Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Olaf Potratz

Ein System zur physikbasierten Interpretation von Gesten im 3D-Raum

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Olaf Potratz

Ein System zur physikbasierten Interpretation von Gesten im 3D-Raum

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Angewandte Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. rer. nat. Kai von Luck
Zweitgutachter: Prof. Dr. rer. nat. Gunter Klemke

Abgegeben am 21.04.2011

Olaf Potratz

Thema der Bachelorarbeit
Ein System zur physikbasierten Interpretation von Gesten im 3D-Raum

Stichworte
Mensch-Maschine-Interaktion, Motion Tracking, Gesten, Gestenerkennung, Physik-Engine

Kurzzusammenfassung
In dieser Arbeit wird ein Softwaresystem zur Interpretation von Gesten im Raum unter Verwendung einer Physiksimulation erstellt. Die Anforderungen für ein solches System werden beschrieben und analysiert. Die Verwirklichung dieser Software erfolgt unter Verwendung eines optischen Motiontrackers und unter der Zuhilfenahme einer Physik-Engine. Darauf aufbauende Verfahren zur Gestenerkennung werden theoretisch beschrieben und teilweise umgesetzt. Erste Versuche mit der Umsetzung werden dokumentiert und bewertet.

Olaf Potratz

Title of the paper
A system for physics-based interpretation of gestures in three dimensions

Keywords
Human-Computer-Interface, Motion Tracking, Gestures, Gesture Recognition, Physics-Engine

Abstract
In this paper, a software system for the interpretation of gestures in a room using a physic simulation is written. The requirements for such a system are described and analysed. The realisation of this software is using an optical motion tracker and the help of a physic engine. Methods for gesture recognition based on this are theoretically described and partially implemented. Initial trials with the implementation are documented and evaluated.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Einen großer Teil menschlicher Kommunikation und Interaktion im alltäglichen Umgang miteinander stellen Gesten dar. Gesten können verbale Kommunikation unterstützen, das bedeutet ihr Nachdruck verleihen. Sie kommen bewusst zum Einsatz, werden teilweise jedoch auch unbewusst von der Person getätigt.
Gesten können aber auch ganz anstelle verbaler Kommunikation treten, diese also vollkommen ersetzen. Diese können Bewegungen mit Armen und Händen umfassen, gleichermaßen kann ein einfaches Handzeichen ebenfalls als Geste verstanden werden.
Die richtige, oder besser gesagt, die im jeweiligen Augenblick korrekte Bedeutung einer Geste zu erkennen, ist schwierig. Eine Geste kann viele unterschiedliche Aussagen haben.
Für eine Gestenerkennung durch ein Computersystem ist die Komplexität von solchen Gesten sehr hoch. Man muss hier starke Abstriche machen und dem Benutzer gewisse Regeln auferlegen, in dessen Rahmen eine Gestenerkennung stattfinden soll. Solche Regeln können mithilfe einer simulierten Umgebung erzielt werden. Die dort vorhandenen Objekte allein dienen nun der eigentlichen Mensch-Maschine-Interaktion. Der Anwender bräuchte im Idealfall keinerlei Controller für die Interaktion, das System würde mit der Simulation selbst das Eingabegerät stellen. Dies wäre ganz im Sinne der „seamless interaction”, die besagt, dass sich die Maschinen den Menschen anpassen und nicht wie bislang die Menschen den Maschinen.
Davon ausgehend, dass unterschiedliche Handlungsweisen dasselbe Ziel verfolgen und dieselben Auswirkungen haben, wäre eine Gestenerkennung auf Basis der Handlungsergebnisse und nicht der vollführten Geste selbst denkbar. Dies ließe sich auch mit der zuvor angesprochenen Simulationsumgebung verwirklichen.

> „Wir müssen unser Handeln danach beurteilen, was wir damit erreichen.”
>
> (Dalai Lama)

Der Ansatz einer Gestenerkennung in dieser Arbeit interpretiert, was durch das Wirken des Benutzers in der Simulation verändert wurde, nicht, wie diese Änderung erzielt wurde.

1.1 Motivation

Momentan werden für die Interaktion zwischen Mensch und Maschine häufig Hilfsmittel wie Zeigergeräte, wie in [Fischer (2007)] beschrieben, verwendet. Da die Kommunikation zwischen Mensch und Maschine allerdings möglichst einfach und intuitiv gehalten werden sollte, ist es wünschenswert, generell auf Controller zu verzichten. Obgleich oftmals hardwaretechnische Eingabegeräte (u.a. WII-Controller) notwendig sind, um bestimmte Problemstellungen, die bei Gestenerkennung auftreten können, abzumildern oder ganz zu beheben. Ein solches Problem ist unter anderem das Start-Ende Problem. Der Ansatz, der in dieser Arbeit verfolgt wird, versucht das oben genannte Problem zu umgehen, indem es nicht die Handlung selbst zur Gestenerkennung heranzieht, sondern deren Auswirkungen betrachtet und so auf die Geste zurückschließen will.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist die Entwicklung und Umsetzung eines Systems zur Erkennung von Gesten im Raum mittels einer physikgestützten Simulation. Alle empfangenen Messdaten werden in einer Physiksimualtion verarbeitet und Resultate zur Gestenerkennung genutzt.

In dem theoretischen Teil der Arbeit wird anhand eines ausgewählten Szenarios eine Problemstellung für Gestenerkennung im Raum aufgestellt. Gesten sollen dann nicht wie üblicherweise mittels ihrer Bewegungsabläufe erkannt werden, sondern vielmehr soll der Benutzer mit einer 3D-Simulation interagieren. Die Positionen seiner Hände werden mittels eines Motiontrackers in die Simulation selbst übertragen. Aufgrund der Auswirkungen, die der Benutzer während der Interaktion mit den in der Simulation bereitstehenden Gegenständen verursacht, können Rückschlüsse auf die ursprünglichen Gesten gezogen werden. Dies geschieht durch die Beobachtung der Gegenstände im Verhältnis zueinander und ihrer internen Zustände. Das oben genannte Verfahren wird analysiert und die notwendigen Eigenschaften, die für eine softwaretechnische Umsetzung relevant sind, werden ermittelt.

Weiterhin wird im praktischen Teil dieser Arbeit ein Softwaresystem unter Berücksichtigung der in der Analyse ermittelten Anforderungen entwickelt. Anhand einiger ausgewählter Gesten soll die grundsätzliche Machbarkeit eines solchen Vorgehens zur Gestenerkennung untersucht und, falls die Möglichkeit besteht, exemplarisch gezeigt werden. Das zu entwi-

ckelnde System wird in mehrere Softwaremodule unterteilt. Die einzelnen Module umfassen die im Folgenden beschriebenen Bereiche. Zunächst werden die Punkte des Motiontrackers aufbereitet. Außerdem ist die Gestenerkennung als solches beinhaltet. Dabei wird eine Physik-Engine zu Hilfe genommen. Am Ende steht eine Grafikausgabe, die dem Benutzer eine abstrakte, also keineswegs fotorealisitsche Anzeige seiner Interaktionen mit den Gegenständen liefert.

1.3 Gliederung der Arbeit

Das erste Kapitel umfasst die Einleitung, in der Motivation und Zielsetzung näher erläutert sind.
Im Anschluss darauf folgt eine Erläuterung der Grundlagen in Kapitel 2. Es werden verschiedene Arten der 2D- und 3D-Gestenerkennung vorgestellt. Weiterhin werden unterschiedliche Techniken für Motion-Capturing (2.2) aufgeführt. Darüber hinaus wird die Arbeitsumgebung beschrieben, in der später die praktische Umsetzung stattfinden soll. In dem zweiten Kapitel werden ebenfalls einige grundlegende Techniken erklärt, welche im weiteren Verlauf der Arbeit Verwendung finden.
Das Kapitel 3 befasst sich mit der Analyse für ein solches System zur Gestenerkennung. Zuerst wird auf der Grundlage von vorhandenen Beispielen ein Anwendungsszenario beschrieben, mit dessen Umsetzung sich diese Arbeit befassen soll. Aus dem Szenario werden einige Gesten exemplarisch ermittelt, welche anschließend spezifischer betrachtet werden. Auf Grundlage der Analyse der Gesten und dem grundsätzlich hier gewählten Ansatz diese zu erkennen, werden die funktionalen Anforderungen für die spätere Umsetzung formuliert.
Das folgende Kapitel wird in zwei Abschnitte geteilt. Kapitel 4 befasst sich im ersten Teil genauer mit dem gewählten Ansatz zur Gestenerkennung; es werden theoretische Überlegungen zur Aufbereitung der Bewegungsdaten und zur Gestenerkennung mittels einer Physiksimulation gemacht. Der zweite Teil des Kapitels beinhaltet die Entwicklung eines Softwaresystems, das die Umsetzung dieser Gestenerkennung ermöglichen kann. Die zugrunde liegenden Architekturmuster werden erläutert, außerdem werden der Aufbau und die Kommunikation der einzelnen Komponenten untereinander beschrieben.
Kapitel 5 befasst sich mit der Realisierung eines Prototyps für die ermittelten Anforderungen. Verschiedene Frameworks von Physik-Engines, die bei der Umsetzung in Java in Frage kamen, werden vorgestellt und bewertet. Dabei werden einige Entwicklungsetappen aufgeführt. Bei einer kurzen Evaluation wird betrachtet, inwieweit die Software die in Kapitel 3 vorgestellten Gesten erkennt. Zum Schluss des Kapitels findet eine kurze Bewertung der technischen Umsetzung statt. Es wird ein Ausblick dahingehend gewährt, welche Erweiterungen mit fortgeschritteneren technischen Mitteln machbar sind.
Das Kapitel 6 ist der letzte Teil dieser Arbeit, welches eine Zusammenfassung enthält. Letztendlich erfolgt ein Ausblick.

Kapitel 2

Grundlagen

Um die Grundlagen für diese Arbeit zu legen, geht Abschnitt 2.1 auf die Gestenerkennung an sich ein. Es beschreibt einige Problemstellungen, die sich ergeben und geht kurz auf einige andere Arbeiten im 2D- und 3D-Bereich ein.
Abschnitt 2.2 stellt verschiedene technische Umsetzungen für dreidimensionales Motiontracking vor. Besonders viel Wert wird hier auf den im Labor aufgebauten Motiontracker gelegt. Der Aufbau sowie das Verfahren des Motiontrackers werden kurz umrissen. Es werden die verschiedenen Möglichkeiten erläutert, mit welchen Mitteln der Motiontracker eingesetzt werden kann.
Im darauffolgenden Abschnitt 2.3 werden 3D- und Physik-Engine vorgestellt. Ihre jeweilige Arbeitsweisen wird kurz umrissen, und die für ihre Berechnung verwendeten Daten werden aufgezeigt. Am Ende dieses Abschnitts wird noch die Trennung der einzelnen Bestandteile der Simulationsdaten zwischen der Grafik und der Physik erläutert.

2.1 Gestenerkennung

Um Gesten erkennen zu können, bedarf es einer Unterscheidung, um welche Art von Geste es sich jeweils handelt. In der Arbeit von Joachim Boetzer [Heitsch (2008)] wird auf eine Klassifizierung der Gesten nach Vladimir Pavlovic eingegangen. Es erfolgt dort eine Aufteilung der Gesten in bewusste und unbewusste Gesten. Die bewussten Gesten wurden dann in kommunikative und manipulative Gesten unterteilt. Im weiteren Verlauf dieser Arbeit wird sich mit Gesten beschäftigt, die in diesem Sinne manipulativ sind.
Die richtige, oder besser gesagt, die im jeweiligen Augenblick korrekte Bedeutung einer Geste zu erkennen, ist schwierig. Eine Geste kann viele unterschiedliche Aussagen haben. Welche genaue Aussage der Initiator verfolgt, lässt sich oft nur im Zusammenhang mit vielen Faktoren erschließen. Einige Faktoren sind unter anderem der Gesteninitiator selbst, die Situation, in der die Geste gemacht wird und der Empfänger der Geste. Darüber hinaus spielen der Zustand der Person, seine Haltung sowie seine Mimik eine wichtige Rolle bei der korrek-

ten Interpretation der Geste.
Bei der Gestenerkennung spielt das sogenannte „Start-Ende-Problem" eine große Rolle. Wie in [Hara (2011)] beschrieben, ergibt sich bei einer kontinuierlichen Aufzeichnung der Bewegung das Problem, dass das System ermitteln muss, ab welchem Zeitpunkt der Aktor bewusst eine Geste beginnt und wann er diese beendet.
Die im Folgenden aufgeführten Verfahren zur Gestenerkennung werden nur kurz angesprochen. Da nach Analyse dieser Verfahren keine Verwendung zum Lösen der in dieser Arbeit gestellten Aufgabe gesehen wurde, wird auf eine tiefergehende Beschreibung verzichtet und auf die entsprechenden Arbeiten verwiesen.

2.1.1 Angewandte Gestenerkennung in 2D

Gestenerkennung in einer zweidimensionalen Umgebung bedeutet, dass die zu interpretierende Geste sich auf zwei Dimensionen beschränkt. Das einfachste Eingabegerät, welches für solche Gesten im Umfeld eines Computers infrage kommt, ist die Maus. Darüber hinaus kommen heutzutage immer mehr und wesentlich komfortabelere Eingabemöglichkeiten hinzu.

Unter anderem im dem Computerspiel Black & White[1] aus dem Jahre 2001 gab es eine einfache 2D-Gestenerkennung. Als Eingabegerät für die Gesten wurde die Maus benutzt. Zu Beginn einer Geste musste die linke Maustaste gedrückt werden, dann musste die Taste während der gesamten Geste gedrückt bleiben. Mit der Bewegung der Maus konnten nun vorher erklärte Formen wie ein Pentagramm, ein Wirbel oder ein Herz gezeichnet werden. Die Geste musste nicht aktiv beendet werden, sobald das System eine Geste erkannt hat, wurde dies dem Benutzer grafisch wie auch akustisch mitgeteilt. Eine Liste der möglichen Gesten und weiterführende Informationen wurden in der Ausarbeitung von Torsten Hein [Hein (2005)] aufgeführt .

Ein weiters beliebtes Eingabegerät sind Touchscreens. Diese sind schon lange keine Seltenheit mehr und finden vorallem in Mobilgeräten häufig Verwendung.
Wie in [Mohammadali Rahimi (2008)] beschrieben, kann als Eingabegerät für 2D-Gesten auch ein Touchscreen dienen. Das Start-Ende-Problem kann hier im Zusammenspiel mit dem Berühren des Touchscreens, dem Start der Geste, und dem Loslassen des Touchscreens, als Ende der Geste, behandelt werden. Ansatz zur Erkennung der Gesten liefern hier die einzelnen Positionen der Berührung über die Zeit. Als mögliche Verfahren zur Gestenerkennung werden in [Mohammadali Rahimi (2008)] das „Hidden Markov Modell", die „Support Vector Machines" und die „Space-Time-Linearisierung" genannt.

