UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

COMPUTERVISUALISTIK

Erweiterung der Wii-Interaktion mit Bewegungssteuerung durch 6 Freiheitsgrade und Demonstration an Beispielapplikationen

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Computervisualistik

vorgelegt von
Fabian Klebe

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Dominik Grüntjens (Institut für Computervisualistik, AG Computergraphik)

Koblenz, im März 2011

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....
(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Institut für Computervisualistik
AG Computergraphik
Prof. Dr. Stefan Müller
Postfach 20 16 02
56 016 Koblenz
Tel.: 0261-287-2727
Fax: 0261-287-2735
E-Mail: stefanm@uni-koblenz.de

UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Aufgabenstellung für die Diplomarbeit

Fabian Klebe

(Matr.-Nr. 204 210 210)

Thema: Erweiterung der WII-Interaktion mit Bewegungssteuerung durch 6 Freiheitsgrade und Demonstration an Beispielapplikation

Die Spielkonsole Nintendo Wii bietet mit dem Wii Remote Controller eine neuartige Bewegungssteuerung für Konsolenspiele. Mit Hilfe von Gyrosensoren in dem Zusatzadapter Wii Motion Plus ist eine Steuerung mit 3 Freiheitsgraden für Rotationen im Raum gewährleistet. Für eine realistische 1:1 Bewegungssteuerung würden aber 3 zusätzliche Freiheitsgrade für Translationen in drei Koordinatenachsen benötigt. In aktuellen Spielen für Nintendo Wii werden diese nicht unterstützt, da Translationen durch Beschleunigungssensoren unzureichend umzusetzen sind.

Ziel dieser Arbeit ist es ein System zu implementieren, das für den Wii Remote Controller eine 1:1 Bewegungssteuerung für alle 6 Freiheitsgrade ermöglicht. Dabei sollen die Rotationen durch die Sensoren des Wii Motion Plus Adapters, die Translationen hingegen durch Stereotracking mit Hilfe der Infrarotkameras zweier zusätzlicher Wii Remotes erreicht werden.

Ein solches System ergibt interessante Anwendungsmöglichkeiten. Insbesondere für Spiele könnte so eine verstärkte Immersion des Spielers erreicht werden, da jede Bewegung direkt im Spiel sichtbar wäre.

Anhand von verschiedenen Beispielapplikationen sollen die Interaktionsmöglichkeiten demonstriert werden.

Die Genauigkeit der Steuerung soll dabei evaluiert werden und mit der herkömmlichen 3 Achsensteuerung im Bezug auf Immersion, Komplexität, Eignung für Spiele und Benutzerfreundlichkeit verglichen werden.

Für das Infrarot Stereotracking soll eine geeignete technische Lösung gefunden werden, z.B. durch ein IR-Array oder am Controller angebrachten Infrarotdioden. Beide Lösungen können verglichen werden.

Schwerpunkte dieser Arbeit sind:

1. Einarbeitung in Ansteuerung der Wiimote
2. Design von Beispielapplikationen
3. Konzeption und Implementierung einer 1:1 Bewegungssteuerung durch Wiimotion Plus und 3D-Infrarot-Stereotracking
4. Technische Lösung für IR-Stereotracking
5. Konzeption und Implementierung der Beispielapplikationen
6. Vergleich und Evaluation der Ergebnisse
7. Dokumentation der Ergebnisse

Koblenz, den 22. August 2010

- Prof. Dr. Stefan Müller-

1 Einführung

Durch die Einführung der Spielkonsole Wii des Konsolen und Spieleherstellers Nintendo gegen Ende 2006 [22] hat sich die Art und Weise, wie Computerspieler mit einem Spiel interagieren können, stark erweitert. Zwar ist auch heute noch die Metapher des klassischen Spielecontrollers mit Tasten und Bewegungsstick vorherrschend, aber der Markt hat das von Nintendo angebotene Konzept der Wiimote, eines Controllers, der auf Bewegungen des Spielers reagiert, alle Erwartungen übertreffend angenommen. Auf einmal war es möglich, ein Golfspiel nicht mehr nur durch das Drücken von Knöpfen eines Gamepads zu steuern, sondern durch das Nachahmen eines tatsächlichen Golfschlags vor dem Fernseher. Aufgrund der hohen Verkaufszahlen ist Nintendo in der aktuellen Konsolengeneration mit der Wii Marktführer vor Sony und Microsoft, was wohl größtenteils der neuartigen Steuerung zu verdanken ist. Mit der Konsole hat sich nicht nur das Konzept der Steuerung geändert, auch die Art der Spiele, die damit möglich sind, hat sich erweitert. Auch bereits bekannte Spielkonzepte wurden mehr oder weniger erfolgreich für die Bewegungssteuerung umgesetzt und erlaubten manchmal völlig neue Gameplayelemente umzusetzen. War beispielsweise der Spieler beim Fußballspiel Pro Evolution Soccer mit dem klassischen Controller darauf beschränkt, stets zu einem Zeitpunkt nur einen Spieler einer Mannschaft zu steuern, so bot die Wii Steuerung auch die Kontrolle über mehrere Spieler an und ermöglichte durch die Pointerfunktion (vgl. Abschnitt 6.2.1) das Passen und Schießen an jede beliebige Stelle auf dem Feld. Hier zeigt sich wieder deutlich, dass das Medium oftmals die Art der Inhalte, also in diesem Fall die Spiele die dafür entwickelt werden, in gewisser Weise bestimmt.

Das im Bezug auf die Steuerung wohl erfolgreichste Wii Spiel war aber Wii Sports, das bereits zur Markteinführung zumindest in Europa jeder Konsole beigelegt wurde. Das Spiel war als Demonstration gedacht, um zu zeigen, welche Bewegungen und Gameplaymetaphern mit der Wiimote möglich sind. Dabei wurden verschiedene Sportarten mit der Steuerung simuliert, unter anderem Tennis, Bowling, Golf, Baseball und Boxen. Zu Demonstrationszwecken eigneten sich diese Art von Spielen sehr gut und auch der Spielspaß schien enorm zu sein.

Zu Beginn ließ sich mit der Wiimote eine 1:1 Umsetzung von Rotationen allerdings nur in begrenztem Umfang umsetzen. Die Beschleunigungssensoren wurden vielmehr dazu eingesetzt, eine ungefähre Aussage über Kraft und Richtung des Schlags treffen zu können. Die Möglichkeiten der Wiimote zu Beginn des Konsolenzyklus, möchte ich anhand des Minispiels Tennis in Wii Sports aufzeigen. Bei diesem Spiel entschied beispielsweise die Kraft des Schlags darüber wie fest der Tennisball geschlagen wurde.

Die Richtung des Schlags, also ob der Ball cross[1] oder longline[2] geschlagen würde entschied sich aber lediglich durch Timing. Traf man relativ früh, so flog der Ball cross, traf man später, so flog er longline. Zusätzlich konnte die Wiimote die ungefähre Richtung des Schlags feststellen, und so konnten auch hohe, tiefe, oder geschnittene Bälle geschlagen werden. Letztendlich wurde also weder eine ausreichende Simulation der Translation des Schlägers, noch der Rotation umgesetzt, da dies mit den technischen Gegebenheiten der Wiimote an sich gar nicht möglich gewesen wäre. Die Idee, zumindest die Position der Wiimote im Raum allein mit Hilfe der Beschleunigungssensoren zu ermitteln, scheint auf den ersten Blick verführerisch. Letztendlich sind diese aber zu ungenau. Bereits kleine Fehler der Messung würden sich im Lauf der Zeit akkumulieren. Anhand der Beschleunigung kann nicht exakt bestimmt werden, ob die Wiimote gerade in Bewegung ist oder stillsteht. Selbst wenn sie in Bewegung ist, muss noch keine Beschleunigung vorherrschen. Man könnte die Position nur ermitteln, wenn alle Beschleunigungen im Laufe der Zeit exakt stimmen würden, dies ist aber beinahe umöglich. Erschwerend hinzu kommt, dass Beschleunigungen selbst dann registriert werden, wenn die Position der Wiimote unverändert bleibt, sie aber auf der Stelle rotiert wird.

Abbildung 1: Minispiel „Tennis“ von Wii Sports [24]

Als Nintendo im Jahr 2009 allerdings den Zusatzadapter MotionPlus veröffentlichte, war es mit dessen Hilfe schließlich möglich, eine wirkliche 1:1-Umsetzung der Rotationen, die der Spieler auf die Wiimote ausübt, umzusetzen. Um MotionPlus zu demonstrieren, wurde damals der Nachfolger von Wii Sports, nämlich Wii Sports Resort, veröffentlicht. Allein die Tatsache, dass hier 12 Minispiele umgesetzt wurden, zeigt schon, dass mit Hilfe der Rotation nun weitaus mehr Applikationen simuliert werden konnten. Ich möchte dies an dem Beispiel der Schwertsimulation in Wii Sports Resort aufzeigen. Hätte man eine Schwertsimulation ohne MotionPlus durchführen wollen, sähe das Ergebnis wahrscheinlich so aus, das die Wiimote lediglich ca. 8 verschiedene Richtungen eines Schlags unterscheiden wür-

[1] schraeg
[2] gerade

de, und jeder Richtung nur eine einzige Schlaganimation zuweisen würde. Höchstens die Intensität des Schlags könnte noch zusätzlich mit Hilfe der Beschleunigungssensoren simuliert werden. Das Spiel würde zwar funktionieren, aber der Spieler würde zu keinem Zeitpunkt das Gefühl haben, ein Schwert zu steuern. Die Schlagmöglichkeiten würden sich schnell wiederholen und die Motivation würde nach kurzer Zeit abflauen.

Mit Hilfe von MotionPlus konnte aber nun sehr genau jede vom Spieler durchgeführte Rotation der Wiimote auf das Schwert im Spiel übertragen werden. So waren die Richtungen, in die das Schwert geführt werden konnte natürlich beinahe unbegrenzt. Es war möglich innerhalb eines Schlags die Richtung zu ändern und so mit dem Schwert zum Beispiel eine Kreisbewegung in der Luft durchzuführen. Man konnte auch beeinflussen, ob man eine weit ausgeholte Bewegung oder nur eine sehr geringe machte. Letztendlich hatte der Spieler zumindest anfangs durchaus das Gefühl ein wirkliches Schwert zu steuern. Nach einiger Zeit fiel den meisten Spielern aber unweigerlich auf, dass lediglich die Rotationen der Wiimote übertragen wurde. Demnach machte es keinen Unterschied, ob man den Controller nur aus dem Handgelenk heraus, oder mit einer richtigen Armbewegung rotierte.

Um aber die Bewegung, die der Spieler mit dem Controller ausführt, wirklich exakt im Spiel zu übertragen, müsste zusätzlich zur Rotation auch noch die Translation der Wiimote im Raum mit in die Simulation einfließen. Dies konnte bis heute noch kein Spiel für die Konsole umsetzen. Die Kombination von Orientierung und Position eines Objekts wird auch als Pose oder Lage bezeichnet [26]. Die Orientierung ergibt sich dabei aus den Rotationen, die Position aus den Translationen, die das Objekt erfährt.

1.1 Die Fragestellung der Diplomarbeit

Im Rahmen meiner Diplomarbeit möchte ich eine Möglichkeit bieten, neben der Rotation auch die Translation der Wiimote im Raum virtuell darzustellen, um somit die Lage des Controllers im dreidimensionalen Raum virtuell abzubilden. Dazu soll die Position des Controllers, der vom Spieler bewegt wird mittels Stereotracking von den Infrarotkameras zweier weiterer Wiimotes erfasst werden. Der Einsatz dieser 1:1 Bewegungssteuerung soll an verschiedenen Beispielapplikationen demonstriert werden. Eine solche Rotations- und Translationssteuerung[3]Die Rotations- und Translationssteuerung wird im Weiteren häufig als R+T-Steuerung bezeichnet, die Rotationssteuerung als R-Steuerung.) könnte die Immersion, also das Gefühl des Spielers im Spielgeschehen einbezogen zu sein, erheblich steigern. Au-

[3](

ßerdem könnte der Spielverlauf durch die zusätzlichen Freiheiten, die die Steuerung gewährt, bereichert werden. Auch könnten diverse Spiele überhaupt erst mit solch einer Steuerung ermöglicht werden. Es stellt sich aber die Frage, ob der Einstieg ins Spiel und der Schwierigkeitsgrad möglicherweise durch die Komplexität der Steuerung erschwert wird. In Abschnitt 4.3 werden deshalb verschiedene Hypothesen im Bezug auf die Beeinflussung der Qualität des Spiels durch die R+T-Steuerung aufgestellt, die dann im Rahmen von Benutzertests evaluiert werden sollen.

1.2 Motivation

Die Motivation, dieses Projekt einer Erweiterung der Wii-Interaktion zu beginnen, kann ich auf verschiedene Ursachen zurückführen. Das Erscheinen der Wii-Konsole versprach auf einmal ganz neue Möglichkeiten der Interaktion mit Konsolenspielen. Die Bewegungssteuerung bot intuitive Interaktionsmetaphern, die oftmals der Realität angepasst sein konnten. Der Wurf einer Bowlingkugel ließ sich nun in einem Konsolenspiel durch eine Bewegung vollführen, ähnlich als hätte man eine echte Bowlingkugel in der Hand.

Ob die Gesten des Spielers wirklich naturgetreu übertragen werden konnten, war dabei natürlich eine wichtige Frage. Es zeigte sich, wie oben beschrieben, dass vor Erscheinen von MotionPlus von wirklich naturgetreuer Übertragung keine Rede sein konnte, und oftmals die im Spiel gezeigten Bewegungen eher einem starren Schema ähnelten. Dennoch war der Spielspaß allein durch das Ausführen der Bewegungen bereits enorm, was durch die hohen Verkaufszahlen der Konsole scheinbar bestätigt wurde.

Trotz der Einschränkungen der Wiimote zeigte aber das Spiel Baseball von Wii Sports andeutungsweise, wie sich eine naturgetreue Übertragung der Bewegungen in einem Computerspiel anfühlen könnte. Bei diesem Spiel wurden nämlich Rotationen direkt übertragen, solange der Spieler die Wiimote mit der Spitze nach oben hielt, ähnlich wie man einen Baseballschläger in der Ausgangsstellung hält. Dass diese Rotationen bereits ohne MotionPlus relativ gut übertragen werden konnten, liegt an den Schwerkraftsensoren in der Wiimote (siehe Abschnitt 6.2.1), die es ermöglichten, bei relativ langsamen Bewegungen in bestimmen Orientierungsbereichen der Wiimote deren Rotationen messen zu können. Zwar konnten damit keine komplexen 360-Grad Rotationen gemessen werden, aber der Spieler bekam bei diesem relativ geringen Bewegungsradius schon ein recht überzeugendes Gefühl der Immersion.

Mit der Veröffentlichung von MotionPlus und der Möglichkeit, Rotationen direkt ins Spielgeschehen zu übertragen, kam bei mir die Idee auf, diese

Abbildung 2: Wii Sports Baseball zeigte schon in Ansaetzen eine direkte Uebertragung der Rotationen des Controllers im Spiel [23]

Technologie selber in einem Projekt anzuwenden. Allerdings gab es lange Zeit keine Möglichkeit, MotionPlus mit den gängigen Bibliotheken wie WiimoteLib (vgl. Abschnitt 3.1) anzusprechen, weshalb ich solch ein Projekt zunächst nicht angehen konnte. Als dann jedoch das Programm Glovepie in der Version 0.43 die Unterstützung von MotionPlus versprach, und sich diese als sehr gelungen herausstellte, wollte ich ein solches Projekt wieder angehen.

Oft hatte ich aber bei Spielen, die MotionPlus unterstützten, das Gefühl, dass Rotationen allein als Interaktionsmetapher nicht ausreichen, um ein wirkliches Empfinden von Immersion beim Spieler zu erzeugen. Zwar fühlten sich die übertragenen Bewegungen zunächst sehr gut an, aber nach einiger Spielzeit bemerkte ich doch, dass eine tatsächliche Übertragung der Position der Wiimote im Raum einfach nicht gewährleistet wurde, dass also die Translation nicht übertragen wurde. Deshalb wollte ich in meinem zukünftigen Projekt eben diese Translation der Wiimote im Raum zusammen mit der durch MotionPlus unterstützten Übertragung der Rotation verknüpfen um eine wirklich detailgetreue Abbildung der Gesten des Spielers im Spiel zu erreichen.

1.3 Überblick

Kapitel 2 stellt zunächst die Geräte, die in diesem Projekt Verwendung finden vor. Dazu gehört natürlich die Wiimote und deren Infrarotkamera, sowie der Zusatzadapter MotionPlus. Kapitel 3 zeigt kurz auf, welche Projekte bereits mit der Wiimote am PC durch eine Community von Hobbyentwicklern umgesetzt wurden. Die Konzepte für die Beispielapplikationen, die letztendlich umgesetzt wurden werden in Kapitel 4 aufgezeigt. Dabei werden Hypothesen formuliert, inwieweit die Bewegungssteuerung Einfluss auf die Qualität der Applikationen nehmen könnte. Das verwendete Framework und die Umsetzung der Beispielapplikationen, inklusive

Beschreibung der wichtigsten Klassen ist in Kapitel 5 zu finden. Das wohl wichtigste Kapitel 6 behandelt die letztendliche Umsetzung der Steuerung, sowohl der Rotationen als auch des Stereotrackings, mit dem die Translationen ermöglicht werden. Die Benutzertests werden im Kapitel 7 genau erläutert. Dabei werden die Ergebnisse bezüglich der in Kapitel 4 aufgestellten Hypothesen interpretiert. Kapitel 8 gibt schließlich einen Ausblick, zu weiteren denkbaren Einsatzmöglichkeiten der freien Bewegungssteuerung, sowie ein persönliches Fazit zu der erreichten Arbeit.

2 Technische Gegebenheiten

Das folgende Kapitel stellt das technische Konzept der Wii-Konsole vor, mit besonderem Bezug auf die Geräte, die auch in diesem Projekt zum Einsatz kamen.

2.1 Die Wii Remote

Abbildung 3: Die Rotationsachsen im lokalen Koordinatensystem der Wiimote

Die Wii Remote, häufig auch mit Wiimote abgekürzt, wurde am 14. Oktober 2005 erstmals auf der Tokyo Game Show vorgestellt [29]. Ihr Design ähnelt auf den ersten Blick dem einer herkömmlichen Fernbedienung, woher auch der Name „Remote“ rührt. Daten versendet sie per Bluetooth.

Sie verfügt über eine Infrarotkamera, mit der bis zu 4 Infrarotpunkte gleichzeitig wahrgenommen werden können. Die Auflösung der Kameras beträgt zwar lediglich 128 x 96 Pixel, allerdings werden die Werte von der Wiimote interpoliert auf eine Auflösung von 1024 x 768 Pixeln. Dass die 4 verschiedenen Infrarotpunkte voneinander unterschieden werden können, also beispielsweise stets dieselbe Infrarotquelle als Punkt 1 wahrgenommen wird, wird durch Bewegungskorrespondenzen erreicht [2].

Weiterhin verfügt die Wiimote über ADXL-330-Beschleunigungssensoren für alle 3 Bewegungsachsen. Leider ist es mit diesen Sensoren alleine äußerst schwierig, genaue Angaben zur Pose der Wiimote zu treffen. Hauptsächlich können damit ungefähre Aussagen über die grobe Richtung gemacht werden, in die bewegt wurde (also hoch, runter, links, rechts, nach vorne, nach hinten). Es kann dabei aber nicht zwischen kurzen, und weitläufigen Bewegungen unterschieden werden. [18] Die Aussagen, die mit den Beschleunigungssensoren der Wiimote bezüglich der Rotation gemacht werden können, sind eingeschränkt. So ist es nur begrenzt möglich, die Orientierung im Raum abzufragen. Die Pitch-Achse kann, aufgrund der Beschleunigung, die die Schwerkraft ausübt, abgefragt werden. Auch die

Roll-Achse kann zum Teil auf diese Weise ermittelt werden, allerdings nur wenn die Wiimote waagerecht gehalten wird[4]. Diese Abfragen sind aber teilweise recht ungenau und nur bei langsamen Bewegungen möglich. Es ist allerdings unter keinen Umständen machbar, den Yaw-Winkel abzufragen, da dieser mit Hilfe der Schwerkraft nicht ermittelt werden kann.

Die Wiimote kann durch verschiedene Adapter erweitert werden, zum Beispiel durch den Zusatzcontroller Nunchuk[5], oder das bereits erwähnte MotionPlus.

2.2 Wii MotionPlus

Abbildung 4: Der MotionPlus Zusatzadapter

Abbildung 5: Links: Wiimote mit angestecktem MotionPlus Zusatzadapter. Mitte und rechts: In diese Wiimote wurde MotionPlus bereits integriert

[4]Bei waagerechter Haltung kann die Schwerkraft helfen, den Rollwinkel zu messen. Bei vertikaler Haltung ist dies allerdings nicht möglich, da die Schwerkraft dann entlang der Längsachse verläuft, um die ja bei Roll gedreht wird.

[5]Der Nunchuk-Controller findet in diesem Projekt keine Verwendung

Wii-MotionPlus ist ein Adapter der als Erweiterung an die Wiimote angesteckt werden kann. Er wurde von Nintendo in Zusammenarbeit mit AiLive[6] entwickelt. Mit diesem Gerät ist es nun möglich, die Orientierung des Controllers in allen 3 Achsen, also Pitch, Yaw und Roll in Echtzeit abzufragen. Rotationen um die x- und y-Achse werden mit Hilfe des MEMS[7]-Gyroskops IDG-600 der Firma InvenSense gemessen. Gyroskope können den Winkel und die Drehrate von Rotationen direkt messen, wobei sie auf lineare Bewegungen und Zittern der Hand nicht reagieren. Das verwendete Gyroskop in MotionPlus ist besonders tolerant gegenüber Erschütterungen, Temperatur- und Feuchtigkeitsschwankungen. [18] In einem Interview („Iwata Asks: Wii MotionPlus“ [5]) erklärten die Entwickler des Adapters, dass MotionPlus mit verschiedenen Sensoren bei verschiedenen Rotationsgeschwindigkeiten arbeitet. Der Grund war, dass die Sensoren für bestimmte Geschwindigkeiten ausgelegt sind, und nur dann auch exakt messen können.

MotionPlus misst die Rotationen der Wiimote in deren lokalen Koordinatensystem. Die Rotationsachsen heißen Neigungs[8]- , Gier- und Rollachse [28]. Im Folgenden Verlauf werden dafür stets die englischen Bezeichnungen, nämlich Pitch, Yaw und Roll verwendet. Pitch bezeichnet eine Rotation um die x-, Yaw um die y-, und Roll um die z-Achse[9]) des lokalen Koordinatensystems. Abbildung 3 zeigt die Orientierung dieser Achsen bei der Wiimote. Wie man sieht bleiben die Rotationsachsen im Bezug zur Wiimote immer gleich, auch wenn diese, vom Weltkoordinatensystem aus betrachtet, geneigt wurde, wie im rechten Bild zu erkennen ist.

2.3 Die Infrarotkamera der Wiimote und die Sensorbar

Die Infrarotkamera der Wiimote kann bis zu 4 verschiedene Infrarotpunkte gleichzeitig unterscheiden. Nimmt die Kamera allerdings mehr als 4 Punkte wahr, wird gar kein Punkt mehr verfolgt. Die Sensoren der Kamera reagieren besonders gut auf Infrarotlicht im Wellenlängenbereich von 940nm [2]. In der Kamera ist deshalb ein Filter integriert, der speziell dieses Licht durchlässt.

Die Sensorbar ist 20 cm lang [29] und wird normalerweise auf oder unter dem Fernseher platziert. Sie verfügt auf der linken und rechten Seite jeweils über 5 Infrarotdioden, die Infrarotlicht im, für den Menschen unsichtbaren, Wellenlängenbereich von 940nm aussenden. Ab ca. einem Me-

[6] AiLive stellt Software für Spielentwickler, speziell im Bereich der Künstlichen Intelligenz und Bewegungserkennung, her.

[7] Micro-Electromechanical Systems

[8] oder auch Nickachse

[9] also die Längsachse der Wiimote

ter Entfernung werden diese 5 Punkte auf beiden Seiten von der Kamera der Wiimote aber lediglich als jeweils ein Punkt wahrgenommen. So ist es möglich mit Hilfe der Sensorbar 2 Punkte in ausreichender Entfernung zu verfolgen. Abbildung 6 zeigt die Sensorbar, wie sie von einer normalen Farbbildkamera wahrgenommen wird. Da die Farbbildkamera Infrarotlicht erkennen kann, werden die Dioden auf dem Bild sichtbar gemacht, für das menschliche Auge, sind sie aber eigentlich unsichtbar.

Die Sensorbar wird hauptsächlich für eine Zeigerfunktion eingesetzt. Dabei wird mit Hilfe der beiden Infrarotpunkte bestimmt, wohin der Spieler mit dem Controller zeigt. Mit Hilfe des Winkels, den die beiden Punkte auf der Bildebene der Kamera bilden, kann zusätzlich die Rotation entlang der Längsachse der Wiimote berechnet werden[10], ein Feature das sich zum Beispiel bei Ego-Shootern durch das Neigen der Waffe zeigt. In diesem Projekt wird die Sensorbar zu Kalibrierungszwecken eingesetzt, eine Pointerfunktion ist nicht vorgesehen.

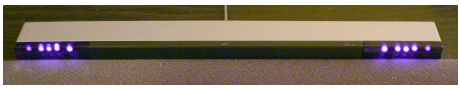

Abbildung 6: Die Sensorbar hat an jeder Seite 5 Infrarotdioden. Ab einem geringen Abstand werden diese nur noch als 2 helle Infrarotpunkte wahrgenommen

[10] Es ist zu beachten, dass die Rotation hier nicht in allen 3 Achsen berechnet werden kann

3 Überblick über Projekte mit der Wiimote am PC

Als Nintendo Wii auf den Markt kam, war die damals eingeführte Bewegungssteuerung im Konsolen und Computerspielemarkt eine absolute Neuerung. Zwar gab es bereits zuvor die technischen Möglichkeiten mit Gyrosensoren oder Beschleunigungsmessern, diese wurden aber noch nicht in realen Spielen eingesetzt. Das Konzept der Wii, konzentrierte sich voll und ganz auf den neuen Controller. Dieser wurde jeder verkauften Konsole beigelegt und eben nicht als zusätzlich zu erwerbendes Add-On angeboten. Dementsprechend groß schien das Risiko zur Markteinführung und viele Analysten und auch damalige Fans von Nintendokonsolen zweifelten daran, dass die Menschen dieses neue Konzept annehmen würden. Nintendo Wii erwies sich aber als voller Erfolg und mit der neuen Steuerung wurde auch die Zielgruppe der Käufer erweitert.

3.1 Verwendung der Wiimote am PC

Offiziell ist die Verwendung der Wiimote von Nintendo zwar nur für die Wii-Konsole bestimmt, aber da es über Bluetooth relativ leicht möglich ist, sie mit dem PC zu verbinden, hat sich im Internet recht bald eine Gemeinschaft[11] von Hobbyentwicklern gefunden, die es ermöglichten, den Controller über Bibliotheken (zB. „WiimoteLib“ [19], siehe 5.1.1) in eigenen Programmen einzubinden. Dabei war die Experimentierfreudigkeit dieser Entwickler meist recht hoch. Ein Pionier bei der Erstellung von recht kreativen Anwendungen für die Wiimote am PC ist mit Sicherheit Johnny Chung Lee von der HCII Carnegie Mellon University [2]. Auf seiner Webseite [10] stellt er drei seiner wichtigsten Applikationen für unterschiedliche Verwendungen der Wiimote vor. Einige dieser Projekte flossen auch als Ideengebung in diese Diplomarbeit mit ein:

- *Fingertracking:* Unter Verwendung eines Infrarotarrays und an den Fingerkuppen des Benutzers angebrachten retroreflektierenden Klebebands (vgl. Abschnitt 6.2.2) werden die Bewegungen der Finger im Raum von einer einzigen Wiimote erfasst, auf eine 2D Ebene abgebildet und somit im 2D-Raum getrackt[12]. Das Klebeband reflektiert dabei das Infrarotlicht und strahlt es zur Wiimote zurück, welche auf dem Monitor liegt. Die Idee zur Verwendung eines Infrarotarrays (siehe Abschnitt 6.2.2) und retroreflektierenden Materials (siehe Abschnitt 6.2.2) in dieser Diplomarbeit entstammt diesem Projekt.

[11] engl: Community

[12] Unter Tracking versteht man das Verfolgen eines Punktes im Raum durch eine oder mehrere Kameras. Dabei ist vonnöten, dessen Position in Weltkoordinaten zu ermitteln

- *Interaktives Whiteboard:* Der Benutzer hat hierbei einen Stift in der Hand, der Infrarotlicht aussendet. Er steht vor einer Leinwand, auf welche ein Monitor ein vom Computer berechnetes Bild wirft. Wenn der Benutzer den Stift sehr nahe an die Leinwand hält, die nun wie eine Tafel fungiert, dann wird das Infrarotlicht reflektiert. Die Infrarotkamera einer Wiimote erkennt dieses Licht, berechnet, wo es sich befindet und zeichnet einen Strich, an diese Stelle. Somit kann der Benutzer virtuell auf der Leinwand malen.

- *Headtracking für Desktop VR-Displays:* Am Kopf des Benutzers werden 2 Infrarot-LEDs angebracht. Eine auf dem Display liegende Wiimote kann somit die Position des Kopfes verfolgen. Bei einer auf dem Monitor ausgegebenen 3D-Szene werden die Bilder nun abhängig von der Blickposition des Betrachters gerendert, sodass der Monitor wie ein Fenster in eine virtuelle Umgebung wirkt. Das Kamerazentrum wird dabei auf die Position des Betrachters gelegt, das Kamerafrustum wird bei Bewegung des Betrachters verändert. Geht der Benutzer beispielsweise näher an den Monitor heran, so wird der Öffnungswinkel [13] der Kamera vergrößert, sodass der Betrachter mehr von der Szene sieht. Allerdings muss der Betrachter den Kopf stets direkt in Richtung der Wiimote richten, sodass der Vektor zwischen Kopf und Wiimote senkrecht auf dem Vektor, den die beiden Infrarotpunkte aufspannen, steht. Tut er dies nicht, so kann die Position nicht mehr richtig berechnet werden.

Abbildung 7: Demonstration von J.C. Lee's Headtracking. Der Monitor wirkt dabei wie ein Fenster in eine virtuelle Welt. [16]

Außerdem gründete J. C. Lee die Webseite *WiimoteProject.com* [11], die ein Forum für Hobbyentwickler bietet, welche sich mit PC-Anwendungen für

[13] engl.: field of view

die Wiimote beschäftigen. Hier werden unter anderem auch nützliche Tipps und die technischen Vorraussetzungen für den Bau von Infarotstiften- und Arrays gegeben.

4 Konzeption

Bevor mit dem tatsächlichen Projekt begonnen werden konnte, mussten in der Konzeptionsphase verschiedene Fragestellungen bezüglich der Umsetzung geklärt werden. Das Projekt der Erweiterung der Wii-Interaktion mit einer 6-Achsen Steuerung stellt verschiedene Anforderungen in unterschiedlichen Bereichen. Zum einen sind diese technischer Natur, wie zum Beispiel die Anforderung, dass die Position der Wiimote im Raum erfasst werden muss. Zum anderen werden Anforderungen an die Beispielapplikationen gestellt, die die Ergebnisse des Projekts demonstrieren sollen.

4.1 Liste der Anforderungen

In der folgenden Liste werden die Anforderungen an das Projekt aufgeführt. Diese lassen sich unterteilen in Anforderunge die erfüllt werden müssen, damit das Projekt als erfolgreich bewertet werden kann, und Anforderungen, die möglichst erfüllt werden sollten.

