Eigenbewegung der Szene kommt. Der Filter verliert dadurch seinen Nutzen und wirkt eher störend. Diese Problematik ist auf verschiedene Fakten zurückzuführen.

Die dynamischen Verzögerungen sind auf einem Multitaskingsystem wie Windows XP aufgrund von Diensten und Anwendungen, die im Hintergrund ablaufen, zu stark schwankend. Diese Schwankungen bewegen sich zwar im Millisekundenbereich, reichen jedoch aus, um die zeitliche Konstanz, die für die Vorhersage von Nöten ist, zu zerstören. Eine Abhilfe dafür könnte ein schnellerer Rechner sein, der auch bei der Darstellung komplexer Szenen noch genügend Reserven für eine einigermaßen konstante Berechnungszeit bietet.

Auch die verwendete Grafikkarte stellte sich beim Test mit einer echten Szene, trotz Übertaktung, als Flaschenhals dar. Da es sich dabei um eine *GeForce*-Karte handelt, stellt diese eigentlich keine Stereofunktionalität zur Verfügung. Diese wurde erst durch die Modifikation des Treibers erreicht, indem die Grafikkarte als professionelle Quadro-Grafikkarte installiert wurde. Dazu wurde ein Tool namens *SoftQuadro4* benutzt, dass unter `http://www.nvworld.ru/docs/sq4e.html` zu finden ist. Dabei kommt es jedoch zu Performanceeinbußen, die nicht mehr zu kompensieren sind, da der Fehler des Vorhersagemechanismus´ bei zu großer Schrittweite für eine praktische Anwendung inakzeptabel wird. Die veränderte GeForce-Installation bringt es nicht im OpenGL Stereo-Mode nichteinmal auf 50 Prozent der Leistung einer Quadro-Grafikkarte der gleichen Generation.

Die Benutzung des Kalman-Filters in einer detaillierten Umgebung ist deshalb auf dem Entwicklungssystem von geringem bis gar keinem Nutzen. Damit der Filter mit einer ansprechenden Szene benutzt werden kann ist ein schnelleres System mit besserer Grafikhardware notwendig. Das Entwicklungssystem, dass zugegebenermaßen nicht einmal mehr im Mittelfeld der auf dem Markt verfügbaren Systeme anzusiedeln ist, reicht für eine echte Anwendung leider nicht aus. Außerdem konnte aufgrund der vielfältigen zu bearbeitenden Problematik nicht ausreichend Zeit auf die Optimierung der Darstellung verwendet werden. Um die Benutzbarkeit der Anwendung letztlich beurteilen zu können, müssen noch weitere Tests auf anderen Hardwarekonfigurationen durchgeführt werden.

7. Mögliche Verbesserungen und Ausblick

Um eine immersive Umgebung zu erschaffen, benötigt man Lösungen in einee Vielzahl von Problemkreisen. Genauso vielfältig sind die denkbaren Verbesserungsmöglichkeiten. Als erstes ist dabei sicher die allgemeine Verbesserung der Hardware zu nennen. Schnellere Prozessoren und Grafiksysteme ermöglichen bessere Darstellungen bei höheren Frameraten und geringerer Latenz, was virtuelle Umgebungen realistischer erscheinen lässt und wie im vorangegangenen Kapitel erkannt wurde, eine anwendungsorientierte Nutzung des Vorhersagealgorithmus erst ermöglicht.

Im praktischen Test (Kap. 6.2) musste leider festgestellt werden, dass das Entwicklungssystem schlicht zu langsam ist um eine optisch annehmbare Umgebung stereoskopisch zu rendern und dabei auch noch eine zeitliche Konstanz zu wahren, die den Einsatz des Kalman-Filters sinnvoll macht. Ein schnelleres System würde Abhilfe für dieses Problem schaffen und ermöglichen, den Kalman-Filter mit zeitlichen Parametern zu nutzen die den Vorhersagefehler so klein halten, dass die Vorhersage sinnvoll ist und trotzdem eine ansprechende Umgebung darzustellen. Da ein solches System leider nicht zur Verfügung stand, ist zum jetztigen Zeitpunkt das off-line-Testen des Verfahren die einzige, wenn auch keine sinnvolle Methode, den Kalman-Filter einzusetzen.