[1] Black & White - Lionhead Studios, veröffentlich von Electronic Arts 2001

2.1.2 Angewandte Gestenerkennung in 3D

Diese Form der Gestenerkennung findet räumlich statt, die Geste selbst muss aber nicht zwingend alle drei Dimensionen ausnutzen. Besondere Herausforderungen ergeben sich hier aus dem Start-Ende-Problem, da ein Tracking der Bewegungen kontinuierlich erfolgt. Johann Heitsch stellt in seiner Arbeit [Heitsch (2008)] ein Framework zur Erkennung dreidimensionaler Gesten vor. Das dort entwickelte System verwendet eine Support Vektor Machine zum Ermitteln der Gesten. Es handelt sich bei der Support Vektor Machine um ein Verfahren zur Mustererkennung, Genaueres kann der Arbeit [Heitsch (2008)] entnommen werden.

2.2 Optische Trackingverfahren

Optische Trackingverfahren bieten die Möglichkeit eine berührungsfreie Verbindung zwischen Mensch und Maschine zur Verfügung zu stellen.
Die meiste Aufmerksamkeit beim Vorstellen verschiedener Verfahren wird hier auf das Verfahren des ARTtrackers gelegt, da dieser in der späteren Umsetzung der Arbeit zum Einsatz kommt.

2.2.1 ARTtracker

Bei dem System der Firma A.R.T.-Advanced Realtime Tracking GmbH handelt es sich um ein optisches Trackingverfahren. Das System besteht aus ARTtrack-Kameras und einer Software names D-Track. In der Arbeit wird das gesamte System umgangsprachlich als ARTtracker bezeichnet.

2.2.1.1 Funktionsweise

Wie in [Boetzer (2008)] beschrieben, stehen für das System sogenannte aktive und passive Marker zur Verfügung. Aktive Marker senden selbstständig einen infraroten Lichtimpuls zu den Kameras, wohingegen die passiven Marker lediglich die von der Kamera ausgesendeten infraroten Lichtimpulse reflektieren. Die Kameras zeichnen die von den Markern ausgehenden aktiven bzw. passiven Lichtpunkte auf. Das System besteht hier aus sechs Kameras. Der D-Track Software sind Abstand und Winkel der Kameras zueinander, nach entsprechender Konfiguration, bekannt [ARTtracker-Anleitung (2007)].
Unter Verwendung der bis zu sechs einzelnen Aufnahmen eines Markers kann nun eine Position des Markers im Raum berechnet werden. Je mehr Kameras eine Aufnahme eines bestimmten Markers liefern können, desto zuverlässiger ist die Position, die von der D-Track Software ermittelt wird.

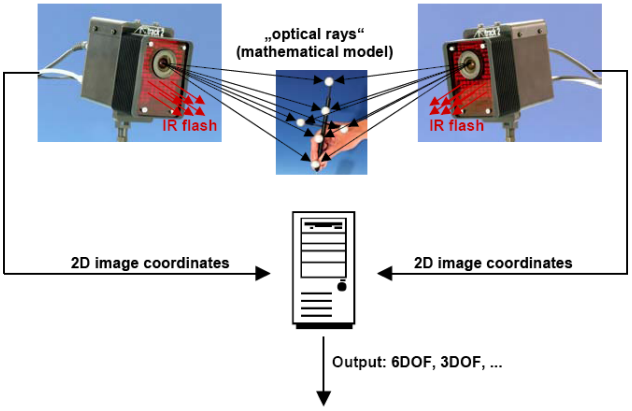

Abbildung 2.1: Funktionsweise Tracker (Quelle: [ARTtracker-Anleitung (2007)])

2.2.1.2 Format der Positionsdaten

Der verwendete Motiontracker liefert, wie auch in [ARTtracker-Anleitung (2007)] erklärt, zwei verschiedene Datentypen. Diese sind abhängig davon, ob einzelne Punkte gefunden wurden oder es sich aber bei den getrackten Strukturen um komplexere starre Objekte handelt. Die beiden Datentypen sind im Folgenden beschrieben, das genaue technische Format dieser Daten kann aus dem technischen Anhang der D-Track Software [DTrack-Anhang (2007)] entnommen werden.

3DOF Die Abkürzung steht für „3 degrees of freedom”, ins Deutsche übersetzt: „3 Freiheitsgrade”. Dies bezieht sich hier auf die drei Raumachsen X, Y und Z. Die Position eines Markers wird durch diese drei Achsen beschrieben.

6DOF Diese Abkürzung steht für „6 degrees of freedom” also 6 Freiheitsgrade. In diesem Format wird zusätzlich zu dem Ort, den 3 Raumachsen X, Y und Z noch die Orientierung, also die jeweilige Rotation um eine der Raumachsen mitgeliefert.

2.2.1.3 Freie Punkte

Freie Punkte sind die simpelste Form der Punktübertragung. Das Motiontrackersystem errechnet lediglich die Position der Punkte aus den Bildern seiner einzelnen Kameras. Jeder übertragene Punkt entspricht einem gefundenen Marker. Es ist kein vorheriges Einmessen nötig. Bei freien Punkten wird jeder einzelne Punkt separat betrachtet. Es ist daher nur eine Aussage über die Position des Punktes im Raum möglich, über die Lage im Raum kann bei einem Punkt allein keine Aussage getroffen werden. Die Übertragung dieser Daten geschieht im Format 3DOF. Abbildung 4.1 zeigt eine Gruppe von freien Markern, da diese keine feste Position zueinander einnehmen.

2.2.1.4 Starre geometrische Körper

Diese Gebilde besitzen mehrere Marker. Wie in Abbildung 2.2 zu sehen ist, sind diese Marker in einer festen Position zueinander angeordnet. Wie man besonders gut bei dem rechten der beiden Fotos in Abbildung 2.2 erkennen kann, sind diese Körper asymetrisch aufgebaut. Vor der Verwendung muss der Körper vom System eingemessen werden [ARTtracker-Anleitung (2007)]. Ist der Körper erfolgreich vermessen worden, werden vom System nicht mehr die einzelnen Positionen der Marker als 3DOF zurückgeliefert, sondern das System errechnet anhand der einzelnen Positionen der Maker im Raum Ort und Lage des Körpers. Dem Benutzer wird nun der errechnete Mittelpunkt des Körpers zurückgegeben, nicht die einzelnen Markerpositionen. Diese Daten werden als 6DOF übermittelt.

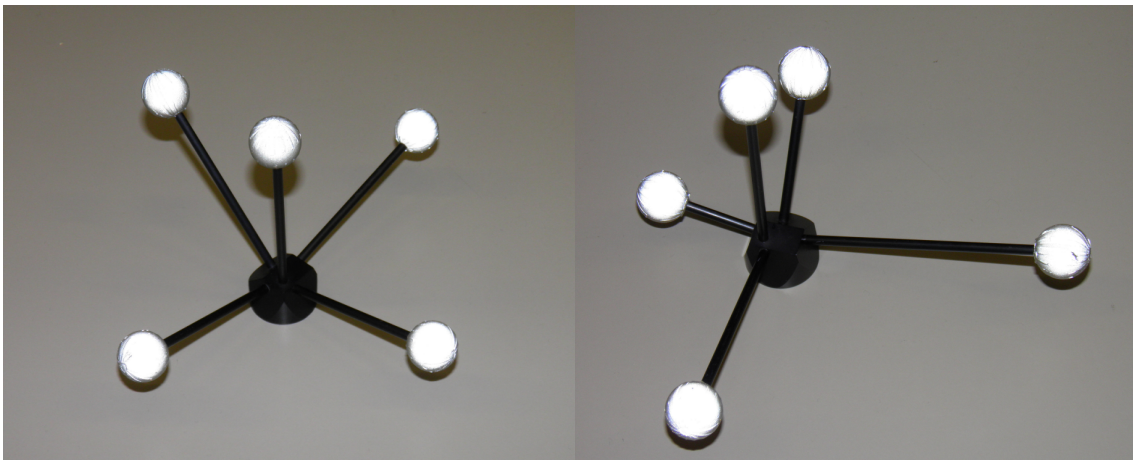

Abbildung 2.2: Starre geometrische Körper

2.2.1.5 Starrer geometrischer Körper mit freien Punkten

Der in Abbildung 2.3 gezeigte Marker-Körper von der Firma A.R.T. GmbH ist eine Erweiterung zu den rein statisch geometrischen Körpern. Es ist ein hybrides Modell. Hier findet eine Positions- und Lagebestimmung weiterhin nur über den starren Teil des Controllers statt. Dieser wurde eingemessen und durch die bestimmten Abstände der einzelnen Punkte zueinander ist nun auch die Position und Lage des errechneten Mittelpunkts möglich. Wie in Abbildung 2.3 zu erkennen ist, befindet sich der starre Teil auf dem Handrücken. Diese Software kennt den Aufbau des Controllers und es gibt daher eine gewisse Erwartungshaltung, wo sich die freien Marker, also die Fingerpunkte befinden müssen, wenn sie den starren Teil auf der Handoberseite gefunden hat. Dieses Verfahren wird durch ein Zusatzmodell der eigentlichen Betreibssoftware des ARTtrackers abgearbeitet. Die Position dieses Objekts wird in einem gesonderten Format zurückgegeben, welches im Prinzip eine Mischung aus 3DOF- und 6DOF-Daten enthält.

Abbildung 2.3: A.R.T.- Finger tracking board (Original entnommen aus [Fingertracking (2006)])

2.2.2 Time of flight Kameras

TOF-Kameras sind 3D-Kamerasysteme, die mit einer Laufzeitmessung des Lichts Distanzen messen. TOF ist eine englische Abkürzung und steht für „time of flight".
Um Distanzen zu messen, wird, wie in [Breuer (2005)] beschrieben, die Szene mittels eines Lichtpulses ausgeleuchtet. Die Kamera misst für jeden Bildpunkt die Zeit, die das Licht zum Objekt und wieder zurück zur Kamera braucht. Die verstrichene Zeit ist direkt proportional zur Distanz zwischen Kamera und dem Objekt. Es ist somit möglich für jeden Bildpunkt die Entfernung des darauf abgebildeten Objektes zu ermitteln. Im Prinzip entsprich dieses Verfahren dem herkömmlichen Laserscanning, nur mit dem Vorteil, dass ein ganzes Bild auf einmal aufgenommen wird und das Objekt nicht sequenziell abgetastet wird.
Kombiniert man eine Farb- und eine TOF-Kamera, wie man auf Abbildung 2.4 erkennen kann, ist es möglich beide aufgenommen Bilder zu verbinden und gemeinsam zu nutzen.
Die in Abbildung 2.5 zu sehenden Bilder sind die einer TOF-Kamera Messung. Das linke Bild zeigt die Frontalansicht, so wie die Kamera die Daten aufgenommen hat. Das rechte Bild zeigt eine seitliche Ansicht, wie sie mithilfe der gewonnen Distanzdaten errechnet wurde.

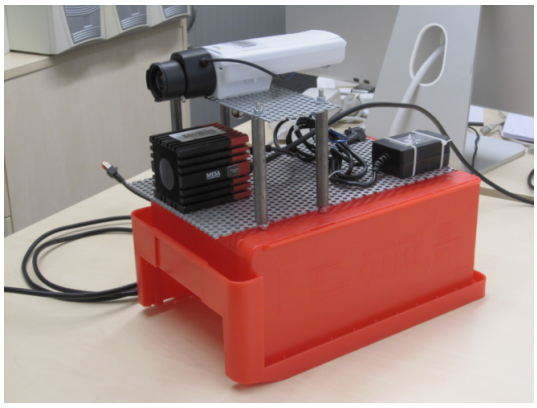

Abbildung 2.4: TOF-Kamera und Farbkamera (Originalfoto von Arne Bernin)

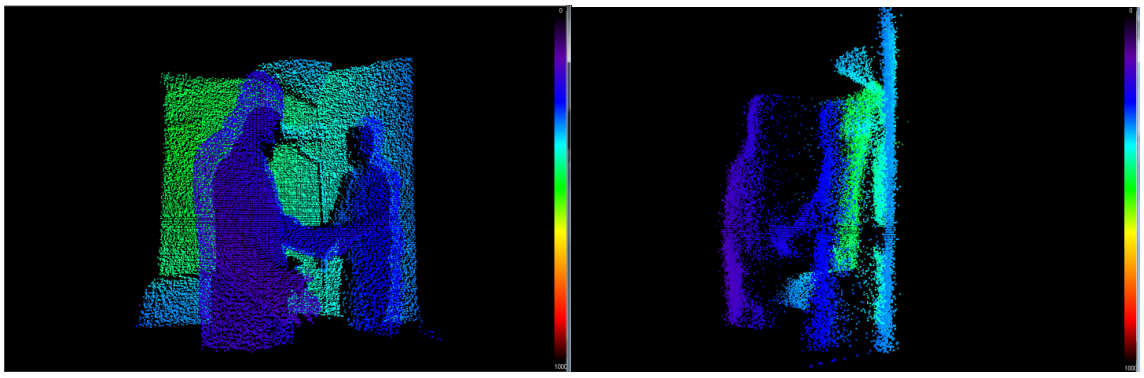

Abbildung 2.5: TOF-Distanzbilddaten Entfernungen von violett (kurze Distanz) bis rot (große Distanz) dargestellt (Die Originalaufnahmen stammen von Arne Bernin.)

2.2.3 Microsoft XBox360 mit Kinect

Der Entwicklungsname der Kinect lautet Project Natal. Sie ist ein Zubehör für die Spielekonsole XBox360 von Mircosoft [XBox360]. Mit diesem Zubehör soll eine Steuerung über Bewegungserkennung ermöglicht werden.
Zur Erkennung von Bewegungen ist die Kinect sowohl mit optischen 3D-Sensoren als auch akustischen 3D-Sensoren ausgerüstet. Als optische Sensoren kommen eine stereokopische Tiefenkamera und eine Farbkamera zum Einsatz.
Außerdem wird eine Reihe von Mikrofonen für die akustische Sensorik verwendet [OpenNI-UserGuide].
Aufgrund der mittlerweile frei zugänglichen Referenztreiber[2] ist die Verwendung der Kinect

[2]OpenNI: www.openni.org
OpenKinect: www.openkinect.org

nun nicht mehr auf die XBox360 beschränkt. Man ist in der Lage die von der Kinect gelieferten Daten frei zu verwenden.