- Die Wiimote soll für den Benutzer als Eingabegerät dienen. Alle in der Anwendung nötigen Interaktionen sollen mit der Wiimote durchführbar sein.
- Die Rotationen der Wiimote im Raum sollen mit Hilfe des Zusatzadapters MotionPlus abgefragt werden und in mindestens einer Applikation eingesetzt werden.
- Aus den ersten beiden Anforderungen folgt, dass eine Möglichkeit gefunden werden muss, auf die Daten, die von der Wiimote und MotionPlus gesendet werden, zuzugreifen und diese zu verarbeiten.
- Bei der Unterstützung der Rotation der Wiimote im Raum sollte möglichst gewährleistet sein, dass das Interaktionsobjekt im virtuellen Raum ähnlich orientiert ist, wie die vom Anwender gesteuerte Wiimote. Deshalb sollte es eine Möglichkeit geben, die Orientierung des Objekts bei ruhigen Bewegungen neu zu kalibrieren.
- Die Position der Wiimote im Raum soll mittels Stereotracking verfolgt werden. Damit soll erreicht werden, die Translation des Eingabegeräts im dreidimensionalen Raum zu übertragen.
- Für das Stereotracking werden zwei Kameras benötigt. Damit für den Spieler möglichst keine Zusatzhardware angeschafft werden muss und sich die eingesetzten Geräte möglichst auf den Bereich Wii und Wiimote beschränken, sollen zwei im Raum positionierte Wiimotes als Infrarotkameras fungieren.

- Da die Kameras der Wiimotes nur Infrarotpunkte verfolgen können, muss die Wiimote des Spielers dieses Infrarotlicht direkt oder indirekt aussenden. Da die Wiimote des Spielers ursprünglicherweise kein Infrarotlicht aussendet, muss eine Möglichkeit gefunden werden, wie dieses erreicht werden kann.
- Dabei sollten an der Wiimote möglichst wenig Gerätschaften wie Kabel oder Dioden angebracht werden, damit sie noch bequem vom Spieler gehalten werden kann.
- Es sollen verschiedene Beispielapplikationen erstellt werden, mit denen das Projekt demonstriert werden kann. Dabei kann es sich um Computerspiele handeln, dies muss aber nicht zwangsweise der Fall sein. So wären auch Applikationen im medizinischen Bereich denkbar.
- In mindestens einer Applikation sollen sowohl Rotation als auch Translation der Wiimote zur gleichen Zeit umgesetzt werden, um eine 1:1-Bewegungssteuerung zu ermöglichen.
- Die Applikationen sollen den Unterschied und die Vor- und Nachteile zwischen den verschiedenen Steuerungsarten, nämlich R-, , und R+T-Steuerung ersichtlich machen.
- Es sollte möglichst mindestens eine Applikation erstellt werden, die ohne Translation der Wiimote im Raum nicht denkbar wäre. Damit kann gezeigt werden, dass eine komplexere Umsetzung einer Bewegungssteuerung neue Spiele mit der Wiimote überhaupt erst möglich macht.
- Um die Steuerungsarten demonstrieren zu können, müssen die Applikationen in einer dreidimensionalen virtuellen Welt stattfinden. Diese sollte möglichst ansprechend visualisiert werden, um die Immersion des Benutzers noch zu verstärken.

4.2 Spielkonzepte

Worauf bei der Auswahl der Spielkonzepte geachtet wurde Die Spielkonzepte wurden mit dem Ziel ausgewählt, die Vor- und Nachteile einer 6-Achsensteuerung zu demonstrieren. Dabei wurde schon vorher zwischen den Vorteilen der R+T-Steuerung, also dem neuen realistischen Spielgefühl, der Immersion und Motivation, und den Nachteilen, nämlich einer möglichen Verkomplizierung der Steuerung, einer geringeren Einsteigerfreundlichkeit oder einem erhöhten Schwierigkeitsgrad abgewägt. Es wurde zuvor überlegt ob der Einbezug der Translation dem Spieler ein neues

Spielgefühl vermittelt. Auch wurden Konzepte in Betracht gezogen, die mit einer reinen R-Steuerung nicht möglich gewesen wären.

mögliche Faktoren, die von einer R+T-Steuerung beeinflusst werden können Ein ganz zentraler Teil des Projekts besteht darin, zu beobachten, welche Faktoren, die die Qualität eines Spiels ausmachen, durch eine R+T-Steuerung beeinflusst werden. In Fachzeitschriften werden Spiele oft nach den Kriterien Spielspaß, Einsteigerfreundlichkeit, Motivation, Qualität der Steuerung, etc. bewertet. Es erschien deshalb konsequent, dass der Erfolg der Einführung einer solchen Steuerung ebenfalls auf diese Kriterien hin überprüft werden sollte. Dies zieht sich wie ein roter Faden durch das gesamte Projekt. Im Folgenden werden diese Faktoren beschrieben.

- *Motivation zu Beginn* Damit ein Spieler aus freien Stücken ein Spiel zum ersten Mal startet, muss eine gewisse Motivation dazu vorhanden sein. Laut dem Medienpädagogen Jürgen Fritz muss eine Person, bevor sie überhaupt Interesse an einem Spiel entwickeln kann, in irgendeiner Weise mit diesem strukturell gekoppelt sein [3]. Das bedeutet, dass der Spieler, der das Spiel zuvor noch nie gespielt hat, in seinem bisherigen Erfahrungsschatz, den er in der realen Welt oder auch in schon bekannten Spielen gesammelt hat, gewisse Schemata vorliegen haben muss, die in irgendeiner Weise mit dem Spiel in Zusammenhang stehen. So muss „der Reizeindruck eines Autofahrspiels [...] entfernte Ähnlichkeit mit dem real erlebten Autofahren“ oder auch mit „Präsentationsformen des Autofahrens in der medialen Welt (also in Filmen und Fernsehserien)“ [3] haben, damit überhaupt eine Nachfrage an einem solchen Spiel entstehen kann. Es wäre umgekehrt auch fraglich, ob jemand, der in seinem ganzen Leben noch kein Fahrzeug gesehen hat, und somit keinerlei Bezug zum Autofahren hat, überhaupt Interesse an einer solchen Autofahrsimulation entwickeln könnte. [3] Es stellt sich im Bezug auf das Projekt also die Frage, ob die R+T-Steuerung das Interesse erhöht, eine der Applikationen zu starten.

- *Einsteigerfreundlichkeit* Dieser Faktor beschreibt, ob die Steuerung zu Beginn eher eine Hürde darstellt, oder der Spieler schon früh das Gefühl entwickelt, die Anforderungen des Spiels beherrschen zu können.

- *Langzeitmotivation* Mit Langzeitmotivation ist gemeint, ob ein Spiel nur kurzfristig den Spieler in seinern Bann schlägt oder ihn auch langfristig motivieren kann, weiter zu spielen.

- *Ist die Steuerung präzise* Damit ist gemeint ob sich das Objekt, das gesteuert wird exakt und dazu auch flüssig bewegt oder ruckelt.
- *Steigt die Immersion?* Immersion wird beschrieben als "das Gefühl, Teil einer virtuellen Welt zu sein“ [13]. Ein wichtiges Ziel der R+T-Steuerung wird mit Sicherheit sein, dem Spieler das Empfinden zu vermitteln, direkt in der virtuellen Welt interagieren zu können und somit in diese einzutauchen. Bei [13] wird Immersion auch als eine sehr ausgeprägte Form der Interaktion beschrieben[14].
- *Wird das Spielen erleichtert oder erschwert?* Mit der Erweiterung der Rotationssteuerung durch die Translation steigt auch die Komplexität. Es stellt sich die Frage, ob dies die Kontrolle über das Spiel letztendlich erschwert, oder sogar erleichtern könnte.
- *Werden die Bewegungen der Wiimote erwartungsgemäß ins Spielgeschehen übertragen?* Um die Immersion zu steigern, ist es wohl wünschenswert, wenn das Objekt im Spiel die Bewegungen der Wiimote möglichst exakt nachahmt.
- *Bietet die Steuerung neue Möglichkeiten, das Spielgeschehen zu beeinflussen?* Die Komplexität der R+T-Steuerung könnte bedeuten, dass der Spieler letztendlich mehr Möglichkeiten hat, das Spielgeschehen zu beeinflussen.
- *Steigt der Spielspaß?* Dies ist die wichtigste Frage, da der Spielspaß letztendlich darüber entscheidet, ob das Spiel dem Spieler gefällt.

4.2.1 Tischtennis

Im Spiel WiiSports Resort gibt es eine Tischtennisapplikation, bei der der Schläger nur durch Rotation bewegt werden kann. Diese Applikation soll dieses Konzept um die Translation erweitern. Der Schläger soll komplett frei im Raum bewegbar sein, also müssen sowohl Rotation als auch Translation gleichzeitig unterstützt werden. Die Tischtennisapplikation soll dem Spieler ein möglichst realitätsnahes Spielgefühl vermitteln. Um Vor- und Nachteile dieser Steuerung zu ermitteln, soll ein zweiter Modus geboten werden, bei dem der Schlag nur durch Rotation durchgeführt wird.

Tischtennis ist eine Sportart, bei der es auf Reaktion, Genauigkeit beim Schlag und vor allem Timing ankommt. Andere denkbare Applikationen, zum Beispiel eine, bei der nur ein Objekt im Raum positioniert werden

[14] dies bezieht sich auf das sogenannte VR-Referenzmodell. Dieses besteht aus 3 Achsen, wobei auf einer Achse die Interaktion aufgeführt wird. Bei besonders hoher Interaktion wird dabei von Immersion gesprochen.

soll, würden weitaus weniger Anforderungen an die Präzision der Simulation stellen. Deshalb ist dies eine Applikation, die Schwachstellen beim Tracking nur schwerlich verzeiht, mit der selbiges aber auch sehr gut demonstriert werden kann.

Beim Tischtennis wird, wie im echten Leben, ein Schlag meistens so durchgeführt, dass der Schläger mit der Schlagfläche nach vorne geführt wird und mit dem Ball kollidiert. Auch in dieser Applikation soll das ermöglicht werden. Der Spieler wird also die Wiimote quer halten und auf die beiden Kameras zu bewegen. Dies bedeutet eine Translation in z-Richtung. Es muss also gewährleistet werden, dass diese Translation in z-Richtung besonders fehlerfrei übertragen wird.

Die Kamera wird so positioniert, dass es wirkt, als würde man selbst vor der Platte stehen. Der Schläger soll dabei stets im Blickfeld bleiben. Eine Lichtquelle soll von oben auf die Platte scheinen, so dass der Tischtennisball stets einen Schatten wirft. Der Spieler soll dadurch leichter die Position des Balls im Raum ermitteln können (vgl. Abschnitt 5.5).

Das Kollisionsverhalten zwischen Schläger und Ball ist von größter Wichtigkeit. Auch dieses soll der Realität angelehnt sein. So darf der Ball bei leichten Kollisionen nicht unrealistisch weit weg geschlagen werden, andererseits aber auch nicht zu leicht. Der Spieler soll ein möglichst intuitives Gefühl dafür entwickeln, wie fest der Ball geschlagen werden muss. Es soll aber sehr wohl auch möglich sein, den Ball bei zu festen Schlägen ins Aus, oder bei zu leichten ins Netz zu schlagen. Das Tischtennisnetz soll deshalb ein tatsächliches Hindernis darstellen, muss also ebenfalls physikalische Eigenschaften besitzen.

Es soll regelmäßig ein neuer Ball zum Spieler geschlagen werden, der diesen dann zurückschlagen kann. Wenn ein einfacher Schwierigkeitsgrad gewählt wird, soll der Ball stets an dieselbe Stelle geschlagen werden und dabei dieselbe Flugbahn unternehmen. Dies erleichtert das Treffen des Balls, da man sich nach einiger Zeit nicht mehr darauf konzentrieren muss, wo der gegnerische Ball hingespielt wird. Da dieser Modus aber nach einiger Zeit an Reiz verlieren wird, soll ein höherer Schwierigkeitsgrad gewählt werden können. Bei diesem wird der Ball an unterschiedliche Stellen der Platte, von unterschiedlichen Positionen aus geschlagen. Der Spieler muss als die Wiimote vor dem Schlag an passende Ausgangspositionen im Raum bewegen. Deshalb ist es außerordentlich wichtig, dass das Trackingverfahren die Translation des Schlägers möglichst detailgenau wiedergibt. Es soll dabei auch möglichst keine Verzögerung zwischen der Bewegung des Spielers und der Bewegung des Schlägers in der virtuellen Welt geben.

Bei Tischtennis hängt viel davon ab, wie der Spieler den Schläger zum Ball hin orientiert. So kann man durch Drehung des Handgelenks den Schläger

in Top-Spin oder Slice-Stellung halten, ihn anwinkeln, und durch die Orientierung bestimmen, ob man Cross oder Long Line spielen will. All dies soll in der Tischtennisapplikation auch möglich sein, wobei Top Spin, oder Slice-Schläge auch von der Physikengine unterstützt werden müssten. Ob dies letztendlich möglich ist, muss sich also zeigen.

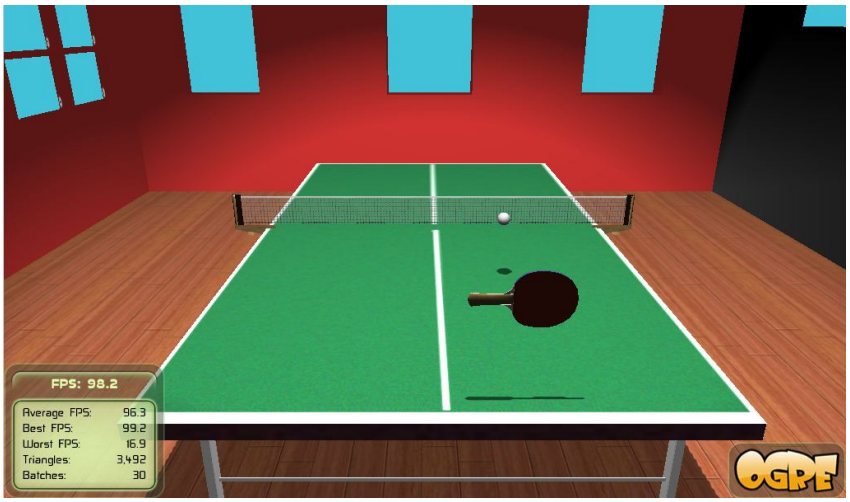

Abbildung 8: Die Tischtennisapplikation

4.2.2 Billiard

Wie im echten Billiard sind die Spielregeln hier folgendermaßen. Der Billiardtisch verfügt über 6 Löcher an den Seiten. Es können 2 Spieler gegeneinander abwechselnd spielen, oder auch 2 Teams, wobei sich die Spieler eines Teams abwechseln. Insgesamt gibt es in dem Spiel 16 Kugeln. Eine weiße Kugel, eine schwarze, und 7 pro Team, die von eben diesem in die Löcher befördert werden sollen. Abbildung 10 zeigt die 4 verschiedenen Kugeltypen. Der Spieler stößt dafür mit dem Billiardqueue die weiße Kugel an, die dann wiederum eine oder mehrere Kugeln des eigenen Teams in die Löcher befördern soll.

Billiard benötigt im echten Leben äußerste Präzision. Allerdings hat der Spieler dort tatsächlich einen langen Queue in der Hand, den er mit der linken Hand abstützen kann. Dies ist mit der Wiimote leider nicht möglich. Man kann sie zwar frei im Raum bewegen, es dürfte aber sehr schwer fallen, sie exakt gerade nach vorne zu stoßen. Deshalb werden die Freiheitsgrade bei Billiard eingeschränkt, damit es nicht passieren kann, dass man aus Versehen die Kugel an der falschen Stelle trifft.

Zunächst sieht der Spieler das Spielfeld von oben. Wenn die A-Taste der Wiimote gedrückt wird, so wird der Queue automatisch in der Nähe der

weißen Kugel platziert. Wird die Taste gedrückt gehalten, so befindet sich der Spieler im Zielmodus. Wenn er die Wiimote rotiert, so rotiert der Queue in geringem Abstand zur Kugel um diese herum und bleibt dabei stets auch auf diese gerichtet. Die Kamera wird dabei stets knapp über dem Queue platziert, so dass die weiße Kugel, wie auch die Kugeln die ins Loch befördert werden sollen, gut sichtbar sind und man gut zielen kann. Lässt man die A-Taste los, so kann man nun den Stoß ausführen. Die Kamera wird dann etwas höher platziert, damit man den Queue gut sehen und beurteilen kann, wie fest man schießt. Wenn der Spieler mit der Wiimote eine Bewegung nach vorne ausführt, so bewegt sich der Schläger in seinem lokalen Koordinatensystem nach vorne, also in Richtung der Kugel. Dabei wird die Rotation komplett unterdrückt. Außerdem kann man den Queue hier nicht nach oben, unten, links oder rechts bewegen, sondern nur nach vorne und hinten, also entlang der Z-Achse des lokalen Koordinatensystems. Dadurch kann es nicht passieren, dass man mit dem Queue aus Versehen nicht direkt auf die Mitte der Kugel zielt.

Abbildung 9: Die Billiardapplikation

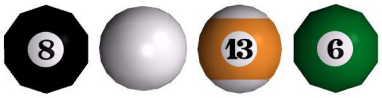

Abbildung 10: Links: schwarze und weiße Kugel, rechts: halbe und volle Kugeln der Teams

4.2.3 Baseball

Die Applikation Baseball soll nicht ein komplettes Baseballspiel simulieren, sondern konzentriert sich einzig und allein auf das Schlagen und Zielen von Bällen mit Hilfe eines Baseballschlägers. Dabei soll genau wie bei Tischtennis ein Modus existieren, der nur die R-Steuerung zulässt und einer für die R+T-Steuerung.

Wenn der Spieler den A-Knopf drückt, wird ein Ball in Richtung des Schlägers geworfen. Im Spiel werden Körbe unterschiedlicher Größe aufgebaut, in die der Ball mit Hilfe des Baseballschlägers geschlagen werden soll. Wenn der Ball fünf mal in einen Korb getroffen wurde, wird ein höherer Schwierigkeitsgrad gestartet. Der Korb, wie auch der Ball werden dann kleiner, wodurch es schwerer sein soll, zu treffen.

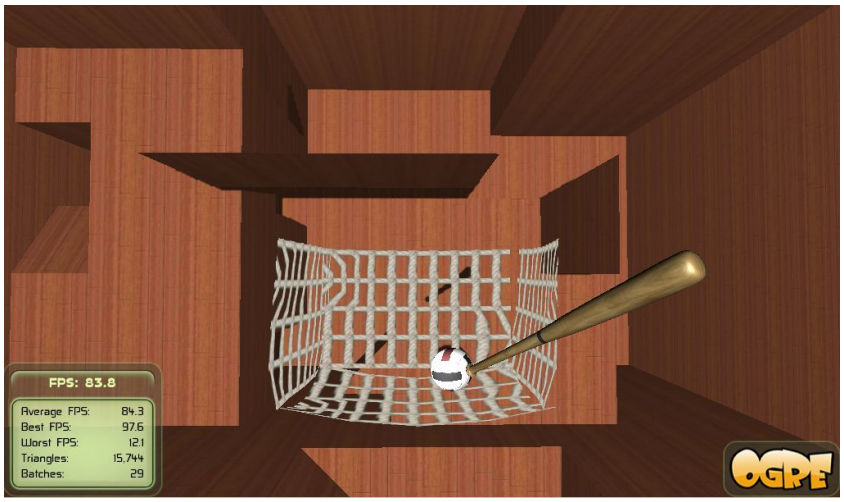

Abbildung 11: Die Baseballapplikation

4.3 Hypothesen bezüglich der Steuerung

Die 3 Applikationen wurden nicht zuletzt deshalb ausgewählt, weil wohl nahezu jeder Spieler diese Sportarten kennt und sofort weiß, was zu tun ist.

Im Folgenden werden einige Hypothesen aufgestellt, die die Auswirkung der R+T-Steuerung auf die Faktoren, die in Abschnitt 4.2 beschrieben wurden, betreffen. Ob diese Hypothesen letztendlich zutreffen, wurde im Verlauf des Projekts anhand der Aussagen von Testpersonen überprüft.

4.3.1 Allgemeine Hypothesen

Zunächst werden einige allgemeine Hypothesen gemacht, die auf alle 4 Applikationen zutreffen sollen.

- *Verbesserung der Rotationen* Durch eine R+T-Steuerung ist prinzipiell jede Bewegung möglich. Auch die Rotationen an sich sollen verbessert werden. Während es bei einer reinen R-Steuerung keinen Unterschied macht, ob man die Drehung aus dem Handgelenk oder in einer großen Bewegung mit dem ganzen Arm vollführt, soll dieser Unterschied bei der R+T-Steuerung sehr wohl zur Geltung kommen.
- *Verbesserung der Immersion* Durch den Einbezug von Rotationen und Translationen ist eine 1:1-Übertragung der Bewegungen, die der Spieler auf die Wiimote ausübt, möglich. Dies wiederum erzeugt beim Spieler ein sehr gutes Gefühl der Immersion. Durch die realistische Interaktion, die seinen tatsächlichen Bewegungen folgt, hat er „das Gefühl, Teil [der] virtuellen Welt zu sein“ [13].
- *Erhöhte Langzeitmotivation* Durch die Komplexität und die zahlreichen Schlagmöglichkeiten, die eine R+T-Steuerung bietet, könnte eine recht hohe Langzeitmotivation entstehen. Gewünscht ist bei einem Tischtennisspiel zum Beispiel, dass im Spiel Schläge möglich sind, die man auch im echten Leben vollführen kann, und diese erhöhte Komplexität den Spieler bei der Stange hält.
- *Höherer Spielspaß* Die große Frage, die sich letztendlich stellt ist, ob der Spielspaß durch die R+T-Steuerung erhöht wird. Wenn die Schwierigkeit, die die hohe Komplexität möglicherweise mit sich bringt nicht zu extrem ausfällt, ein gutes Verhältnis zwischen Anspruch und Flow-Erlebnissen beim Spieler herrscht, dann könnte der Spielspaß recht hoch sein. Auch die möglicherweise hohe Immersion könnte Spielspaß mit sich bringen, allerdings nur, wenn die Steuerung nicht Selbstzweck ist, sondern dem Spielgeschehen tatsächlich weitere Tiefe bringt.

4.3.2 Hypothesen bezüglich der Spielkonzepte

Tischtennis Da beim Tischtennis im realen Leben der Schlag nicht hauptsächlich durch eine Rotationsbewegung, sondern vielmehr auch durch ein Bewegung des ganzen Körpers nach vorne geschieht, wobei der Schläger meist in einem bestimmten Winkel zum Ball gehalten wird, ist die Hoffnung, dass genau dieses Verhalten durch die Translation erst ermöglicht wird. Bei der R-Steuerung würde der Computer den Part der Translation

übernehmen müssen, sofern dieser nötig ist, um den Schläger in eine Ausgangsposition zu bringen, in der ein Schlag auf den Ball überhaupt erst möglich ist. Dadurch fühlt sich der Benutzer zusätzlich eingeschränkt. Dadurch dass der Spieler bei einer R+T-Steuerung den Schläger selbst in eine günstige Ausgangsposition bringen muss, führt das zu einer erhöhten Aufmerksamkeit, und möglicherweise auch zu erhöhter Langzeitmotivation.

Weiterhin könnte eine R+T-Steuerung mehr Möglichkeiten bieten, den Ball an eine bestimmte Stelle zu schlagen. Kann der Spieler den Tischtennisschläger nämlich nur rotieren, dann entscheidet das Timing, in welche Richtung der Ball nach dem Schlag fliegt. Trifft man den Ball recht früh, dann trifft man ihn demnach auch weiter vorne. Der Schläger ist dabei in einem solchen Winkel, dass der Ball Cross, also schräg geschlagen wird. Abbildung 12 veranschaulicht solch einen Cross-Schlag, der durch Rotation zustandegekommen ist. Trifft man den Ball dagegen später, dann geht dies nur wenn man den Schläger in einer geöffneten Haltung hält. In diesem Fall würde er gerade aus, also Long-Line fliegen. Die Richtung des Schlags wird also letztendlich durch Timing bestimmt. Bei einer R+T-Steuerung hat man aber die freie Wahl, wie früh man den Ball trifft und kann dennoch an jede beliebige Stelle zielen. Man kann den Schläger einfach in einer beliebigen Orientierung halten und dabei eine Bewegung zum Ball hin vollführen. Die Orientierung des Schlägers und die Richtung des Schlags bestimmen dann, wo der Ball hinfliegt, man kann den Ball aber früh oder auch spät treffen. Abbildung 13 verdeutlicht diese Schlagmöglichkeit. Man hat aber zusätzlich weiterhin die Möglichkeit, den Schläger wie bei der R-Steuerung einfach nur zu rotieren.

Dies ist nur eines von vielen Beispielen zusätzlicher Schlagmöglichkeiten. Vorstellbar wären auch angeschnittene Bälle, sofern die Physikengine diese realisieren kann.

Billiard Die Hypothese bei diesem Spiel ist, dass bei der Steuerung weniger manchmal mehr sein kann. Billiard bedarf sehr hoher Präzision. Wer schon einmal am Computer mit der Maus eine Billiardsimulation gespielt hat, weiß, dass zwar das Zielen an sich kein Problem darstellt, dass aber die Kugel, die durch Kollision der weißen Kugel ins Loch befördert werden soll, an exakt der richtigen Stelle getroffen werden muss, was einiges an vorheriger Planung und Abschätzung bedarf. Hinzu kommt, dass der Spieler eine Wiimote und keinen Billiardqueue in der Hand hat. Er hat also nicht die Möglichkeit, den Queue mit der linken Hand zu fixieren und mit der rechten Hand zu stoßen. Aus diesem Grund soll die Komplexität der Steuerung so weit vereinfacht werden, dass der Spieler sich voll und Ganz auf das Zielen konzentrieren kann, und beim Stoßen nur darauf achten muss, wie fest er die Bewegung durchführt. Möglicherweise leidet

Abbildung 12: Beispielschlag bei R-Steuerung. Frühes Treffen des Balls führt zu Cross-Schlag (mit Maya gerendert)

dadurch zwar die Immersion, aber der Spielspaß könnte steigen.

Weiterhin basiert der Stoß beim Billiard auf einer Bewegung mit der Wiimote nach vorne. Dies ist also eine Translation, die nur mit Hilfe des Stereotrackings möglich ist. Die Hypothese ist also, dass das Stereotracking bestimmte Spiele, wie in diesem Fall Billiard, überhaupt erst möglich macht.

Baseball Die Sportsimulation Baseball wäre mit Sicherheit auch mit einer Rotationssteuerung denkbar. Allerdings sind es die Feinheiten der Bewegungen, die die R+T-Steuerung ermöglicht, und die dem Spiel so mehr Tiefe und Langzeitmotivation verleihen könnten.

Abbildung 13: Beispielschlag bei R+T-Steuerung. Die Schlagrichtung ist vom Timing unabhängig (mit Maya gerendert)

5 Umsetzung

In diesem Kapitel wird die Umsetzung des Projekts beschrieben. Dabei wird allerdings noch nicht auf die Umsetzung der Steuerung eingegangen, die dann im nächsten Kapitel beschrieben wird. Zunächst wird das Framework beschrieben, danach werden die Klassen und ihre Funktionalitäten erläutert, und dabei auch darauf eingegangen, wie die einzelnen Spiele programmiert wurden.

5.1 Das verwendete Framework

5.1.1 GlovePie

Es existieren verschiedene Programme für Windows die sich mit der Wiimote verbinden und deren Daten auslesen können. So kann man beispielsweise die Bibliothek „WiimoteLib“ [19] in C++ in ein Projekt einbinden und das eigene Programm so mit der Wiimote verbinden. Allerdings unterstützen die wenigsten Bibliotheken oder Programme den Zusatzadapter MotionPlus. Eine Ausnahme bildet hier die Skriptsprache GlovePie, die ich in meinem Programm wegen der MotionPlus-Unterstützung verwende.

Beispiel: Ein einfaches GlovePie Skript:

Um beispielsweise auf den aktuellen Gierwinkel zuzugreifen, kann man diesen einer Variable zuweisen nach der Form:

[caption=init3D]Name2 var.GyroYaw = Wiimote1.MotionPlus.GyroYaw

Wie in Abschnitt 2.2 erwähnt, verfügt MotionPlus über verschiedene Sensoren die bei unterschiedlichen Rotationsgeschwindigkeiten eingesetzt werden. Allerdings unterstützt GlovePie nur einen einzigen Sensor, weshalb bei schnellen Bewegungen die richtige Orientierung nach einiger Zeit verloren geht. Aus diesem Grund wird im Programm stets neu kalibriert, wenn die Wiimote ruhig gehalten wird.

5.1.2 Ogre

Um die dreidimensionalen Welten der verschiedenen Applikationen darzustellen, wird die Open Source Grafikengine Ogre verwendet. Diese kann entweder mit Open-GL Oder Direct-3D betrieben werden. Objekte werden in Ogre mit Hilfe von Szenegraphen verwaltet. Ogre ist nicht auf die Verwendung in Spielen beschränkt, sondern fand bereits Einsatz in Simulationen, Lernsoftware, interaktiver Kunst, wissenschaftlichen Visualisierun-

gen ua. [20] Obwohl Ogre oft in Spielen verwendet wird, ist es primär keine Spielengine. So fehlt die Unterstützung von Sound oder physikalischen Berechnungen. Diese Features müssen erst in Form von entsprechenden Bibliotheken eingebunden werden, die teilweise speziell für die Engine geschrieben wurden. Im Kernbereich von Ogre, der schnellen grafischen Darstellung, ist es im Vergleich zu anderen Open Source Spieleengines aber oft überlegen. Es existiert eine recht große Community im Netz[15], allerdings lässt die Dokumentation teilweise zu Wünschen übrig. In diesem Projekt wird Ogre in Visual Studio verwendet und dabei unter C++ programmiert.

Die Verknüpfung zwischen Ogre und Glovepie Um die Daten, die Glovepie von der Wiimote ausliest im Programm nutzen zu können, muss Ogre auf diese zugreifen können. Allerdings gibt es in GlovePie keine Funktion um Daten direkt an andere Programme zu senden. Es besteht lediglich die Möglichkeit, die Daten in eine Textdatei zu schreiben, die dann von einem anderen Programm ausgelesen werden könnte. Dieser Schreib- und Lesevorgang wäre aber mit unnötigem Aufwand verbunden und würde sich im Bezug auf die Geschwindigkeit der Übertragung als Flaschenhals erweisen.

In meinem Projekt lese ich deshalb diese Daten pro Frame, der gerendert wird, neu aus dem Speicher aus.

5.1.3 OgreBullet

Wie bereits erwähnt stellt Ogre an sich keine Spielengine, sondern lediglich eine Grafikengine dar. Deshalb bietet es keine direkte Unterstützung für physikalische Berechnungen. Es gibt aber verschiedene Physikengines die in Ogre eingebunden werden können. Unter anderem die Bullet Engine, die in der rennomierten US amerikanischen Universität Massachusetts Institute of Technology (MIT) erstellt wurde.

Bullet ist eine Open Source Physik Engine und bietet Kollisionserkennung und Simulation der Bewegung fester, den sogenannten Rigid Bodies, und weicher Körper, die als Softbodies bezeichnet werden. [25] Als Rigid Bodies unterstützt die Engine die geometrischen Formen Kugel, Würfel, Zylinder, Kegel aber auch die konvexe Hülle jedes beliebigen Körpers. Dabei können die Körper in allen Achsen skaliert werden, eine Kugel muss also letztendlich nicht komplett rund sein. Weiterhin bietet Bullet an, den Physikobjekten bestimmte Bedingungen[16] zuzuweisen. So kann ein Objekt zum

[15] Ogre-Forum: http://www.ogre3d.org/forums/
[16] engl.: constraints

Beispiel an einer bestimmten Stelle im Raum oder auch an einem anderen Objekt befestigt werden, oder anderweitig in seiner Bewegungsfreiheit eingeschränkt werden.