Eine weitere Verbesserungsmöglichkeit liegt ebenfalls auf der Hardwareseite. Durch den Einsatz von Beschleunigungssensoren am HMD könnten Vorhersageverfahren, wie das von Azuma [Azu95] zum Einsatz kommen, die eine bessere Vorhersage als der Algorithmus von Liang [LSG91] versprechen.

Ebenso sind noch erheblich Verbesserungen beim Rendering der Szenen möglich. Zwar wurde durch den Einsatz von vorkompilierten OpenGL-Displaylisten schon eine erste Geschwindigkeitssteigerung erreicht, dennoch sind Optimierung zweifelsfrei möglich und auch notwendig, damit das hier erstellte System auch auf Rechner der Leistungsklasse des Entwicklungssystems sinnvoll zum Einsatz kommen kann.

Eine interessante Möglichkeit zur Optimierung der Grafikdarstellung könnte auch nVidia' kürzlich eingeführte Grafikprogrammiersprache *cG* sein, da sie laut nVidia auf den NV25-Chips, auf der die hier benutzte GeForce4 beruht, eine deutlich bessere Performance bietet als OpenGL. Ob cG eine Stereounterstützung ohne Treibermodifikationen auf den GeForce-Karten bietet ist jedoch ungewiss.

A. Pflichenheft für 3D-Applikation

Die Anwendung soll eine prototypische Implementierung einer dreidimensionalen Umgebung sein. Dabei werden folgende Anforderungen an das System definiert:

- Die Applikation soll das Laden von Objekten im 3D-Studio MAX™Format .3ds unterstützen um verschiedene Umgebungen bereitzustellen.
- Die Ausgabe soll auf dem HMD vom Typ Cybermind hi-Res900™3D erfolgen (Kapitel 4.1).
- Die Orientierung im Raum wird durch den im HMD vorhandenen Tracker InterTrax^2 bestimmt (Kapitel 4.1.3).
- Bei der Darstellung der Umgebung soll die korrekt “off-axis”-Methode zur Stereoprojektion zum Einsatz kommen (Kapitel 2.6.3).
- Die Bewegung im Raum soll über die Tastatur gesteuert werden, da das verwendete HMD ohnehin kabelgebunden ist.
- Die bei der Bewegung des Kopfes entstehende Latenz soll durch den Einsatz des Verfahrens von Liang [LSG91] kompensiert werden.

B. Die visuelle Testumgebung

Neben den im Kapitel 5 beschriebenen Softwarekomponenten wurde zur Erstellung dieser Arbeit eine weitere Anwendung implementiert. Dabei handelt es sich um eine Windows-Anwendung auf Basis der *Microsoft Foundation Classes (MFC)*. Sie dient zur Visualisierung von Quaternionenanteilen und ermöglicht es sowohl den originalen Graph als auch den Vorhersagegraph darzustellen. Die Abbildungen 34 und 35 im Kapitel 6 sind Screenshots dieser Anwendung. Die Graphen können nach belieben verschoben und gezoomt werden. Dieses Tool war besondern in der Anfangsphase der Kalman-Filter Entwicklung eine große Hilfe, da sie die Möglichkeit bietet die Qualität der Filtervorhersage visuell einzuschätzen um ein Gefühl für die Größenordnungen der Parameter zu erhalten.

Die Anwendung kann die erstellten Benutzerprofile laden und eine Vorhersage mit frei wählbaren Parametern durchführen. Dazu bietet sie ein Fenster in dem der zu visualisierende Graph ausgewählt werden kann und alle Parameter des Vorhersagealgorithmus' manuell eingestellt werden können (Abb. B). Ein Button löst die Vorhersage aus und blendet den Graph der Vorhersage im Hauptfenster ein.

Abbildung 36: Kalman-Fenster

Außerdem lassen sich mit der Anwendung auch noch Benutzerprofile in beliebiger Samplingfrequenz erzeugen. Das geschiet zwar blind, war aber bei den ersten Gehversuchen mit dem HMD von Bedeutung, da anfangs Komplikationen bei der Bestimmung der Orientierung, in Form von plötzlichen Richtungswechseln, auftraten. Des weitern wurde diese Funktion benutzt um festzustellen, ob der Tracker *driftet* und ob die von InterSense propagierte *zero noise*, also die völlige Konstanz der Orientierungswerte, wenn dar Tracker in Ruhe ist, der Realität entspricht. Das HMD mit dem InterTrax^2 wurde fixiert und der Recorder nahm über einen Zeitraum von 30 Minuten die Orientierung in eine Datei auf. Dabei konnte festgestellt werden, dass tatsächlich keinerlei Messungenauigkeiten auftreten wenn der Tracker in Ruhe verbleibt.