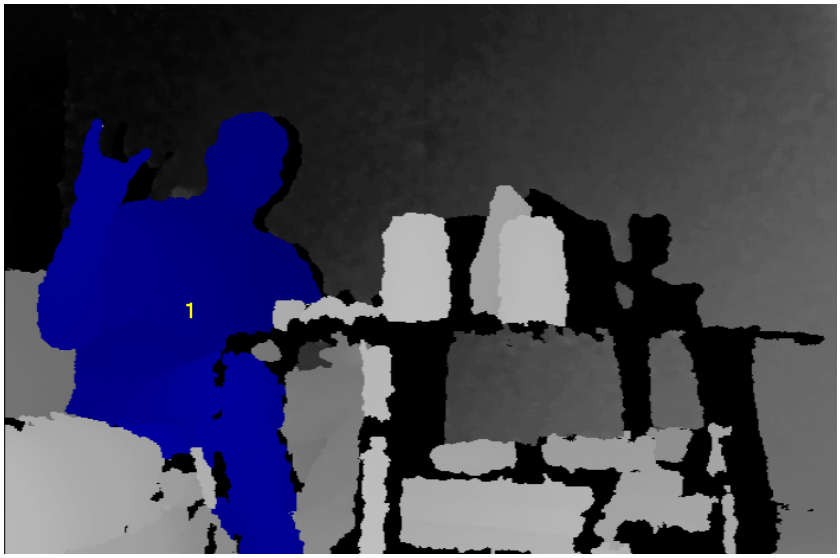

Abbildung 2.6: Kinectaufnahme mit OpenNI und einem Windows PC (Quelle: [Kinect])

2.3 3D- und Physik-Engine

Beide Enginearten sind sich in ihren Aufgabenbereichen vergleichbar. Die 3D-Engine soll dem Software-Entwickler die dreidimensionale Darstellung seiner Objekte abnehmen und die dafür anfallenden Berechnungen erledigen. Die Physik-Engine will dies für das physikalische Verhalten der in einer Simualtion befindlichen Objekte tun.
Die Datenhaltung in der 3D- wie auch der Physik-Engine ist sehr ähnlich. Sie ist eine Baumstruktur. Das heißt Daten, die auch als Entitäten bezeichnet werden können, werden dort eingehängt. Will man mehrere Entitäten gruppieren, hängt man diese zusammen unter denselben Knoten. Anschließend wird dieser Knoten dann in die Baumstruktur eingefügt.

2.3.1 3D-Engine

Eine 3D-Engine errechnet aus den geometrischen Werten und den visuellen Eigenschaften der Oberfläche dieser Objekte ein dreidimensionales Abbild. Die Berechnung findet auf der Grafikkarte unter Verwendung eines entsprechenden Treibers statt.
3D-Engines sind plattformabhänig. Damit plattformunabhängige Sprachen wie Java eine 3D-Engine nutzen können, bedarf es spezieller APIs, die einen Zugriff auf die eigentliche 3D-

Engine ermöglichen. Eine solche API ist JOGL[3]. Es bietet dem Javaentwickler den Zugriff auf die OpenGL[4] Bibliotheken.
Es werden für die Grafikberechnung Eigenschaften wie Farbe, Transparenz und Reflexion verwendet. Zur komplexeren Darstellung werden dabei auch Texturen auf die Oberflächen der darzustellenden Objekte gelegt.
Darüber hinaus nimmt die Simulationsumgebung noch Einfluss auf die grafische Präsentation. Dabei nehmen die allgemeine Helligkeit und unterschiedlichen Lichtquellen, die sich zusätzlich im Simulationsraum befinden, Einfluss auf das entstehende Bild.

2.3.2 Physik-Engine

Eine Physik-Engine ist meist ein eigenständiges Modul innerhalb einer Softwareanwendung, deren Berechnungen zum Teil in eigens dafür bereitgestellter Hardware stattfindet. Diese Hardware wird dann in Form von Steckkarten vertrieben oder wie in diesem Beispiel [NvidiaPhysX] direkt als Bestandteil der Grafikkarten mitgeliefert. Auch hier gilt für platformunabhänige Sprachen wie Java, dass sie auf spezielle APIs angewiesen sind, wenn sie eine systemnahe Physik-Engine nutzen wollen.
Physik-Engines werden zur Simulation physikalischer Prozesse und der Berechnung physikalischen Verhaltens von Objekten genutzt.
Wie in Abbildung 2.7 zu sehen ist, kollidiert das türkisfarbene Vehikel mit der Würfelwand, die dann zum Teil in sich zusammenstürzt. Die einzige Eingabe des Anwenders hier war die Fahrtrichtung und die Beschleunigung des Vehikels. Die Kollision mit der Wand und das weitere Verhalten werden ausschließlich von der dort verwendeten Physik-Engine berechnet.

Es gibt unterschiedliche Arten der Berechnung. Sie sind abhängig von den physikalischen Modellen, die der Berechnung zugrunde liegen. In dieser Arbeit kämen dazu das Physikmodell der starren Körper und das Modell der nicht elastischen Physik mit Deformationen der Körper infrage.

[3] www.jogamp.org/jogl
[4] www.opengl.org

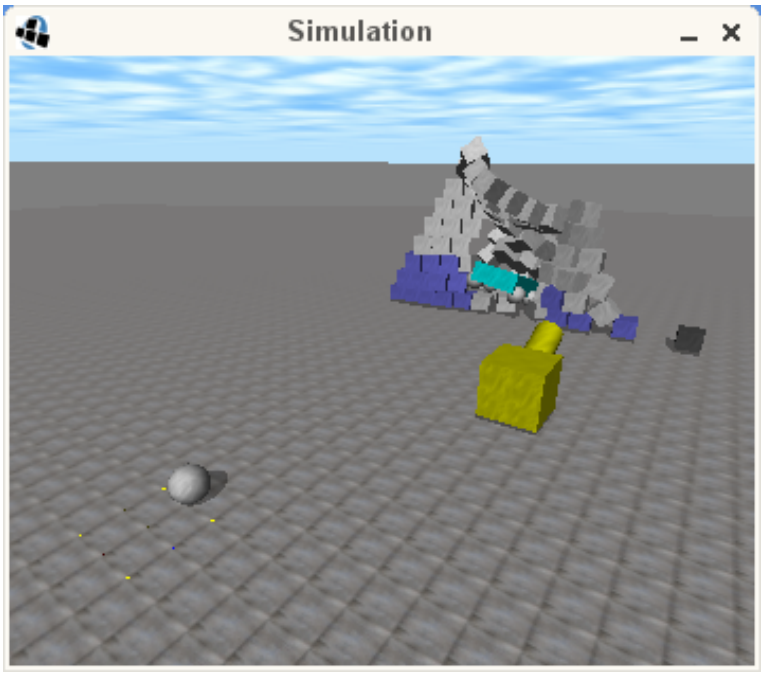

Abbildung 2.7: ODE Physik-Engine in Aktion (Quelle: [ODE4J])

Die im Verlauf der Arbeit vorgestellten Frameworks und deren verwendeten Physik-Engines arbeiten ausschließlich mit dem Modell der starren Körper. Die hier errechneten Kräfte dienen der Bewegung der Körper und deren Interaktion untereinander. Wie der Begriff „starre Körper" aber schon schließen lässt, wird ein Einwirken der Kräfte auf den Körper selbst und somit eine mögliche Deformation hier nicht betrachtet.
Die physikalischen Eigenschaften, wie Größe, Masse, Beschleunigung und Bewegungsrichtung werden von der Physik-Engine entgegengenommen und zur Berechnung des Verhaltens hinzugezogen. Einige Physik-Engines unterstützen darüber hinaus auch Materialeigenschaften. Diese beziehen sich auf die Oberfläche der Körper und dort vor allem auf die Reibungseigenschaften der Körper, die maßgeblich das Bewegungsverhalten bei Kontakt zu anderen Körpern in der Simulation beeinflussen. Es gibt zwei Arten von Reibung, nämlich die Haftreibung und die Gleitreibung. Unter dem Begriff Haftreibung versteht man die Kraft, die bei dem Versuch einen ruhenden Körper zu verschieben, entgegenwirkt.
Bei einem Körper, der in Bewegung ist, wirkt die Gleitreibung der Bewegung selbst entgegen. Entscheidend für die auftretenden Reibungskräfte sind einerseits die Reibungskoeffizienten der beiden Materialen, die aneinander reiben und andererseits die Anpresskraft, mit der diese beiden Materialen aufeinander gepresst werden.

2.3.3 Trennung von Grafik und Physik

Für ein Datenobjekt wird je eine Entität in der 3D- und in der Physik-Engine erzeugt. Einige objektbezogene Daten können dabei identisch sein, müssen es aber nicht.
Die von der Physik-Engine errechneten Änderungen der Entitäten, wie beispielsweise die Lage im Raum, werden nach der Berechnung von der verwendeten 3D-Engine grafisch dargestellt.
Die physikalische Entität und sein grafisches Gegenstück müssen nicht zwingend dieselben räumlichen Ausdehnungen besitzen. Wie in Abbildung 2.8 gezeigt, können Physik und Grafik zum Teil voneinander abweichen. Das physikalische Objekt wird hierbei möglichst einfach gehalten. Das für den Anwender später sichtbare Verhalten des simplen Physikobjekts unterscheidet sich kaum von dem eines Physikobjekts, das ähnlich detailliert aufgebaut wäre wie das Grafikobjekt. Ein häufiger Grund für diese Art von Abstraktion ist eine Ersparnis von Rechenzeit innerhalb der Physik-Engine.
Die Physik-Entität erhält die ID, die Abmessung und die spezifischen physikalischen Eigenschaften, wie Gewicht, Oberflächenreibung und Anzahl der Unterteilungen des Physikkörpers.
Die grafische Entität erhält ebenfalls die ID, die Abmessung und die Eigenschaften, die für die grafische Darstellung von Interesse sind. Interssant sind unter anderem die Farbe der Oberfläche, das Verhalten bei Lichteinfall und die Anzahl der Unterteilungen, die für die Grafik-Entität verwendet werden.

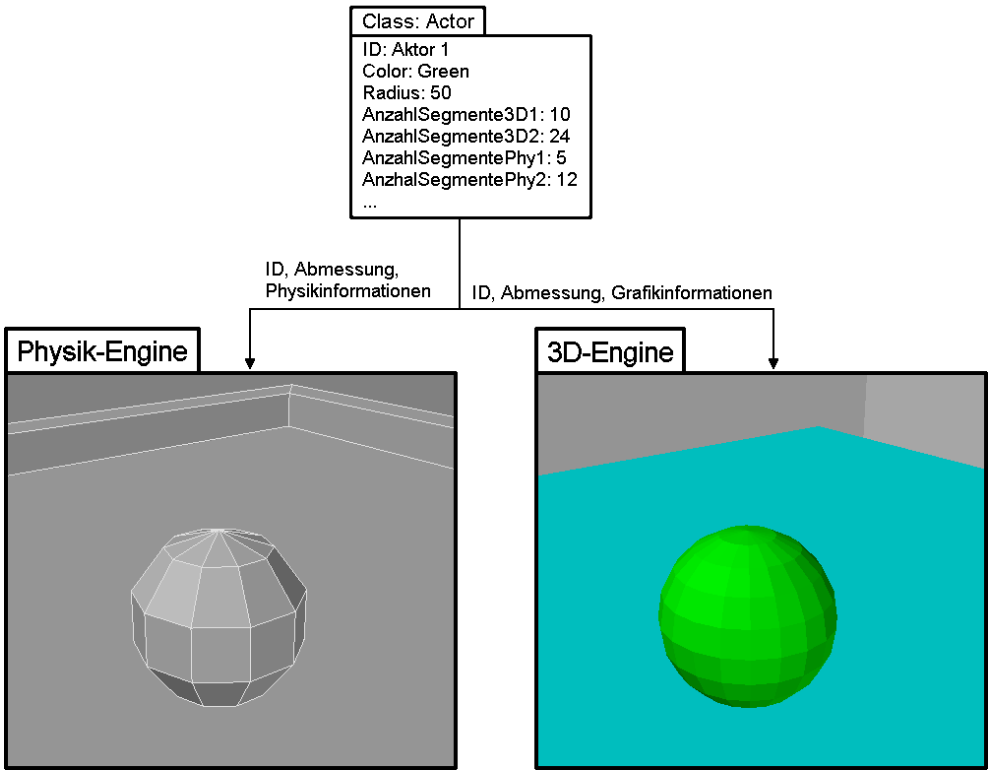

Abbildung 2.8: Aufteilung in Physik-Entität und Grafik-Entität

2.4 Zusammenfassung

Dieses Grundlagenkapitel befasste sich mit drei Hauptthemen. Der erste Teil 2.1 des Kapitels erläuterte das Thema Gestenerkennung. Es wurden verschiedene Verfahren der Gestenerkennung aufgeführt. Zudem wurde das „Start-Ende-Problem" vorgestellt, auf das im weiteren Verlauf der Arbeit noch einmal eingegangen wird.
Im Abschnitt 2.2 des Kapitels wurde auf optische Trackingverfahren eingegangen. Der ARTtracker als optisches Trackingsystem wurde besonders ausführlich beschrieben, da dieses System später auch zur Anwendung kommen soll. Darüber hinaus wurde auf andere Verfahren kurz eingegangen.
Im letzten Teil 2.3 dieses Kapitels wurden die grundlegenden Funktionen und Aufgaben einer 3D- und einer Physik-Engine erläutert. Auch wurde in 2.3.3 auf die Trennung zwischen Grafik und Physik eingegangen, indem erläutert wurde, welche Bestandteile der Daten für die grafische und welche für die physikalische Repräsentation in der Simulation nötig sind.

Kapitel 3

Analyse

Die in (3.1) aufgeführten, bereits umgesetzten Projekte sollen eine Grundlage für das in (3.2) entworfene Szenario bilden. Die hier vorgenommene Anforderungsanalyse nimmt auf das in Abschnitt 3.2 beschriebene Beispielszenario Bezug. Die Analyse soll später die Grundlage für ein zu entwickelndes Softwareprogramm bilden, welches die in (3.3) aufgeführten Gesten erkennen kann und auch den weiter in (3.4) festgehaltenen Anforderungen entspricht.

3.1 Reale Beispiele

Im folgenden Unterkapitel werden zwei bereits durchgeführte Projekte vorgestellt, anhand derer in Abschnitt 3.3 ein eigenes Szenario entwickelt werden soll.