Obwohl die Engine Open Source ist wurde sie unter anderem in kommerziellen Spielen wie GTA 4 und Red Dead Redemption verwendet. Aber auch in Animationsfilmen und 3D Visualisierungs- und Animationssoftware[17] kam sie bereits zum Einsatz. [25]

Ogre implementiert Bullet nur teilweise unter Einbindung der Bibliothek OgreBullet. Diese ermöglicht eine einfache Implementation der wichtigsten Features der Engine. Dagegen werden komplizierte Berechnungen, wie zum Beispiel die von Softbodies, nicht unterstützt. Es ist aber auch möglich, über einige Umwege innerhalb von Ogre auf die originale Bullet Engine zuzugreifen (siehe unten).

Ursprünglicherweise sieht der Betrachter nur die Geometrieobjekte, die die Grafikengine zeichnet. Die dazugehörigen Physikobjekte, die meist weniger Geometrie besitzen werden standardmäßig nicht angezeigt. Oftmals ist es für die Entwicklung aber nötig, diese Objekte zu sehen. Dies leistet der sogenannte Debug Drawer, der die Physikobjekte in Form von weißen Linien darstellt. In dem Projekt kann dieses Feature auf Knopfdruck angeschaltet werden. Abbildung 15) zeigt den Debug Drawer in Aktion.

[21]

Verwendung der ursprünglichen Bullet-Bibliothek statt OgreBullet In diesem Projekt wurde Wert auf einige Features von Bullet gelegt, die von OgreBullet jedoch nicht unterstützt werden. So sollten zum Beispiel sogenannten Softbodies (siehe unten, Paragraph Softbodies) eingebunden werden sowie das Feature, aus Geometrieobjekten Physikobjekte zu erzeugen. Bullet bietet im Gegensatz zu Ogre Bullet aber noch weitere Eigenschaften.

Es war über einige Umwege möglich, die ursprüngliche Bullet Engine einzubinden. Allerdings können deren Physikobjekte nicht ohne weiteres mit Ogre Objekten verbunden werden, um sie grafisch anzeigen zu können. Hierfür ist eine Wrapperklasse *BtOgre* nötig, die die Pose eines Bullet Physikobjekts im dreidimensionalen Raum mit einem Ogre Geometrieobjekt verbinden kann. Mit dieser Klasse ist es auch möglich, ein Ogre Mesh in ein Bullet Mesh umzuwandeln. So können zum Beispiel Terrains in Ogre erstellt werden und darauf basierend das zugehörige Physikobjekt (Rigidbody) in Bullet automatisch erstellt und verwendet werden. Meist bietet es sich an zu einem komplexen Modell, welches grafisch angezeigt wird, ein weniger komplexes Modell zu erstellen, aus welchem dann das Physikob-

[17] zum Beispiel der Open Source Software Blender

jekt generiert wird, da Objekte mit viel Geometrie auch mehr Rechenzeit benötigen.

Rigid Bodies Rigid Bodies sind im Gegensatz zu Softbodies feste Körper, die nicht deformiert werden können. Bei der Kollisionsabfrage prüft Bullet in bestimmten zeitlichen Abständen, ob ein Objekt teilweise in ein anderes eindringen würde. Ist dies der Fall, wird berechnet, inwieweit sich die Objekte voneinander abstoßen, wobei neben den kinetischen Kräften einige spezifische Parameter der Körper in die Berechnung eingehen, wie zum Beispiel die Elastizität (vgl. Abschnitt 5.3.3. Problematisch ist allerdings, wenn die Volumen der Körper zu klein sind im Vergleich zur Geschwindigkeit mit der sie auftreffen. Wenn sich die Objekte so schnell bewegen, dass sie sich innerhalb zweier Berechnungsschritte schon komplett durchdrungen haben, so kann Bullet keine Kollision erkennen. Es ist deshalb anzuraten, einen guten Kompromiss zwischen der Anzahl der Berechnungsschritte pro Sekunde, der Größe der Physikobjekte und deren Geschwindigkeiten zu finden.

Man kann Rigid Bodies als kinematische Körper definieren. Solche Objekte besitzen keine Masse und können deshalb nicht zu Boden fallen. Sie reagieren nicht auf Kollisionen mit anderen Körpern, können selbst aber auf andere Körper einwirken. Für die vom Spieler gesteuerten Objekte wurden Kinematic Bodies verwendet, damit diese nicht zu Boden fallen und vom Spieler bewegt werden können. Ein großes Problem bei den Kinematic Bodies war, dass man nicht durch die Veränderung der Masse der Körper beeinflussen konnte, wie stark ein Rigid Body von einem Kinematic Bodz weggestoßen wird. Man konnte dies nur sehr eingeschränkt durch die Schwerkraft und die Elastizität der Körper manipulieren.

Soft Bodies Softbodies sind Körper, die unter Einwirkung von Kräften deformiert werden können (Quelle vielleicht). In den Applikationen wurden Softbodies für die Simulation und Darstellung von Netzen in den Applikationen Tischtennis und Baseball verwendet.

Wie oben beschrieben unterstützt OgreBullet keine Softbodies, das im späteren Verlauf des Projekts verwendete ursprüngliche Bullet jedoch schon. Neben der Wrapperklasse *BtOgre* war eine weitere Klasse „BtOgreSoftbody“ nötig, damit die Objektgeometrie, die ja zur Ogre Engine gehört, auch auf die Verformungen des Softbodies reagieren kann. Mit der Klasse kann man ebenfalls aus einem Ogre Mesh ein Bullet Softbody erstellen. Um diesen dann durch das Ogre Mesh grafisch sichtbar zu machen werden pro Frame die Positionen der Vertices des Ogre-Mesh entsprechend den Positionen der Knoten des Bullet Softbody angepasst. Somit wird die Geome-

trie, die grafisch dargestellt wird, entsprechend des Softbody verändert. Weiterhin muss auch die Position und Orientierung des Knotens, an dem die Geometrie hängt, also das Objekt an sich, dem Softbody entsprechend angepasst werden. Damit ist es also möglich, Softbodies auch in Ogre zu verwenden. Allerdings geht dies bei komplexeren Objekten häufig zu Lasten der Framerate.

Soft Body-Patches Man kann mit Hilfe von *BtOgreSoftbody* also einen Softbody aus gewöhnlicher Geometrie erzeugen. Dieser reagiert allerdings auf Schwerkraft und würde in einer Simulation einfach zu Boden fallen. Um aber beispielsweise ein Tischtennisnetz zu simulieren, müssten die 4 Ecken des Netzes fest positioniert sein. In Bullet ermöglichen dies sogenannte Softbody-Patches die über die Funktion createPatch() erzeugt werden. Der Funktion werden die Positionen der 4 Ecken übergeben, sowie die Anzahl der Unterteilung des Netzes in x- und y- Koordinaten. Die Klasse BtOgreSoftbody bot allerdings keine Möglichkeit, um aus selbst erzeugter Geometrie solch ein Patch zu erstellen. Ich habe deshalb ein Mesh erzeugt, das denselben Aufbau hat wie das Patch, dass ich verwenden will. Weiterhin habe ich die Funktion von BtOgreSoftbody insofern angepasst, dass sie nicht mehr einen Softbody aus dem übergebenen Mesh erzeugt, sondern stattdessen ein gewöhnliches Patch. Es musste darauf geachtet werden, dass das Patch dieselbe Auflösung hat wie das Mesh. Nun konnte auch hier das Mesh pro Frame dem Patch angepasst werden. Problematisch war aber, dass die Reihenfolge der Vertices und der dazugehörigen Texturkoordinaten im Mesh nicht der Reihenfolge der Knoten im Knotenbuffer des Softbodies entspricht. Deshalb wurden die Polygone in der Applikation in komplett falscher Anordnung angezeigt. Um dies zu berichtigen mussten in der Mesh-Datei manuell die Reihenfolge der Texturkoordinaten sowie die Pointer, die die Polygone erzeugen verändert werden.

5.1.4 Shadersprache

5.2 Erstellung der 3D-Objekte

Um eine visuell ansprechende virtuelle Spielumgebung zu erzeugen, wurden alle Spielobjekte vorher als 3D-Modelle mit der Animationssoftware Autodesk Maya[18] erzeugt. Hilfreich dabei waren viele Tutorials, die während des Studiums durchgeführt wurden. Bei 3D-Meshes in Spielen sollte man einen geeigneten Kompromiss bei der Polygonanzahl finden, um die Objekte zwar möglichst detailliert aussehen zu lassen, aber dennoch

[18] http://www.autodesk.de/adsk/servlet/pc/index?siteID=403786&id=14657512

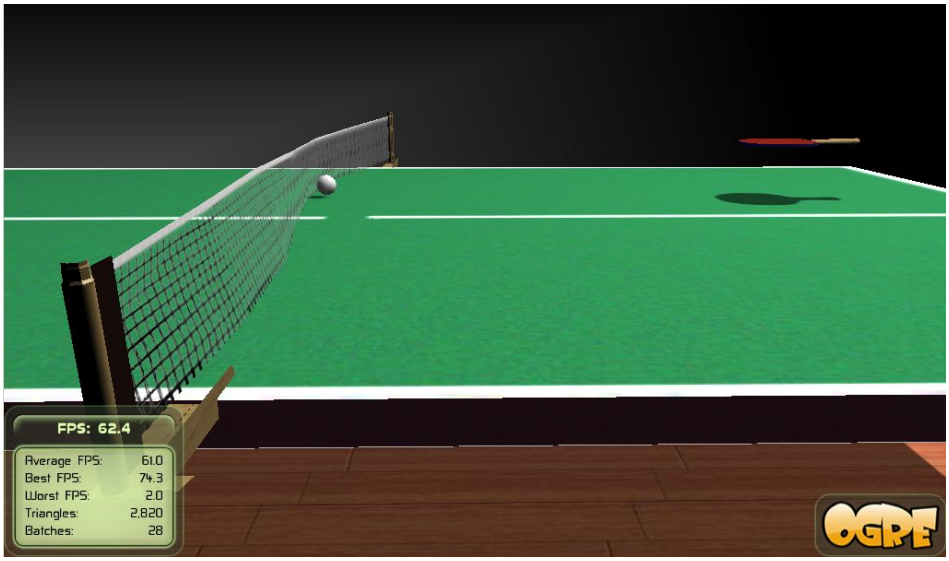

Abbildung 14: Um die Verformung von Softbodies grafisch darzustellen muss die Geometrie des Meshs dem Physikobjekt entsprechend transformiert werden.

nicht so viel Geometrie zu verwenden, dass die Framerate darunter leidet. Bei dieser Aufgabe war vor allem das Buch *"Maya for Games"* [4] hilfreich, das ausschließlich auf die Erstellung und Texturierung von 3D-Objekten in Spielen eingeht.

5.3 Allgemeine Klassen

Im Folgenden Abschnitt werden Klassen meines Projekts beschrieben, die von allen oder den meisten Applikationen verwendet werden.

5.3.1 Die Klasse Main

Die Klasse *Main* erbt von einer von Ogre bereits zur Verfügung gestellten Basisklasse *BaseApplication*. Letztere stellt die Grundfunktionalitäten der Grafikengine Ogre zur Verfügung. Dazu gehört zum Beispiel der Szenegraphenmanager, der alle Transformations-, Geometrie-, und Gruppenknoten verwaltet. Weiterhin unterstützt *BaseApplication* bereits die Kamera, die ebenfalls der Szenegraphenmanager verwaltet, den Viewport sowie EventListener die auf Tastatur- und Mausabfragen reagieren. Sie erzeugt den FrameListener, der auf Events reagiert sobald ein Fenster erzeugt wurde. Außerdem stellt sie abstrakte Methoden zur Verfügung, die von *Main* überschrieben werden sollen. Dazu gehört eine Funktion *createScene()*, die

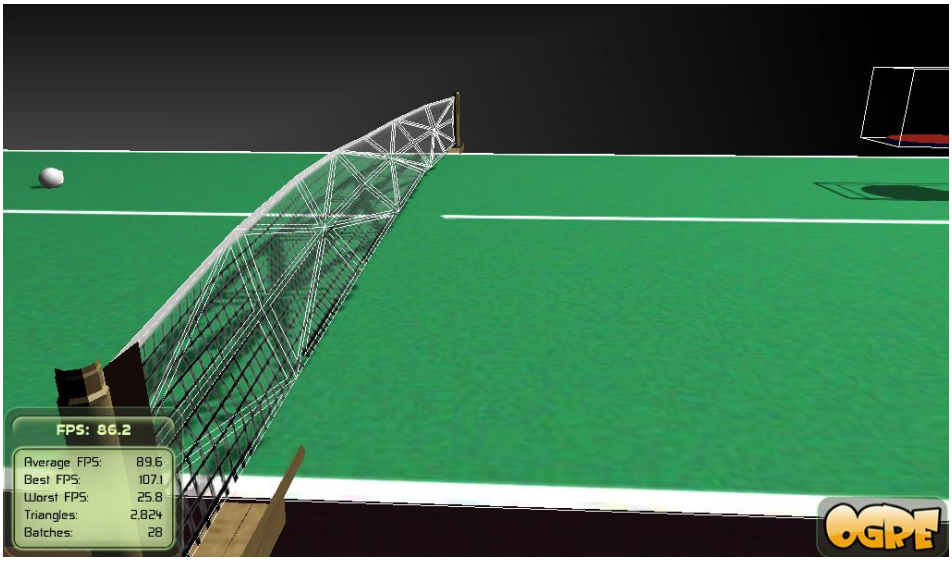

Abbildung 15: Mit Hilfe des Debug Drawers lassen sich die Knoten des Softbodies grafisch anzeigen. (weiße Linien am Tischtennis Netz)

zu Beginn die Szene erzeugt, also unter anderem alle darin enthaltenen Geometrieobjekte und die Beleuchtung, sowie alle Anweisungen an das Programm, die nur zu Beginn ausgeführt werden müssen. Außerdem gibt es die von *Main* zu überschreibende Funktion *frameRenderingQueued*, die auf den FrameListener reagiert und vor jedem Renderingschritt eines Frames wird eine weitere Funktion *processUnbufferedInput* gestartet, die weitere Abläufe pro Frame neu berechnet. Will man beispielsweise ein Objekt laufend rotieren lassen, so wird diese Anweisung in *processUnbufferedInput()* gegeben.

Aus *Main* heraus werden die verschiedenen Applikationen gestartet. Da die 3D-Welten unterschiedlich skaliert sind, verwaltet *Main* unterschiedliche Skalierungsfaktoren, die den jeweiligen Applikationen übergeben werden.

Damit physikalische Berechnungen möglich sind, erzeugt *Main* die Physikumgebung Bullet, die über eingebundene Bibliotheken angesprochen wird. Wie in Abschnitt 5.1.3 beschrieben, wurde im Projekt zu Beginn OgreBullet verwendet, später dann allerdings auf die Verwendung des originalen Bullet mit Hilfe einer Wrapperklasse umgestiegen, um Softbodies zu ermöglichen und aus Meshes Physikobjekte zu erzeugen. Ein wichtiges Attribut das dabei gesetzt wird ist die Größe der Schwerkraft, die mit dem Skalierungsfaktor angepasst wird. Eine kleinere Welt benötigt auch eine geringere Schwerkraft, um dasselbe realistische Verhalten zu bewahren. Für

Softbodies wird eine eigener Schwerkraftswert festgesetzt[19]. Dieser muss allerdings nicht skaliert werden. Weiterhin wird in *Main* als Physikobjekt eine Art Boden erzeugt, der dafür sorgt, dass Objekte nicht unendlich in die Tiefe fallen.

Weiterhin reagiert *Main* auf Tastatureingaben und sendet diese an die jeweilige Applikation weiter. Es wird auch die Funktionalität ermöglicht, dass die Kamera mit der Maus gesteuert [20] und mit den Pfeiltasten der Wiimote bewegt werden kann. *Main* sorgt auch dafür, dass die Klasse *Menu* gestartet wird, durch die auf Tastendruck ein Art Menü gestartet werden kann. Auch eine Konsole, die Ausgaben erzeugen kann, wird erzeugt.

5.3.2 Die Klasse Game

Die Klasse *Game* ist die Oberklasse, von der jede Applikation erbt. Sie definiert wichtige Verhaltensweisen, die für jede Applikation gelten.

Die Klasse stellt zum Beispiel ein Objekt der Klasse *Schläger* zur Verfügung, da jede Applikation letztendlich über solch ein Objekt verfügen soll, welches vom Spieler mit der Wiimote gesteuert werden kann. Dieses wird aber erst in der jeweiligen Applikation über einen Konstruktor erzeugt, da es unterschiedliche Schlägertypten gibt, welche ihre eigenen von der Schlägerklasse abgeleiteten Klassen besitzen. Da in allen Spielen Schatten verwendet werden, definiert die Funktion *setShadow()*, wie diese im Spiel generiert werden (Beschreibung der Schatten in Abschnitt.

5.3.3 Die Klasse Ball

Sowohl bei Tischtennis, Billiard wie auch Baseball werden Bälle geschlagen. Deshalb wurde eine Klasse *Ball* geschrieben, die von allen Spielen verwendet werden kann. Mit Hilfe der Klasse wurden unter anderem Fußbälle, Volleybälle, Tischtennisbälle, und Billiardkugeln simuliert.

Der Klasse kann ein Mesh übergeben werden, das möglichst die geometrische Form eines beliebigen Balles haben sollte. Das Mesh wird über die Funktion „setType()“ übergeben. Weiterhin übergibt man der Klasse den Manager des Szenegraphen, den Manager der Physik, und die Startposition des Balles. Damit die Klasse möglichst einfach verwendet werden kann, kann auch die Größe des Balles angegeben werden. Problematisch ist nämlich, dass jedes Mesh das eingelesen wird, eine unterschiedliche Ausdeh-

[19] Diese Schwerkraft sorgt zum Beispiel dafür, dass ein in der Luft hängendes Netz in der Mitte etwas durchhängt

[20] dies ist für den Spieler allerdings nicht von Belang

nung haben kann. Um nun zwei verschiedene Meshes unterschiedlicher Größe auf die selbe Ausdehnung zu skalieren, müsste man demnach ein kleineres Mesh mit einem anderen Wert skalieren, als ein größeres Mesh. Die Klasse Ball übernimmt diese Funktion automatisch. Man kann also verschiedene Meshes unterschiedlicher Größe übergeben. Solange man aber den selben Wert als Größe festsetzt, werden alle Meshes automatisch auf die selbe Größe skaliert. Damit ist die Festlegung der Größe unabhängig von der ursprünglichen Größe des Meshes.

Aufgrund des übergebenen Meshes erzeugt die Klasse ein Physikobjekt in Form eines Balls. Dieses Physikobjekt hat allerdings nicht dieselbe Komplexität wie das Mesh, sondern verfügt über eine relativ geringe Anzahl von Polygonen. Dies ist für die Simulation eines Balles allerdings ausreichend und verbraucht weniger Rechenzeit als ein komplexeres Modell beanspruchen würde. Das erzeugte Physikobjekt hat zunächst exakt die Größe des übergebenen Meshs. Wird dieses im Nachhinein skaliert muss auch das Physikobjekt entsprechend skaliert werden. Das erzeugte Physikobjekt ist ein Rigid Body, da die Geometrie im Verlauf der Simulation nicht verändert wird, wie dies bei Soft Bodies der Fall ist.

Damit eine an die jeweilige Applikation angepasste Simulation möglich ist, kann über verschiedene Set-Funktionen die Masse, Elastizität, Reibung, sowie das sogenannte „Angular Damping“ und „Linear Damping“ des Physikobjekts eingestellt werden. Es folgt eine kurze Erklärung dieser fünf Attribute:

- *Mass (Masse):* Damit lässt sich einstellen, wie sich verschiedene Objekte bei Kollisionen beeinflussen. Kollidieren in einer Simulation Kugeln unterschiedlicher Masse, so wird eine Kugel niedrigerer Masse von einer Kugel höherer Masse weiter weggestoßen, als eine Kugel mit derselben Masse wie die der kollidierenden Kugel. Dementsprechend verhält sich auch die Kollision zwischen Kugeln [21] und Soft Bodies. Wird die Masse des Soft Bodies (zum Beispiel ein Tischtennisnetz) im Verhältnis zur Kugel erhöht, so reagiert der Soft Body weniger stark durch Veränderung der Geometrie auf die Kollision. Ist die Masse eines Softbody Patch im Verhältnis zur Kugel zu niedrig, so wird die Kugel vom Patch nicht aufgehalten, sondern kann einfach hindurchgehen.

- *Restitution (Elastizität):* Mit Hilfe des Attributs Restitution, also Elastizität, lässt sich zusätzlich anpassen, wie stark ein Ball von einem Kollisionobjekt bei gleichbleibender Masse der beiden Objekte abprallt.

- *Friction (Reibung):* Eigentlich kann man durch das Attribut Reibung

[21] welche, wie erwähnt, aus Rigid Bodies bestehen

bewirken, wie sehr ein Körper in Bewegung von einem Objekt, wie zum Beispiel dem Boden, abgebremst wird. Dafür ist allerdings nötig, dass das Physikobjekt den Boden nicht nur mit der Kante eines Polygons, sondern mit der ganzen Fläche berührt. Dies ist allerdings bei einem Ball bei *Bullet* meist nicht der Fall, da aufgrund von dessen Form der Boden meist nur mit der Kante oder der Ecke zwischen verschiedenen Polygonen des Balles berührt wird. Deshalb wirkt sich Reibung auf das Verhalten des Balles überhaupt nicht aus, was auch durch Tests widergespiegelt werden konnte.

- *Angular Damping:* Da sich Reibung in *Bullet* also nicht auf das Verhalten des Balles auswirkt, wird ein anderes Attribut benötigt, asonsten würde ein am Boden rollender Ball niemals an Geschwindigkeit verlieren und stets weiter rollen. Das Attribut Angular Damping übernimmt nun diese Funktion. Je höher es gesetzt wird, desto stärker verliert ein am Boden rollender Ball an Geschwindigkeit.

- *Linear Damping:* In der realen Welt wird ein geworfener Ball durch den Luftwiderstand leicht abgebremst. Dies kann auch in *Bullet* durch das Attribut Linear Damping realisiert werden.

Mit Hilfe dieser Attribute und der zuvor bereits eingestellten Schwerkraft lässt sich ein relativ realistisches Verhalten von verschiedenen Balltypen simulieren.

Über die Funktion throwBall() lassen sich Bälle mit unterschiedlichen Geschwindigkeiten und Richtungsvektoren von einer Startposition aus werfen. Die Flugbahn des Balles wird zusätzlich von der eingestellten Schwerkraft und dem Attribut „Linear Damping“ (siehe oben) beeinflusst. Mit der Funktion setActivationState() kann man einstellen, ob das Physikobjekt stets aktiviert sein soll oder bei Ruhelage deaktiviert würde. An ein aktiviertes Physikobjekt können stets über manuelle Anweisungen im Programm Kräfte übertragen werden, wie zum Beispiel das Werfen eines Balles auf Knopfdruck, bei dem eine Anfangsgeschwindigkeit gesetzt wird. Ein deaktiviertes Physikobjekt würde auf eine solche Anweisung nicht reagieren. Allerdings reagiert ein deaktiviertes Objekt dennoch auf Kollisionen mit anderen Objekten. In den Applikationen hat es sich bewährt, Objekte wie den vom Spieler gesteuerten Schläger oder einen Ball, der auf Knopfdruck geworfen werden kann stets im aktivierten Zustand zu belassen. Objekte wie Billiardkugeln, die nur auf Kollisionen reagieren müssen, können dagegen bei Ruhelage deaktiviert werden um Rechenzeit zu sparen.

5.3.4 Die Klasse Schlaeger

Die Klasse *Schlaeger* ist eine der Wichtigsten im Programm. Mit ihrer Hilfe kann jede Applikation einen oder mehrere Schläger implementieren, die vom Spieler über die Wiimote gesteuert werden.

Von dieser Klasse erben verschiedene weitere Klassen, um spezialisierte Funktionen der jeweiligen Schläger zu spezifizieren. Deshalb gibt es für Billiardqueue, Tischtennis- und Baseballschläger eigene Unterklassen.

Die Klasse *Schlaeger* benötigt ein von der aufrufenden Applikation festgesetztes Mesh, welches die grafische Repräsentation des Schlägers darstellt. Die Klasse generiert daraus ein Physikobjekt in Form eines Würfels. Für die erstellten Simulationen war diese geometrische Form optimal, um ein glaubwürdiges Kollisionsverhalten bei geringer Rechenzeit zu realisieren. Standardmäßig wird das Physikobjekt exakt auf die Größe des Bounding Volumes des Meshs gesetzt. Beim Tischtennisschläger ergab sich allerdings das Problem, dass das Volumen des Balls zu groß im Verhältnis zum Volumen des Schlägers war. In Bullet führt dies dazu, dass Kollisionen bei schnellen Bewegungen des Schlägers teilweise nicht erkannt werden, weil das Physikobjekt des Balls in keinem Simulationsschritt in das Physikobjekt des Schlägers eindringt. Grundsätzlich gilt, dass die Ausdehnung des Schlägers in allen Dimensionen größer sein sollte als die Ausdehnung des Balls. Deshalb wird das Physikobjekt des Tischtennisschlägers etwas dicker gemacht, als das Mesh des Schlägers, gleichzeitig aber so platziert, dass die Seite, mit der geschlagen wird, auch mit dem Physikobjekt übereinstimmt, die andere Seite des Schlägers jedoch nicht.

Das erzeugte Physikobjekt ist ein Rigid Body, da die Geometrie des Schlägers im Verlauf der Simulation unverändert bleibt. Weiterhin wird das Objekt als kinematisch definiert (siehe Abschnitt 5.1.3). Der Grund liegt darin, dass der Schläger nicht auf die Schwerkraft reagieren soll, sonst könnte er vom Spieler nur unzureichend gesteuert werden. Wenn die Wiimote in Ruhelage gehalten wird, so schwebt der Schläger also in der Luft. Weiterhin werden kinematische Objekte bei Kollisionen mit anderen Rigid Bodies, wie dem Ball, oder Softbodies, wie dem Netz, nicht beeinflusst. Umgekehrt können sie andere Objekte jedoch beeinflussen. Schlägt der Spieler also mit dem Schläger auf einen Ball, so wird der Ball abgestoßen, der Schläger allerdings reagiert weiterhin nur auf die Eingabe des Spielers.

Schlaeger kümmert sich ganz bewusst nicht um die Ermittlung der Transformationsdaten, mit denen das Objekt bewegt werden soll. Diese Aufgabe obliegt der Klasse *SchlaegerControl*. *Schlaeger* bietet aber Funktionen an, um mit Hilfe der Transformationsdaten das Objekt zu bewegen. Der Schläger hängt an einem Transformationsknoten, der die Position und Orientierung

speichert. Mit der Funktion translate() lässt sich der Knoten und damit das Mesh im Raum verschieben. Mit der Funktion rotate() lässt sich der Knoten im lokalen Koordinatensystem drehen. Dies geschieht deshalb im lokalen Koordinatensystem, da auch MotionPlus die Yaw, Pitch und Roll Werte im lokalen Koordinatensystem der Wiimote misst.

Es gibt einen Flag, der zwischen zwei Modi umschaltet. Die Rotation muss nämlich ab und zu neu kalibriert werden, weil manchmal die Orientierung des Objekts nicht mehr mit der tatsächlichen Orientierung der Wiimote übereinstimmt. Der Flag setzt fest, ob gerade kalibriert wird oder nicht. *Schlaeger* verwaltet für die Kalibrierung einen zusätzlichen Knoten, hier als Orientierungsknoten bezeichnet, der mit der Orientierung der Wiimote im Raum übereinstimmen soll. Wenn kalibriert wird, dann wird der Orientierungsknoten neu gesetzt und der Transformationsknoten des Meshs nicht sofort, sondern nach und nach dem Orientierungsknoten angenähert. Der Vorteil dabei ist, dass der Schläger beim Kalibrieren nicht ruckartig neu orientiert wird, sondern in einer weichen Bewegung diese Ausrichtung einnimmt. Die Kalibrierung der Rotation wird genauer beschrieben in den Abschnitten 6.1.1 und 6.1.2. Es gibt 2 Funktionen, die den Orientierungsknoten neu setzen. Eine kalibriert nur den Pitch- und Roll-Winkel, eine andere kalibriert den Yaw-Winkel, da dies nur mit Hilfe der Sensor Bar funktioniert. Wenn der Transformationsknoten wieder mit dem Orientierungsknoten übereinstimmt, wird der Kalibriermodus ausgeschaltet.

Damit das Physikobjekt auch dem Mesh folgt, dafür sorgt die Funktion updateBullet(). Diese ermittelt die aktuelle Transformation des Meshs und weist sie dem Physikobjekt zu. Über verschiedene Getter und Setter Funktionen können physikalische Eigenschaften des Schlägers wie zum Beispiel Restitution (vgl. Abschnitt 5.3.3) gesetzt werden. Die Klasse stellt noch einige Funktionen mehr zur Verfügung, die aber für das Verständnis des Programms und die Steuerung teilweise nicht relevant sind.

5.3.5 Die Klasse SchlaegerControl

Die Klasse *SchlaegerControl* benötigt von dem Skriptprogramm aus Glovepie, das im Hintergrund läuft die nötigen Daten, die von der Wiimote berechnet wurden. Allerdings gibt es keine Möglichkeit, Glovepie direkt in das eigene Programm einzubinden und somit direkten Zugriff auf die Daten zu erhalten. Glovepie bietet jedoch an, die Daten in eine Textdatei zu schreiben, aus der andere Programme dann lesen können. Es ist allerdings von größter Wichtigkeit, dass auf diese Daten möglichst verzögerungsfrei zugegriffen werden kann. Der Vorgang des Schreibens und Lesens einer Textdatei ist allerdings mit erheblichen Verzögerungen verbunden, weshalb sich diese Art der Datenübergabe für das Projekt nicht eignen würde.

Deshalb sucht *SchlaegerControl* nach dem Prozess, der zu dem Programm GlovePie gehört. Innerhalb dieses Prozesses werden dann die jeweils benötigten Daten direkt aus dem Speicher ausgelesen. Dies ist sehr effizient und funktioniert ohne jedwede Verzögerung.

SchlaegerControl kümmert sich vor allem um die Ermittlung der Transformationsdaten. Die Rotationswinkel werden direkt von GlovePie über MotionPlus ermittelt, die Translation muss allerdings mit Hilfe von Algorithmen berechnet werden. Als Eingabe dienen dabei lediglich die Bildkoordinaten, die die Infrarotkameras von dem Infrarotlicht sendenden Controller des Spielers aufgenommen haben. Wie diese Transformationsdaten berechnet werden, wird detailliert in Kapitel 6 beschrieben. Hier wird nur grob darauf eingegangen. *SchlaegerControl* berechnet den Weltpunkt des verfolgten Objekts und aus diesen Daten wiederum, wie weit das Objekt bewegt werden muss. Über eine Setter-Funktion kann entschieden werden, wie schnell der Schläger entlang jeder Achse transliert wird. Wird die Translation entlang der y-Achse dabei auf 0 gesetzt, so kann nur noch auf der $x/z-$Ebene bewegt werden. Die Translation übernimmt dann die Klasse *Schlaeger*.

Mit den von GlovePie ermittelten Rotationswinkeln Pitch, Yaw und Roll wird das Objekt über die entsprechende Funktion der Klasse *Schlaeger* gedreht. Genauer wird dies in Abschnitt 6.1 beschrieben. Wenn der Spieler auf die Infrarotdioden der SensorBar zeigt, wird die Funktion in *Schlaeger* aufgerufen, die den Yaw-Winkel des Objekts in Weltkoordinaten neu kalibriert (siehe Abschnitt 6.1.2). Immer wenn der Schläger relativ schnell bewegt wurde, und er sich danach wieder in Ruhelage befindet, wird der Pitch und Roll-Winkel neu kalibriert, wie in Abschnitt 6.1.1 beschrieben.