Abbildung 37: Das Aufnahmefenster

Die Anwendung nutzt genauso wie die virtuelle Umgebung die Klassen des Kalman-Filters um die Vorhersage zu treffen und diente im Zuge der schriftlichen Erstellung dieser Arbeit nur noch zum Erzeugen der erwähnten Screenshots.

Der Vollständigkeit halber ist in Abbildung 38 das UML-Diagramm der Klassen der Testumgebung dargestellt.

Abbildung 38: UML-Diagramm der Klassen der Anwendung kalmanGraph

C. Literaturverzeichnis

[AB95] AZUMA, Ronald ; BISHOP, Gary: A Frequency-Domain Analysis of Head-Motion Prediction. In: *Computer Graphics* 29 (1995), Nr. Annual Conference Series, S. 401–408

[AWB01] ALLEN, B. D. ; WELCH, Greg ; BISHOP, Gary. *Tracking: Beyond 15 minutes of Thought*. ACM SIGGRAPH 2001 Course Notes. 2001

[Azu95] AZUMA, Ronald T.: Predictive Tracking for Augmented Reality. 1995 (TR95-007). – Forschungsbericht. no note

[Beu02] BEUTELSPACHER, Albrecht: *Lineare Algebra*. 5. Auflage. Friedr. Vieweg und Sohn Verlagsgestellschaft mbH, Braunschweig/Wiesbaden, 2002

[Dee92] DEERING, Michael: High Resolution Virtual Reality. In: *Proceedings of SIGGRAPH '92*, 1992, S. 195–202

[Dev02] DEVELOPERS@SOURCEFORGE. *The 3D Studio File Format Library Version 1.2.0*. http://lib3ds.sourceforge.net. 2002

[FDF+94] FOLEY, James D. ; VAN DAM, Andries ; FEINER, Steven K. ; HUGHES, John F. ; PHILLIPS, Richard L.: *Grundlagen der Computergrafik*. 1. Auflage. Addison-Wesley (Deutschland) GmbH, 1994

[Kal60] KALMAN, Rudolph E.: A New Approach to Linear Filtering and Prediction Problems. In: *Journal of Basic Engineering* (1960), S. 35–45

[Kre02] KRENGEL, Ulrich: *Einführung in die Wahrscheinlichkeitstheorie und Statistik*. 6. Auflage. Friedr. Vieweg und Sohn Verlagsgestellschaft mbH, Braunschweig/Wiesbaden, 2002

[LS] LEE, Jehee ; SHIN, Sung Y. *General Construction of Time-Domain Filters for Orientation Data*

[LSG91] LIANG, J. ; SHAW, C. ; GREEN, M.: On Temporal-Spatial Realism in the Virtual Reality Environment. In: *Proceedings ACM UIST'91 4th Annual ACM Symposium on User Interface Software and Technology* (1991), S. 19–25

[May79] MAYBECK, Peter S.: *Mathematics in Science and Engineering*. Bd. 141: *Stochastic models, estimation, and control*. 1. Auflage. 1979

[Min95] MINE, Mark R.: Virtual Environment Interaction Techniques. 1995 (TR95-018). – Forschungsbericht

[MJ94] MIN, P. ; JENSE, H. *Interactive stereoscopy optimization for head-mounted displays*. 1994

[OCMB95] OLANO, Marc ; COHEN, Jonathan D. ; MINE, Mark R. ; BISHOP, Gary: Combatting Rendering Latency. In: *Symposium on Interactive 3D Graphics*, 1995, S. 19–24, 204

[PKP99] PRINZ, Peter ; KIRCH-PRINZ, Ulla: *C++ Lernen und professionell anwenden*. 1. Ausgabe. MITP-Verlag GmbH, 1999

[SDB00] SWINDELLS, Colin ; DILL, John ; BOOTH, Kellogg S.: System lag tests for augmented and virtual environments. In: *UIST*, 2000, S. 161–170