3.1.1 Aarhus by Light

Aarhus by Light war ein Projekt, welches zwischen Februar und März 2008 an der Konzerthalle von Aarhus (Dänemark) stattfand.

Die bei Aarhus by Light stattfindende Interaktion mit dem System benötigt kein personalisiertes Eingabegerät. Personen, die sich auf den farbigen Bereichen des Bodens befinden, werden von Kameras erfasst. Der Teil vom farbigen Untergrund, der von den Aktoren verdeckt wird, wird farbig wiedergegeben. Die so aufgezeichneten Silhouetten der Personen wurden dann zum Teil spiegelverkehrt oder aber auch mehrfach (siehe Abbildung 3.1) auf der Außenfassade der Konzerthalle dargestellt.
Bei Aarhus by Light sind vor allem zwei Tatsachen für das spätere Szenario von Interesse. Erstens, dass eine, wenn auch zugegebenermaßen rudimentäre Interaktion mit dem Computersystem ohne weitere Hilfsmittel erfolgt. Das zweite Augenmerk wird auf die Dimension gelegt, in der der Aufbau stattfand. Wie man sehr gut in Abbildung 3.2 sehen kann, erstreckt

sich der Aufbau auf einen größeren Bereich, wobei nur innerhalb der farbigen Flächen eine Aufnahme der Personen durch das System erfolgte.

Abbildung 3.1: Eingangsbereich der Konzerthalle in Aarhus (Quelle: [Aarhus1])

Abbildung 3.2: Luftbildaufnahme der Konzerthalle in Aarhus (Quelle: [Aarhus2])

3.1.2 VR/URBAN - SMSlingshot

Bei SMSlingshot handelt es sich um einen Nachfolger vom Projekt „spread.gun" [spreadgun2010] das auf dem Medienfassade Festival 2008 [MediaFacadesFestival] in Berlin präsentiert wurde. SMSlingshot ist eine Erweiterung für VR/Urban's digitalen Interventions-Aktivismus.
SMSlingshot [Smslingshot] hat sich zur Aufgabe gesetzt, mithilfe einer digitalen Steinschleuder Informationen über öffentliche Bildschirme zu verbreiten. Aufgrund der immer zahlreicher vorhandenen digitalen Werbeflächen im städtischen Raum war der Wunsch aufgekommen, mit der dort verbreiteten Information zu interagieren. Mit dem Projekt soll es ermöglicht werden, digitale Information zu kommentieren, indem man diesen öffentlichen Bildschirm selbst als Präsentationsfläsche nutzt.

Abbildung 3.3: SMSlingshot in Aktion (Quelle: [VR/URBAN])

Die Skizze in Abbildung 3.4 zeigt den grundsätzlichen Aufbau von SMSlingshot (siehe [Smslingshot]). Der Aufbau selbst umfasst einen oder mehrere dieser Steinschleudern nachemfundenen Controller, eine Kamera und einen Projekter, mit dessen Hilfe eine Media-Fassade erschaffen wird.

Der Controller in Abbildung 3.5 hat zwei Funktionen:

- Zum Ersten kann mittels einer kleinen Telefontastatur eine kurze Zeichenkette eingegeben werden.
- Zum Zweiten verfügt der Controller über einen Laserpointer.

Der von dem Controller ausgehende Laserstrahl wird auf eine Wand gerichtet. Weiterhin wird der so entstehende Laserpunkt von der aufgestellten Kamera erfasst. Das System wird somit in die Lage versetzt die Position zu ermitteln, auf welche mit dem Controller gezielt wird. Betätigt der Anwender nun den Auslöser am Controller, wird an die Stelle des Laserpunktes ein Farbklecks mit der vorher in den Controller eingegebenen Zeichenkette projeziert [zwille2010].

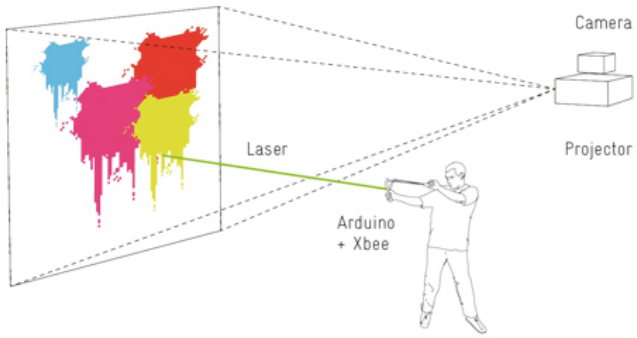

Abbildung 3.4: Schematischer Aufbau SMSlingshot (Quelle: [vrurbanSchema])

Abbildung 3.5: Das Eingabegerät bei SMSlingshot - Die Zwille (Quelle: [vrurbanZwille])

3.2 Szenario

Das folgende Szenario ist inspiriert aus Teilen der beiden oben genannten Projekte „Aarhus by Light" und „SMSlingshot". Auf dieser Grundlage soll ein mögliches Beispielszenario vorgestellt werden.
Es soll ein System entworfen werden, mit dem es ähnlich einfach ist, eine Mensch-Maschine-Kommunikation zu realisieren wie bei „Aarhus by Light", möglichst ohne speziellen Controller. Im Gegenstatz zu den beiden oben genannten Projekten soll hier aber einen Schritt weitergegangen werden, und zwar die Ermöglichung einer weiterführenden Kommunikation mit dem System. Diese Kommunikation wird durch einfache Gesten erfolgen.
Der Benutzer steht vor einer großen Projektionsfläche, auf der er einen simulierten Raum sieht. In der Mitte des Raums befindet sich eine Plattform, auf der ein Quader liegt. Tritt der Benutzer in den Messbereich der Anlage ein, werden die Hände des Benutzers erfasst und ebenfalls in diesem Raum abgebildet. Der Benutzer ist nun in der Lage mithilfe seiner Hände mit dem Quader zu interagieren. Die Stellvertreter der Hände innerhalb des Simulationsraums können den Quader berühren und bewegen und anheben. Dabei werden Gesten im Zusammenhang mit der Interaktion mit dem Quader ermittelt. Der Benutzer kann als Beispiel nach dem Quader greifen, ihn anheben und somit bereits Gesten beschreiben.
Wird vom System eine Geste erkannt, soll diese dem Benutzer visuell mitgeteilt werden.

3.3 Gesten

Aus dem Beispielszenario in Kapitel 3.2 sollen hier vier mögliche Gesten exemplarisch genauer beschrieben und analysiert werden. Inspiriert durch SHRDLU[5], geschrieben von Terry Winograd [Winograd (1972)], soll die Namensgebung wie folgt lauten:
Der in der Simulation verwendete Quader wird 'Block' genannt. Die Hände des Anwenders werden als Aktoren bezeichnet. Wenn speziell auf eine Hand Bezug genommen wird, wird diese ensprechend der Hand, um welche es sich handelt, als 'Left-Hand', die linke Hand oder als 'Right-Hand', die rechte Hand bezeichnet. Die Bodenplatte, auf der der Block zum Liegen kommt, wird 'Table' genannt.
Im Folgenden werden die vier beispielhaften Gesten aufgelistet.

[5] SHRDLU - http://hci.stanford.edu/~winograd/shrdlu/

3.3.1 Schieben

Die Geste des Schiebens wird realisiert, indem der Benutzer eine flache Hand formt und diese so weit bewegt bis eine Seite seiner Hand den Block berührt. Der Block liegt dabei bewegungslos auf dem Boden. Bewegt man die Hand jetzt weiter in Richtung des Blocks, so gibt der Block nach. Im Ergebnis bewegen sich Block und Hand in dieselbe Richtung. Der Block bleibt dabei auf dem Table liegen, er wird geschoben. Stoppt man die Handbewegung, endet gleichzeitig auch die Bewegung des Blocks.

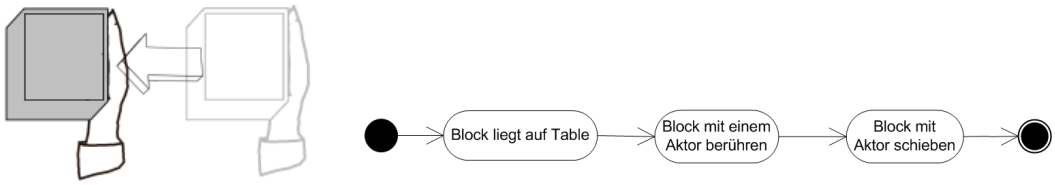

Abbildung 3.6: Schieben

3.3.2 Greifen

Die Formulierung „greifen” ist mehrdeutig. Sie kann sowohl für das Greifen mit einer Hand als auch das Greifen mit zwei Händen stehen. In dieser Arbeit wird das Wort „greifen” und auch der Begriff „Greifgeste” immer im Zusammenhang mit beidhändigem Greifen genannt, sofern es nicht im Einzelfall explizit anders beschrieben wird.
Bei der Greifgeste werden beide Hände an die sich jeweils gegenüberliegenden senkrecht stehenden Seiten des Blocks gelegt. Dabei ist es unerheblich, ob die beiden Hände gleichzeitig oder nacheinander den Block berühren, solange die erste berührende Hand in dem Moment noch Kontakt zum Block hat, wenn die zweite Hand den Block berührt. Bei dieser Geste ist die Lage des Blocks im Raum ohne Bedeutung. Das heißt, es kann außer Acht gelassen werden, ob der Block Kontakt zum Table hat oder nicht. Der Anwender kann also nach dem Block am Boden als auch in der Luft greifen.

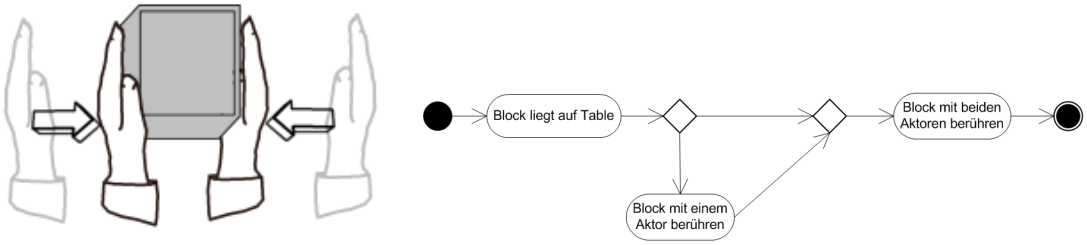

Abbildung 3.7: Greifen

3.3.3 Heben

Die Hebegeste geht aus der Greifgeste hervor. Genauso wie beim Greifen müssen beide Hände den Block an sich gegenüber liegenden Seiten berühren. Die Greifgeste ist somit eine Vorbedingung der Hebegeste. Werden die Hände nun nach oben bewegt, folgt der Block den Händen. Der Block wird angehoben. Beim erstmaligen Ausführen der Geste wird der Block für gewöhnlich auf dem Boden liegen, diesen also berühren. Für die Hebegeste an sich ist aber das vorherige Berühren des Tables unerheblich. Das bedeutet, es wäre theoretisch möglich, den Block auch in der Luft zu greifen und ihn von dieser Position aus weiter anzuheben.

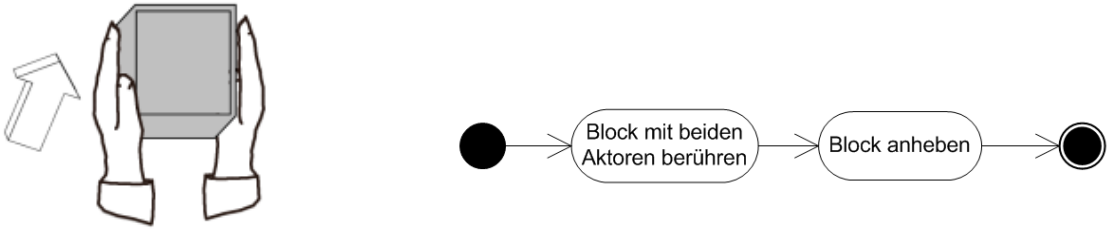

Abbildung 3.8: Heben

3.3.4 Loslassen

Ein Loslassen liegt vor, wenn der Block mit beiden Händen gegriffen wird und sich eine oder beide Hände von dem Block lösen. Der Block kann sich zu diesem Zeitpunkt in der Luft befinden, oder auf dem Boden liegen. Die Loslassgeste kann auf die Greif- oder auf die Hebegeste folgen. Beide Gesten für sich sind eine Vorbedingung für diese Geste.

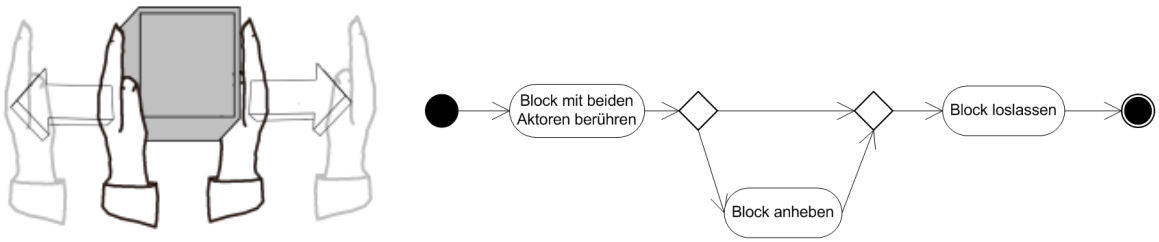

Abbildung 3.9: Loslassen

3.4 Funktionale Anforderungen

In diesem Teil werden die funktionalen Anforderungen ermittelt, beginnend mit den Anwendungsfällen.

3.4.1 Anwendungsfälle

Wie in dem Szenario in (3.2) beschrieben, sollen einfache Gesten im Raum erkannt werden. Jede der vier Gesten kann in einem Anwendungsfall abgebildet werden.

Block schieben Der Block liegt auf dem Table, ein Aktor berührt den Block seitlich und schiebt diesen über den Table.

Block greifen Der Anwender berührt den Block mit seinen beiden Aktoren. Die Berührung findet auf sich jeweils gegenüberliegenden Seiten statt.

Block anheben Der Block wird bereits vom Anwender mittels beider Aktoren gegriffen und wird dann angehoben. Der Verlust des Kontakts zwischen Block und Table spielt hier keine Rolle.

Block loslassen Ein oder beide Aktoren berühren den Block nicht mehr. Falls der Block zuvor angehoben wurde, fällt er zu Boden.