5.4 Die Applikationsklassen

Jede Applikation verfügt über eine eigene Klasse, die von *Game* erbt. Die Applikationen werden aus *Main* heraus gestartet. Bei allen Spielen musste auf die Größenverhältnisse der Objekte geachtet werden. Außerdem musste eine angemessene Skalierung der Welt gefunden werden, damit einerseits die Physikengine gut funktioniert und andererseits die Schatten richtig dargestellt werden. Deshalb wurden alle Objekte, aber auch Geschwindigkeiten, wie die der Kamera oder von Objekten mit einem Skalierungsfaktor, den die *Main*-Klasse festsetzt angepasst (vgl. Abschnitt 5.3.1). *Main* stellt außerdem ein Objekt der Klasse Schläger zur Verfügung, dass die jeweilige Applikation schließlich erzeugt. Die Klasse sorgt dafür, dass dieses Objekt mit Hilfe von *SchlaegerControl* oder durch Tastatureingaben bewegt werden kann.

5.4.1 Die Klasse Tischtennis

Die Klasse *Tischtennis* ist für die Funktionalität der Tischtennisapplikation zuständig.

Dazu erzeugt sie zunächst die Beleuchtung und alle statischen Geometrieobjekte, die in diesem Spiel vorkommen, also die Tischtennisplatte und die sie umgebende Sporthalle. Es wird an der Position der Platte ein Physikobjekt erzeugt, das allerdings nur ein einfacher Quader ist, der so positioniert ist, das der Tischtennisball auf der Platte aufspringen kann. Diesem Physikobjekt wird die passende Elastizität[22] (vgl. Abschnitt 5.3.3) zugewiesen, die zusammen mit der Schwerkraft und der Elastizität des Balls für ein realistisches Sprungverhalten des Balls auf der Platte sorgt. Damit die Tischtennisplatte nicht herunterfällt wird sie als *Kinematic Body* definiert (vgl. Abschnitt 5.1.3).

Das Tischtennisnetz wiederum besteht aus einem Geometrieobjekt und einem Softbody-Patch. Über die Klasse *BtOgreSoftbodies* wird in Echtzeit das Geometrieobjekt des Netzes dem Softbody-Patch entsprechend deformiert. *BtOgreSoftbodies* wurde für dieses Projekt erweitert, um diese Funktionalität zu unterstützen . Der Klasse muss ein Pointer auf die Vertices und die Indizes übergeben werden, welche die Polygone des Meshs definieren, damit die Geometrie pro Frame verändert werden kann. Da aus den Geometriedaten allerdings kein Softbody-Patch erstellt werden kann, sondern dieses speziell für diese Applikation manuell erstellt wird, stimmten die Indizes und Texturkoordinaten des Meshs nicht mit denen des Patches überein. Deshalb mussten die Koordinaten des Meshs extra dafür angepasst werden (vgl.: Abschnitt 5.1.3). Abbildung 14 zeigt, wie das Netz auf die Kollision mit dem Ball reagiert.

Weiterhin wird über die Klasse Ball ein Tischtennisball erzeugt, mit dem entsprechenden Physikobjekt und den nötigen Eigenschaften, um ein realistisches Kollisions-, und Flugverhalten zu ermöglichen (vgl.: Abschnitt 5.3.3). Die Eigenschaft Linear Damping erwies sich als besonders wichtig, da es ohne diese schwieriger war, den Ball auf die gegnerische Seite zurückzuschlagen, da er oft über die Platte hinausflog. Durch die Dämpfung ist es letztendlich möglich, dass mit einer größeren Bandbreite von Kräften, mit denen der Spieler gegen den Ball schlägt, dieser auf der gegnerischen Hälfte aufkommt. Da der Ball nur selten auf dem Boden rollt, ist die Eigenschaft Angular Damping nicht wirklich wichtig.

Die Applikation sorgt dafür, dass während dem Spiel in bestimmten zeitlichen Abständen laufend ein neuer Ball zum Spieler in unterschiedlichen

[22] engl.: Restitution

Flugbahnen geworfen wird [23]. Dafür werden für jede dieser Flugbahnen unterschiedliche Anfangspositionen gespeichert, von denen aus geworfen wird sowie passende Vektoren, die die Wurfrichtung- und Geschwindigkeit festsetzen. Ein Zufallsgenerator wählt aus, welche Flugbahn geschlagen wird, der Tischtennisball [24] wird an die Anfangsposition gesetzt und anhand des initialen Vektors geworfen.

Tischtennis erzeugt über ein Objekt des Typs *TTSchlaeger*, das von *Schlaeger* erbt, einen Tischtennisschläger. Es werden 2 Modi angeboten: Einen, bei dem dieser Schläger mit der Klasse *SchlaegerControl* lediglich rotiert, und einer bei dem er zusätzlich auch transliert werden kann. Dies ist wichtig für die späteren Tests, um diese unterschiedlichen Spielabläufe und den Spielspaß bewerten zu können. Bei R+T-Steuerung setzt *Tischtennis* fest, wie schnell der Schläger bewegt werden kann. Wird lediglich die R-Steuerung verwendet, so rotiert der Schläger um einen Punkt, der sich außerhalb des Schlägers befindet. Bei der R+T-Steuerung verhält sich die Sache anders. Weil das retroreflektierende Material (siehe 6.2.2 vorne an der Wiimote angebracht ist, wird beim Rotieren der Wiimote bereits eine Translation erfasst, da sich die Stelle mit dem Material beim Rotieren im Raum bewegt. Der Rotationspunkt wird deshalb bei der R+T-Steuerung weiter vorne angesetzt, um dies auszugleichen.

Problematisch war in dieser Applikation das Verhältnis zwischen der Dicke der Schlägerfläche, dem Durchmesser des Balls und den Geschwindigkeiten, mit der diese Objekte kollidieren. Da die Durchmesser beider Körper recht klein sind, diese aber mit hoher Geschwindigkeit kollidieren, wurde oftmals keine Kollision festgestellt (vgl. Abschnitt 5.1.3 unter Rigid Bodies). Man hätte demnach die Zahl der Berechnungsschritte pro Sekunde extrem erhöhen müssen, wodurch die Framerate in starke Mitleidenschaft gezogen worden wäre. Um dies zu umgehen, wurde stattdessen das Volumen des Physik-Quaders, das den Schläger umgibt vergrößert. In Abbildung 16 ist mit Hilfe des Debug Drawers zu erkennen, wie das Physikobjekt positioniert wurde, so dass dennoch mit der Schlägerfläche geschlagen werden kann. Die Dicke des Schlägers wird einfach auf der Schlägerseite, die gerade nicht mit dem Ball kollidieren kann, vergrößert. Wenn der Schläger gedreht wird, so erkennt das Programm, dass höchstwahrscheinlich mit der anderen Schlägerseite geschlagen wird, und das Physikobjekt wird entsprechend im Verhältnis zum Geometrieobjekt verschoben.

[23] es sieht aus als würde der Ball geschlagen, jedoch ist kein gegnerischer Schläger sichtbar

[24] Es wird keine neuer Ball erzeugt, sondern es handelt sich stets um dasselbe Geometrie- und Physikobjekt

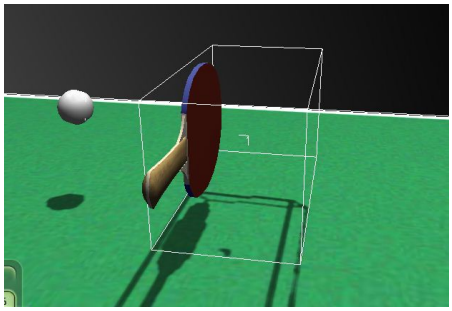

Abbildung 16: Das Physikobjekt des Schlägers wurde vergrößert, damit Kollisionen stets erkannt werden.

5.4.2 Die Klasse Billiard

Die Klasse *Billiard* ist für die Billiardapplikation zuständig. Sie erzeugt anfangs die Beleuchtung und die statischen Objekte, wie zum Beispiel eine Bar die den Billiardtisch umgibt, inklusive Theke, Tische und Stühle (siehe Abbildung 18). Der Billiardtisch besteht aus einem relativ detaillierten Mesh (siehe Abbildung 19) und einem weniger komplexen Physikobjekt, einem Kinematic Body. Letzteres wird aus einem in Maya erzeugten Objekt generiert, das dem Billiardtisch angepasst wurde. Überall dort, wo eine Kugel am Tisch kollidieren kann, wurde die Geometrie recht komplex gewählt. An Stellen wie zum Beispiel der Außenseite des Tisches, wurde sie extrem vereinfacht, um keine Framerates einzubüßen. Damit die Kugeln in die 6 Löcher des Tischs fallen können, und danach nicht mehr sichtbar sind, wurde die Physikgeometrie so gestaltet, dass die Kugeln unterhalb des Tischbodens im Objekt zum liegen kommen. Außerdem wurde eine unsichtbare Bande erzeugt, die verhindert, dass Kugeln aus dem Tisch herausgeschlagen werden können. Das Physikobjekt wurde in 3 Objekte aufgeteilt. Eines für den Billiardboden, eines für die Bande und eins für die Begrenzungen der 6 Löcher. Der Grund dafür ist, dass diese Bereiche alle eine unterschiedliche Elastizität benötigen, und man einem einzigen Physikobjekt nur einmal diesen Wert zuweisen könnte. Die Bande benötigt nämlich eine sehr hohe Elastizität, damit die Kugeln wie beim echten Billiard daran abprallen, der Boden und besonders die Löcher benötigen gar keine Elastizität, andernfalls wäre es sehr schwer die Kugeln einzulochen, wenn diese abprallen würden.

Weiterhin wird ein Billiardqueue durch die Klasse *BilliardQueue* erzeugt, die von *Schlaeger* abgeleitet ist. Mit dem Queue soll die weiße Kugel gestoßen werden können. Das den Queue umgebende Physikobjekt ist auch

[25] mit Maya gerendert
[26] mit Maya gerendert

Abbildung 17: Billiardtisch von oben[25]

Abbildung 18: Diese Bar umgibt den Billiardtisch[26]

in diesem Fall ein einfacher Quader und Kinematic Body. Es gab auch hier Probleme mit den Größenverhältnissen dieses Quaders zu den Kugeln, sodass er in der Breite etwas dicker gemacht wurde, als der eigentlich sichtbare Queue.

Billiard ermöglicht, bei Druck der *A*-Taste den Zielmodus zu aktivieren. Dabei wird der Queue in geringem Abstand vor die weiße Kugel platziert. Durch Rotieren der Wiimote kann nun gezielt werden, dabei rotiert der Queue nach links und rechts um die Kugel herum, allerdings nicht nach oben und unten, und zielt dabei stets auf die weiße Kugel. Die Kamera bleibt dabei stets hinter dem Queue um eine gute Sicht beim Zielen zu ermöglichen. Während man sich im Zielmodus befindet, wird das Physikobjekt ausgeschaltet, damit man nicht andere Kugeln mit dem Queue aus Versehen wegbewegen kann, während dieser sich um die Kugel dreht. Wird die Taste losgelassen, kann der Spieler den Queue nun nach vorne und hinten stoßen, allerdings wird dabei nur das Mesh bewegt. Erst wenn der Spieler kurz die B-Taste gedrückt hat, wird auch das Physikobjekt bewegt und ein Stoß gegen die weiße Kugel ist möglich. Der Grund dafür ist,

Abbildung 19: Oben: Die Positionierung der Kamera im Zielmodus. Unten: In diesem Modus kann gestoßen werden, die Kamera rückt etwas nach oben.

dass es vorkommen kann, das man, sobald man aus dem Zielmodus herausgegangen ist, die Wiimote nach vorne bewegt und aus Versehen einen Stoß ausführt, der so nicht geplant war. Prinzipiell kann der Spieler den Stoß gegen die Kugel auch erstmal üben, und erst wenn er bereit ist, drückt er B um zu stoßen.

Beim Stoß wird keine Rotation erlaubt, und die Translation wird lediglich in der Z-Ebene ermöglicht. Es kann also nur nach vorne und hinten gestoßen werden.

Weiterhin erzeugt *Billiard* alle Kugeln, die es auch beim richtigen Billiard gibt, also 7 volle für das erste Team, 7 halbe für das zweite, eine schwarze Kugel und eine weiße, die gestoßen werden soll. Alle Kugeln werden durch die Klasse *Ball* erzeugt und haben dieselben physikalischen Eigenschaften, insbesondere die gleiche Masse und Elastizität. Letztere entscheidet darüber wie stark die Kugeln voneinander abprallen. Wenn eine Billiardkugel auf eine andere, die sich in Ruhestellung befindet, trifft, geht relativ wenig Bewegungsenergie verloren, sprich die getroffene Kugel erhält beinahe dieselbe Geschwindigkeit mit der die erste aufgetroffen ist. Die erste Kugel bleibt dagegen fast stehen. Um dies zu ermöglichen, wurde die Elastizität

relativ hoch gesetzt. Außerdem verfügen die Kugeln über etwas Angular Damping (vgl. Abschnitt 5.3.3), damit sie nicht ewig weiter rollen, sondern langsam durch die Reibung auf dem Boden des Tisches abgebremst werden.

Die physikalischen Eigenschaften des Tisches, der Kugeln und des Queues mussten also exakt angepasst werden. Dabei wurden zahlreiche Tests unternommen, wobei auf ein einem echten Billiardspiel nachempfundenes Kollisionsverhalten geachtet wurde. Auch der Tisch bekam relativ viel Elastizität zugewiesen, denn bei einem echten Billiardspiel verlieren die Kugeln beim Abprallen an der Bande kaum Bewegungsenergie.

Wenn die weiße Kugel aus Versehen in einem Loch versenkt wird, wird diese automatisch wieder auf dem Tisch platziert, sobald der Spieler die A-Taste für den Zielmodus drückt. Probleme gab es teilweise mit den Schatten. Diese konnten aber gelöst werden, wie in Abschnitt 5.5 beschrieben wird.

5.4.3 Die Klasse Baseball

Die Klasse *Baseball* ist für das Baseballspiel verantwortlich.Dabei wird auch hier zunächst die komplette Szene erzeugt. Erwähnenswert ist dabei eine Art Korb, in den der Spieler mit dem Schläger Bälle schlagen kann. Der Korb besteht aus 3 Patches an den Seiten und einem Patch der den Boden bildet. Die Patches wurden ähnlich erzeugt wie das Tischtennisnetz (vgl. Abschnitt 5.4.1 und 5.1.3). Die Schwerkraft, welche nur für die Softbodies gilt, wurde so gesetzt, dass diese etwas in der Mitte durchhängen.

Der Baseball, und das zugehörige Physikobjekt wird über die Klasse *BaseballBat* erzeugt, die von *Schlaeger* erbt. Auch beim Baseball gibt es einen Modus mit R- und einen mit R+T-Steuerung. Ersterer spielt sich ähnlich wie einige Spiele von WiiSports Resort (siehe Einleitung 1), die nur auf Rotationen der Wiimote basieren. Bei der R+T-Steuerung legt die Klasse zuvor fest, wie schnell der Schläger bewegt werden kann. Wenn der Spieler die A-Taste drückt, wird ein Ball in Richtung des Schlägers geworfen.

Das Spiel registriert, ob mit dem Ball ein Netz getroffen wurde. Dies wird durch Bounding Volumes ermöglicht. Dabei wird getestet, ob sich der Ball, der geschlagen wurde, innerhalb oder außerhalb des Bounding Volume, das den Korb umgibt, befindet. Diese Tests sich recht einfach durchzuführen. Wenn der Spieler eine bestimmte Trefferzahl erreicht hat, wird eine höhere Schwierigkeitsstufe gestartet. Der Ball, der nun geschlagen wird, sowie das Netz, werden etwas kleiner, was weitere Treffer erschwert. Dazu wird das Objekt des Netzes gelöscht und ein neues erzeugt.

5.5 Darstellung von Schatten

Schatten sind in den Applikationen von nicht zu unterschätzender Relevanz. Da der Spieler bespielsweise einen Tischtennisschläger frei im Raum bewegen kann, muss er ein Gefühl dafür entwickeln können, wo sich dieser im Raum befindet. Wenn der Schläger nun einen Schatten auf die Platte wirft, kann besser abgeschätzt werden, wie hoch sich der Schläger über der Platte befindet. Es entsteht auch ein besseres Gefühl für Tiefe, man kann also die Entfernung des Objekts besser abschätzen, wie zum Beispiel auf Abbildung 8 zu erkennen ist. Ogre unterstützt unter anderem Stencil Shadows, die relativ leicht eingebunden werden können. In meiner Engine verwende ich zur Schattenberechnung jedoch das Shadowmappingverfahren und eine Filtermaske, die weiche Schatten, sogenannte Softshadows berechnet, die insgesamt schöner aussehen. Diese Art von Schatten wird von Ogre nicht offiziell unterstützt, sondern stellt eine Erweiterung durch einen Shader dar, der erst eingebunden werden musste. Der Shader musste erst für die entsprechenden Applikationen optimiert werden, um visuell ansprechende Schatten darstellen zu können. Problematisch war teilweise das Verhältnis der Größe der Welt zu den Einstellungen im Shader. Wenn zum Beispiel bei einer eher kleinen Welt zu große Objekte in der Nähe waren, warfen plötzlich andere Objekte keinen Schatten mehr. Dies hat mit Details im Shader zu tun, auf die hier allerdings nicht eingegangen werden soll. Um in allen Applikationen Schatten anzeigen zu können, mussten einige Einstellungen im Shader angepasst und teilweise die 3D-Welt auf eine Größe skaliert werden, in der dann alle Schatten sichtbar waren. Es gab auch das Problem, das Objekte, die sehr nah am Boden sind, einen weniger dunklen Schatten werfen. Der Grund war ein Offset-Floatwert, der bestimmte Schattenartefakte verhindern sollte. Man konnte diesen Wert aber entsprechend anpassen um das Phänomen zu verhindern. Leider waren teilweise Treppenstufen an den Kanten der Schatten sichtbar, die von der geringen Auflösung der Schattentextur herrühren. Es gibt Grafikkarten, die diese Pixel automatisch interpolieren, aber auf dem Rechner, auf dem das Programm ausgeführt wurde, wird dies nicht unterstützt. Das Problem konnte verringert werden, wenn man die Filtermaske erhöhte, allerdings ging dies auch zu Lasten der Framerate.

6 Umsetzung der 6 Achsen Steuerung

In diesem Kapitel wird detailliert erläutert, wie die freie Bewegungssteuerung im Projekt letztendlich umgesetzt wurde.

6.1 Umsetzung der Rotation mit Wii-MotionPlus

MotionPlus berechnet, wie in Abschnitt 2.2 beschrieben, in regelmäßigen Zeitabständen die Drehrate und den Drehwinkel der Rotationen bezogen auf das lokale Koordinatensystem der Wiimote. Die Rotationen werden dabei in den 3 Achsen Yaw, Pitch und Roll im Winkelmaß gemessen, die in Abbildung 3 zu sehen sind. Wenn das GlovePie-Skript gestartet wird und die Wiimote sich in Ruhelage befindet, so haben Yaw, Pitch und Roll zunächst den Wert 0. Dreht man die Wiimote, so werden die gemessenen Drehwinkel laufend auf die alten Werte addiert. Die Klasse *Schlaegercontrol* fragt pro Frame die aktuellen Winkel für Yaw, Pitch und Roll ab. Dann wird die Differenz zwischen den aktuellen Drehwinkeln und den Drehwinkeln des letzten Frames gebildet. Somit erhält man die Winkel in den 3 Achsen, mit denen das Objekt für den aktuellen Frame rotiert werden muss. Die Rotation erfolgt dann mit Hilfe der Funktion rotate() der Klasse *Schlaeger*.

6.1.1 Kalibrieren des Pitch und Roll-Winkels bei ruhigen Bewegungen

Zwar werden die Rotationsgeschwindigkeiten und Drehwinkel mit Hilfe von MotionPlus sehr genau berechnet. Allerdings erkennt MotionPlus während dieser Bewegungen nicht die tatsächliche Orientierung der Wiimote im Raum. Es werden lediglich aus der Orientierung des letzten Frames und den aktuell gemessenen Rotationen die neue Ausrichtung berechnet. Wenn die Orientierung im letzten Frame allerdings schon falsch war, so wird auch die neu berechnete Ausrichtung falsch sein. Außerdem reagieren die Sensoren bei sehr schnellen Bewegungen manchmal nicht richtig, wodurch falsche Rotationsgeschwindigkeiten berechnet werden.

Es ist deshalb nötig, zu Beginn und in regelmäßigen Zeitabständen die tatsächliche Orientierung der Wiimote zu berechnen, und die Ausrichtung des Objekts dementsprechend wieder anzugleichen. Dies bezeichnet man als Neukalibrierung. Man unterscheidet auch hierbei die Rotationswinkel Yaw, Pitch und Roll, allerdings arbeitet man dabei meist in Weltkoordinaten. Yaw bezeichnet eine Rotation nach links und rechts in Weltkoordinaten, Pitch eine Rotation nach oben und unten in Weltkoordinaten. Roll wird allerdings in lokalen Koordinaten der Wiimote angegeben. Für die Berechnung der Pitch- und Roll-Werte ist MotionPlus nicht nötig, dies erledigt

die Wiimote. Aufgrund der Schwerkraft, kann sie Pitch erkennen, wenn die Wiimote ruhig gehalten wird. Die Schwerkraft wirkt auf die Wiimote wie eine Beschleunigung nach unten, und die Wiimote registriert, wohin diese Beschleunigung in ihrem lokalen Koordinatensystem wirkt. Wirkt die Schwerkraft zum Beispiel entlang der Längsachse der Wiimote, so bedeutet das, dass der Spieler mit dem Controller nach oben oder nach unten zeigt [27]. Auf diese Weise kann die Wiimote auch den Rollwinkel berechnen, dieser wird allerdings im lokalen Koordinatensystem der Wiimote angegeben. Roll kann aber nur berechnet werden, wenn der Spieler nicht direkt nach oben oder unten zeigt. In diesem Fall würde nämlich die Schwerkraft entlang der Längsachse zeigen, und beim Rollwinkel wird ja um die Längsachse gedreht. Wenn die Drehachse also mit der Richtung der Schwerkraft übereinstimmt, könnte der Winkel nicht mehr ermittelt werden. Der Spieler sollte zum Kalibrieren deshalb möglichst nicht nach oben oder unten, sondern am besten nach vorne zeigen.

Die Klasse *ArmControl* prüft, ob der Controller sehr schnell bewegt wurde. Wenn dies passiert ist, ist die Gefahr gegeben, dass die Orientierung nicht mehr stimmt. In diesem Fall wird gewartet, bis sich die Wiimote in Ruhelage befindet und der Spieler dabei nicht zu stark nach oben oder unten zeigt. Erst dann ist eine exakte Berechnung von Pitch und Roll möglich. Diese Werte werden von GlovePie abgefragt und mit Hilfe der Kalibrierfunktion der Klasse *Schlaeger* wird das Objekt langsam neu orientiert, wie in Abschnitt 5.3.4 beschrieben.

Wie in Abschnitt 2.2 beschrieben, kann MotionPlus eigentlich mit unterschiedlichen Sensoren bei unterschiedlichen Rotationsgeschwindigkeiten arbeiten. Benutzt man MotionPlus aber unter GlovePie, dann ist nur ein Sensor aktiv. Aus diesem Grund ist es öfter nötig, neu zu kalibrieren, weil die Sensoren bei sehr schnellen Bewegungen ungenau arbeiten.

6.1.2 Kalibrieren des Yaw-Winkels mit der Sensorbar

Nachdem nun die Kalibrierung der ersten zwei Rotationsachsen, Pitch und Roll, gelöst wurde, ist zusätzlich wichtig, von Zeit zu Zeit die dritte Achse, nämlich Yaw, in Weltkoordinaten zu kalibrieren. Dieser Winkel ändert sich beispielsweise, wenn der Spieler eine Drehung nach rechts und links macht. Der Yaw-Winkel kann mit MotionPlus oder den Beschleunigungssensoren der Wiimote allerdings nicht berechnet werden. Wie bereits erwähnt, bestimmt die Wiimote den Neigungs- und Rollwinkel mit Hilfe der Schwerkraft. Die Schwerkraft kann aber nicht dafür genutzt werden, zu

[27] Die Kraft entlang der Längsachse kann schließlich in 2 verschiedene Richtungen zeigen, zeigt sie in die negative Richtung, dann zeigt der Spieler mit dem Controller nach oben

prüfen, ob der Spieler nach rechts oder links zeigt. Um Yaw zu kalibrieren ist also eine andere Herangehensweise nötig.

Ich habe mich in meiner Applikation dafür entschieden, die Sensorbar (siehe Abschnitt 2.3) für die Kalibrierung zu verwenden. Dieses Gerät liegt auf dem Monitor oder Fernseher und kann entweder von der Wii-Konsole betrieben werden, oder man verwendet eine Sensorbar, die mit Batterien betrieben wird. Wenn die IR-Kamera der Wiimote beide Infrarotpunkte der Sensorbar zur gleichen Zeit erfasst, so greift die Pointerfunktion. Diese ist mit der Funktion eines Mauszeigers vergleichbar. GlovePie fragt also den x-Wert der Pointerfunktion ab. Zeigt man mit der Wiimote in Richtung der Sensorbar aber möglichst weit nach links, so liegt der Wert bei 0. Nun kann man die Wiimote bis zu einem Winkel von ca. 40 Grad nach rechts drehen um den maximalen x-Wert zu erreichen. Der Bereich, in dem die Pointerfunktion greift, hat also einen Öffnungswinkel von 40 Grad. Der Yaw-Winkel kann demnach allein aus dem x-Wert der Pointerfunktion berechnet werden. Zeigt der Spieler genau in die Mitte der Sensorbar, wird der Schläger im Spiel so ausgerichtet, dass er genau nach vorne, entlang der Z-Achse zeigt. Zeigt man weiter nach links, wobei die Sensorbar immer noch im Blickfeld ist, wird ein entsprechender Yaw-Winkel berechnet. Die Klasse *Schlaeger* übernimmt die Berechnung des Yaw-Winkels aus den x-Koordinaten der Pointerfunktion. Daraufhin wird der Schläger nach und nach der neu berechneten Orientierung angenähert.

In einer älteren Version wurde ein Algorithmus geschrieben, der aus den Bildkoordinaten der gemessenen Infrarotpunkte berechnet, wie weit der Spieler nach links oder rechts zeigt. Später wurden diese etwas aufwändigen Berechnungen durch die Pointerfunktion ersetzt, die ganz einfach von der Wiimote abgefragt werden kann. Die Idee, den Yaw-Winkel auf diese Art zu kalibrieren entstammt Beobachtungen, die ich im Spiel Wii-Sports Resort gemacht habe. Auch hier wurde bei der Schwertsimulation das Schwert immer dann wieder sehr genau in Richtung des Bildschirms ausgerichtet, wenn der Spieler auf den Fernseher, und damit auf die Sensorbar zeigte.

6.2 Stereotracking

Nachdem die Umsetzung von Rotationen geklärt wurde, soll nun die Umsetzung der Translation der Wiimote erläutert werden. Um diese Translation in die jeweilige Applikation zu übertragen, ist zunächst einmal nötig, pro Frame die Position der Wiimote, die vom Spieler gesteuert wird, zu bestimmen. Dies kann durch das sogenannte Stereotracking realisiert werden. Hierbei werden 2 Kameras verwendet, die an unterschiedlichen Positionen aufgestellt sind und denselben Punkt, also die Wiimote, verfolgen.

Die Kameras liefern dabei unterschiedliche Bildpositionen, an denen der Punkt wahrgenommen wurde. Das sogenannte Rekonstruktionsproblem bezeichnet die Aufgabe, aus den beiden korrespondierenden Bildpunkten die Weltkoordinaten des gesuchten Punktes zu berechnen. Die Lösung dieses Problems, also die Berechnung des Weltpunktes, wird als Triangulierung bezeichnet [14]. Man unterscheidet zwischen achsenparallelem und konvergentem Stereotracking.

Zunächst wird auf die eingesetzten Kameras eingegangen, und erläutert, wie die Wiimote von den Kameras erkannt werden kann. Dann wird das achsenparallele und schließlich das konvergente Stereotrackingverfahren erläutert. Danach wird erklärt, wie das Stereotracking letztendlich ins Projekt eingebunden wurde.

6.2.1 Die Kameras

Als Kameras dienen in diesem Projekt die Infrarotkameras zweier weiterer Wiimotes. Warum wurde hier wieder auf Wiimotes zurückgegriffen? Zum einen, weil alle Gerätschaften für das Projekt möglichst aus dem Bereich der Wii-Konsole stammen sollen und nicht extra neue Kameras angeschafft werden sollen, wenn diese ohnehin schon in Form der Wiimote gegeben sind. Außerdem ist der große Vorteil der Wiimote, dass es sich um Infrarotkameras handelt. Diese nehmen lediglich Infrarotlicht wahr und filtern alle anderen Lichtquellen aus. Wenn also ein Objekt verfolgt werden soll, so muss dieses lediglich auf irgendeine Weise Infrarotlicht aussenden und die Wiimotes geben die Bildkoordinaten des aufgenommenen Lichtpunktes aus. Damit entfällt der komplette Bildverarbeitungsanteil, der bei den meisten Trackingverfahren nötig wäre. Würden die Kameras keine exakten Bildkoordinaten des Punkts, sondern stattdessen ein normales Kamerabild ausgeben, so müsste in diesem Bild zunächst das zu verfolgende Objekt mittels Bildverarbeitungsalgorithmen gesucht werden. Dieser Aufwand entfällt völlig. Weitere Informationen zu den Infrarotkameras werden in Abschnitt gegeben.

Bei Verwendung der Wiimote bei der Wii-Konsole ist es möglich, die Sensitivität der Kameras einzustellen. Bei GlovePie wird solch eine Einstellung leider nicht unterstützt. Es wäre interessant gewesen, ob sich dadurch das Tracking nochmal verbessern ließe.

6.2.2 Die Wiimote als Infrarotlichtquelle

Damit die Kameras die Wiimote des Spielers verfolgen können, muss diese auf irgendeine Art und Weise Infrarotlicht aussenden. Die Wiimote sendet

allerdings im Originalzustand kein Infrarotlicht aus. Dies ist ein weit verbreiteter Irrtum. Die Wiimote kann Infrarotlicht erkennen, aber nicht aussenden. Es wurden verschiedene Möglichkeiten ausprobiert, aus der Wiimote eine Infrarotlichtquelle zu machen.

Methode 1: Infrarotdioden an der Wiimote Eine Möglichkeit ist, Infrarotdioden direkt an die Wiimote anzubringen. Dies hat zwar den Vorteil, dass die IR-Dioden direkt von den IR-Kameras wahrgenommen werden und so kein Licht auf dem Umweg zu einer Reflektionsquelle verlorengeht. Auch ist der Stromverbrauch recht gering, da weniger Dioden als in Möglichkeit 2 (siehe unten) benötigt werden.