[SGLB] SCHUTTER, Joris D. ; GEETER, Jan D. ; LEFEBVRE, Tine ; BRUYNINCKX, Herman. *Kalman Filters: A Tutorial*

[Sho85] SHOEMAKER, K.: Animating rotation with quaternion curves. In: *Computer Graphics 19* (1985), Juli, S. 245–254

[ST95] SCHMIDT, R.F. ; THEWS, G.: *Physiologie des Menschen*. 26. Auflage. Springer Verlag Berlin, 1995

[Sut68] SUTHERLAND, Ivan: A Head-Mounted Three Dimensional Display. In: *Fall Joint Computer Conference, AFIPS Conference Proceedings 33* (1968), S. 757–764

[Swa99] SWANKE, John E.: *Visual C++ MFC Programming by Example*. 1. Auflage. R&D Books, 1999

[VJ00] VIOLA, Joseph J. L. ; JR. *A Discussion of cybersickness in virtual environments*. 2000

[WB94] WARE, Colin ; BALAKRISHNAN, Ravin: Reaching for objects in VR displays: lag and frame rate. In: *ACM Transactions on Computer-Human Interaction* 1 (1994), Nr. 4, S. 331–356

[WB02] WELCH, Greg ; BISHOP, Gary. *An Introduction to the Kalman-Filter*. 2002

[WGP98] WARE, C. ; GOBRECHT, C. ; PATON, M. *Dynamic Adjustment of Stereo Display Parameters*. 1998

[Wie49] WIENER, Norbert: *Extrapolation, Interpolation, and Smoothing of Stationary Time Series, with Engineering Applications*. 1. Auflage. John Wiley & Sons, 1949

[WND97] WOO, Mason ; NEIDER, Jackie ; DAVIS, Tom: *OpenGL Programming Guide. The Official Guide to Learning OpenGL, Version 1.1*. Zweite Auflage. Addison-Wesley Publishing Company, 1997

[WND99] WOO, Mason ; NEIDER, Jackie ; DAVIS, Tom: *OpenGL Programming Guide. The Official Guide to Learning OpenGL, Version 1.2*. Erste Auflage. Addison-Wesley Publishing Company, 1999

[WSO99] WHITE, David ; SCRIPBNER, Kenn ; OLAFSEN, Eugene: *MFC Programming with Visual C++ 6 Unleashed*. 1. Auflage. Sams Publishing, 1999

[Wun] WUNSCH, Saad C. *Predictive Head Tracking for Virtual Reality*

D. Abbildungsverzeichnis

E. Tabellenverzeichnis

Tabellenverzeichnis

F. Glossar

3DS *3D Studio MAX™*: 3D-Animationssoftware der Firma *discreet*

API *Application Programming Interface*: Programmier- und Anwendungsschnittstelle

AR *Augmented Reality*: Unterbegriff der VR, beschreibt Fusion von realer und virtueller Welt

DOF *Degree of Freedom*: Anzahl der Achsen auf denen eine Bewegung im Raum stattfinden kann

CAVE *CAVE Automatic Virtual Environment*: Kubischer Raum, der von Projektionsflächen begrentzt wird

EKF *Extended Kalman Filter*: Verfahren zur Schätzung nicht-linearer Systeme

FOV *Field of View*: Das überblickbare Gesichtsfeld

GLUT *OpenGL Utility Toolkit*: Systemunabhängige Programmierbibiliothek für OpenGL

HID *Human Interface Device*: Protokoll zur Anbindung von Eingabegeräten via USB

HMD *Head-Mounted-Display*: siehe Abbildung 3, Seite 7

KF *Kalman Filter*: Verfahren zur Schätzung linearer Systeme

LCD *Liquid Crystal Display*: Flüssigkristallanzeige die in HMDs Verwendung findet

OpenGL *Open Graphics Library*: cross-platform API für 3D-Rendering

SDK *Software Development Kit*: Zusammenstellung von Werkzeugen, Dokumentationen und Bibliotheken zur Softwareentwicklung

UML *Unified Modeling Language*: Sprache zur Beschreibung von Softwaresystemen

USB *Universal Serial Bus*: Bussystem zum Anschließer verschiedenster Perepherie an einen Computer

VR *Virtual Reality*: virtuelle Realität

G. Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfmittel verfaßt habe. Die Arbeit wurde in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Berlin, 12. Mai 2003

Daniel Grabert