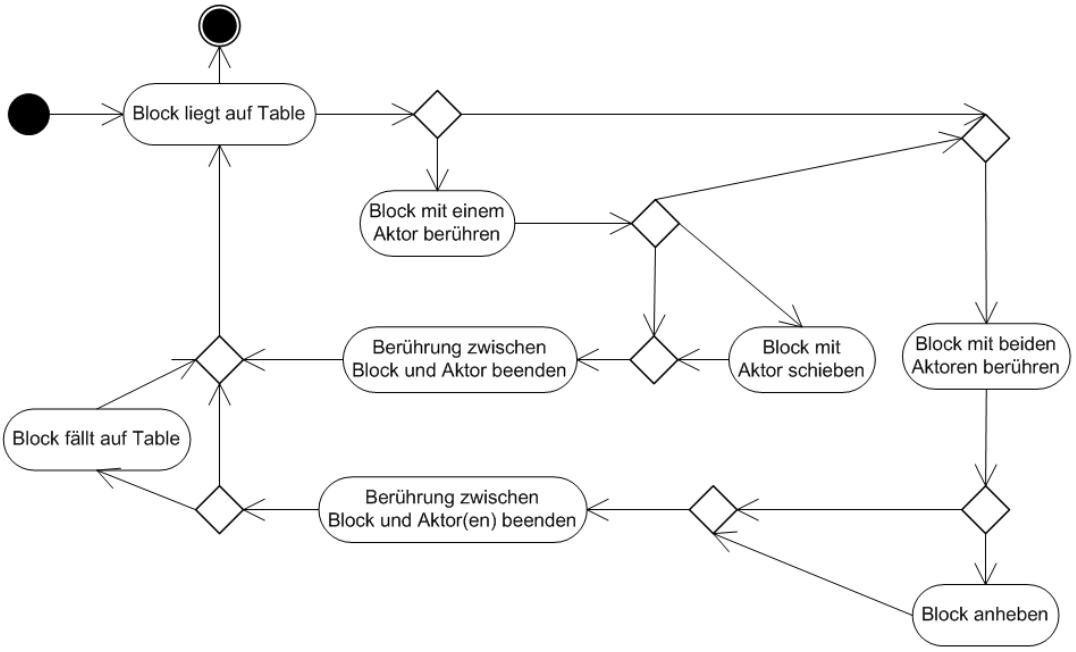

Abbildung 3.10: Aktivitätsdiagramm aller Gesten

3.4.2 Verwendete Techniken

Alle Anwendungsfälle stellen bestimmte technische Anforderungen an das zu entwickelnde System.

Aktor-Tracking Die beiden Hände des Anwenders treten hier als Aktoren auf. Für alle Anwendungsfälle ist es nötig, dass die Positionen der Hände im Raum zuverlässig dreidimensional erkannt werden. Die Positionen müssen regelmäßig neu erfasst und an das System weitergeleitet werden.

Kollisionserkennung Die Positionen der Hände werden in der Simulation durch Stellvertreterobjekte dargestellt. Diese Stellvertreterobjekte werden als Aktoren bezeichnet. Die Aktoren interagieren mit dem nur in der Simulation existierenden Block. Um festzustellen, wann sich die Aktoren mit der Position des Blocks überschneiden, wird eine Kollisionserkennung zwischen den Aktoren und dem Block benötigt.

Schwerkraft Um ein mögliches Anheben des Blocks so intuitiv wie möglich zu gestalten, sollte es den normalen Erwartungen der Benutzer möglichst nahe kommen. Real erwartet man bei einem Versuch etwas anheben zu wollen, dass dieser Handlung eine

Kraft entgegenwirkt. Wenn diese Kraft überwunden wurde, kann man den Block auch wirklich anheben. Lässt man den Block bei einem Hebevorgang los, erwartet man, dass dieser wieder zu Boden fällt.

Reibung Eine Möglichkeit das Heben zu realisieren, ist die Verwendung von Reibung. Wenn die auftretenden Reibungskräfte zwischen den Händen und dem Block größer sind als die Schwerkraft, welche auf den Block einwirkt, ist man in der Lage mit seinen Händen den Block anzuheben, indem man einfach die Hände nach oben bewegt.

Die oben genannten Anwendungsfälle werden in Abbildung 3.11 zusammengefasst.

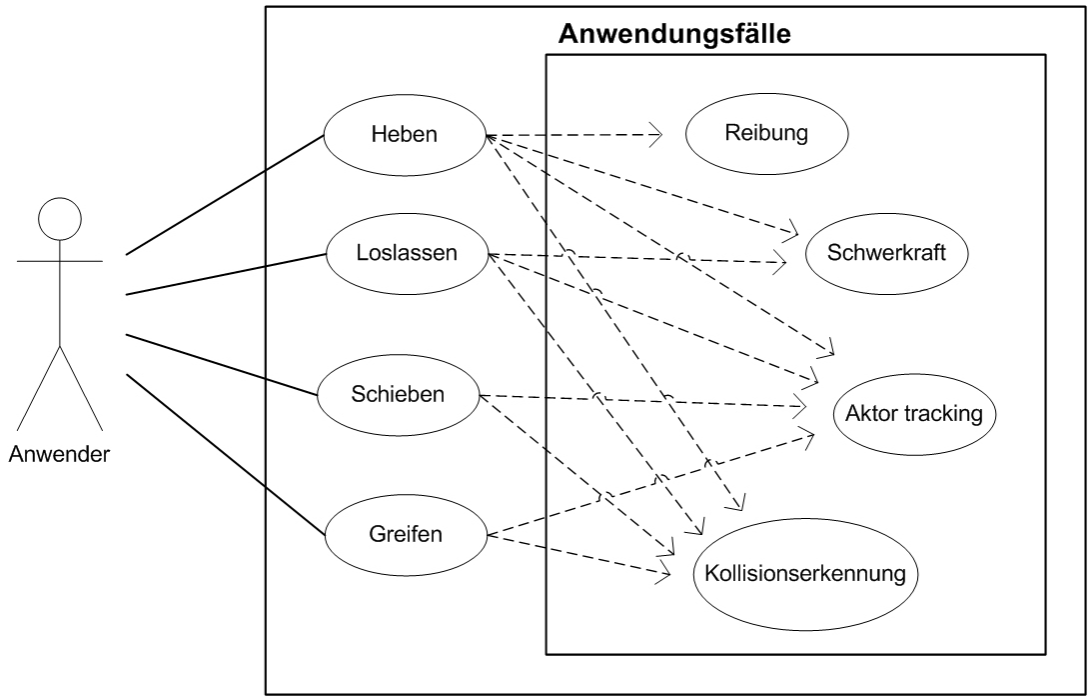

Abbildung 3.11: Usecase Diagramm

3.5 Zusammenfassung

In diesem Kapitel wurden in Abschnitt 3.1 die Projekte „Aarhus by light" und „SMSlingshot" beschrieben, welche Teil der Überlegungen sind, die in das spätere Szenario eingeflossen sind.
Das Szenario in (3.2) bildet die Grundlage der weiteren Arbeit. Es wird ein Szenario beschrieben, in dem der Benutzer mittels einfacher Gesten mit einem System kommunizieren kann. Die Kommunikation soll möglichst ohne weiteren Controller, den der Benutzer in seinen Händen hält, erfolgen.
In Abschnitt 3.3 wurden einige Gesten textuell, wie auch mithilfe kleiner Diagramme beschrieben, die im Szenario (3.2) möglich sind.
Auf Grundlage der Gesten wurden die funktionalen Anforderungen in (3.4) ermittelt. Gemeinsame Anforderungen zum Umsetzen der Gesten wurden herausgefiltert und zum Schluss in einem Usecase-Diagramm 3.11 abgebildet.

Kapitel 4

Design

Der erste Teil dieses Kapitels befasst sich detaillierter mit der Funktionalität, die ein System leisten muss, das den Anforderungen aus Kapitel 3 genügen will. Es wird auf Aufbereitung und Auswertung der vom Motiontracker übermittelten Positionsdaten eingegangen. Danach folgt die nähere Beschreibung einer Simulation, in der die Gestenerkennung stattfinden soll. Die für ein möglichst realistisches Verhalten der simulierten Gegenstände im Physikraum verwendeten Funktionen werden detailliert erklärt. Es wird kurz auf die grafische Ausgabe eingegangen. Anschließend werden einige Umsetzungsvorschläge für die Gestenerkennung vorgestellt, die infrage kommen könnten.
Teilkapitel 4.2 befasst sich mit der Architektur, die für die Umsetzung der Software verwendet wird. Es folgt eine Erläuterung, wie das MVC-Entwurfsmuster bei der Umsetzung zum Tragen kommt. Darüber hinaus werden die einzelnen Komponenten des Systems beschrieben und der Arbeitsablauf der Software wird aufgezeigt.

4.1 Funktionalität

Im Folgenden werden mögliche Funktionsweisen für die geforderte Gestenerkennung mittels einer Physiksimulation aufgeführt. Die Verfahren werden derjenigen Reihenfolge vorgestellt, in welcher sie später teilweise in der Software zur Verarbeitung der Daten zum Einsatz kommen würden. Zuerst soll beschrieben werden, wie die Daten in das System übertragen werden. Danach erfolgt eine Beschreibung unterschiedlicher Verfahren zur Aufbereitung der Daten. Im Anschluss wird die Simulation besprochen und die sich daraus ergebenden Verfahren um Gesten zu erkennen.
Es sollen hier auch die nötigen weiterführenden Funktionalitäten dargestellt werden, die zur Umsetzung einer bestimmten Vorgehensweise notwendig sind. Aus den hier aufgeführten Techniken werden später einige derer ausgewählt und dann softwaretechnisch umgesetzt.

4.1.1 Abgrenzung

Die Umsetzung erfolgt mithilfe des ARTtrackers, und einem Controller mit passiven Markern. Andere Methoden zur Bewegungserkennung wie Time of flight Kameras, der Kinect oder mithilfe von Computer-Vision gestütze Verfahren werden im Weiteren nicht berücksichtigt.

4.1.2 Eingabegerät

Auch wenn nicht ganz auf ein Eingabegerät verzichtet werden kann, soll dieses doch möglichst einfach gehalten sein. Der Benutzer trägt zwei Handschuhe mit jeweils fünf Markern. Die Marker sind an den Finterspitzen der Handschuhe angebracht, wie in Abbildung 4.1 zu sehen ist. Im Gegensatz zu den statischen Markern (2.2.1.4) sind hier die Abstände und Positionen der Marker untereinander variabel. Es handelt sich hier also um freie Marker, das bedeutet jeder Marker muss einzeln betrachtet werden. Die Übertragung aus dem ARTtracker erfolgt wie in (2.2.1) beschrieben und die Daten liegen somit als 3DOF vor (2.2.1.2).

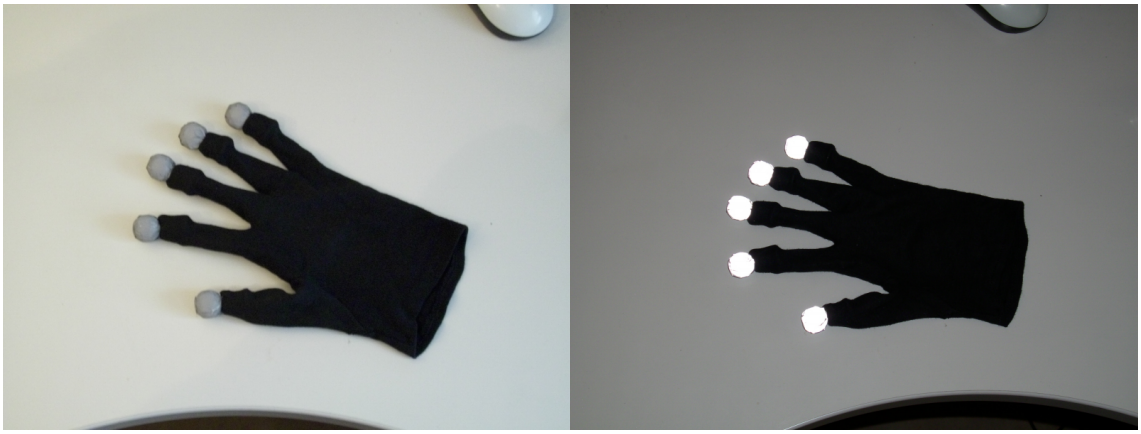

Abbildung 4.1: Selbstgebauter Handschuh mit Markern links bei normalem Lichteinfall, rechts mit Blitz fotografiert

4.1.3 Auswertung der Punktdaten

Die vom Motiontracker übermittelten Daten sind verrauscht. Das bedeutet die Daten können Punkte enthalten, die nicht zu den Handschuhen gehören. Darüber hinaus kommt es vor, dass in einigen Frames nicht alle Punkte der Handschuhe gefunden werden und diese somit auch nicht übertragen wurden. Dieses Datenrauschen gilt es möglichst zu minimieren und herauszufiltern.
Die einzige vom System mitgeliferte Hilfe bei der Aufbereitung der Punkte sind die PunktIDs.

Punkt IDs Der Punkttracker liefert seine Punkte mit eindeutigen Identifikationsnummern aus. Solange die Trackingsoftware sicherstellen kann, dass über mehrere Frames der Punkt immer derselbe ist, bekommt dieser Punkt in den Daten auch dieselbe ID. Bei den ersten Versuchen mit dem Punkttracker stellte sich heraus, dass die Trackingsoftware häufig die Punkte verliert und neue IDs vergibt. Eine zuverlässige Auswertung und somit Ermittlung der Aktoren, allein auf diesem Verfahren basierend, ist damit nicht möglich. Das Verfahren spielt also keine wesentliche Rolle in der Auswertung der Punktdaten. Jedoch können übereinstimmende IDs, sofern sie vorhanden sind, als Hilfe zur Wiedererkennung von Punktwolken hinzugezogen werden. Wenn ein Teil der Punktwolke bereits bekannte IDs enthält, kann auf vorher erfasste Punktwolken geschlossen werden.

4.1.3.1 Messbereich

Zuerst sollen die Punkte entfernt werden, die scheinbar nicht von den Handschuhen stammen.

Raumbegrenzung Zunächst wird ein Raum innerhalb des von dem Motiontracker erfassten Bereichs ausgewählt. Danach werden alle Punkte außerhalb dieses Raums verworfen. Dieses Verfahren entfernt somit Punkte, die zum Beispiel durch Reflexionen und Spiegelungen von Geräten im Labor außerhalb des relevanten Messbereichs erzeugt werden. Ist der Messbereich mit der Simulationsumgebung abgestimmt, werden nur Messpunkte innerhalb des durch die Simulation erzeugten Raums dargestellt. Dem Anwender wären so die Grenzen des Messbereichs schnell ersichtlich und er könnte obendrein diesen bewusst verlassen ohne ungewollte Änderungen in der Simulation hervorzurufen.