Es ergeben sich aber schwerwiegende Probleme: Da der Öffnungswinkel des Lichts der Dioden recht klein ist, sind ca. 10 Dioden nötig, die so an der Wiimote platziert werden müssen, dass immer mindestens eine Diode wahrgenommen werden kann, die möglichst direkt auf die Kameras zeigt. Das Anbringen der Dioden an die Wiimote gestaltet sich aber sehr schwierig, da sowohl Batterien als auch Widerstände und Dioden, sowie ein An/Aus-Schalter angebracht werden müssen. Es hat sich beim Test gezeigt, dass zu viele an der Wiimote angebrachte Gerätschaften beim Spielen recht störend wirken, da diese im Weg sind und sich der Controller nicht mehr so angenehm greifen lässt. Man kann die Dioden zwar an kleinen Lochplatinen verlöten, um das Ganze aufgeräumt wirken zu lassen, aber dennoch wären zu viele Kabel im Weg. Ein weiteres großes Problem ist, dass die Dioden aus größerer Entfernung relativ schlecht erkann werden und das Tracking hier teilweise ausfallen würde. Übertragen auf die jeweilige Applikation äußert sich das dann in einem Ruckeln des Objekts, da die Dioden manchmal kurz erkannt werden, dann aber wieder nicht.

Methode 2: Infrarotarray und retroreflektierendes Material Aufgrund der genannten Probleme wurde eine andere, weitaus besser funktionierende Variante verwendet. Dabei werden 2 Infrarotarrays gebaut. Die Arrays bestehen aus einer bestimmten Anzahl von Infrarotdioden, die auf eine Lochplatine gelötet wurden. In der Mitte des Arrays befindet sich ein Loch, durch welches die Wiimote blicken kann. Das Array sendet also relativ viel Infrarotlicht aus. An die Spitze der Wiimote wird nun ein Aufsatz angebracht, dessen Oberfläche aus retroreflektierendem Material besteht. Dieses Material reflektiert das Infrarotlicht zur Lichtquelle, also zu den Arrays und damit auch zu den Infrarotkameras der beiden anderen Wiimotes, zurück. Abbildung 20 zeigt den groben Aufbau.

Diese Methode bringt viele Vorteile. Es müssen außer dem Aufsatz keinerlei Dioden oder weitere Gerätschaften an die Wiimote angebracht werden.

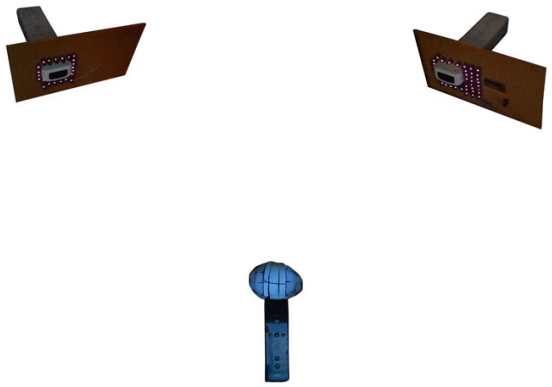

Abbildung 20: Die Kameras blicken durch Infrarotarrays, der Aufsatz auf der schwarzen Wiimote reflektiert das Infrarotlicht zu den Kameras zurück

Man kann für den Aufsatz verschiedene Formen festlegen und testen welche am besten verfolgt werden kann. Ein großer Vorteil ist, dass man die Größe des Aufsatzes festsetzen kann, und damit die Größe der Oberfläche, die Licht reflektiert. So kann man sehr leicht dafür sorgen, dass auch aus größerer Entfernung das Objekt noch gut verfolgt werden kann.

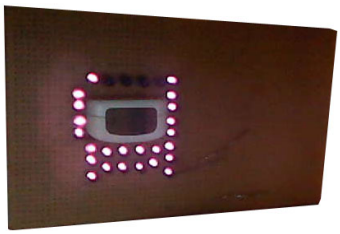

Abbildung 21: Ein Array im angeschalteten Zustand. Die Wiimote blickt durch ein Loch in der Mitte des Arrays

Das InfrarotArray Abbildung 21 zeigt eines der Infrarotarrays im angeschalteten Zustand. Hier wurden lediglich 24 Dioden verbaut, wobei jeweils 8 Dioden mit einem Widerstand in Reihe, und 3 Reihen parallel geschaltet wurden. Das Array für die zweite Kamera ist ähnlich aufgebaut. Betrieben werden beide Arrays mit einem 12 Volt Adapter. Da die Schaltung ideal auf die Dioden abgestimmt ist, strahlen diese sehr hell[28]. Die Kamera, mit der das Bild aufgenommen wurde, kann die Infrarotstrahlung wahrnehmen, das menschliche Auge jedoch nicht. Somit kann das Infrarolicht den Spieler auch nicht ablenken. Man hätte noch mehr Infrarotdioden verbauen können, aber diese haben in Verbindung mit dem reflektierenden

[28]Die Schaltung wurde mit Hilfe der Webseite http://ledcalculator.net/ berechnet

Aufsatz vollkommen ausgereicht, um ein robustes Tracking zu verwirklichen.

Die Infrarotdioden Die für die Arrays verwendeten Infrarotdioden sind von der Marke Vishay TSAL 6400. Diese wurden von Johhny Chun Lee und vielen Besuchern des Forums WiimoteProject.com als ideal für die Verwendung mit der IR-Kamera der Wiimote angepriesen. Letztere verfügt über einen Filter, der hauptsächlich Infrarotlicht im Wellenlängenbereich um 940nm [2] durchlässt. Die Dioden haben exakt diese Peakwellenlänge und sind von daher ideal geeignet. Die Dioden verfügen über einen Abstrahlwinkel von 50 Grad, was für IR-Dioden relativ hoch ist. Allerdings strahlen sie nur direkt nach vorne sehr stark. Um den Abstrahlwinkel zu erhöhen kann man die Köpfe der LEDs mit Schleifpapier anrauhen. Allerdings war dies nicht nötig, da das Tracking mit dem Array ohnehin sehr gut funktioniert hat.

Abbildung 22: Infrarotdiode der Marke Vishay TSAL 6400, die für das Infrarotarray verwendet wurden

Das retroreflektierende Material Retroreflektion bedeutet, dass die Reflektoren das meiste Licht direkt zur Lichtquelle zurückreflektieren, unabhängig von dem Winkel, in dem es auf die Reflektoren auftrifft [27]. Im Straßenverkehr dienen retroreflektierende Folien zum Beispiel der Sicherheit. Da der Autofahrer nicht weit von seiner eigenen Lichtquelle entfernt sitzt, kann er das reflektierte Licht der Reflektorfolien gut sehen.

Es ist leicht vorstellbar, dass Spiegel vollkommen ungeeignet wären, da sie das Licht nur in eine bestimmte Richtung reflektieren. Da Reflektoren aber zur Lichtquelle reflektieren, kann in deren Nähe auch das gesamte reflektierende Objekt erkannt werden. Je größer dieses Objekt, desto leichter kann es erkannt werden. Das Material lässt sich im Gegensatz zu Infrarotdioden auf verschiedene Arten sehr leicht an die Wiimote anbringen.

Zunächst wurde für dieses Projekt selbstklebende reflektierende Folie der Firma 3M getestet. Allerdings konnte diese das Infrarotlicht nicht besonders gut reflektieren und spiegelte teilweise zu stark. Letzten Endes wurde dann einfaches reflektierendes Klettband verwendet, das eigentlich zum

Anbringen an Kleidung gedacht ist, um nachts gut gesehen werden zu können. Dieses Klettband konnte in einem gewöhnlichen Fahradgeschäft bezogen werden. Es eignete sich für das Tracking hervorragend. Abbildung 23 zeigt dieses Material.

Abbildung 23: Das verwendete reflektierende Material

Aufbau eines reflektierenden Aufsatzes Damit die Wiimote das Infrarotlicht zu den beiden Arrays und damit auch zu den Kameras zurückstrahlen kann, muss das reflektierende Material irgendwie an die Wiimote angebracht werden. Dabei wurden zahlreiche Möglichkeiten ausprobiert. Zunächst wurde versucht, das Material an die Spitze der Wiimote anzukleben, so dass mehrere Seiten abgedeckt werden. Das Problem dabei ist, dass das Material keine runde Form einnimmt, sondern eher eine würfelförmige. Damit kann man die Wiimote auch so halten, dass das Infrarotlicht an keiner Stelle senkrecht auf eine Seite des Würfels auftrifft. Allerdings reflektiert das Material immer noch am besten, wenn das Licht senkrecht auftrifft. Das Resultat war deshalb, dass das Tracking manchmal ausfiel um dann abrupt wieder zu funktionieren. Ein großes Problem ist auch, dass der Spieler mit der Wiimote nicht nach hinten ausholen kann, da dann seine Finger die Sicht der Kameras auf das Material verdecken.

Deshalb wurde ein Aufsatz aus Styropor geformt, der vorne auf die Wiimote aufgesteckt werden kann. Auf diesen Aufsatz wurde das Material geklebt. Es zeigte sich, dass ein großer Aufsatz, der viel Fläche für das Material bietet, aus großer Entfernung weitaus besser erkannt wird als ein kleiner. Zunächst wurde ein kugelförmiger Aufsatz getestet, was schon recht gut funktioniert hat. Das beste Ergebnis brachte aber ein eiförmiger Aufsatz. Der große Vorteil war, dass man mit der Wiimote weit ausholen konnte. In diesem Fall zeigt der Controller auf die von den Kameras abgewandte Seite. Der Aufsatz konnte aber immer noch sehr gut erkannt werden, da er groß genug war um nicht von den Fingern verdeckt zu werden. Weiterhin bietet das Ei den Vorteil, dass man es in beinahe allen Orientierungen halten kann und stets ein Bereich der Fläche senkrecht zu den Lichtstrahlen des Arrays steht und somit sehr stark reflektiert. Abbildung 24 zeigt den an der Wiimote angebrachten Aufsatz. Da es sich lediglich um Styropor und reflektierendes Band handelt, ist er zudem sehr leicht.

Im Folgenden Verlauf des Kapitels werden die verschiedenen Trackingverfahren beschrieben.

Abbildung 24: Dieser Aufsatz mit dem reflektierenden Material wurde auf der Wiimote angebracht. Aufgrund der Größe und der Form eines Eies kann der Aufsatz sehr gut auch aus ungünstigen Winkeln verfolgt werden.

6.2.3 Achsenparalleles Stereotracking

Beim achsenparallelen Tracking werden die beiden Kameras exakt parallel zueinander ausgerichtet. Hierbei ist die anschließende Berechnung der Triangulierung stark vereinfacht. Abbildung 25 zeigt den schematischen Aufbau beim achsenparallelen System. Auf der linken Seite befindet sich Kamera 1, auf der rechten Kamera 2.

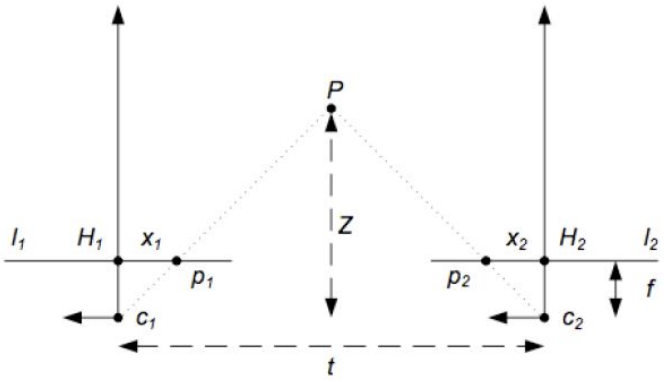

Abbildung 25: Rekonstruktionsproblem bei parallelen optischen Achsen [15]

Im Folgenden werden die Bezeichnungen erläutert:

- c_1, c_2 : Kamerazentren der beiden Kameras
- H_1, H_2 : Hauptpunkte
- *f:* Brennweite der Kameras

- *t:* Abstand zwischen den Kamerazentren
- p_1, p_2 : korrespondierende Bildpunkte in den Bildebenen
- x_1, x_2, y_1, y_2 : Koordinaten der korrespondierenden Bildpunkte in x- und y-Achse

Die Anwendung des Strahlensatzes führt zu

$$\frac{Z}{f} = \frac{t}{(x_1 - x_2)}$$

$x_1 - x_2$ ist die Disparität d.

Daraus folgt:

$$Z = f\frac{T}{d}$$

Für die restlichen Parameter gilt:

$$X = \frac{Z}{f}x_1$$

$$Y = \frac{Z}{f}y_1$$

Die Disparität d wird kleiner, wenn sich das Objekt von den Kameras entfernt. Umgekehrt steigt aber die Genauigkeit der Berechnungen, wenn sich das Objekt den Kameras nähert, da eine größere Disparität eine genauere Berechnung zulässt. Die Triangulierung ist beim achsenparallelen Stereotracking also eine relativ einfache Berechnung. Problematisch ist allerdings, dass eine exakt parallele Ausrichtung der beiden Kameras nahezu unmöglich ist [14].

Umsetzung von achsenparallelem Tracking im Projekt Zu Beginn des Projekts wurde das achsenparallele Tracking verwendet. Die Bildkoordinaten der Infrarotkameras der beiden Wiimotes wurden von Glovepie ermittelt und die Klasse *SchlaegerControl* hat diese aus dem Speicher ausgelesen. In dieser Klasse wurde oben genannter Algorithmus dieses Verfahrens eingesetzt. Leider war das Ergebnis nicht zufriedenstellend. Theoretisch müsste die Disparität bei einer Translation, die exakt orthogonal zur Blickrichtung der beiden Kameras verläuft, unverändert bleiben. In Wirklichkeit aber traten Abweichungen bei der Disparität im Verlauf der Bewegung hervor, die letztendlich zu einem teilweise starken Zittern des Objekts in der Applikation führten, welches sich verstärkte, je weiter die Wiimote

von den Kameras wegbewegt wurde. Um dieses Zittern in der Applikation zu vermeiden, musste interpoliert werden, und zwar umso stärker je größer die Entfernung zu den Kameras war. Eine starke Interpolation führte allerdings zu einer Verzögerung bei der Übertragung der Translation auf das Spielobjekt. Dadurch fühlte sich die Steuerung sehr schwammig an, so als würde der Spieler den Schläger an einem Gummiband hinter sich herziehen. Anstatt zu interpolieren, besteht allerdings die Möglichkeit, das Objekt relativ nah vor den Kameras zu bewegen. Das Problem dabei wiederum ist der sehr geringe Bewegungsradius, je mehr man sich den Kameras nähert. Außerdem besteht das Problem, wie in Abschnitt 6.2.2 beschrieben, dass das Objekt, das verfolgt werden soll relativ groß ist. Große Objekte, die Infrarotlicht aussenden, können von den Kameras aus der Nähe allerdings nicht mehr verfolgt werden, da die Wiimote dann nicht mehr einen, sondern mehr als 4 Punkte gleichzeitig identifiziert. Das größte Problem war aber nach wie vor das Zittern, welches sich negativ auf die physikalischen Berechnungen auswirkte, diese so veränderte, dass sie nicht den Erwartungen des Spielers entsprachen. Aus diesem Grund wurde das achsenparallele Tracking fallen gelassen und durch ein konvergentes Tracking ersetzt.

6.2.4 Konvergentes Stereotracking

Abbildung 26 zeigt den Aufbau eines konvergenten Stereotrackingsystems mit 2 Wiimotes. Hierbei können die Kameras in beliebiger Positionierung und Orientierung aufgebaut werden, solange beide Sicht auf das zu verfolgende Objekt haben. Allerdings setzt dieses Verfahren voraus, dass so-

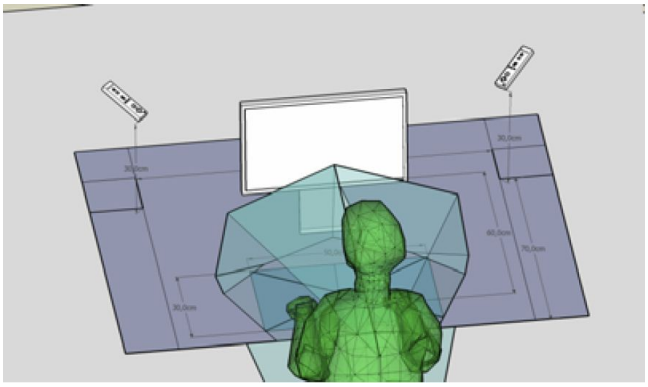

Abbildung 26: konvergentes Stereotracking mit 2 Wiimotes [8]

wohl die intrinsischen wie auch die extrinsischen Kameraparameter bekannt sind. [8] Die intrinsischen Parameter beschreiben die optischen, geometrischen und digitalen Eigenschaften der jeweiligen Kamera. Diese Pa-

rameter ändern sich bei Bewegung der Kamera nicht. Dazu gehören:

- *H:* Hauptpunkt: Dies ist der Ursprung des Bildkoordinatensystems, der meist nicht mit dem Schnittpunkt der optischen Achse der Kamera übereinstimmt.
- d_x, d_y : Größe der Sensorelemente (Pixelgrößen)
- k_1, k_2 : Radiale Verzerrung der Linse
- *f:* Brennweite
- *s:* Scherung: Die $x-$ und $y-$Bildachsen sollen im Idealfall exakt orthogonal zueinander sein. Dies ist nicht immer der Fall. Die Scherung s gibt demnach an wie schief die Achsen der Kamera zueinander stehen [31].

Um von Bild- in Pixelkoordinaten umrechnen zu können existiert die sogenannte K-Matrix, die die eben definierten Parameter mit Ausnahme der radialen Verzerrung enthält. K ist folgendermaßen definiert: [15]

$$K = \begin{pmatrix} \frac{F}{d_x} & s & H_x \\ 0 & \frac{F}{d_y} & H_y \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

Laut [9] müssen der Scherungsfaktor s und die radiale Verzerrung k_1 und k_2 für die Kalibrierung der Wiimote-Kameras nicht in Betracht gezogen werden, da bei den von der Wiimote ausgegebenen Daten diese Faktoren bereits kompensiert wurden.

Die extrinsischen Parameter beschreiben die Position und Orientierung einer Kamera bezüglich des Weltkoordinatensystems durch:

- *t:* Translationsvektor vom Ursprung des Weltkoordinatensystems zum optischen Zentrum der Kamera
- *R:* ist eine 3×3-Rotationsmatrix und gibt die Orientierung der Kamera im Bezug auf das Weltkoordinatensystem an

Die Rotationsmatrix und der Translationsvektor können in einer gemeinsamen Matrix V[29], die also die extrinsischen Kameraparameter enthält zu-

[29] V steht für View-Matrix

sammengefasst werden:

$$V = \begin{pmatrix} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} & R & & t \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2}$$

Beim Stereotracking wird meist der Ursprung des Weltkoordinatensystems in den Ursprung einer der beiden Kameras gesetzt, um mit dem Translationsvektor direkt den Abstand zwischen den Kameras zu bekommen. Damit bliebe nur noch eine Rotation und Translation für eine Kamera zu berechnen übrig. [15]

Um die Kameraparameter herauszufinden ist es nötig, die Kameras zuvor zu kalibrieren.

6.2.5 Kamerakalibrierung

Man kann die Verfahren der Kamerakalibrierung, also der Bestimmung der intrinsischen und extrinsischen Kameraparameter, grob in zwei verschiedene Kategorien einteilen [31]:

- *Photogrammetrische Kalibrierung*[30]*:* Bei dieser Art der Kamerakalibrierung wird ein Kalibriermuster, dessen Geometrie im dreidimensionalen Raum bekannt ist, mit der entsprechenden Kamera beobachtet. „Der Zusammenhang zwischen[...]“ den Weltkoordinaten des Musters und den entsprechenden Pixelkoordinaten der Kamera, die sogenannte Homographie, „[...] beinhaltet die gesuchten Kameraparameter“ [15]. Meist besteht das Muster aus zwei bis drei zueinander orthogonal ausgerichteten Flächen. Abbildung 27 zeigt ein solches Muster. Es gibt auch eine Methode, bei der eine Fläche durch eine bekannte Translation bewegt wird. Diese Verfahren sind aufgrund der benötigten Ausrüstung relativ teuer und aufwändig [31].

- *Verfahren zur Selbstkalibrierung*[31]*:* Annstatt ein Kalibriermuster einzusetzen, werden bei diesem Verfahren von der zu kalibrierenden Kamera mindestens 3 verschiedene Bilder einer starren Szene aus unterschiedlichen Positionen und Perspektiven aufgenommen. Korrespondente Punkte in diesen Bildern werden gesucht anhand derer

[30] engl: Photogrammetric calibration
[31] engl: Self-calibration

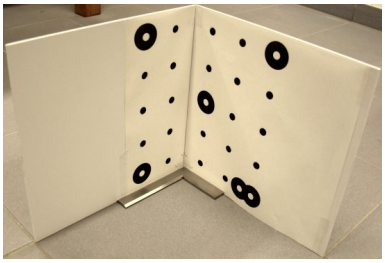

Abbildung 27: Beispiel für ein Kalibriermuster mit dreidimensionaler Ausdehnung. Die zu berechnenden intrinsischen und extrinsischen Kameraparameter sollen in diesem Fall dazu dienen dreidimensionale Objekte zu digitalisieren

die intrinsischen und extrinsischen Parameter bestimmt werden können[32]. Allerdings erweisen sich laut [31] diese Verfahren als nicht besonders robust, weshalb nicht immer verlässliche Werte ermittelt werden können.

Zur Berechnung der internen Parameter wird in diesem Projekt die Methode von Zhang [31] verwendet. Diese kann nicht eindeutig einer der beiden Kategorien zugeordnet werden. Zwar wird ein Kalibriermuster verwendet, allerdings werden dessen Daten nicht im 3D- sondern im 2D-Raum angegeben. „Im Vergleich mit klassischen Verfahren ist das [von Zhang] vorgestellte Verfahren bedeutend flexibler und im Vergleich zu Selbstkalibrierungsverfahren weitaus robuster.“[33] [31]

Als Kalibrierungsmuster dient im Falle dieses Projekts eine rechteckige Platte auf deren 4 Ecken jeweils eine Infrarot-LED angebracht ist, die durch die Wiimotes wahrgenommen werden können [34]. Die Platte wird nun von jeder Kamera an unterschiedlichen Positionen und in unterschiedlichen Ausrichtungen aufgenommen. Die Position und Orientierung der Kameras wird dabei nicht verändert, lediglich die der Platte. Dabei müssen bei mindestens einer Aufnahme alle 4 LEDs für beide Kameras sichtbar sein, da aus dieser Aufnahme später die extrinsischen Parameter berechnet werden. Mit Hilfe des EasyCalibrationTools[35] von Zhang [30] werden aus den Aufnahmen und der Angabe der Größe der Platte die internen Kamera-

[32] zum Beispiel mit Hilfe des 8-Punkte Algorithmus mit dem die sogenannte Fundamentalmatrix bestimmt werden kann. In diese gehen die intrinsischen und extrinsischen Kameraparameter ein [15]

[33] Zitat aus dem Englischen übersetzt

[34] Nach der Methode von Zhang werden meist planare Muster mit mehr als 4 bekannten Punkten verwendet. Da die Wiimote allerdings nur 4 Infrarotpunkte gleichzeitig verfolgen kann, besteht hierbei eine Einschränkung

[35] http://research.microsoft.com/en-us/downloads/7e9de40f-06db-452c-a0f2-4fabb4f20f52/

paramter der jeweiligen Kamera berechnet. Zusätzlich wird zu jeder Aufnahme eine Transformationsmatrix V ausgegeben, die die zur Aufnahme zugehörige Transformation vom Koordinatensystem, das durch das Kalibriermuster aufgespannt wird, ins Kamerakoordinatensystem wiedergibt. Dies entspricht den zur Aufnahme gehörigen extrinsischen Parametern. [8]

Aus diesen zu jeder Aufnahme berechneten extrinsischen Parametern wird nun die Aufnahme, bei der das Muster von beiden Kameras gleichzeitig erkannt wurde, als Basis-Koordinatensystem ausgewählt. Dieses Koorndinatensystem wird dann im Folgenden als Weltkoordinatensystem angenommen. Der Ursprung dieses Koordinatensystems liegt dort, wo das Kalibriermuster im Raum vor die Kameras gehalten wurde. Abbildung 28 zeigt ein solches, durch ein Muster (die 4 als rote Kugeln gekennzeichneten Punkte) aufgespanntes Koordinatensystem. Da die zugehörigen Transformationsmatrizen vorliegen, können hieraus die Positionen und Orientierungen der beiden Kameras, also die extrinsischen Parameter berechnet werden.

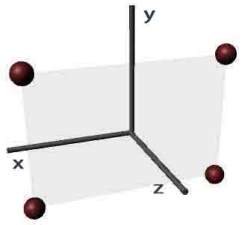

Abbildung 28: Kalibriermuster, das ein Weltkoordinatensystem aufspannt. Die 4 roten Punkte stellen die 4 Infrarotdioden des Musters dar.

Es besteht aber auch die Möglichkeit, eine der Kameras als Weltkoordinatensystem auszuwählen. Über die beiden Transformationsmatrizen die die beiden Aufnahmen geliefert haben, kann die Transformation der zweiten Kamera in den Ursprung der ersten Kamera berechnet werden. Je nachdem, wie das Weltkoordinatensystem gewählt wird, ergeben sich also mit den passenden Transformationsmatrizen, die das Easy Calibration Tool liefert, die extrinsischen Kameraparameter.[7]

Im Folgenden soll ein grober Überblick gegeben werden, wie nach der Methode von Zhang mit Hilfe des EasyCalibrationTools die intrinsischen und extrinsischen Kameraparameter berechnet werden.

Das Kalibriermuster Laut der technischen Dokumentation des Tools [31] soll für gewöhnliche Farbbildkameras ein 2D-Muster auf Papier ausgedruckt, und auf eine flache Platte geklebt werden. Da die Wiimote aller-

dings nur Infrarotlicht erkennt, werden in diesem Fall Dioden verwendet. Abbildung ... zeigt das verwendete Muster. Etwas problematisch ist dabei, dass die Diode selbst im Raum eine dreidimensionale Ausdehnung hat, also ist der Lichtpunkt, den sie ausstrahlt nicht so flach wie ein Punkt auf einem Blatt Papier. Dies hat sich bei den Ergebnissen des Kalibrierverfahrens allerdings nicht negativ bemerkbar gemacht.

6.2.6 Berechnung der Kameraparameter nach der Methode von Zhang

Wie bereits beschrieben ist für das Verfahren von Zhang, welches beim EasyCalibrationTool verwendet wird ein zweidimensionales Kalibriermuster notwendig, das in unterschiedlichen Orientierungen vor die Kamera gehalten wird. Die Transformationen, die das Muster dabei unterläuft müssen vorher nicht bekannt sein.

Es sollte vermieden werden, das Muster bei den unterschiedlichen Aufnahmen parallel zu dem einer anderen Aufnahme zu halten, also lediglich zu verschieben ohne die Orienterung zu ändern. Dies kann das letztendliche Kalibrierergebnis laut Zhang [31] verfälschen. Wenn solch eine reine Translation bekannt wäre, wäre eine Kalibrierung zwar möglich, in unserem Fall ist sie allerdings nicht bekannt.

Wie bereits in Abschnitt 6.2.4 beschrieben wird mit Hilfe der K-Matrix (siehe Formel (1)) und der Viewmatrix V (siehe Formel (2)) ein Weltpunkt in homogenen Koordinaten $\vec{P} = [x, y, z, 1]^T$ in einen Bildpunkt $\vec{p} = [u, v, 1]$ in Pixelkoordinaten umgewandelt, wobei s ein unbekannter Skalierungsfaktor ist:

$$s \cdot \vec{p} = K \cdot V \cdot \vec{P}$$

$$s \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K \cdot V \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{3}$$

Dabei können wir die Spaltenvektoren von V einzeln angeben:

$$V = [r_1 \quad r_2 \quad r_3 \quad t]$$

Für $(K^{-1})^T$ bzw. $(K^T)^{-1}$ wird im Folgenden die Kurzform K^{-T} verwendet.

Wir können nun davon ausgehen, dass unser Kalibriermuster das Weltkoordinatensystem insofern aufspannt, als die Position und Orientierung der

beiden einander entsprechen. Abbildung 28 zeigt, dass der Ursprung des Weltkoordinatensystems an der Position des Kalibriermusters liegt und die Orientierung mit der Ausrichtung des Musters übereinstimmt. Wir können also annehmen, dass das Muster an der Stelle $z = 0$ des Weltkoordinatensystems liegt. [31] Gleichung 3 kann demnach spezifiziert werden:

$$s \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K \cdot [r_1 \quad r_2 \quad r_3 \quad t] \cdot \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix} \tag{4}$$

$$= K \cdot [r_1 \quad r_2 \quad t] \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Die Weltpunkte des Kalibriermusters sind also von vornherein ebenso bekannt, wie deren zugehörige Bildpunkte. Wir können demnach feststellen: Ein Punkt des Musters wird auf den entsprechenden Bildpunkt durch eine Homographie H

$$H = K \cdot [r_1 \quad r_2 \quad t]$$

abgebildet. [31] Diese Homographie lässt sich folgendermaßen schätzen:

Schätzung der Homographie zwischen dem Muster und seiner Projektion auf die Bildebene Es seien P_i ein Weltpunkt des Musters und p_i dessen Abbildung in Pixelkoordinaten. Es besteht die Moeglichkeit, die Homographie H durch Minimieren der folgenden Gleichung zu ermitteln.

$$min_H \sum_i ||p_i - \hat{p_i}||^2$$

wobei

$$\hat{p_i} = \frac{1}{\bar{h_3}^T P_i} \begin{pmatrix} \bar{h}_1^T P_i \\ \bar{h}_2^T P_i \end{pmatrix}$$

wobei $\bar{h}_i$ die i-te Zeile von H darstellt. Diese Gleichung kann mit dem Levenberg-Marquardt-Algorithmus berechnet werden, der aber eine initiale Schätzung der Homographie erfordert. Diese kann folgendermaßen ermittelt werden.

Es sei $x = (\bar{h}_1^T \quad \bar{h}_2^T \quad \bar{h}_3^T)$, dann kann Gleichung (3) umformuliert werden als

$$\begin{pmatrix} P^T & 0^T & -uP^T \\ 0^T & P^T & -vP^T \end{pmatrix} x = 0$$

Da wir bei einer Aufnahme 4 verschiedene Punkte des Musters betrachten, können wir dabei 4 solcher Gleichungen aufstellen. Wir können diese in Matrixform schreiben, wobei die linke Matrix zu einer Gesamtmatrix L ausgebaut wird, die eine $2 * 4 \times 9$ Matrix darstellt. Also gilt:

$$L\,x = 0$$

Da wir die 4 Weltpunkte P_i des Kalibriermusters kennen, ebenso wie die Bildkoordinaten u und v, ist L bereits gegeben. Die gesuchte Matrix x, die bis auf einen unbekannten Skalierungsfaktor definiert sein soll kann nun gefunden werden. Sie ist der rechte Singulärvektor von L korrespondierend zum kleinsten Singulärwert von L. Das selbe Ergebnis erhält man, wenn man den Eigenvektor von $L^T L$ korrespondierend zum kleinsten Eigenwert von L nimmt. Laut Zhang [31] ist L numerisch instabil, da sowohl Pixel, als auch Weltkoordinaten in die Matrix eingehen. Deshalb ist es angebracht zuvor eine Normalisierung der Matrix durchzuführen, um bessere Ergebnisse für x, und demnach für die Homographie zu erreichen.

Berechnung der internen und externen Parameter Nachdem die Homographie H nun also berechnet wurde, können wir deren einzelne Spaltenvektoren definieren als

$$H = [h_1 \quad h_2 \quad h_3] = \lambda\, K\, [r_1 \quad r_2 \quad t] \tag{5}$$

wobei λ ein unbekannter Skalierungsfaktor ist. h_1, h_2, h_3 ist bekannt, die restlichen Variablen der Gleichung, also die rechte Seite, müssen noch gefunden werden.

Da die Spaltenvektoren von Rotationsmatrizen, also auch r_1 und r_2 orthonormal sind [15] und das Skalarprodukt zweier orthonormaler Vektoren 0 ergibt, gilt:

$$h_1^T K^{-T} K^{-1} h_2 = 0 \tag{6}$$

$$h_1^T K^{-T} K^{-1} h_1 = h_2^T K^{-T} K^{-1} h_2 \tag{7}$$

Wir definieren das K-Matrizenprodukt aus (6) und (7) als:

$$B = K^{-T} K^{-1} = \begin{pmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{pmatrix}$$

Ausformuliert ergibt diese Matrix [36]:

$$B = \begin{pmatrix} \frac{1}{F_x^2} & -\frac{s}{F_x^2 F_y} & \frac{v_0 s - u_0 F_y}{F_x^2 F_y} \\ -\frac{s}{F_x^2 F_y} & \frac{s^2}{F_x^2 F_y^2} + \frac{1}{F_y^2} & -\frac{s(v_0 s - u_0 F_y)}{F_x^2 F_y^2} - \frac{v_0}{F_y^2} \\ \frac{v_0 s - u_0 F_y}{F_x^2 F_y} & -\frac{s(v_0 s - u_0 F_y)}{F_x^2 F_y^2} - \frac{v_0}{F_y^2} & \frac{(v_0 s - u_0 F_y)^2}{F_x^2 F_y^2} + \frac{v_0^2}{F_y^2} + 1 \end{pmatrix} \tag{8}$$

[36] Die Variablen F_x, F_y, s, H_x, H_y beziehen sich auf die K-Matrix aus (1)

Dabei ist B symmetrisch und kann deshalb durch den Vektor

$$b = [B_{11}, \quad B_{12}, \quad B_{22}, \quad B_{13}, \quad B_{23}, \quad B_{33}]^T$$

ausgedrückt werden. Wir definieren weiterhin den i-ten Spaltenvektor von H als $h_i = [h_{i1} \quad h_{i2} \quad h_{i3}]^T$ und können nun feststellen:

$$h_i^T \cdot B \cdot h_j = v_{ij}^T \cdot b$$

Dabei sei

$$\begin{aligned} v_{ij} = [&h_{i1}h_{j1}, \quad h_{i1}h_{j2} + h_{i2}h_{j1}, \quad h_{i2}h_{j2}, \\ &h_{i3}h_{j1} + h_{i1}h_{j3}, \quad h_{i3}h_{j2} + h_{i2}h_{j3}, \quad h_{i3}h_{j3}]^T \end{aligned} \tag{9}$$

Wir können die beiden Gleichungen (6) und (7) nun umformen:

$$\begin{pmatrix} v_1{}_2^T \\ (v_{11} - v_{22})^T \end{pmatrix} b = 0 \tag{10}$$

Da letztenendes eine Anzahl von n Aufnahmen betrachtet werden, kann die erste Matrix aus (10) anhand der Homographien der weiteren Aufnahmen erweitert werden zu

$$V \cdot b = 0$$

wobei V eine $2n \times 6$-Matrix und aufgrund der ermittelten Homographien bekannt ist. Dieses Gleichungssystem ist bei einer Aufnahmenanzahl von $n \geq 3$ bis auf einen unbekannten Skalierungsfaktor eindeutig lösbar, indem man den rechten Singulärvektor von V korrespondierend zum kleinsten Singulärwert als Lösung nimmt. Damit erhalten wir die Variablen in b und können nun die K-Matrix mit den intrinsischen Parametern ermitteln.

Ermittlung der intrinischen Parameter Aus der ausformulierten B-Matrix in (8), die bis auf einen Skalierungsfaktor λ definiert ist, ergibt sich die Berechnung der intrinsischen Parameter folgendermaßen:

$$\begin{aligned} v_0 &= \frac{(B_{12}B_{13} - B_{11}B_{23})}{(B_{11}B_{22} - B_{12}^2)} \\ \lambda &= B_{33} - [B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})]/B_{11} \\ F_x &= \sqrt{\lambda/B_{11}} \\ F_y &= \sqrt{\frac{\lambda B_{11}}{B_{11}B_{22} - B_{12}^2}} \\ s &= -B_{12}F_x^2F_y/\lambda \\ u_0 &= \frac{sv_0}{F_y} - \frac{B_{13}F_x^2}{\lambda} \end{aligned}$$

Ermittlung der extrinsischen Parameter Als nächstes können die extrinsischen Parameter R unf t nach folgenden Gleichungen berechnet werden. Diese ergeben sich aus der Homographiegleichung (5)

$$\begin{aligned} r_1 &= \lambda K^{-1} h_1 \\ r_2 &= \lambda K^{-1} h_2 \\ r_3 &= r_1 \times r_2 \\ t &= \lambda K^{-1} h_3 \end{aligned} \tag{11}$$

Dabei ist $\lambda = 1/||K^{-1}h_1|| = 1/||K^{-1}h_2||$. [31]

Optimierung der Parameter nach der Methode der maximalen Wahrscheinlichkeit[37] Die eben berechneten extrinsischen und intrinsischen Kameraparameter könnten nun daraufhin überprüft werden, ob sie wirklich exakt dafür geeignet sind die Weltpunkte P_i auf die Pixelkoordinaten p_i abzubilden. Wegen Bildrauschen wird dies bei einem solchen Test aber wohl nicht der Fall sein[38]. Deshalb sollten die Parameter durch Minimieren des folgenden Algorithmus optimiert werden. Dabei ist n die Anzahl der Aufnahmen und m die Anzahl der Punkte bei jeder Aufnahme, in unserem Falle also 4.

$$\sum_{i=1}^{n} \sum_{j=1}^{m} ||p_{ij} - \hat{p}(K, R_i, t_i, P_j)||^2 \tag{12}$$

$\hat{p}(K, R_i, t_i, P_j)$ entspricht dabei dem mit Hilfe der ermittelten Homographie auf Pixelebene abgebildeten Weltpunkt (siehe Gleichung (4)), der aber wegen Rauschen nicht unbedingt mit dem von der Kamera ausgegebenen Bildpunkt p_{ij} übereinstimmt. (12) stellt ein nichtlineares Optimierungsproblem dar und wird bei Zhang [31] nach dem Levenberg-Marquardt Algorithmus gelöst. Dieser bekommt als Eingabe, neben den Bildpunkten p_{ij} und Weltpunkten des Musters P_j[39], die eben berechneten initialen Kameraparameter K sowie die zu jeder Aufnahme zugehörigen extrinsischen Parameter $\{R_i, t_i | i = 1...n\}$. Dabei werden die Rotationsmatrizen nach der Rodrigues-Formel [31] parametrisiert, sodass die Rotation durch einen Vektor angegeben wird. Die Richtung des Vektors entspricht der Rotationsachse und die Länge dem Rotationswinkel. [31] Das EasyCalibrationTool kann die radiale Verzerrung der Linse mit einbeziehen, allerdings ist dies in unserem Fall nicht nötig (siehe Abschnitt 6.2.4)

[37] engl: maximum likelihood estimation

[38] Dieser Fehler wird auch als Rückprojektionsfehler bezeichnet [1].

[39] es ist zu beachten, dass die Weltpunkte des Musters bei jeder Aufnahme gleich bleiben, da das Muster dabei jeweils ein neues Weltkoordinatensystem aufspannt (siehe Abbildung 28). Deshalb wäre die Angabe P_{ij} unnötig, es reicht P_j anzugeben

6.2.7 Triangulierung

Hat man die intrinsischen und extrinsischen Kameraparameter bestimmt kann die Aufnahme eines zu verfolgenden Punkts, der in beiden Kameras sichtbar sein muss, durchgeführt werden. Um die Weltkoordinaten des Punkts zu errechnen wird eine Triangulierung durchgeführt. Dafür müssen die Pixelkoordinaten, die von den Wiimotes geliefert werden zunächst mit Hilfe der Matrix K (siehe Formel (1)) in Bildkoordinaten umgerechnet werden [15].

die Bildkoordinaten $\vec{b_p}$ werden durch folgende Formel berechnet:

$$\vec{bp} = K^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

Bei diesen Bildkoordinaten hat z stets den Wert 1 und die $x-$ und $y-$ Koordinaten sind entsprechend skaliert. Aus den Bildkoordinaten und der Transformationsmatrix V, die die errechneten extrinsischen Kameraparameter R und t vom Welt- ins Kamerakoordinatensystem enthält (siehe Formel (2)) kann der Richtungsvektor bestimmt werden [7]:

$$\vec{r} = V^{-1} \cdot \vec{bp}$$

Die Position in Weltkoordinaten der Kamera kann folgendermaßen berechnet werden:

$$\vec{p} = V^{-1} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Mit Hilfe des Richtungsvektors und der Kameraposition lässt sich die Gerade in Weltkoordinaten berechnen die vom Kamerazentrum und dem verfolgten Punkt aufgespannt wird: [8]

$$\overline{g} = \vec{p} + a\vec{r}$$

Die gesuchte Weltkoordinate des Punktes ergibt sich dann aus dem Schnittpunkt der beiden Geraden die die linke und rechte Kamera jeweils aufspannen. Da sich die Geraden allerdings aufgrund geringfügiger Abweichungen meist nicht schneiden, wird der Punkt in Weltkoordinaten, der zu beiden Geraden den geringsten Abstand hat, als der gesuchte Punkt angenommen. [7]

Im Folgenden wird die Berechnung dieses Punktes erläutert. Zunächst muss der kürzeste Abstand zwischen den beiden Geraden L_1 und L_2 berechnet werden. Die Geraden werden folgendermaßen definiert:

$$L_1 : \vec{g} = \vec{a} + s\vec{b} \tag{13}$$

$$L_2 : \vec{g} = \vec{c} + t\vec{d}$$

Der Abstand zwischend den beiden Geraden kann auf folgendem Weg ermittelt werden [12]: Die Punkte $\vec{a}$ und $\vec{c}$ liegen jeweils auf einer der beiden Geraden, also beschreibt die Differenz der beiden einen Vektor

$$(\vec{a} - \vec{c})$$

der den Abstand zwischen beiden Geraden in einer bestimmten Richtung definiert, allerdings nicht unbedingt in der Richtung die orthogonal zu beiden Geraden liegt. Demnach beschreibt dieser Vektor nicht unbedingt den kürzesten Abstand zwischen den Geraden. Weiterhin gilt:

$$\vec{u} = \frac{(\vec{b} \times \vec{d})}{|\vec{b} \times \vec{d}|}$$

beschreibt einen Einheitsvektor $\vec{u}$, der senkrecht zu beiden Geraden steht. Das Skalarprodukt eines Vektors $\vec{a}$ mit einem Einheitsvektor $\vec{b}$ wiederum beschreibt die Länge der Projektion von $\vec{a}$ auf $\vec{b}$. Bildet man demnach das Skalarprodukt

$$g = (\vec{a} - \vec{c}) \cdot \frac{(\vec{b} \times \vec{d})}{|\vec{b} \times \vec{d}|}$$

so wird also die Länge der Projektion eines Vektors, der den Abstand zwischen den Geraden in einer bestimmten Richtung beschreibt, auf den Einheitsvektor, der senkrecht zu beiden Geraden verläuft, gebildet. Damit erhält man den kürzesten Abstand g zwischen beiden Geraden. [12]

6.2.8 Berechnung des nähesten Weltpunkts zwischen 2 Geraden im Raum

Möglichkeit 1: Um nun die beiden Punkte auf den Geraden zu finden, die den geringsten Abstand voneinander haben (im Folgenden als $\vec{p_1}$ und $\vec{p_2}$ bezeichnet) müssen die unbekannten Parameter s und t der Geradengleichungen aus Formel (13) ermittelt werden. Dazu wird eine dritte Gerade L_3 angenommen. die die Verschiebung von L_1 um den Vektor $g\vec{u}$ darstellt:

$$L_3 : \overline{g} = g\vec{u} + \vec{a} + s\vec{b}$$

L_3 schneidet nun L_2 mit Sicherheit in dem Punkt $\vec{p_2}$, der auf L_2 letztendlich gesucht wird. Man kann die beiden Geraden L_3 und L_2 also gleichsetzen:

$$g\vec{u} + a + s\vec{b} = \vec{c} + t\vec{d}$$

Dies ist ein lineares Gleichungssystem mit den 2 unbekannten Variablen s und t und kann mit dem Gaußschen Eliminationsverfahren gelöst werden.

Man kann 2 verschiedene Komponenten der Vektoren (zum Beispiel x und y) nehmen und für jede Komponente eine lineare Gleichung erstellen, damit das Gleichungssystem gelöst werden kann. Sind die beiden Parameter s und t also gefunden, ergeben sie durch Einsetzen in die jeweiligen Geradengleichungen L_1 und L_2 die beiden gesuchten Punkte $\vec{p_1}$ und $\vec{p_2}$. Der Weltpunkt $\vec{p_w}$, der letztendlich gesucht wird ist dann der Mittelpunkt des Vektors, der von den beiden Punkten aufgespannt wird:

$$\vec{p_w} = \vec{p_1} + \frac{\vec{p_2} - \vec{p_1}}{2}$$

[12]

Es gibt allerdings 2 weitere effizientere Umsetzungen um s und t zu ermitteln. Da diese nur eine Umformung der oben genannten Möglichkeit darstellen, wird hier nur der Algorithmus aufgeschrieben. Dieser fand auch Verwendung beim konvergenten Stereotracking in diesem Projekt:

Zunächst wird der Vektor gebildet der von den beiden Positionen der Geraden aufgespannt wird:

$$\vec{P_{21}} = \vec{c} - \vec{a} \tag{14}$$

Anschließend wird durch ein Kreuzprodukt der beiden Richtungsvektoren ein Vektor gebildet, der senkrecht zu beiden Geraden steht:

$$\vec{M} = \vec{d} \times \vec{b} \tag{15}$$

Dieser Vektor wird mit sich selbst Skalarmultipliziert:

$$m_2 = \vec{M} \cdot \vec{M}$$

Anschließend wird durch ein Kreuzprodukt ein Vektor $\vec{R}$ gebildet, der senkrecht zum Abstandsvektor aus (14) und zum Vektor aus (15) steht.

$$\vec{R} = \vec{P_{21}} \times (\vec{M}/m_2)$$

Um s und t zu berechnen wird das Skalarprodukt aus $\vec{R}$ und den Richtungsvektoren der Geraden gebildet.

$$s = \vec{R} \cdot \vec{d}$$

$$t = \vec{R} \cdot \vec{b}$$

[17]

Die beiden Geradengleichungen und die Parameter s und t liefern uns also die Punkte auf den Geraden die nun als $\vec{g_1}$ und $\vec{g_2}$ bezeichnet werden sollen. Der gesuchte Weltpunkt $\vec{P}$ ist dann der Mittelpunkt der Strecke, die von den beiden Punkten aufgespannt wird:

$$\vec{P} = \vec{g_1} + ((\vec{g_2} - \vec{g_1}) * 0.5);$$

Damit ist der Weltpunkt erfolgreich berechnet worden.

6.3 Umsetzung des Stereotrackings im Projekt

Im Folgenden wird beschrieben, wie das oben beschriebene konvergente Stereotracking im Projekt umgesetzt wurde. Es werden Klassen beschrieben, die die Kalibriermethode nach Zhang implementieren, sowie die Berechnung des Weltpunkts des verfolgten Objekts durchführen. Weiterhin wird auf den Aufbau des Stereokamerasetups eingegangen.

6.3.1 Einbindung der Kalibriermethode in das Projekt

Zusammenfassung Die Kamerakalibrierung nach Zhang erfordert für das Projekt die folgenden Schritte:

- Die x- und y-Koordinaten des Kalibriermusters werden für die Kamerakalibrierung benötigt und deshalb in einem Textfile namens *model.txt* gespeichert. Es handelt sich dabei um Weltkoordinaten. Die z-Koordinate ist allerdings nicht vonnöten, da die Punkte alle auf der $z = 0$ Ebene liegen. Die Koordinaten werden durch Vermessung exakt ermittelt. Letztendlich ist aber nur das genaueSeitenverhältnis des von den 4 Punkten aufgespannten Rechtecks für ein erfolgreiches Kalibrieren ausschlaggebend.

- Zunächst müssen die beiden Wiimotes, wie vom Benutzer gewünscht, positioniert und orientiert werden. Dabei müssen sie einen gemeinsam sichtbaren Bereich[40] aufspannen, wie in Abbildung 26 zu sehen.

- Mit beiden Kameras wird nun das Kalibriermuster einmal aufgenommen. Das Muster besteht aus 4 Infrarotdioden, die ein Rechteck bilden. Die Aufnahme geschieht auf Tastendruck des Benutzers. Diese erste Aufnahme soll dabei stets das Weltkoordinatensystem für das spätere Tracking aufbauen. Die externen Kameraparameter, die also später für diese Aufnahme berechnet werden, gehen in das Tracking mit ein.

- Nun werden für jede einzelne Kamera weitere Aufnahmen gemacht. Es ist nicht mehr nötig, dass beide Kameras das Muster sehen kön-

[40] also ein gemeinsames Frustum

nen. Dies war nur bei der ersten Aufnahme nötig, aus der die externen Parameter berechet werden. Es ist sogar besser, wenn nur eine Kamera das Muster sehen kann, da man es dann in extremeren Orientierungen halten kann. Dies wiederum verbessert die Kalibrierung enorm. Es müssen mindestens 2 weitere Aufnahmen pro Kamera gemacht werden, um die Kameraparameter berechnen lassen zu können. Mehr Aufnahmen sind aber von Vorteil, unter der Bedingung, dass das Kalibriermuster in unterschiedlichen Orientierungen gehalten wird.

- Die Kameras der Wiimotes werden auch hier über GlovePie angesteuert. Bei der ersten Aufnahme entstehen $8x$- und $8y$-Koordinaten[41]. Diese werden in eine Textdatei namens *output.txt* in eine Zeile geschrieben. Die weiteren Aufnahmen produzieren weitere Koordinaten, diese werden ebenfalls so in eine Zeile des Textfiles geschrieben, das in der linken Hälfte der Zeile die Werte der ersten Kamera und in der rechten die der zweiten Kamera stehen.

- Wenn nun eine Applikation gestartet wird, so führt diese zunächst einen Textparser aus, der aus *output.txt* liest.

- Der Textparser erkennt, ob es sich um die Daten der ersten oder der zweiten Kamera handelt. Er schreibt die Daten jeder Aufnahme einer Kamera in verschiedene Textfiles[42].

- Die Textfiles werden in einem gemeinsamen Ordner gespeichert.

- Nun ruft die Applikation das EasyCalibrationTool auf. [43] Dem Tool werden die verschiedenen Aufnahmedaten der ersten Kamera in Form der Textfiles übergeben und zusätzlich die Welkoordinaten des Kalibriermusters in Form der Datei *model.txt*.

- Daraus berechnet das Tool einmal die internen und für jede Aufnahme die externen Kameraparameter der ersten Kamera und schreibt sie in eine Datei *result.txt*. Anschließend werden die letzten beiden Schritte auch für die zweite Kamera durchgeführt.

- Die Applikation liest dann das zu jeder Kamera gehörige *result.txt* und bekommt so die jeweiligen internen Kameraparameter. Aus der ersten Aufnahme werden die externen Parameter gewonnen, da diese Aufnahme von beiden Kameras gleichzeitig durchgeführt wurde.

[41] 2 Kameras $* 4$ aufgenommene Infrarotdioden

[42] die mit *data1.txt* für die erste Aufnahme, *data2.txt* für die zweite, etc. beschriftet sind

[43] Es handelt sich dabei um ein Kommandozeilentool, das deshalb recht einfach von externen Programmen ausgeführt werden kann.

Wichtig ist, dass die Reihenfolge der Koordinaten in den verschiedenen Textfiles der einzelnen Aufnahmen auch der Reihenfolge der Koordinaten in *model.txt* entspricht. Wenn also der linke untere Punkt des Arrays an erster Stelle in *model.txt* steht, muss dieser von einer Kamera aufgenommen Punkte im zugehörigen Textfile auch an erster Stelle aufgeführt werden. Leider ist die Reihenfolge der aufgenommenen Punkte der Wiimotes aber nicht immer gleich. Mal wird der linke untere Punkt des Arrays als erstes ausgegeben, mal ein anderer. Der Textparser sorgt dafür, dass die Punkte entsprechend sortiert werden.

Implementierung des Textparsers - Die Klasse CalibrationMgr Um die Daten für das EasyCalibrationTool aufzubereiten wurde ein Textparser geschrieben, der durch die Klasse *CalibrationMgr* verwaltet wird.

Zunächst wird gezählt, wieviele Aufnahmen in *output.txt bestehen*, was durch die Anzahl der darin enthalten Zeilen gegeben ist. Die ersten 8 Koordinaten einer Zeile gehören zur ersten, die letzten 8 zur zweiten Kamera. Jede Zeile wird gelesen und daraus jeweils die Daten der ersten und der zweiten Kamera extrahiert und in separate Textfiles geschrieben. Dabei muss, wie im oberen Abschnitt beschrieben, *CalibrationMgr* dafür sorgen, die Punkte so zu sortieren, dass ihre Reihenfolge der in *model.txt* entspricht. Deshalb wird jeder Punkt einer der 4 Ecken zugewiesen[44]. Anschließend werden die Punkte in der richtigen Reihenfolge in die Textdatei geschrieben. Nun wird das EasyCalibrationTool gestartet und diesem alle *data.txt*-Dateien der ersten Kamera, sowie *model.txt* übergeben. Das selbe wird für die zweite Kamera mit den dafür zugehörigen Eingabedaten wiederholt. Das Tool speichert das Ergebnis in weiteren Textfiles. Diese werden von *CalibrationMgr* wiederum ausgelesen und daraus die intrinsischen Parameter in zwei verschiedene Matrizen K (siehe Formel 1) für jede Kamera geschrieben. Da in den beiden *result.txt*-Dateien die jeweils ersten 9 Werte die extrinsischen Parameter der ersten Aufnahme beschreiben, werden auch diese extrahiert und in zwei Matrizen V (siehe 2) geschrieben. Die extrinsischen und intrinsischen Parameter können nun der Klasse *SchlaegerControl* übergeben werden. Diese kann dann mit deren Hilfe die Position der Wiimote im Raum verfolgen.

6.3.2 Berechnung des Weltpunktes in der Klasse *SchlaegerControl*

Die Klasse *Schlaegercontrol* bekommt also die intrinsischen und extrinsischen Parameter jeder Kamera von der Klasse *CalibrationMgr* geliefert. Weiterhin werden pro Frame die Bildkoordinaten der Infrarotkameras von Glove-

[44] also „unten links“, „unten rechts“, „oben links“, „oben rechts“

Pie abgefragt. Die Bildkoordinaten entstehen durch die Abbildung des Infrarotlichts, welches die Wiimote aussendet, auf die Bildebene jeder Kamera. Aus den Koordinaten werden nun die Geraden berechnet, die von den Kameras und dem Weltpunkt aufgespannt werden. Dies geschieht nach dem Algorithmus, der in Abschnitt 6.2.7 beschrieben wurde. Da sich die Geraden im Raum nicht schneiden, wird nun der Punkt im Raum gesucht, der den kürzesten Abstand zu den Geraden aufweist. Die Methode dieser Berechnung wurde bereits in den Abschnitten 6.2.7 und 6.2.8 erläutert. Dieser Punkt ist nun der gesuchte Weltpunkt des Objekts. Da das verfolgte Objekt die Infrarotlicht aussende Wiimote des Spielers ist, haben wir damit die Position des Controllers im Raum erfolgreich bestimmt. Nun muss nur noch der Abstand zur Position des letzten Frames berechnet werden und mit diesem Wert die Translationsfunktion der Klasse *Schlaeger* aufgerufen werden. Damit wird der Schläger im Spiel bewegt.

6.4 Zusammenfassung des Trackingverfahrens

Zusammenfassend werden noch einmal die wichtigsten Punkte des Stereotrackingverfahrens, das letztendlich verwendet wurde, aufgezählt. Es wurde konvergentes Stereotracking verwendet, da achsenparalleles Tracking zu zitternden Objekten im Spiel geführt hatte. Selbst wenn die Wiimote still gehalten wurde war ein Zittern zu erkennen. Beim konvergenten Tracking ist dieses Zittern nur noch dann geringfügig vorhanden, wenn sich der Spieler nach rechts und links bewegt. Hält man die Wiimote still im Raum, bewegt sie nach vorne und hinten oder unten und oben, so ist beinahe gar kein Zittern mehr auszumachen. Da die Kameras beim konvergenten Tracking in unterschiedlichen Orientierungen und an unterschiedlichen Positionen aufgebaut werden, ist eine Kamerakalibrierung nötig, die mit Hilfe des EasyCalibrationTools erreicht wurde.

Es werden Infrarotarrays verwendet und ein Aufsatz mit reflektierendem Material auf die Wiimote angebracht. Der Aufsatz wird aufgrund seiner Größe und Form sehr gut erkannt. Damit ist das Tracking sehr robust und fällt nicht einfach kurzfristig aus. Das Objekt im Spiel bewegt sich deshalb flüssig. Auch ist es möglich, die Wiimote in unterschiedlichen Orientierungen zu halten, ohne das das Tracking ausfällt. In den Tests, die im nächsten Kapitel folgen, sollen die Personen das Tracking bewerten.

Technische Aufstellung der Infrarotkameras Da beide Kameras nur einen sehr geringen Öffnungswinkel von ca 43 Grad haben, sollte der Benutzer mindestens 1,3 Meter von den Kameras entfernt sein. Nur wenn beide Kameras das Objekt sehen können, kann ein Tracking erfolgen. Der Raum,

in dem getrackt werden kann, besteht also aus dem Schnittbereich der beiden Kamerafrusta. Beim achsenparallelen Tracking sollte man die Kameras möglichst nah aneinander platzieren um einen möglichst großen gemeinsamen Sichtbereich der Kameras zu verwirklichen und so den Bewegungsradius des Spielers zu vergrößern.

Letztendlich wurde aber ohnehin das konvergente Tracking verwendet. Hier hat es sich bezahlt gemacht, die Kameras im Abstand von ca 70 bis 80 Zentimeter aufzubauen. Beide Kameras liegen auf einer ebenen Fläche, blicken also nicht nach oben oder unten. Der Winkel zwischen den beiden Kameras beträgt ca. 20 Grad. Damit wurde ein recht großer Bewegungsradius erreicht. Der Spieler bewegt sich meist in einem Raum von 2 Quadratmetern, der bei diesem Aufbau dann abgedeckt wird, wenn man ca 1,5 Meter von den Kameras entfernt ist. Abbildung 29 zeigt den Aufbau der Kameras und den Controller mit dem Aufsatz.

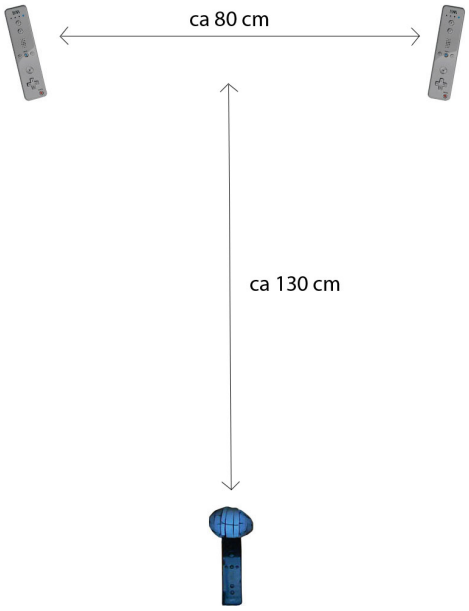

Abbildung 29: Die Wiimotes wurden im Abstand von ca 80 Zentimeter aufgebaut. Der Winkel zwischen den beiden Kameras beträgt ca 20 Grad.

Positionierung des Reflektionsmaterials an der Wiimote und sich ergebende Folgen Idealerweise müsste der Punkt, der für die Translation verfolgt wird, das Handgelenk des Spielers sein. Der reflektierende Aufsatz befindet sich allerdings an der Spitze der Wiimote. Dies führt zu dem Problem, dass eine Rotationsbewegung zusätzlich als Translation wahrgenommen wird. Der Grund ist, dass sich die Spitze der Wiimote, bei der Rotation im Raum bewegt, da der Punkt um den rotiert wird normalerweise das

Handgelenk des Spielers ist. Um zu vermeiden, dass Rotationen auch zu Translationen führen, muss der Punkt, um den rotiert wird, bei dem Objekt im Spiel nicht dort hingesetzt werden, wo man den Schläger normalerweise greift, sondern weiter vorne. Beim Tischtennisschläger beispielsweise wird das Rotationszentrum ans Ende des Schlägers gesetzt. Dadurch wird das Problem der Tranlationen bei Rotationsbewegungen einigermaßen ausgeglichen. Es kann aber störend wirken, wenn die Translation plötzlich ausfällt, weil der reflektierende Aufsatz aus irgend einem Grund verdeckt wird. Dann ist das Rotationszentrum plötzlich nicht mehr am Griff, sondern am Kopf des Schlägers, was eher nicht gewünscht ist.

7 Evaluation

Um zu ermitteln, inwieweit die umgesetzte R+T-Steuerung den Spielablauf und die Qualität der Spiele beeinflusst, wurden Tests mit verschiedenen Personen durchgeführt. Diese sollten zunächst die Spiele ausprobieren und hinterher eine Fragebogen beantworten. Im folgenden Abschnitt wird zunächst erklärt, unter welcher Prämisse die Tests mit den verschiedenen Personen durchgeführt wurden und nach welchen Kriterien die Fragen des Fragebogens erstellt wurden. Es wird darauf eingegangen wie die Tests schließlich durchgeführt wurden und was die Testpersonen für Aufgaben in den Spielen meistern sollten. Im weiteren Verlauf werden die Ergebnisse der Fragebögen aufgezeigt, und diese Ergebnisse schließlich interpretiert, und daraufhin überprüft, ob sie die Hypothesen aus Abschnitt 4.3 untermauern oder diesen widersprechen.

Die Testpersonen Es wurde darauf geachtet, dass nicht nur Personen die Spiele ausprobieren konnten, die ohnehin schon viel Erfahrung mit Konsolen- oder Computerspielen haben. Es wurden auch Personen getestet, die so gut wie nie oder sehr selten mit Spielen in Berührung kommen, insbesondere um die Einsteigerfreundlichkeit bewerten zu können.

7.1 Die Tests

Alle Testpersonen sollten die Spiele Tischtennis, Baseball und Billiard eingehend ausprobieren. Dabei sollten verschiedene Steuerungsmodi getestet werden, besonders unter dem Aspekt, die R+T-Steuerung mit anderen Modi vergleichen und bewerten zu können. Bei Tischtennis und Baseball sollten die Spieler also neben der R+T- auch die Rotationssteuerung und teilweise auch einen Modus zu spielen, bei dem nur Translation verwendet wurde.

Es wurde bei allen Tests das konvergente Stereotrackingverfahren verwendet. Bei den allermeisten Tests wurde außerdem der Aufsatz, der in Abschnitt 6.2.2 beschrieben wird eingesetzt. Während der Evaluationsphase wurde die Form dieses Aufsatzes verbessert, so dass das Tracking bei den späteren Tests sehr robust verlief. Die Infrarotkameras hatten am Ende überhaupt kein Problem mehr, den Aufsatz zu verfolgen. Da das Tracking insbesondere bei der Applikation Tischtennis bei den allerersten Tests aufgrund des damals noch nicht verwendeten Aufsatzes noch nicht sehr robust funktionierte, gehen bei Fragen zu Tischtennis diese Antworten nicht in das Endergebnis ein. Billiard funktionierte allerdings auch ohne Aufsatz schon sehr gut, also waren die Antworten für diese Applikation von Beginn

an repräsentativ und gingen ins Ergebnis mit ein.

7.2 Der Fragebogen

Der komplette Fragebogen ist im Anhang zu finden. Die wichtigsten Fragetypen werden nun zusätzlich erläutert, und beschrieben warum bestimmte Fragen ausgewählt wurden.