4.1.3.2 Punktwolkenanalyse

Die hier folgenden Verfahren zur Findung und Erkennung von Punktwolken können entweder miteinander kombiniert oder nacheinander zur Anwendung kommen, das bedeutet es wird ein Verfahren auf die Ergebnismenge des zuvor benutzten Verfahrens angewendet.

Erwartungshaltung an Anzahl von Punkten und Wolken Als Erwartungshaltung wird von zwei Händen mit jeweils maximal fünf Markern ausgegangen. Jede Hand hat also fünf Punkte, die pro Abtastframe maximal erkannt werden. Es wird ein Minimum von drei vorhandenen Punkten pro Hand festgelegt, damit diese selbst als Wolke gelten können. Wird mehr als ein Frame als Grundlage der Wolkenfindung verwendet, erhöht sich die Anzahl der Punkte folgendermaßen: bei einem Frame fünf Punkte pro Hand, bei zwei Frames 10 Punkte, bei drei Frames 15 Punkte usw. Auch die Mindestanforderungen steigen im selben Verhältnis Anzahl Frames x drei Punkte. Dies erlaubt

die Möglichkeit auch Frames zu akzeptieren, in denen durch ungünstige Umstände eventuell nur zwei Punkte der Hand übertragen wurden, wenn die darauffolgenden Frames dann wieder vier Punkte tracken. Die Anzahl der zu findenden Punktwolken bleibt konstant zwei, da zwei Hände im Einsatz sind. Ein und diesselbe Hand kann sich in zwei aufeinanderfolgenden Frames nicht soweit von der vorherigen Position entfernen, dass sie nicht als eine Wolke gelten würde. Dies setzt eine entsprechend hohe Abtastrate voraus, die mit dem ARTtracker bei einer Abtastrate von bis zu 60 Bildern pro Sekunde gegeben ist (siehe [ARTtracker-Anleitung (2007)]).

Erwartungshaltung an Bewegung der Wolke Davon ausgehend, dass es sich bei den zu verfolgenden Punktwolken um zwei Hände handelt, ergibt sich eine Erwartungshaltung an diese Wolken. Die beiden Punktwolken können nicht ohne weiteres verschwinden, unter der Voraussetzung, dass die Hände den Messbereich der Anlage nicht verlassen. Darüber hinaus werden die Hände einen kontinuierlichen Weg beschreiben. Die Hände und somit auch die Punktwolken können zwischen zwei Messungen nur eine bestimmte Wegstrecke zurückgelegt haben, daher springen die Punktwolken nicht. Misst man Bewegungsrichtung und Geschwindigkeit, kann man eine Aussage darüber machen, wo die Wolke sich ungefähr bei der nächsten Messung befinden muss. Dieser Bereich, in der die Wolke vermutet wird, vergrößert sich entsprechend der Zeit, die bis zur nächsten Messung verstreicht.

Maximaler Abstand zweier Finger Die Ausdehnung der zu erwartenden Punktwolke wird maßgeblich durch den maximalen Abstand der voneinander am weitesten entfernten Marker bestimmt. Wie in Abbildung 4.2 gezeigt, befinden sich diese beiden Marker auf dem Daumen und dem kleinen Finger. Spreizt man die Hand, kann man diesen maximalen Abstand ermitteln und in die spätere Bewertung einfließen lassen.

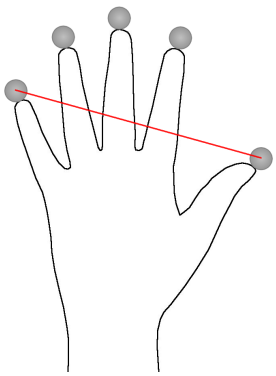

Abbildung 4.2: Maximaler Abstand von Daumen und kleinem Finger

Maximale Fläche der umspannenden Marker Eine weitere Möglichkeit die Wolke dahingehend zu analysieren, ob es sich ihrer Zusammensetzung nach um eine Punktwolke einer Hand handelt, ist den die Wolke umschließenden Flächeninhalt zu betrachten. Hierbei werden immer nur zwei der drei Dimensionen zur Berechnung hinzugezogen. Es werden also insgesamt drei Berechnungen vorgenommen. Wie die Abbildung 4.3 verdeutlicht, wird eine flache Hand beispielhaft angenommen, um dann die Punkte miteinander zu verbinden. Angelehnt an den ConvexBottom-Algorithmus entnommen von [GeoLab] werden die Punkte nun verbunden. Im Gegensatz zum ConvexBottom-Algorithmus müssen nicht immer die Extremwerte der X-Achse Verwendung finden. Bei einer flachen Hand wären die Extremwerte wohl bei den beiden Fingern zu finden, die auch den maximalen Abstand zueinander haben. Von dort aus müsste man dann die folgenden Punkte in eine Reihenfolge bringen und danach verbinden. Hier mit den Nummern in Abbildung 4.3 gekennzeichnet.

Dieses Verfahren trifft eine Aussage darüber, ob es sich bei einer Wolke um eine Hand handeln könnte oder eventuell nicht zugehörige Punkte gefunden wurden. Solange der Flächeninhalt nicht größer der einer flachen ausgestreckten Hand ist, kann es sich um eine Punktwolke von einer Hand handeln. Sollte der berechnete Flächeninhalt jedoch größer sein, kann es darauf hindeuten, dass einer oder mehrere Punkte nicht zu der eigentlichen Punktwolke der Hand gehören. Dieses Verfahren kann sich als hilfreich erweisen, wenn sich beide Hände dicht beieinander befinden und man mögliche Punkte zwischen den beiden errechneten Mittelpunkten der Wolken zuweisen möchte.

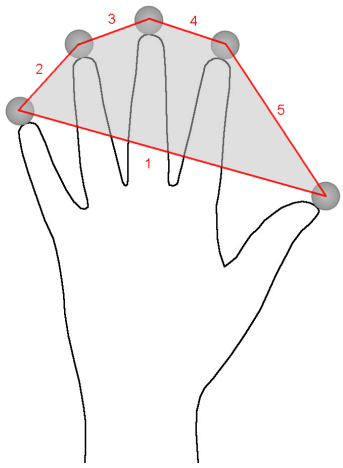

Abbildung 4.3: Maximaler Fächeninhalt einer Punktwolke bei ausgestreckter Hand. Die Zahlen zeigen die Reihenfolge, in der die Punkte miteinander verbunden werden, um ein Polygon zu bilden und anschließend die Fläche zu berechnen.

Schwerpunkte der Wolke bestimmen

Nachdem alle übermittelten Punkte zu Punktwolken zusammengefasst oder in Einzelfällen aus der Berechnung entfernt wurden, können die Punktwolken selbst betrachtet werden. In einem ersten Schritt wird der Schwerpunkt der Punktwolke ermittelt. Um den Schwerpunkt zu berechnen, wird das arithmetische Mittel jeweils aus den x-, y- und z-Koordinaten der Punkte mit folgender Formel gebildet.

$$X_s = \frac{1}{n} \sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + x_3 + ... + x_n}{n}$$

Die so erhaltene Punktkoordinate im Raum ist der Schwerpunkt der Wolke und wird als Mittelpunkt des Aktorobjekts verwendet.

4.1.4 Simulation

Die Simulation der Datenobjekte erfolgt durch Erzeugen einer jeweiligen Entität in der 3D- und der Physik-Engine.
Die Eigenschaften zur grafischen Darstellung werden an die 3D-Engine weitergereicht, die zur physikalischen Darstellung hingegen an die Physik-Engine. Einige Eigenschaften wie die Abmessungen der Datenobjekte sind für beide Entitäten von Bedeutung, siehe hier auch (2.3.3).
Die Manipulation der Entitäten durch den Benutzer erfolgt ausschließlich über die Physik-Engine. Eine direkte Manipulation der Aktoren ist aber unmöglich. Der Benutzer kann durch das Tracking seiner Handschuhe eine Manipulation der Aktoren anregen und eine neue Position vorgeben. Ob die Aktoren diese Position einnehmen können, entscheidet aber die Physik-Engine. Ein Erzwingen der Manipulation ist somit durch den Benutzer nicht möglich. Die tatsächliche Position der Hände und der stellvertretenden Aktoren in der Simulation kann daher abweichen. Die 3D-Entitäten werden bei Veränderung der Position der entsprechenden Physikentitäten angepasst, dies erfolgt über den Observer (siehe 4.2.2).
Im Folgenden wird gesondert auf die beiden Simulationen eingegangen.

Physiksimulation

Anforderungen an die Physik-Engine

Die unten angegebenen Anforderungen sind aus benötigten Eigenschaften aus Kapitel 3 hergeleitet und gegebenenfalls erweitert worden.

Kollisionserkennung Die Physik-Engine muss das Berühren oder Überlappen zweier oder mehrerer geometrisch/physikalischer Objekte unter physikalischer Belastung zuverlässig erkennen und eine angebrachte Kollisionsbehandlung erbringen. Mit physikalischer

Belastung ist in diesem Fall gemeint, dass eine Kraft ein geometrisch/physikalisches Objekt gegen ein anderes drückt. Dieses Objekt darf die Oberfläche des sperrenden Objekts nicht durchdringen. Beim Testen verschiedener Physik-Engines ist es vorgekommen, dass Objekte nach kurzer Zeit oder aber bei genügend Kraftaufwand die eigentlich undurchlässige Oberfläche durchdrungen haben.
Die Kollisionserkennung soll darüber hinaus die Möglichkeit an den Benutzer liefern, dass auch Kollisionen von Objekten gemeldet werden, die sich berühren aber nicht mehr bewegen.

Masse Objekten kann eine Masse frei zugewiesen werden. Dabei ist es auch erforderlich, dass Objekten die Masse 0, also keine Masse, zugeordnet werden kann. Diese Objekte sind somit erst einmal von den Effekten der Schwerkraft befreit. Es gibt folgende zwei Möglichkeiten die Masse eines Objektes festzulegen:

1. Man legt fest, wie viel ein Körper pro Kubikeinheit wiegt, gibt dann an, welche Ausmaße das eigentliche Objekt später besitzen soll und anhand dieser Werte errechnet die Physik-Engine die Masse des Körpers.
2. Es werden erst die Abmessungen des Körpers festgelegt. Dann errechnet die Physik-Engine eine Default-Masse und man setzt diese Masse nachträglich auf einen gewünschten Wert. Diese Variante hat den Vorteil, dass man unabhängig von den Ausmaßen eines Körpers immer mit derselben Masse rechnen kann. Außerdem wird die Konfiguration des Versuchsaufbaus entscheidend vereinfacht.

Schwerkraft Standartmäßig wird die Schwerkraft mit der Schwerkraft der Erde (9,81m/s²) initialisiert. Für einen Versuchsaufbau ist es von Vorteil, wenn man diesen Wert beliebig verändern kann. Einzelne Physikobjekte sollten von den Auswirkungen der Schwerkraft ausgenommen werden können.

Joints Eine Möglichkeit zwei Körper innerhalb der Physiksimulation miteinander zu verbinden, sind sogenannte Joints (siehe auch [Smith (2002)]), wie in Abbildung 4.4 dargestellt. Der Joint wird jeweils am Mittelpunkt des Physikkörpers befestigt. Er arbeitet ähnlich wie eine Feder (siehe [Roßberger (2008)]). Man kann dem Joint eine Länge mitgeben, sodass dieser nun versucht beide Körper genau in diesem Abstand voneinander fernzuhalten. Über- oder unterschreiten die Körper nun den vom Joint vorgegebenen Abstand, übt der Joint eine Kraft auf diese Körper aus. Zieht man einen Körper in eine Richtung, wirkt der Joint mit einer Kraft auf den zweiten Körper und dieser wird nachgezogen. Gelingt das Nachziehen zum Beispiel aufgrund einer Kollision mit einem anderen Körper nicht, bleibt diese Kraft bestehen und wirkt nun auch als eine Druckkraft auf das kollidierende Objekt.
Es gibt verschiedene Arten von Joints, einige von ihnen arbeiten nur längs bestimmter

Achsen oder erlauben nur Rotationen um bestimmte Achsen. Andere Joints hingegen arbeiten ähnlich einer starren Achse, sie übertragen auch Rotationen längs des Joints auf das andere Objekt.
Für den Versuchsaufbau wird der einfachste Joint verwendet, er besitzt keinerlei Einschränkungen und erlaubt jegliche Bewegungen und Rotationen. Er überträgt keine Rotationen, sondern nur Zugkräfte. Am ehesten lässt sich dieser Joint mit einem Gummiband vergleichen.

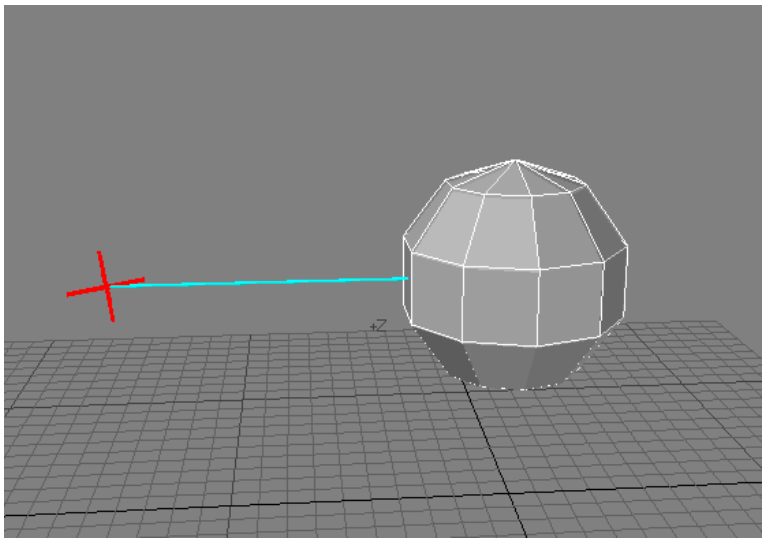

Abbildung 4.4: Joint (türkis) verbindet den Punkt (rot) und den Aktor

Physikentität ohne Körper Jeder Physikkörper verfügt über einen Mittelpunkt. Voraussetzung für die Abbildung der tatsächlichen Position des Mittelpunktes der Punktwolke ist ein Objekt, das nur aus einem Punkt besteht. Dieser Punkt existiert in der Physik, besitzt aber keinen Körper. Demzufolge darf er weder von Schwerkraft beeinflusst werden noch Kollisionen verursachen. Gleichwohl muss es möglich sein, dieser Entität unter Zuhilfenahme eines Joints an eine andere Physikentität zu koppeln, siehe Abbildung 4.4.

Reibung & Oberflächeneigenschaften Ebenso wie man auch in einer 3D-Grafikengine Körpern bestimmte Oberflächeneigenschaften, wie beispielsweise Farbe, Glanz und Spiegelung geben kann, ist es auch in einer Physik-Engine entscheidend, welche physikalischen Eigenschaften die Oberflächen eines Objektes besitzen. Ausschlaggebend ist dann, welche Haft- bzw. Gleitreibung diese verschiedenen Materialen besitzen, siehe hier für auch 2.3.2.

Grafiksimulation

Die Anforderungen an die Grafiksimulation sind sehr einfach gehalten. Die aus der Physik-Engine errechneten Positionen der Körper sollen grafisch dargestellt werden. Es folgt eine Übertragung aller Werte, also die für eine dreidimensionale Darstellung benötigten Daten, von der Physik-Engine an die 3D-Engine.
Die Darstellung der Entitäten durch die 3D-Engine soll einfach gehalten sein. Größtenteils werden nur Farben verwendet, nur in Ausnahmefällen werden Objekte mit Texturen versehen. Optische Feinheiten wie Glanz, Reflexion oder Transparenz der Oberflächen der zur Anzeige gebrachten Entitäten werden nicht benötigt.
Abgesehen von den Entitäten, muss die 3D-Engine noch über die Möglichkeit verfügen Texte darzustellen. Dies wird für die Ausgabe der erkannten Geste und für eventuelle weitere Kommunikation mit dem Anwender benötigt. Die Physik-Engine liefert der 3D-Engine den Namen der Geste, sobald sie erkannt wurde. Angezeigt wird die Grafiksimulation über die in Abbildung 4.5 gezeigte Powerwall.

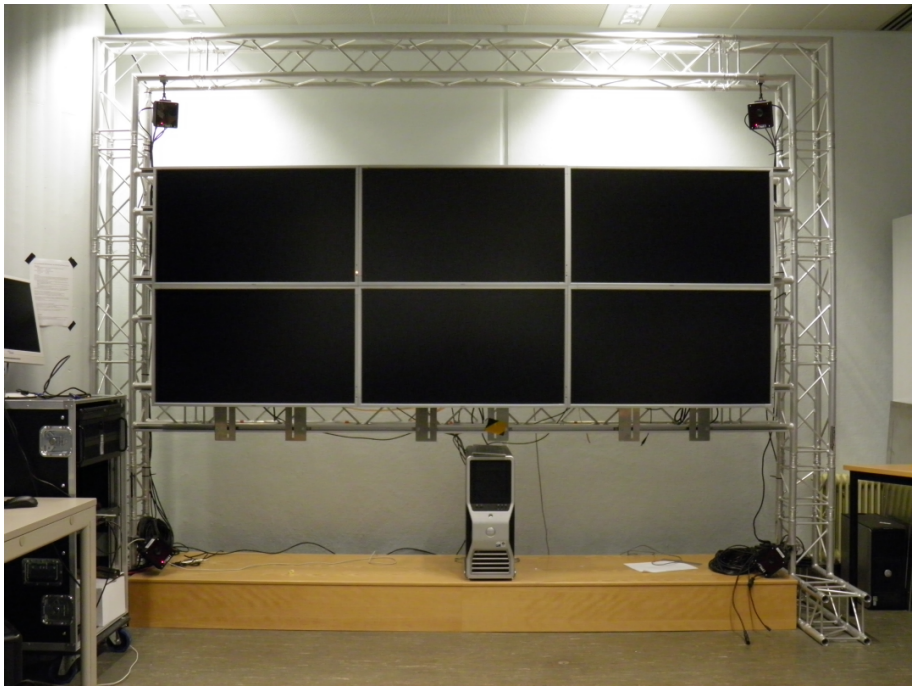

Abbildung 4.5: Powerwall (Stand 17.03.2011)

4.1.5 Gestenerkennung im Modell

In der Simulation stehen die vier folgenden Objekte zur Verfügung: die beiden Aktoren, die mithilfe des Controllers die Bewegung der Hände im Simulationsraum abbilden, den zu manipulierenden Block und den für den Block undurchdringlichen Table. Die durch die Physik-Engine errechnete Schwerkraft lässt den Block auf den Table fallen. Diese Konstellation wird nach einem Start oder einem Neustart der Simulation vorgefunden. Körper, die sich berühren, bilden eine Gruppe von Körpern. Im weiteren Text wird dieser Verbund als Assembly[6] bezeichnet. Eine Ausnahme bildet jedoch der Table. Objekte, die den Table berühren, bilden keinen Verbund mit dem Table. Ein Objekt registiert lediglich, ob es den Table berührt. Ein Assembly hingegen aus Block und zwei Aktoren, das den Table berührt, ist ein anderes Assembly als eines, das nur aus Block und den beiden Aktoren besteht. Solange diese Berührung zwischen den Aktoren und dem Block anhält, werden diese Körper nur gemeinsam betrachtet.
Es gibt nun drei verschiedene Möglichkeiten auf dieses Assembly einzuwirken:

- Im Grundsatz ist es möglich das Assembly wieder aufzulösen. Folglich beenden die Objekte ihre Berührung und die Baugruppe zerfällt wieder in ihre Einzelkomponenten.

- Eine andere Möglichkeit auf diese Baugruppe einzuwirken, ist, dass ein weiteres Objekt im Simulationsraum das Assembly berührt. Somit hört das vorherige Assembly auf in seiner Form zu existieren und es entsteht ein Neues aus dem Verbund der drei Objekte. Bei diesem Verfahren gibt es jetzt zwei Varianten. Die Erste ist eine einfachere aber auch ungenauere Art. Sie betrachtet nur die in der Baugruppe vorhandenen Einzelteile und schließt dann auf das Assembly. Die Zweite, in der Umsetzung wesentlich anspruchsvollere Variante ist, dass nicht nur die Anwesenheit eines Bauteils mitentscheidend ist, sondern auch seine Position im Assembly. Reihenfolge und Anordnung der Bauteile bestimmen hier über das Assembly, es kann also mehrere bauteilgleiche Baugruppen geben, nur ihr Zusammenbau ist unterschiedlich. Aus diesem Sachverhalt kann man eine weitere Möglichkeit der Gestenerkennung ableiten. Als Beispiel seien hier die Greifgeste (3.3.2) und die Präsentiergeste (4.7) genannt. Beide Gesten bestehen jeweils aus Block und zwei Aktoren.

- Als weitere Variante ist das Einwirken auf das Assembly von außen zu nennen. Hierbei wird das Assembly in seiner Existenz nicht verändert, nur die Zustände innerhalb der Baugruppe. Man kann beispielsweise die Position im Raum verändern, also es bewegen.

Im Zusammenhang der hier zu erkennenden Gesten und den möglichen Assemblys kann man nun die Gesten in unterschiedliche Gruppen aufteilen. Zum einem gibt es Gesten, die

[6] Assembly: Englisch für „Baugruppe", ein Assembly wird aus mehreren Bauteilen zusammengesetzt und kann auch wieder in diese zerlegt werden.

sich mit einem Assembly mehrfach ausführen und erkennen lassen, die Schiebe- und die Hebegeste sind Beispiele dafür. Das bedeutet man kann das Assembly schieben bzw. anheben, diese Bewegung unterbrechen, die Geste damit beenden und wieder mit der Bewegung fortfahren und somit die gleiche Geste ein weiteres Mal ausführen. Bei diesen Handlungen ändert sich jeweils nur ein interner Zustand, nämlich die Positionswerte der Punkte auf der X-Achse bzw. auf der Y-Achse.
Eine andere Gruppe von Gesten lassen sich nicht ohne weiteres wiederholt ausführen. Als ein Beispiel kann hier die Greifgeste (3.3.2) genannt werden. Ein Objekt kann nur einmal gegriffen werden, es muss dann erst einmal wieder losgelassen werden, bevor man erneut nach ihm greifen kann. Die Greifgeste wird nicht durch eine Zustandsänderung eines Assemblys erkannt, sondern durch die Erschaffung eines Assemblys. Das Assembly zum Erkennen dieser Geste besteht aus dem Block und den beiden Händen. Die Entscheidung, ob der Block oder das resultierende Assembly dabei den Table berührt, spielt keine Rolle.
Auch das Auflösen eines solchen Assemblys kann dazu benutzt werden, um auf eine Geste zu schließen, zum Beispiel auf die Loslassgeste (3.3.4). Wird das „Greif-Assembly" wieder zerlegt, lässt sich darauf schließen, dass der Benutzer den Block wieder losgelassen haben muss.

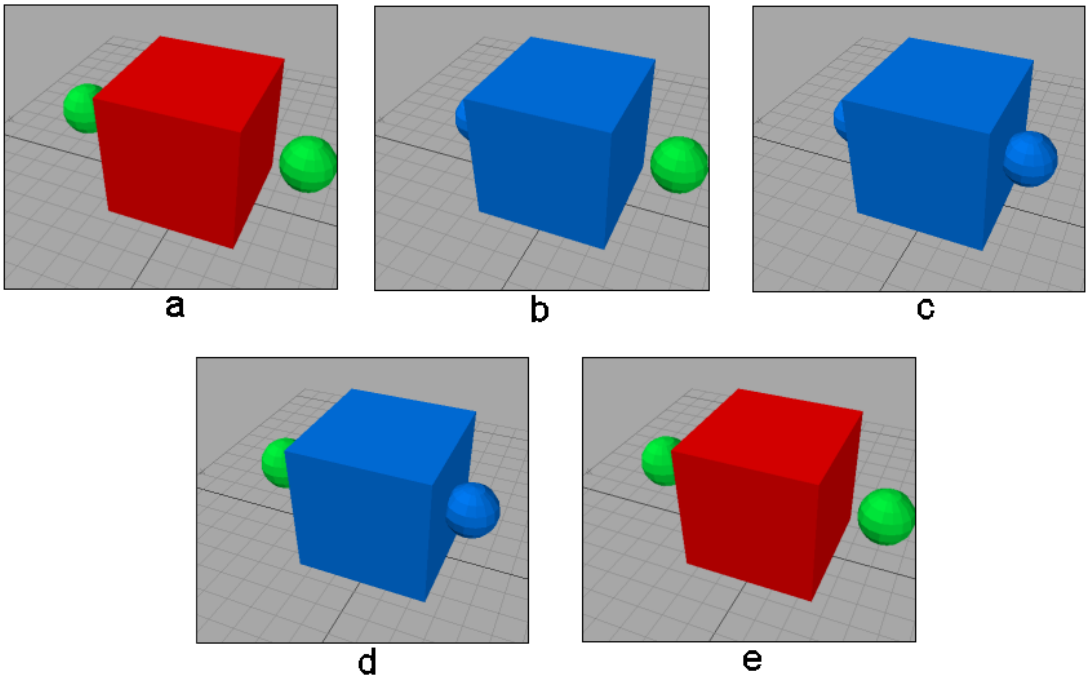

Abbildung 4.6: Bilderreihe, die eine Folge von Assemblys zeigt. Das entstandene Assembly ist jeweils blau eingefärbt.

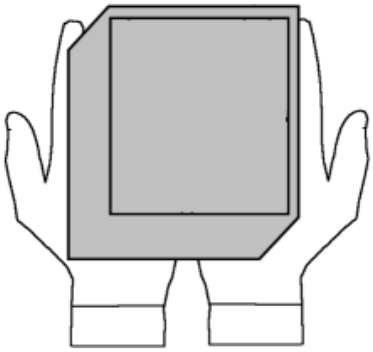

Abbildung 4.7: Präsentier-Geste

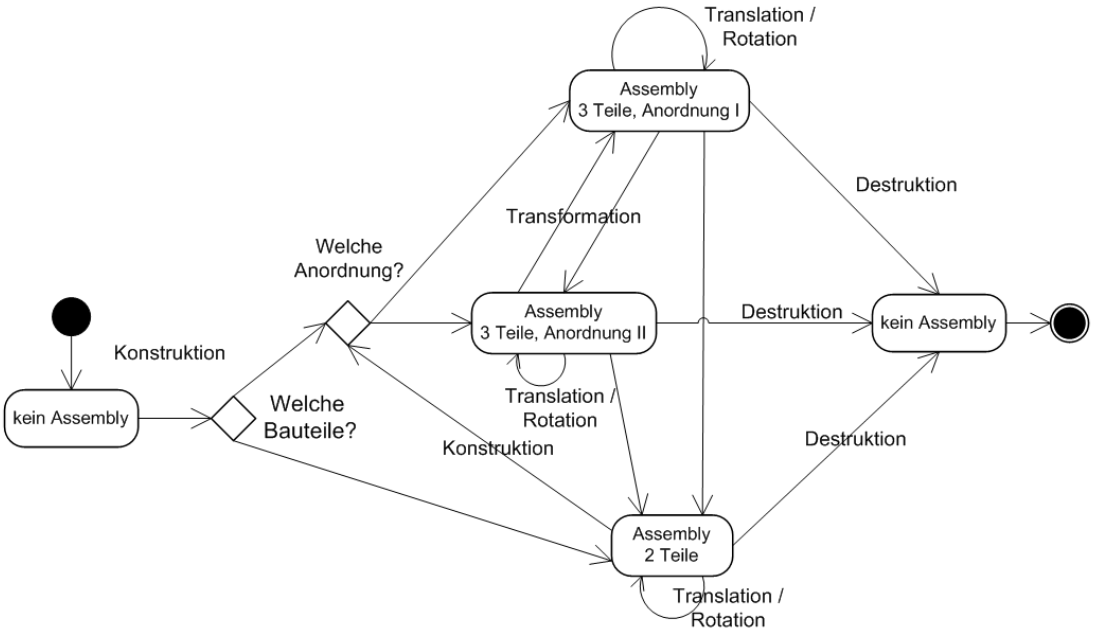

Abbildung 4.8: Zustandsübergänge Assembly

Gruppen der Handlungen für Gesten:

Man kann die Gesten nun in verschiedene Handlungsgruppen einteilen. Eine Gruppe beschreibt, welche Handlung notwendig ist, um auf die Geste schließen zu können. Die einzelnen Gruppenmitglieder werden im Folgenden erklärt.

Konstruktion Auf die Geste wird durch die Konstruktion eines Assemblys und die daran maßgeblich beteiligten Einzelkomponenten geschlossen. Dabei wird nicht betrachtet, welches und ob bereits ein anderes Assembly vorher existierte.