Zunächst wurde gefragt ob der Spieler auch in seiner Freizeit häufig Konsolen oder Computerspiele spiele, und ob er bereits Erfahrung mit Wii und MotionPlus hätte. Dadurch konnte man testen, ob die Anforderungen der Applikationen den Personen leichter fallen würden, die bereits Spielerfahrung in anderen Spielen besitzen. Weiterhin wurde nach der Motivation vor Beginn des Spiels gefragt. Dies ist von daher interessant, um zu testen, ob bestimmte Erwartungen erfüllt oder eher enttäuscht wurden. Anschließend sollten die Spieler bewerten, wie sich die unterschiedlichen Steuerungsmodi auf die Immersion auswirken, bewerten ob in den einzelnen Spielen die Bewegungen des Controllers erwartungsgemäß übertragen wurden und ob die R+T-Steuerung präzise sei. Es wurde auch gefragt, ob Verzögerungen, oder ein Zittern des Objekts im Spiel festgestellt wurden. Falls ja, so wurde gefragt, ob sich eines dieser Probleme negativ auf das Spielgefühl auswirkte. Diese Fragen wurden gestellt, um bewerten zu können, ob die Steuerung auch in den Augen der Benutzer angemessen funktioniert.

Schließlich wurden nochmal spezifische Fragen zu den einzelnen Applikationen gestellt. Bei Tischtennis und Baseball sollte die Schwierigkeit und die Kontrolle, die der Spieler mit der Steuerung über den Spielverlauf hat, bewertet werden. Deshalb wurde gefragt, ob es einfach war, mit den verschiedenen Steuerungsmodi einen Ball zu treffen, bzw. ihn in eine bestimmte Richtung zu zielen. Das Tischtennisspiel sollte nocheinmal mit R+T-Steuerung getestet werden, wobei in diesem Fall mehr oder weniger stark bei den Bildkoordinaten der IR-Kameras interpoliert wurde. Die Spieler sollten dann bewerten, ob die daraus entstandene Verzögerung der Translation sich negativ auf das Spielgefühl auswirkte.

Jede Applikation sollte nach Spielspaß und Langzeitmotivation bewertet werden. Bei Tischtennis und Baseball wurde gefragt, ob die Spieler der Meinung wären, dass sie mit R+T-Steuerung mehr Möglichkeiten hätten, Einfluss auf die Flugbahn des Balles zu nehmen. Zum Schluss wurde noch gefragt, welche Applikation dem Spieler am besten gefallen hätte und welche man wohl am längsten spielen würde.

7.3 Ergebnisse und Interpretationen

Insgesamt wurden über 25 Fragen gestellt, die sich im Anhang finden lassen. Die wichtigsten Antworten werden nun in Diagrammform präsentiert, jedoch nicht alle, da dies den Rahmen sprengen würde. Es gehen jedoch alle Antworten in die nun folgende Interpretation mit ein.

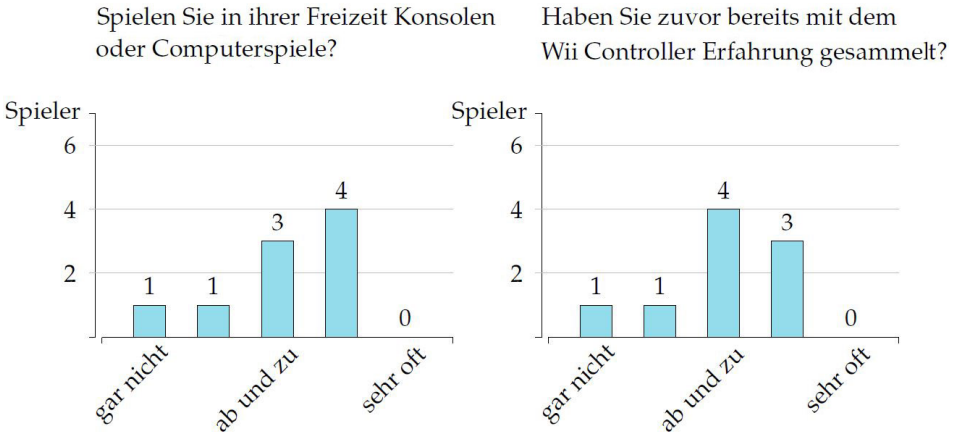

Abbildung 30

Fragen zur Spielerfahrung Zunächst wurden die Testpersonen zu ihrer Erfahrung mit Computerspielen im Alltag befragt. Abbildung 30 zeigt dazu 2 Ergebnisse. Die meisten Personen spielten ab und zu bis häufig, wenige spielten sehr selten oder gar nicht. Sehr ähnlich sieht es mit der Spielerfahrung mit dem Wii-Controller aus. Allerdings hatten nur die wenigsten Spieler Erfahrung mit dem Zusatzadapter MotionPlus gesammelt. Das liegt wahrscheinlich daran, dass dieser erst 3 Jahre nach Erscheinen der Wii-Konsole auf den Markt kam. Es zeigt sich aber, dass nicht nur erfahrene Spieler getestet wurden. Die Leute wurden auch nach Ihrer Motivation zu Beginn der Spiele gefragt, also bevor sie anfingen zu spielen. Diese war bei fast allen Personen hoch bis sehr hoch, was zeigt, dass die freie Bewegungssteuerung Neugier weckt. Eine Person, die auch sonst keine Computerspiele spielt, gab aber an, zu Beginn gar keine Motivation gehabt zu haben, sei dann aber letztendlich positiv überrascht gewesen, was sich auch daran zeigte, dass sie den meisten Applikationen hohe Spielspaßwertungen gab.

Fragen zur Immersion Die Spieler sollten bewerten, wie sehr sie sich in die virtuelle Welt einbezogen fühlten, wie sehr sie also das Gefühl hatten, tatsächlich in der Welt als Spieler interagieren zu können. Diese Frage

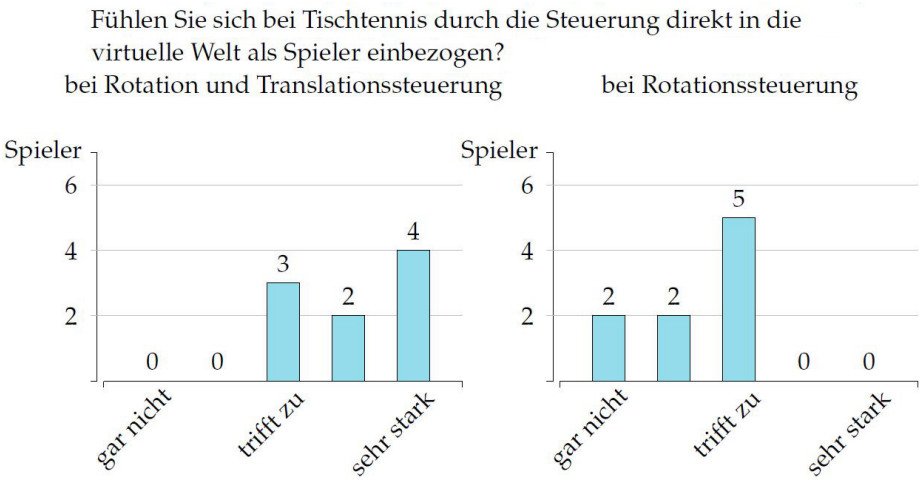

Abbildung 31

bezieht sich also auf die Immersion. Zu jeder der 3 Applikationen wurde diese Frage gestellt, bei Tischtennis und Baseball sollte aber zusätzlich die R+T-Steuerung mit der R-Steuerung verglichen werden. Dies ist ganz besonders interessant, um beurteilen zu können, ob die Erweiterung der Rotationssteuerung durch die Translation tatsächlich eine Verbesserung der Immersion bewirkt. Und tatsächlich wurde die Immersion der R+t-Steuerung viel höher bewertet, als bei der R-Steuerung, wie es in Abbildung 31 bezüglich des Tischtennisspiels zu sehen ist. 4 Spieler gaben sogar an, sich sehr stark in die Welt eingebunden zu fühlen. Man muss fairerweise anmerken, dass die Tischtennisapplikation auch für die R+T-Steuerung optimiert worden ist. Wäre von Anfang an nur eine Rotationssteuerung geplant gewesen, wäre diese wohl auch noch verbessert worden. Abbildung 32 zeigt wie sehr sich die Spieler bei Billiard und Baseball ins Spiel einbezogen fühlten. Bei Billiard sind die Werte ebenfalls sehr hoch, obwohl man dort nur relativ wenig Freiheitsgrade bei der Steuerung hat. Allerdings ist die Steuerung wohl so auf das Billiardspiel optimiert worden, dass es gerade wegen der wenigen Freiheitsgrade sehr gut spielbar ist. Außerdem wurden die Kameras beim Zielen und Stoßen so gesetzt, dass man stets viel Überblick hat. Dadurch dass der Spieler die Kamera beim Zielen durch Bewegen der Hand drehen kann, entsteht wohl zusätzliche Immersion. Baseball dagegen wurde weniger gut bewertet. Die Spieler hatten hier manchmal große Probleme, den Ball richtig zu treffen, wodurch Frust entsteht. Darunter könnte die Immersion leiden. Es kann auch sein, dass die Bewegungen des Schlägers hier zu schnell eingestellt wurden, und der Spieler dies deshalb als unrealistisch empfand.

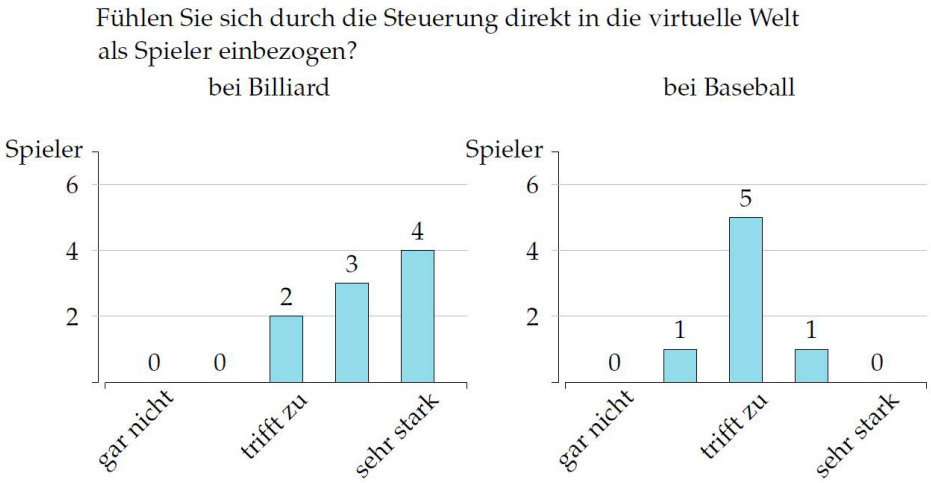

Abbildung 32

Fragen zur Steuerung Als nächstes sollten die Spieler die Steuerung an sich bewerten. Dies war besonders interessant, da ich zur Zeit, als ich noch das achsenparallele Tracking eingesetzt habe das Zittern des Schlägers im Spiel sehr störend fand. Als ich dies mit Interpolation zu unterdrücken versuchte, empfand ich die Verzögerung, die diese mit sich bringt wiederum sehr störend. Mit dem konvergenten Tracking wurde das Zittern aber stark verringert. Es tritt jedoch noch teilweise auf, wenn man den Schläger nach links und rechts bewegt, also parallel zur x/y-Ebene des Weltkoordinatensystems. Mit dem konvergenten Tracking wurde die Interpolation allerdings komplett weggelassen, der Schläger müsste sich also sehr direkt steuern lassen. Die Spieler wurden deshalb nach dem Zittern gefragt, und ob sie eine Verzögerung bemerkt hätten. Abbildung **??** zeigt, dass die meisten Spieler gar kein Zittern bemerkt hatten, wahrscheinlich weil dieses nur entlang einer Achse auftritt. Wenige hatten aber doch ein Zittern bemerkt und dieses auch als leicht störend empfunden. Abbildung 33 zeigt das meist gar keine, bzw. sehr geringe Verzögerungen bemerkt wurden, der Schläger also direkt auf die Bewegungen reagierte.

Die Spieler sollten bewerten, wie gut ihre Bewegungen bei der R+T-Steuerung bei den Applikationen Tischtennis und Baseball übertragen wurden. Damit ist gemeint, ob zum Beispiel der Tischtennisschläger im Spiel relativ genau die Bewegungen der Wiimote nachahmt. Ebenso sollten sie die Steuerung bei Billiard bewerten, bei dem keine R+T-Steuerung verwendet wird. Abbildung 34 zeigt dabei einen Vergleich zwischen den Applikationen Tischtennis und Baseball. Wie man sieht wurde Tischtennis dabei zu gleichen Anteilen gut und sehr gut bewertet. Dies zeigt wohl, dass die R+T-Steuerung sich für dieses Spiel gut eignet und großen Anklang gefunden hat. Auch bei Billiard wurde die Steuerung größtenteils gut bewertet. Allerdings kann

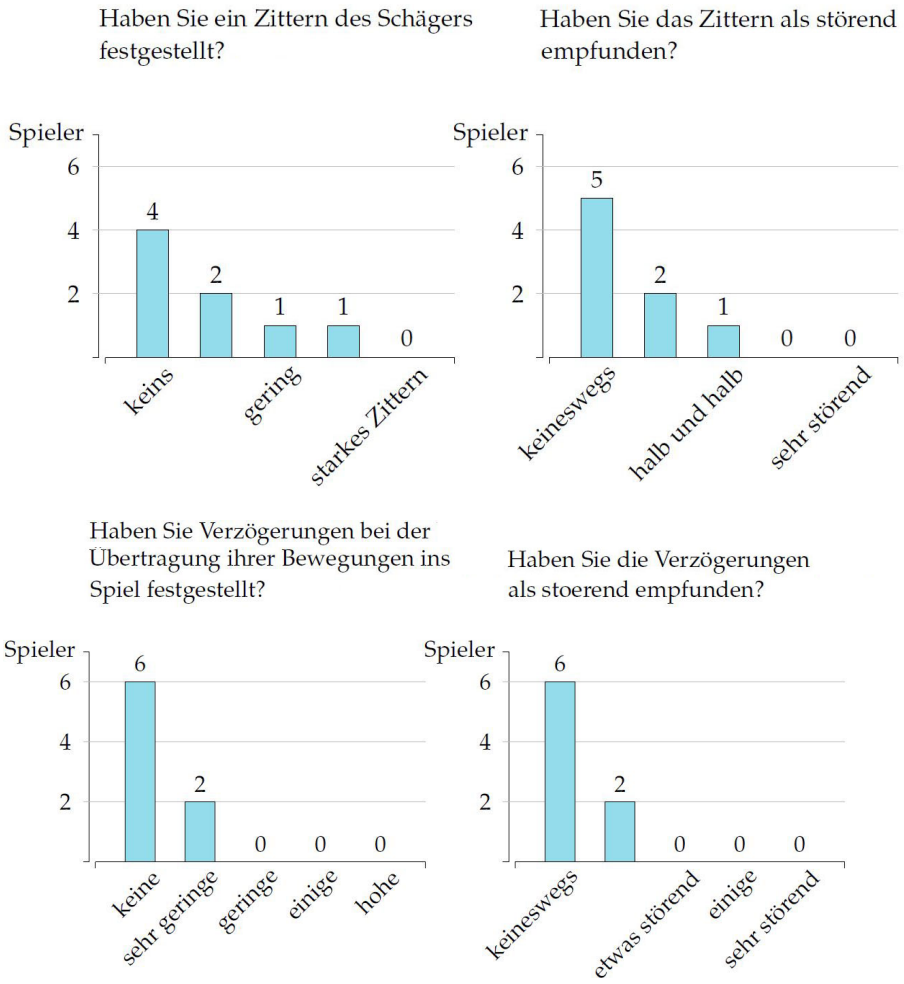

Abbildung 33

man hier auch nur Zielen und anschließend entlang einer Bewegungsachse Stoßen. Es würde hier also nicht so stark auffallen, wenn irgendetwas mit der Steuerung nicht stimmt. Bei Tischtennis könnte es, wenn zuvor schlecht kalibriert wurde, zum Beispiel passieren, dass der Schläger nicht exakt den Bewegungen des Spielers folgt, er sich zum Beispiel nach links bewegt, der Schläger dabei aber etwas nach vorne geht. So etwas kann bei Billiard mit der eingeschränkten Bewegungsfreiheit allerdings nicht passieren. Das Diagramm zeigt aber, dass die Personen zufrieden damit waren, wie der Queue auf das nach vorne Stoßen des Spielers reagiert. Bei Baseball wurde die Übertragung der Bewegung auf den Schläger allerdings etwas schlechter bewertet. Die meisten Personen fanden die Umsetzung dort mittelmäßig bis gut.

In einer zusätzlichen Frage wurde nach der Präzision der Steuerung ge-

fragt. Damit ist zum Beispiel gemeint, ob der Schläger im Spiel ruckelt, weil beispielsweise das Tracking zwischenzeitlich kurz ausfällt, oder ob er sich im Gegenteil sehr flüssig bewegt. Abbildung 35 zeigt das Ergebnis. Wie man sieht wurden die Bewegungen des Tischtennisschlägers als präzise und flüssig wahrgenommen. Der Grund hierfür liegt hauptsächlich in der Verwendung von Infrarotarrays und dem relativ großen, für die Kameras stets gut sichtbaren reflektierenden Aufsatz. Bei Billiard wurde die Präzision ebenfalls recht gut bewertet, bei Baseball allerdings schlechter, da der Schläger sich in den Augen der Spieler zu schnell bewegt hatte. Außerdem war er manchmal zu weit von der Kamera entfernt, und deshalb nicht mehr so gut sichtbar.

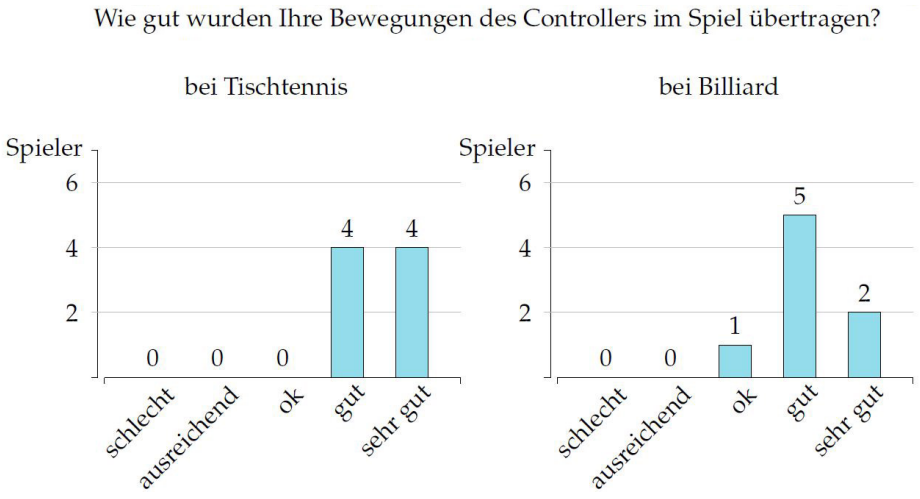

Abbildung 34

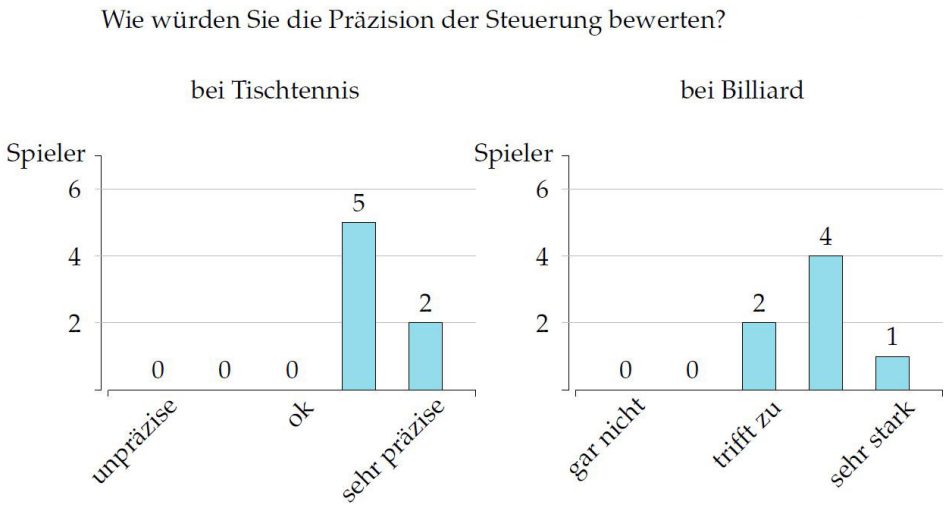

Abbildung 35

Fragen zur Kontrolle Nun folgten Fragen, die hauptsächlich auf den Schwierigkeitsgrad der Spiele, aber auch auf die Kontrolle abzielten. Es wurde gefragt, ob die Spieler bestimmte Schläge ausführen konnten, und ob es ihnen leicht oder schwer fiel, zu zielen. Daraus kann man gut auf die Komplexität der Steuerung rückschließen, und ob die R+T-Steuerung eher als Hürde oder als Erleichterung im Gegensatz zur R-Steuerung empfunden wird.

Die meisten Fragen bezogen sich hier auf die Applikation Tischtennis, weil der Spieler dabei volle Bewegungsfreiheit hat und gleichzeitig den Ball treffen und zielen können muss.

Abbildung 36 zeigt, dass es einigen Spielern beim Tischtennis sehr leicht fiel, den Ball zu treffen, einige hatten dabei aber große Probleme. Dies zeigt, dass die R+T-Steuerung nicht unbedingt die einsteigerfreundlichste Methode darstellt. Allerdings fiel es manchen Leuten auch bei der Rotationssteuerung nicht leicht, den Ball überhaupt zu treffen. Dazu muss gesagt werden, dass der reflektierende Aufsatz leider die Infrarotkamera des Controllers verdeckt, und deshalb ab und zu abgenommen werden musste, um den Yaw-Winkel wieder neu zu kalibrieren (vgl. Abschnitt 6.1.2). Unzulänglichkeiten dieser Art stören leider den Spielfluß und erschweren die Kontrolle. Auch das genaue Zielen des Balles fiel manchen Testpersonen nicht immer leicht, wie Abbildung 37 zeigt. Einige kamen dabei aber sehr gut zurecht. Ob das Zielen mit der R-Steuerung leichter, als mit der R+T-Steuerung fiel, lässt sich nicht eindeutig bestimmen.

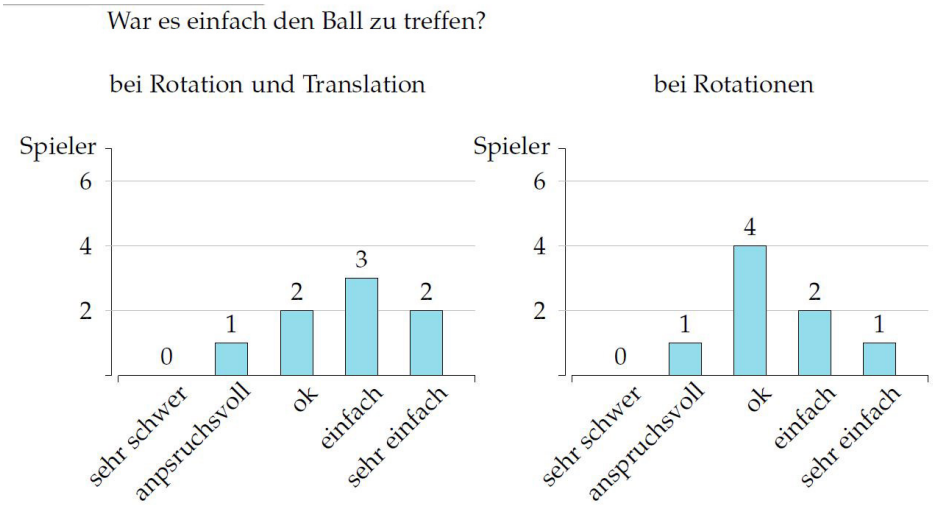

Abbildung 36

Die Personen wurden aber auch gefragt, wie schwer es war, grob die Richtung des Balles zu bestimmen, also diesen cross, oder long line zu schlagen. Dies fiel den Leuten bei der R+T-Steuerung relativ leicht, bei der R-

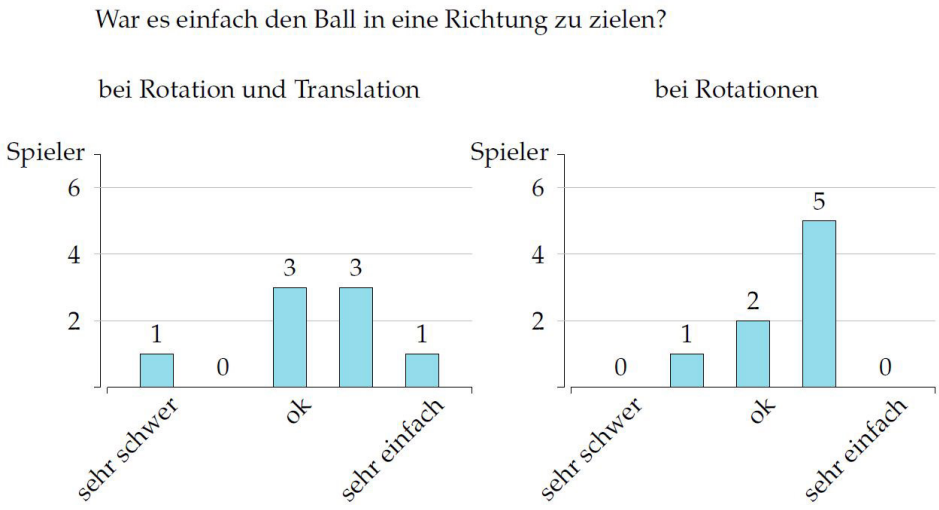

Abbildung 37

Steuerung dagegen etwas schwerer. Daran kann man sehen, wie einfach es ist, den Schläger so zu orientieren, dass man den Ball gerade oder schräg schlägt. Zu bestimmen, ob der Ball lang oder kurz geschlagen wurde, fiel den meisten mittelmäßig schwer. Dies kann zum Beispiel mit der Kraft bestimmt werden, mit der man den Schläger nach vorne bewegt. Man muss dabei anmerken, dass die Physikengine teilweise recht ungenau arbeitet. Sie berechnet, wie viel Kraft der Schläger gegen den Ball aufwendet. Hier könnte man möglicherweise beim Spieldesign eingreifen, und das Spiel vereinfachen, indem man bei jeder Kollision mit dem Ball die Kräfte nicht komplett der Physikengine überlässt, sondern so beeinflusst, dass der Ball in den meisten Fällen auch auf die gegnerische Platte fliegt. Bei Tischtennis sah man aber auch, dass einige Spieler zu Beginn zwar Probleme hatten überhaupt abschätzen zu können, wie fest sie den Ball schlagen könnten, oder wie stark sie die Flugrichtung beeinflussen konnten. Mit etwas mehr Spielerfahrung konnten viele den Ball aber immer besser kontrollieren. Einige Personen sollten auch einen Modus ausprobieren, bei dem der Ball nicht immer an dieselbe Stelle, sondern zufällig an unterschiedliche Stellen auf der eigenen Platte fliegt. Dabei muss der Spieler den Schläger stets in eine neue passende Ausgangsposition bringen. Es stellte sich heraus, dass die Personen große Probleme hatten, den Ball dann überhaupt zu treffen. Möglicherweise waren die Bälle aber auch zu extrem gewählt, man hätte den Modus also noch vereinfachen können. Außerdem ist es schwer, trotz der Schatten, exakt auszumachen, wo sich der Ball jeweils im Spiel befindet, da der Spieler nicht dieselben Tiefeninformationen hat, die er im richtigen Leben kennt.

Als nächstes sollte herausgefunden werden, ob die Personen das Gefühl

hätten, bei Tischtennis mit der R+T-Steuerung insgesamt mehr Schlagmöglichkeiten zu haben. Dabei sollten die Spieler zunächst mit der R+T-Steuerung, und anschließend mit der R-Steuerung einen Cross-, und dann einen Long Line Schlag ausführen. Danach wurden sie gefargt, auf welche Art und Weise sie die Flugbahn des Balls bestimmen konnten. Fast alle Spieler bemerkten, dass es bei der Roationssteuerung eher auf Timing ankam und wie sie sich zum Ball stellten. Bei der R+T-Steuerung fiel den meisten auf, dass sie die Flugbahn auch durch die Orientierung des Schlägers beeinflussen konnten, und durch die Richtung, in die sie den Schläger bewegten. Den Leuten war also durchaus bewusst, dass die R+T-Steuerung komplexer ist, aber dadurch auch mehr Möglichkeiten bietet.

Billiard wurde wegen der vereinfachten Steuerung ein geringerer Schwierigkeitsgrad attestiert, wie in Abbildung 38 zu sehen ist. Den Leuten fiel es von Beginn an relativ leicht, im Zielmodus die Kugel so anzuvisieren, dass sie bei einem Stoß in die gewünschte Richtung rollt. Allerdings heißt das nicht, dass die Personen mit Leichtigkeit alle Kugeln in die Löcher versenken konnten. Dies ist allerdings beim Billiard im echten Leben auch nicht gerade einfach.

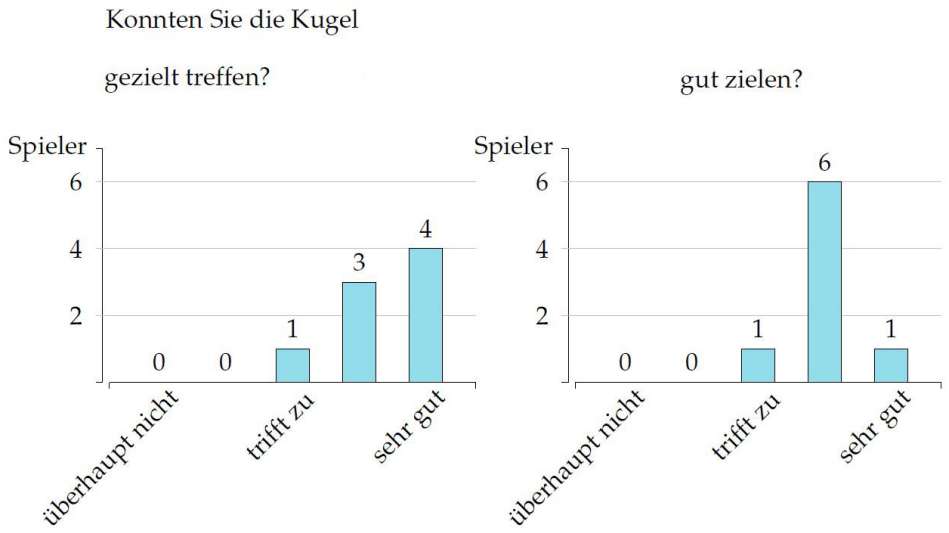

Abbildung 38

Bei Baseball dagegen fiel es den meisten Leuten mit R+T-Steuerung bedeutend leichter, den Ball zu treffen, als mit der R-Steuerung. Zielen fanden die meisten aber recht schwer. Den meisten Spielern fiel es mittelmäßig leicht, den Ball auch in einen Korb zu treffen, was ja das Ziel des Spiels ist. Viele bemängelten, dass der Ball zu schnell zum Spieler angeflogen kam, und sich der Schläger zu schnell bewegte. Hier besteht also noch Verbesserungsbedarf. Insgesamt wurde der Schwierigkeitsgrad als zu hoch bewertet.

Einige Testpersonen sollten auch ausprobieren, wie sich Interpolation auf das Verhalten des Tischtennisschlägers auswirkt. Die Interpolation wäre zwar nicht nötig gewesen, aber die Antworten waren von Interesse, um abwägen zu können, ob sich das konvergente Stereotracking auch wirklich positiv bemerkbar macht, indem das Zittern des Objekts verringert und damit Interpolation unnötig gemacht wird. Tatsächlich beklagten die Personen, dass sich die Steuerung mit Interpolation schwammig anfühlte, und es schwieriger sei, den Ball zu treffen und die Kraft, die man auf den Ball ausüben will, zu kontrollieren.

Fragen zu Spielspaß und Langzeitmotivation Sehr wichtig sind natürlich die Fragen zu Spielspaß und Langzeitmotivation. Hier ist auch wichtig, die R+T-Steuerung mit der R-Steuerung zu vergleichen. Ebenso wie bei der Immersion wurde Tischtennis bei der R+t-Steuerung sehr viel Spielspaß attestiert, die R-Steuerung dagegen wurde relativ schlecht bewertet, wie in Abbildung **??**TTSpielspassDiag) zu sehen. Auch die Billiardsimulation wurde beim Spielspaß hoch bewertet. Einige Personen sprachen Billiard auch hohe Langzeitmotivation zu, zum Teil auch weil man das Spiel abwechselnd in Teams gegeneinander spielen kann. Tischtennis und Baseball wurde weniger Langzeitmotivation zugesprochen. Dies liegt auch daran, dass es keinen Mehrspielermodus gibt, bei dem man gegen einen echten menschlichen Gegner antreten kann, die Möglichkeiten sind von daher eingeschränkt. Auch sind alle Applikationen natürlich nicht mit Vollpreisspielen zu vergleichen, die selbstverständlich eine höhere Langzeitmotivation bieten. Die Spieler gaben aber an, die Tischtennisapplikation mit der R+T-Steuerung wohl länger spielen zu wollen, als mit der R-Steuerung.

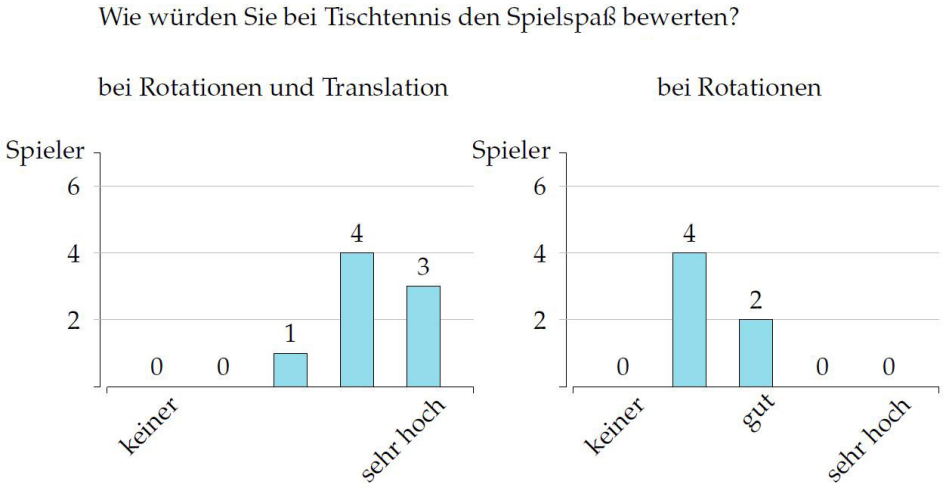

Abbildung 39

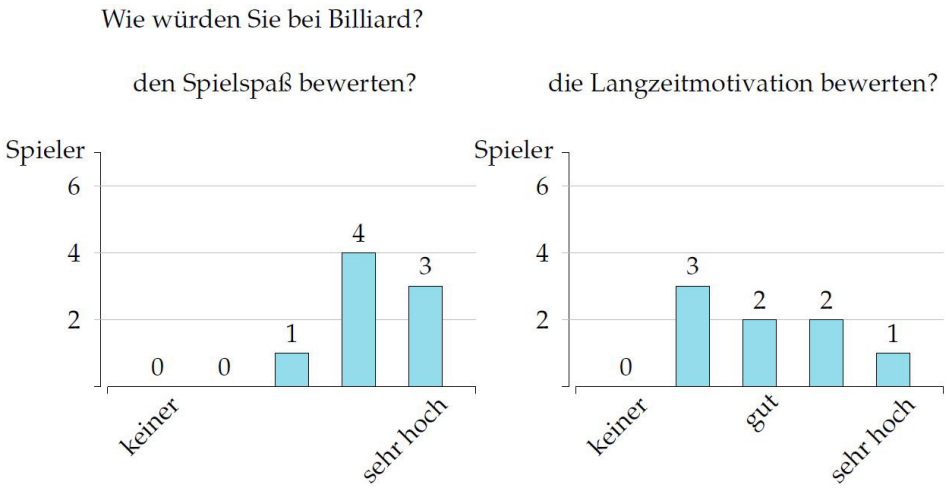

Abbildung 40

Abschließende Fragen Zum Schluss wurde gefragt welches Spiel den Testpersonen am wenigsten, und welches ihnen am besten gefallen hätte. Abbildung 41 zeigt, dass Billiard dabei knapp vor Tischtennis liegt, Baseball aber wohl den letzten Platz verdient. Bei der Frage, welches Spiel man wohl auf Dauer am längsten spielen würde, gaben die meisten Billiard den Vorzug, Tischtennis bekam dabei den zweiten Platz.

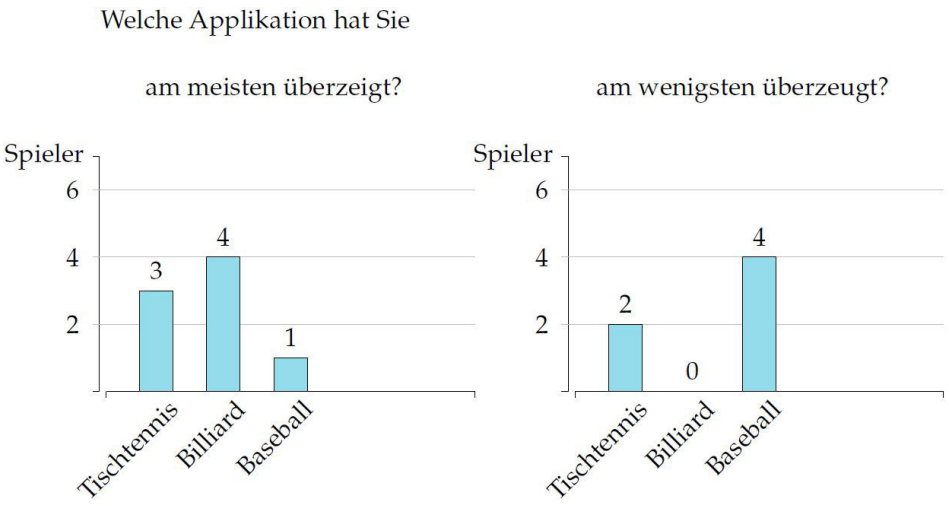

Abbildung 41

7.4 Zusammenfassung

Im Folgenden sollen die Schlussfolgerungen nochmals zusammengefasst werden. Die Hypothese, dass die R+T-Steuerung die Immersion erhöhen würde, wurde von den Testpersonen eindeutig bestätigt. Dies trifft vor allem bei Tischtennis zu, etwas weniger auch bei Baseball. Die hohe Immersion liegt wahrscheinlich daran, dass der Spieler bei der R+T-Steuerung ein sehr direktes Feedback hat. Der Tischtennisschläger macht beinahe exakt die Bewegungen, die der Spieler gerade mit der Wiimote vollführt. Aufgrund der Infrarotarrays und des reflektierenden Aufsatzes sind die Bewegungen dabei sehr flüssig. Dies wurde von den Testpersonen auch weitgehend bestätigt.

Es zeigt sich aber auch, dass die R+T-Steuerung recht hohe Komplexität mit sich bringt, und es für den Spieler eine Zeit lang dauert, bis er sich eingewöhnt hat. Je mehr Bewegungen man machen kann, desto mehr kann man auch falsch machen. Manchen Spielern fiel die Steuerung zu Beginn recht schwer, was auf eine geringe Einsteigerfreundlichkeit schließen lässt. Es bedarf also durchaus etwas Übung, bei Tischtennis den Ball zu treffen und zu zielen.

An der Billiardsimulation kann man sehen, dass weniger manchmal mehr ist. Die eingeschränkte Bewegungsfreiheit eignete sich gut für dieses Spiel. Spielspaß, Langzeitmotivation und Kontrolle, aber auch die Immersion wurden recht hoch bewertet. Hätte der Spieler hier die volle Kontrolle über den Queue, wäre das exakten Treffen der weißen Kugel wohl zu schwer geworden. Diese muss ja wiederum auch die Kugeln des eigenen Teams an der richtigen Stelle mit der richtigen Kraft treffen. Man hätte dies die Testpersonen wohl noch ausprobieren lassen können, allerdings wären die Tests dann weitaus aufwändiger geworden.

Die Langzeitmotivation der Steuerung ist recht schwer zu bewerten. Da das Tischtennisspiel zum Beispiel keinen zweiten Spieler erlaubt, kann das Ganze nach einiger Zeit eintönig wirken. Darauf kann aber noch nicht auf die Steuerung geschlossen werden.

Das wichtigste Kriterium eines Spiels ist aber dennoch der Spielspaß. Hier haben die Applikationen Billiard und Tischtennis recht hohe Werte eingefahren, wobei bezeichnend ist, das Tischtennis mit der freien R+T-Steuerung besser bewertet wurde als bei der R-Steuerung. Dies heißt jedoch nicht, dass alle Spiele automatisch von einer freien Bewegungssteuerung profitieren. Die Billiardsimulation beweist das Gegenteil. Dennoch wäre Billiard ohne die Erweiterung mit der Translation so nicht möglich gewesen.

8 Ausblick

8.1 Weitere Einsatzmöglichkeiten für das System

Neben den vorgestellten und umgesetzten Beispielapplikationen ließen sich mit der R+T-Steuerung, die ich im Rahmen des Projekts umgesetzt habe, zahlreiche weitere Einsatzmöglichkeiten finden. Einige Möglichkeiten sollen nun vorgestellt werden.

Pacman Dies ist eine Applikation, die eigentlich im Rahmen dieser Diplomarbeit beinahe fertig entwickelt wurde. Der Spieler sollte dabei mit Hilfe eines Schlägers einen Ball durch ein Labyrinth „schieben", indem er diesen hin und herstößt. Mit dem Ball konnten Münzen eingesammelt werden, die im Level verteilt sind. Es gab einen Level ohne und einen mit Höhenunterschieden, der sich demnach etwas schwerer spielte, da der Ball in Tälern steckenbleiben konnte. Weiterhin existierten, ähnlich wie beim original Pacman, zwei Gegner, die ebenfalls durch Bälle repräsentiert wurden. Diese rollten vom Computer gesteuert durch das Level. Immer wenn sie an einer Wand anstießen, konnten sie die Richtung zufällig wechseln. Abbildung 42 zeigt einen Screenshot aus dem Spiel. Die Kamera betrachtet das Spielgeschehen von oben, sodass es ähnlich wie beim 2D-Pacman aussieht, die Spielwelt ist aber eigentlich dreidimensional. Der Schläger sollte dabei immer auf der Höhe des Balls bleiben, und konnte vom Spieler nur entlang der $x/z-$Ebene bewegt werden. Das Spiel funktionierte auch, allerdings gab es zum Zeitpunkt als die Tests durchgeführt wurden noch einige Probleme mit der Steuerung. Der Schläger sollte eigentlich automatisch in einigem Abstand dem Ball folgen, damit der Spieler nicht umständlich die Wiimote zum Ball bewegen muss. Über Bounding Volumes sollte dann getestet werden, ob geschlagen wurde. In diesem Moment sollte der Schläger dem Ball nicht mehr folgen, da sich das Physikobjekt sonst kontinuierlich innerhalb des Balls befinden würde und diesen bewegen würde. Letztendlich gab es einige Probleme, da der Schläger nicht immer wie gewünscht dem Ball folgte. Da diese Fehler bei den Tests noch nicht behoben waren, wurde die Applikation nicht mehr Teil dieser Diplomarbeit. Dennoch bietet es sich an, das Spiel weiterzuentwickeln.

Wiimote kontrollierte MRI-Visualisierung Im Bereich der medizinischen Bildbearbeitung produzieren Techniken wie Computertomographie und Magnetic Resonance Imaging (MRI) heutzutage dreidimensionale Ansichten des menschlichen Körpers [6]. Um diese sogenannten Volumendaten betrachten zu können, existieren verschiedene Interaktionsmetaphern. So lässt sich ein solches Volumen beispielsweise mit Maus oder Tastatursteue-

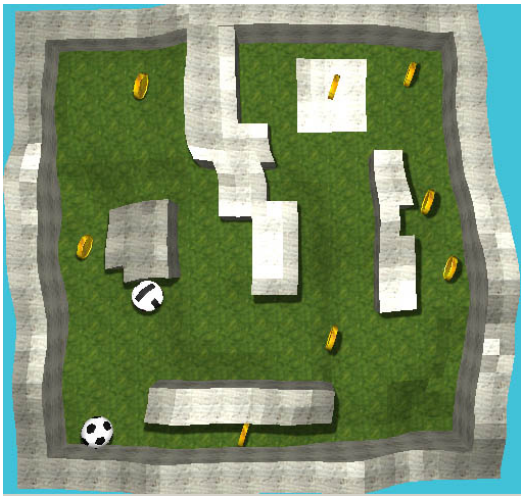

Abbildung 42: Die Pacmanapplikation

rung stufenlos drehen, oder mit dem Mausrad heran- und herauszoomen. Das entwickelte R+T-Steuerungsmodell mit der Wiimote wäre allerdings ideal geeignet, um die bereits vorhandenen Systeme durch ein einfacheres System zu ersetzen. Von Vorteil wäre besonders die exakte Rotation, die mit MotionPlus ermöglicht wird. So könnten die Rotationen der Wiimote im Raum, die der Benutzer durchführt auf ein Objekt wie beispielsweise ein Gehirn übertragen werden. Vorteilhaft wäre hier, das der Benutzer ganz genau bestimmen kann, wie schnell sich das Objekt dreht. Es würde sich beinahe anfühlen, als hätte man das Objekt selbst in der Hand und könnte es von allen Seiten aus betrachten.

Google Earth Interaktion mit der Wiimote die Webapplikation Google Earth bietet dem Benutzer ein dreidimensionales vereinfachtes Modell der Erde mit Kartenfunktion. Es besteht die Möglichkeit in jeden Bereich des Planeten herein zu zoomen und so auch Städte oder Flora und Fauna in einer dreidimensionalen Umgebung zu betrachten. Möglicherweise könnte das Programm auch mit dem vorgestellten System bedient werden. Durch Rotation der Wiimote ließe sich die Welt drehen. Durch Translation bei gedrückter A-Taste könnte man sich schließlich nach oben, unten, rechts links und durch eine Bewegung nach vorne und hinten auch herein- und herauszoomen.

WiiMote steuert Kamera Mit Hilfe der Rotation, die durch MotionPlus gemessen wird, ließe sich auch eine Kamera wie in einem 3D-Shooter steuern. Im Laufe des Projekts habe ich dies kurz getestet. Diese Steuerungsart

bietet eine hohe Immersion, da selbst kleinste Rotationen der Hand in die Kamerasteuerung umgesetzt werden und für den Spieler subjektiv keine Verzögerung wahrnehmbar ist.

Problematisch ist allerdings, dass der Spieler keine 180 Grad Drehung machen kann, ohne den Blick auf den Bildschirm zu verlieren. Deshalb müsste man man hier mit einem Trick abhelfen. Nur wenn der Spieler eine Taste gedrückt hält, ließe sich die Kamera drehen, danach könnte man die Wiimote wieder in eine andere Orientierung versetzen, ohne die Kamera zu rotieren.

8.2 Persönliches Fazit

Rückblickend lässt sich sagen, dass das Projekt, also die Implementierung der Steuerung und auch die Entwicklung der Applikationen mit anschließendem Test, äußerst interessant und abwechslungsreich war. Die zu Beginn gestellten Anforderungen wurden eigentlich alle erfüllt, auch wenn der Weg zum Ziel teilweise über Umwege führte. So war anfangs eigentlich geplant, lediglich das achsenparallele Tracking zu verwenden. Erst als dieses Probleme mit der Steuerung offen legte, bin ich zum konvergenten Stereotracking umgestiegen. Erstaunlich war auch, dass einige Konzepte, wie zum Beispiel die Verwendung des reflektierenden Aufsatzes erst mit dem konvergenten Tracking richtig funktionieren wollten. Bei den Tests, bei denen es anfangs noch Probleme gab, hat sich der Aufsatz schließlich besonders bezahlt gemacht.

Hierbei sei abschließend noch auf ein Video verwiesen, das die Tischtennisapplikation in Aktion zeigt[45]. Dabei ist der Spieler sichtbar, sodass man vergleichen kann, wie die Bewegungen im Spiel umgesetzt wurden. Ein weiteres Video zeigt das Billiardspiel[46].

Wie man sehen kann, funktionierte das Stereotracking am Ende sehr gut, und die Wiimote konnte aus vielen verschiedenen Positionen und Orientierungen verfolgt werden. Die meisten Testpersonen bestätigten ein immersives Spielerlebnis durch die Steuerung. Allerdings sollte die Einsteigerfreundlichkeit noch verbessert werden, und bei manchen Applikationen war der Schwierigkeitsgrad durch die Komplexität teilweise zu hoch. Die Steuerung hat also nicht nur Vorteile, wirkte sich aber im Vergleich zur ausschließlichen Verwendung von Rotationen positiv auf den Spielspaß aus. Das bedeutet letztendlich natürlich nicht, dass die freie Bewegungssteuerung allen anderen Steuerungsmethoden überlegen ist. Vielmehr bietet sie

[45] Video der Tischtennisapplikation (das Video startet nach 10 Sekunden): http://www.youtube.com/watch?v=vMQFJ7bpbo4

[46] Video der Billiardapplikation: http://www.youtube.com/watch?v=2Jh2JP1XLbQ

ein neues, den meisten Menschen unbekanntes Spielerlebnis, dass ebenso wie das altbekannte Konzept der Gamepad- oder der Maus- und Tastatursteuerung seine Daseinsberechtigung hat. Jede Steuerungsmethode bietet ihre Vor- und Nachteile, neue Interaktionsmetaphern, wie die freie Bewegungssteuerung, können die Welt der Konsolen- und Computerspiele jedoch bereichern.

Anhang

Fragebogen

Name _____ Alter _____

Allgemeines

1. Spielen Sie in ihrer Freizeit Konsolen oder Computerspiele?

gar nicht		ab und zu		sehr oft
☐	☐	☐	☐	☐
1	2	3	4	5

2. Haben Sie zuvor bereits mit dem Wii Controller Erfahrung gesammelt?

gar nicht				sehr oft
☐	☐	☐	☐	☐
1	2	3	4	5

3. Haben Sie bereits Erfahrung mit dem Zusatzadapter MotionPlus gesammelt?

gar nicht	ab und zu	sehr oft
☐	☐	☐

Fragen zu den 4 Applikationen

4. Wie wuerden Sie ihre Motivation zu Beginn bewerten?

keine	wenig	ok	hoch	sehr hoch
☐	☐	☐	☐	☐

5. Fühlen Sie sich durch die Steuerung direkt in die virtuelle Welt als Spieler einbezogen?

	gar nicht		trifft zu		sehr stark
bei Tischtennis	☐	☐	☐	☐	☐
bei Billiard	☐	☐	☐	☐	☐
bei Baseball	☐	☐	☐	☐	☐

6. Wie gut wurden Ihre Bewegungen des Controllers im Spiel übertragen?

	schlecht		ok		sehr gut
bei Tischtennis	☐	☐	☐	☐	☐
bei Billiard	☐	☐	☐	☐	☐
bei Baseball	☐	☐	☐	☐	☐

7. Wie wuerden Sie die Praezision der R+T-Steuerung bewerten?

	unpraezise		ok		sehr praezise
bei Tischtennis	☐	☐	☐	☐	☐
bei Billiard	☐	☐	☐	☐	☐
bei Baseball	☐	☐	☐	☐	☐

Allgemeines zur Steuerung

8. Haben Sie Verzögerungen bei der Übertragung ihrer Bewegungen ins Spiel festgestellt?

	keine	sehr geringe	geringe	einige	hohe
bei Rotationen	☐	☐	☐	☐	☐
bei Translationen	☐	☐	☐	☐	☐

9. Wenn ja, haben Sie die Verzögerungen als störend empfunden?

	keineswegs störend	nicht störend	mehr oder weniger	ziemlich störend	sehr störend
bei Rotationen	☐	☐	☐	☐	☐
bei Translationen	☐	☐	☐	☐	☐

10. Hat sich bei der Translation des Schlägers ein Zittern im Spiel bemerkbar gemacht?

	trifft keineswegs zu	trifft nicht zu	halb und halb	trifft zu	trifft stark zu
Ich habe ein Zittern festgestellt	☐	☐	☐	☐	☐
Wenn ja: Das Zittern hat beim Spielen sehr gestört	☐	☐	☐	☐	☐

Tischtennis

11. War es einfach, den Ball zu treffen?

	sehr schwer	anspruchsvoll	ok	einfach	sehr einfach
bei Rotation	☐	☐	☐	☐	☐
bei Rotation und Translation	☐	☐	☐	☐	☐
bei R+T und unterschiedlichen Baellen	☐	☐	☐	☐	☐

12. War es einfach, den Ball zu zielen?

	sehr schwer	anspruchsvoll	ok	einfach	sehr einfach
bei Rotation	☐	☐	☐	☐	☐
bei Rotation und Translation	☐	☐	☐	☐	☐
bei R+T und unterschiedlichen Baellen	☐	☐	☐	☐	☐

Nun folgt ein Test ob und wie sie mit Rotationssteuerung einen bestimmten Schlag ausfuehren koennen:

13. Versuchen Sie die folgenden Baelle zunaechst Long Line und dann Cross zu schlagen. Auf welche Art und Weise konnten sie die Flugrichtung des Balls beeinflussen?

	Rotation: wahr	Rotation: falsch	Rotation und Translation: wahr	Rotation und Translation: falsch
Ich stelle mich zum Ball und versuche ihn moeglichst spaet/frueh zu treffen	☐	☐	☐	☐
Ich kann die Richtung beeinflussen, indem ich den Schlaeger in einem bestimmten Winkel halte und ihn in eine bestimmte Richtung bewege	☐	☐	☐	☐

14. Nun derselbe Test mit Rotation und Translation: Auf welche Art und Weise konnten sie die Flugrichtung des Balls beeinflussen?

	wahr	falsch
Ich stelle mich zum Ball und versuche ihn moeglichst spaet/frueh zu treffen	☐	☐
Ich kann die Richtung beeinflussen, indem ich den Schlaeger in einem bestimmten Winkel halte und ihn in eine bestimmte Richtung bewege	☐	☐

15. Konnten Sie gut bestimmen, ob der Ball Long Line oder Cross geschlagen wird?

	gar nicht	schlecht	ok	gut	sehr gut
bei Rotation	☐	☐	☐	☐	☐
bei Rotation und Translation	☐	☐	☐	☐	☐

16. Konnten Sie gut bestimmen, ob der Ball lang oder kurz geschlagen wird?

	gar nicht	schlecht	ok	gut	sehr gut
bei Rotation	☐	☐	☐	☐	☐
bei Rotation und Translation	☐	☐	☐	☐	☐

Baseball

17. War es einfach, den Ball zu treffen?

	sehr schwer	anspruchsvoll	ok	einfach	sehr einfach
bei Rotation	☐	☐	☐	☐	☐
bei Rotation und Translation	☐	☐	☐	☐	☐

18. War es einfach, den Ball zu zielen?

	sehr schwer	anspruchsvoll	ok	einfach	sehr einfach
bei Rotation	☐	☐	☐	☐	☐
bei Rotation und Translation	☐	☐	☐	☐	☐

19. War es einfach, den Ball in einen Korb zu treffen?

	sehr schwer	anspruchsvoll	ok	einfach	sehr einfach
bei Rotation	☐	☐	☐	☐	☐
bei Rotation und Translation	☐	☐	☐	☐	☐

Billiard

20. Konnten Sie die Kugel gezielt treffen?

	ueberhaupt nicht		trifft zu		sehr gut
bei allen Freiheitsgraden	☐	☐	☐	☐	☐
bei der wenigen Freiheitsgraden	☐	☐	☐	☐	☐

21. Konnten Sie die Kugel gut zielen?

	ueberhaupt nicht		trifft zu		sehr gut
bei allen Freiheitsgraden	☐	☐	☐	☐	☐
bei der wenigen Freiheitsgraden	☐	☐	☐	☐	☐

Zum Schluss

22. Wie würden Sie den Spielspaß bewerten bei?

	keiner	gering	ok	hoch	sehr hoch
Tischtennis mit R+T	☐	☐	☐	☐	☐
Tischtennis mit Rotation	☐	☐	☐	☐	☐
Billiard	☐	☐	☐	☐	☐
Baseball	☐	☐	☐	☐	☐

23. Wie würden Sie die Langzeitmotivation bewerten bei?

	keiner	gering	ok	hoch	sehr hoch
Tischtennis mit R+T	☐	☐	☐	☐	☐
Tischtennis mit Rotation	☐	☐	☐	☐	☐
Billiard	☐	☐	☐	☐	☐
Baseball	☐	☐	☐	☐	☐

24. Welche Applikation hat sie...

	Tischtennis	Billiard	Baseball
am meisten ueberzeugt?	☐	☐	☐
am wenigsten ueberzeugt?	☐	☐	☐

25. Welche Applikation wuerden Sie wohl am laengsten spielen?

Tischtennis	Billiard	Baseball
☐	☐	☐

Literatur

[1] BÖTTCHER, MARCUS: *Structure from motion mit einem starrgekoppelten Kamerasystem*. , Christian-Albrechts-Universität zu Kiel, Institut für Informatik, 2007. http://www.mip.informatik.uni-kiel.de/tiki-download_file.php?fileId=846.

[2] DHEIN, ANDREAS: *Multimodale Präsentations- und Interaktionstechniken mit dem Nintendo Wii Controller*. , Universitaet Koblenz, Fachbereich 4, September 2009.

[3] FRITZ, JUERGEN: *Wie virtuelle Welten wirken - Ueber die Struktur von Transfers aus der medialen in die reale Welt*. FEHR, JUERGEN FRITZ / WOLFGANG (): *Computerspiele - virtuelle Spiel- und Lernwelten*, Bonn, 2003. Bundeszentrale für politische Bildung.

[4] INGRASSIA, MICHAEL: *Maya for games : modeling and texturing techniques with Maya and Mudbox*. Focal Press Elsevier Inc., 1 , 2009.

[5] IWATA, SATORU: *Iwata Asks: Wii MotionPlus*. Webseite. http://us.wii.com/iwata_asks/wiimotionplus/vol1_page1.jsp.

[6] JONATHAN LAM, CHRISTOPHER COLLINS, BILL KAPRALOS ANDREW HOGUE MIGUEL A. GARCIA-RUIZ: *Wiimote-Controlled Stereoscopic MRI Visualization With Sonic Augmentation*. . http://portal.acm.org/citation.cfm?id=1920778.1920825&coll=DL&dl=GUIDE&CFID=6829610&CFTOKEN=50234177.

[7] KREJPOWICZ MARCUS, NGUYEN SONG THU: *Implementierung einer Bibliothek zur einfachen Nutzung eines 3D-Tracking-Systems unter Verwendung von zwei Wiimotes*. Webseite, 2009. http://www.assembla.com/wiki/show/kp08_wiimote.

[8] KREJPOWICZ MARCUS, NGUYEN SONG THUY: *Implementierung einer Bibliothek zur einfachen Nutzung eines 3D-Tracking-Systems unter Verwendung von zwei Wiimotes*. . http://141.76.66.100/mg//_downloads/_files/KPIK_3D_Tracking_Paper.pdf.

[9] KREYLOS, OLIVER: *Wiimote Hacking - Technology*. Webseite. http://idav.ucdavis.edu/~okreylos/ResDev/Wiimote/Technology.html.

[10] LEE, JOHNNY CHUNG: *Wiimote Projects*. Webseite. http://johnnylee.net/projects/wii/.

[11] LEE, JOHNNY CHUNG: *WiimoteProject.com*. Webseite. http://www.wiimoteproject.com/index.php.

[12] MCRAE, GRAEME: *Equation of Lines in 3D - distance between two skew lines, and closest points*. Webseite. http://2000clicks.com/MathHelp/GeometryPointsAndLines3D.aspx.

[13] MUELLER, STEFAN: *Virtuelle Realitaet und Augmented Reality - Vorlesungsskript - Einfuehrung*. Universitaet Koblenz Landau, Institut für Informatik, 2007. http://www.mip.informatik.uni-kiel.de/tiki-download_file.php?fileId=846.

[14] STEFAN, AMELING: *Fragenkatalog Rechnersehen*. http://userpages.uni-koblenz.de/~fsinfko/drupal-6.10/documents/FragenkatalogSS2005Rechnersehen.pdf, 2004. Universitaet Koblenz.

[15] STEFAN, AMELING: *Struktur aus Bewegung (Skript Zusammenfassung)*, 2007.

[16] UNBEKANNT: *Bild: JC Lee's Headtracking*. http://divillysausages.com/files/links/sept_08/johnny_chung_lee.jpg.

[17] UNBEKANNT: *Coordinates of closest points on a pair of skew lines : Graphics*. Webseite. http://objectmix.com/graphics/133793-coordinates-closest-points-pair-skew-lines.html.

[18] UNBEKANNT: *InvenSense - MEMS Gyroscopes for Gaming*. Webseite. http://invensense.com/mems/gaming.html, Abrufdatum: 18.3.2010.

[19] UNBEKANNT: *Managed Library for Nintendo's Wiimote - WiimoteLib v1.7*. http://wiimotelib.codeplex.com/releases/view/21997.

[20] UNBEKANNT: *Ogre 3D.org*. http://www.ogre3d.org.

[21] UNBEKANNT: *Ogre Add-ons Forum - Ogre Soft Bodies*. Webseite. http://www.ogre3d.org/addonforums/viewtopic.php?f=12t=7476.

[22] UNBEKANNT: *Wii - Wikipedia*. Webseite, http://de.wikipedia.org/wiki/Wii.

[23] UNBEKANNT: *Wii Sports Baseball Picture*. http://tobi.darkmonasteria.de/fitohnediaet/wii/images/3_Wii_Sports_Baseball2.jpg.

[24] UNBEKANNT: *Wii Sports Tennis Picture*. http://static.rp-online.de/layout/showbilder/45563-screenshot_wii_sports_tennis_03.bmp.

[25] UNBEKANNT: *Wikipedia - Bullet (Software)*. http://en.wikipedia.org/wiki/Bullet_(software).

[26] UNBEKANNT: *Wikipedia - Pose (Technik)*. Webseite. http://de.wikipedia.org/wiki/Pose_(Technik), Abrufdatum: 2.12.2010.

[27] UNBEKANNT: *Wikipedia - Retroreflexion*. Webseite. http://de.wikipedia.org/wiki/Retroreflexion, Abrufdatum: 2.3.2011.

[28] UNBEKANNT: *Wikipedia - Roll-Nick-Gier-Winkel*. Webseite. http://de.wikipedia.org/wiki/Roll-Nick-Gier-Winkel, Abrufdatum: 2.12.2010, Version: Juli 2010.

[29] UNBEKANNT: *Wikipedia - Wii Remote*. Webseite. http://en.wikipedia.org/wiki/Wii_Remote, Abrufdatum: 10.11.2010.

[30] ZHENGYOU, ZHANG: *A Flexible New Technique for Camera Calibration*. http://research.microsoft.com/en-us/um/people/zhang/calib/, 1998. Webseite.

[31] ZHENGYOU, ZHANG: *A Flexible New Technique for Camera Calibration - Technical Report*. USA, Dezember 2009. http://research.microsoft.com/en-us/um/people/zhang/Papers/TR98-71.pdf, Englisch.

Abbildungsverzeichnis