Destruktion Es wird durch die Auflösung eines bestimmten Assemblys auf die Geste geschlossen. Dabei wird nicht weiter betrachtet, ob aus dem Handeln andere Assemblys entstehen oder keins mehr vorhanden ist.

Transformation Alle an dem Assembly beteiligten Einzelkomponenten bleiben erhalten, nur ihre Position zueinander verändert sich. Wie in Abbildung 4.9 grafisch verdeutlicht, rutschen die Aktoren von den Seiten des Blocks unter den Block und halten ihn somit in der Luft.

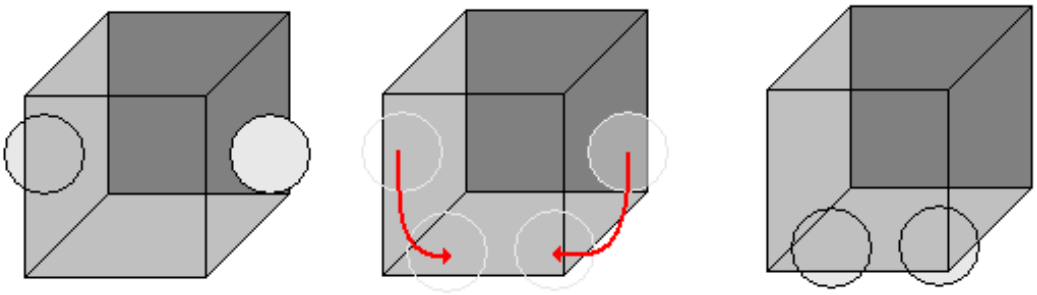

Abbildung 4.9: Transformation des Greifassemblys zu einem Präsentierassembly

Manipulation Die Manipulation betrifft sowohl die Translation als auch die Rotation des Assemblys, es bleibt als solches bestehen, nur innere Zustände des Assemblys werden geändert. Die Translation betrifft die Bewegung längs der drei Achsen[7] im Raum. Die Rotation findet um einen nicht weiter spezifizierbaren Punkt im Raum statt. Die Rotation wie auch die Translation werden durch Kraftausübung auf das Assembly verursacht. Die Kraftausübung findet durch die Aktoren, die Teil des Assemblys sind, statt. Je nachdem, in welche Richtung und mit welcher Intensität diese Kräfte ausgeübt werden, wird z.B. der Punkt, um den die Rotation stattfinden soll, von der Physiksimulation

[7] X-Achse (Längsachse),Y-Achse (Querachse) und Z-Achse (Vertikalachse)

frei bestimmt.
Komplexere Bewegungsabläufe können unter Zuhilfenahme der in Abschnitt 4.1.7 beschriebenen Trajektorie erkannt werden. Um bestimmte Manipulationen einer Geste zuzuordnen, werden die entsprechenden Trajektorien, die diese Manipulation beschreiben, vorher aufgezeichnet und mit der späteren tatsächlichen Flugbahn des Assemblys verglichen.
Wenn das Assembly in seiner Art bestehen bleibt und unter der Bedingung, dass die Trajektorie dies auch ermöglicht, ist diese Art von Gesten mehrfach hintereinander erkennbar.

Die Aufteilung der Gesten in die entsprechenden Handlungsgruppen kann aus Tabelle 4.1 entnommen werden.

Geste	Art der Handlung
Schieben	Manipulativ
Greifen	Konstruktiv
Heben	Manipulativ
Loslassen	Destruktiv

Tabelle 4.1: Gesten

4.1.6 Weitere Arten der Gestenerkennung

Entscheidungsmatrix Die vier hier zu erkennenden Gesten werden in Zustände untergliedert. Im Prinzip ist es dem Verfahren mit den Assemblys ähnlich, allerdings verzichtet man auf die explizite Errichtung von Assembly-Objekten, die einen Teil der Zustände abbilden würden.
Um nun mithilfe von Zuständen die gesuchten Gesten zu finden, bedarf es zwei Zustandtabellen, da viele der Gesten bei Zustandsänderungen gefunden werden. Es werden also die Zustände des letzten Durchlaufs gespeichert und dann mit den Zuständen verglichen, die bei der aktuellen Berechnung ermittelt wurden. Die Kombination aus „Alt"-Zuständen und „Neu"-Zuständen lassen sich die Übergänge ermitteln und so die entsprechenden Gesten finden. Ein Beispiel dazu gibt es in Kapitel 5 in der Tabelle 5.2.

Automat Die andere Möglichkeit unter Verwendung von Zuständen der einzelnen Objekte innherhalb der Simulation auf eine Geste zu schließen, ist ein Automat.
Die eigentliche Geste ist hier als Übergang zwischen dem Start- und dem Ziel-Knoten definiert.
Um zwischen den Zuständen in der Simluation und dem Zustand des Automaten zu

trennen, werden in diesem Textabschnitt die Zustände der Simualtion als Eingabe bezeichnet. Ein Knoten würde genau eine mögliche Kombination von Eingaben innerhalb der Simulation beinhalten.
Beim Hinzufügen einer Eingabe würde der Knoten über eine Kante verlassen werden und in einen neuen Knoten übergehen, es sei denn, die Kante, die beschritten wird, führt nicht wieder zum Ausgangsknoten.
Der Automat muss vollständig spezifiziert sein [Automaten], es muss also aus jedem Zustand (Knoten), für jede Eingabe eine Zustandsübergangsfunktion (Kante) geben.
Beim Start eines Durchlaufs wird der jetzige Zustand anhand seines Knoten gespeichert. Die Eingaben werden nacheinander in den Automanten gegeben, der nach jeder Eingabe seinen Zustand anpasst. Dabei wäre die Reihenfolge der Eingabe egal, auch das mehrfache Einsetzen einer Eingabe wäre ohne Auswirkung auf das Endergebnis möglich.
Die eigentliche Auswertung steht am Ende. Nach Abschluss der Eingabe wird der jetzige Knoten mit dem zuvor gespeicherten Start-Knoten verglichen, existiert ein Übergang, wurde die entsprechende Geste gefunden.
Dieses Verfahren wird recht schnell unübersichtlich, sobald mehrere Objekte an der Simulation teilnehmen und somit eine Vielzahl von Zuständen und auch Knoten vorhanden ist.

4.1.7 Weiterführende Gestenerkennung

Die bisherigen Gesten, die erkannt werden könnten, sind räumlich sehr eingeschränkt. Unter Verwendung von Trajektorien in Kombination mit den bislang vorgestellten Verfahren ist es jedoch möglich auch komplexere Gesten zu erkennen, die alle drei Dimensionen verwenden.

Trajektorie

Mit dem physikalischen Begriff Trajektorie, auch Flugbahn genannt, wird eine Ortskurve beschrieben, entlang der sich ein Punkt oder der Schwerpunkt eines Körpers bewegt. Im Gebiet der Dynamik und Kinematik wird die Trajektorie zur Untersuchung in einem zeitabhängigen Verlauf betrachtet. Die Trajektorie in Abbildung 4.10a wird im Weiteren in ihre drei Dimensionen zerlegt (siehe Abb. 4.10b bis 4.10d). Es wird hier die Position der jeweiligen Achse in Bezug auf die Zeit gesetzt.

Um eine Geste unter Verwendung von Trajektorien zu erkennen, kann man den folgenden Weg wählen. Im Vorfeld wird eine Trajektorie beschrieben. Diese Bahnkurve wird dann einer Geste zugeordnet. Bei der späteren Gestenerkennung wird diese Trajektorie zum Vergleich benötigt.

Beim Start einer Bewegung, die eine mögliche Geste beinhalten könnte, werden die Positionen des Mittelpunkts des Blocks über die Zeit aufgezeichnet. Eine Möglichkeit, um das Start-Ende Problem zu umgehen, wäre die Festlegung, dass zum Start einer Geste der Block mit beiden Aktoren „gegriffen" werden muss. Beim Ende einer Geste würde man den Block dann wieder loslassen. Somit würden Start und Ende einer Geste durch die jeweilige Greif- und Loslassgeste definiert.
Die so gewonnenen Positionsdaten werden in zeitlicher Reihenfolge miteinander verbunden und bilden somit auch eine Trajektorie. Will man nun die eben aufgezeichnete Trajektorie mit der gespeicherten gegenüberstellen, vergleicht man jeweils die einzelnen Bewegungsgrafen auf den Achsen miteinander. Die so entstehenden drei Funktionsgrafen-Paare müssen einander angeglichen werden. Der einzelne Start der Trajektorien auf der jeweiligen Achse mit dem Wert 0 beginnt zum Zeitpunkt t0. Siehe hierfür Abbildung 4.10e.
Man kann nun beide Bahnen miteinander vergleichen. Wenn man eine Geste nur dann als gefunden definiert, wenn alle drei Funktionsgrafen-Paare übereinstimmen, kann man die Geschwindigkeit der Auswertung erhöhen, indem man vorrangig nach Ausschluss-Kriterien bei den Vergleichen sucht. Trifft eins der Paare nicht zu, müssen die anderen zwei Paare für die zu vergleichende Geste nicht mehr untersucht werden. So könnte man unter Verwendung eines Entscheidungsbaums die Anzahl der infrage kommenden Gesten schnell verringern.

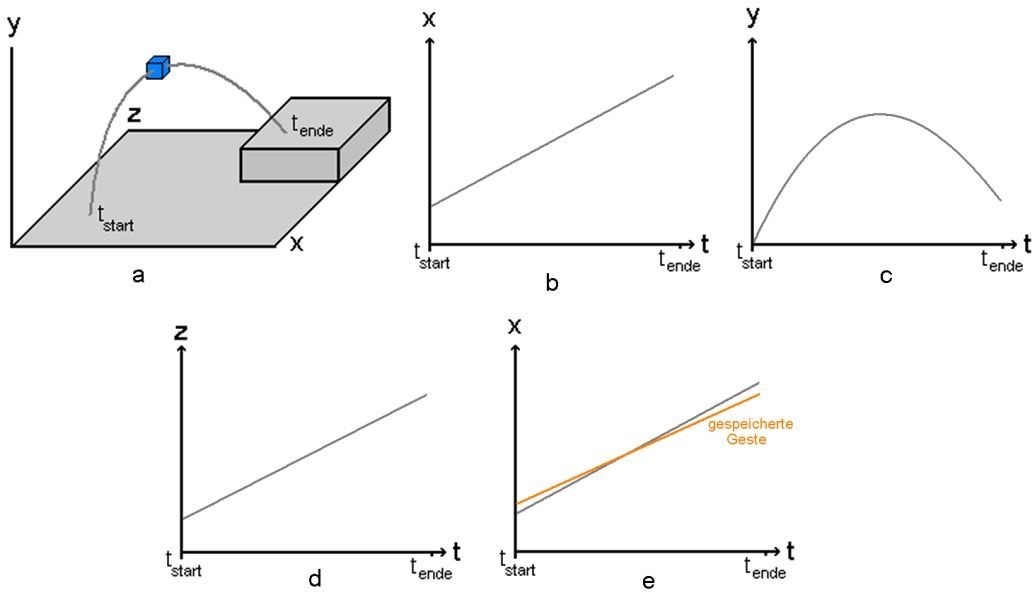

Abbildung 4.10: Trajektorie

4.2 Software-Architektur

Die zu entwerfende Software soll folgende Aufgaben erfüllen:

- Annahme und Aufbereitung von Eingabedaten
- Simulation der Auswirkungen der Eingabedaten
- Erkennung von Gesten auf Basis der Simulation
- Grafische Darstellung der Simulation

Die oben genannten Aufgaben können in einzelnen Modulen abgearbeitet werden.
Es empfiehlt sich für die Umsetzung das MVC[8] Entwurfsmuster zu verwenden.

4.2.1 MVC-Entwurfsmuster

Das MVC-Entwurfsmuster, entnommen aus [Fowler (2003)] und [Erich Gamma (2001)] wurde von Trygve Reenskaug Ende der 1970er Jahre für die Programmiersprache Smalltalk vorgestellt. In seiner Basis-Variante besitzt das MVC-Muster drei Rollen. Die Model-Rolle enthält alle wichtigen Daten.
Die View-Rolle übernimmt die Anzeige der im Model gehaltenen Daten für die Benutzerschnittstelle.
Der Controller nimmt die Eingaben des Benutzers entgegen, manipuliert die Daten im Model und sorgt für eine Aktualisierung der View, nachdem die Daten im Model geändert wurden.
Die einseitigen Kommunikationsverbindungen der Komponenten sind aus der Abbildung 4.11a zu entnehmen.
Darüber hinaus gibt es eine MVC-Variante mit aktiver Model-Komponente. Diese Variante ist nützlich, falls Änderungen im Model vorkommen, ohne dass die Controller-Komponente einbezogen ist. In einem solchen Fall muss die View-Komponente über mögliche Änderungen der Daten in dem Model informiert werden. Die Kommunikation zwischen Model und View erfolgt hierbei über das sogenannte Observer-Entwurfsmuster. Der Observer[9] beobachtet das Model und informiert die View über mögliche Änderungen im Model. Daraufhin aktualisiert die View ihre Darstellung. In Abbildung 4.11 rechts wird das MVC-Muster mit aktivem Model dargestellt. Die Abbildung 4.12 zeigt den sequenziellen Ablauf des MVC-Entwurfsmusters mit Observer.

[8] MVC = Model, View, Controller
[9] Observer = Beobachter

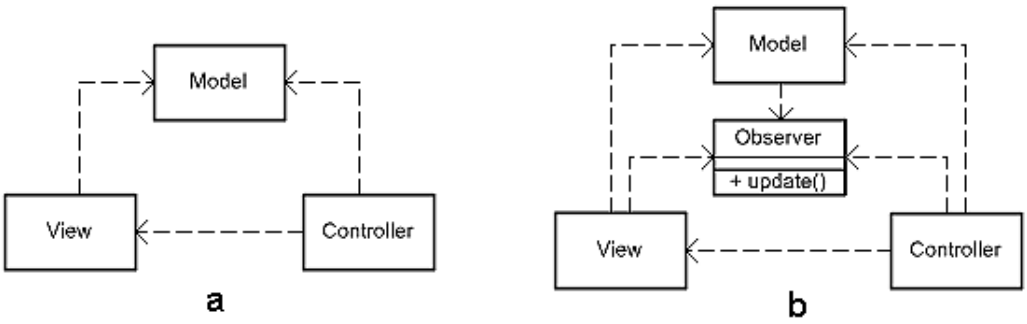

Abbildung 4.11: MVC-Muster rechts, MVC-Muster mit Observer links

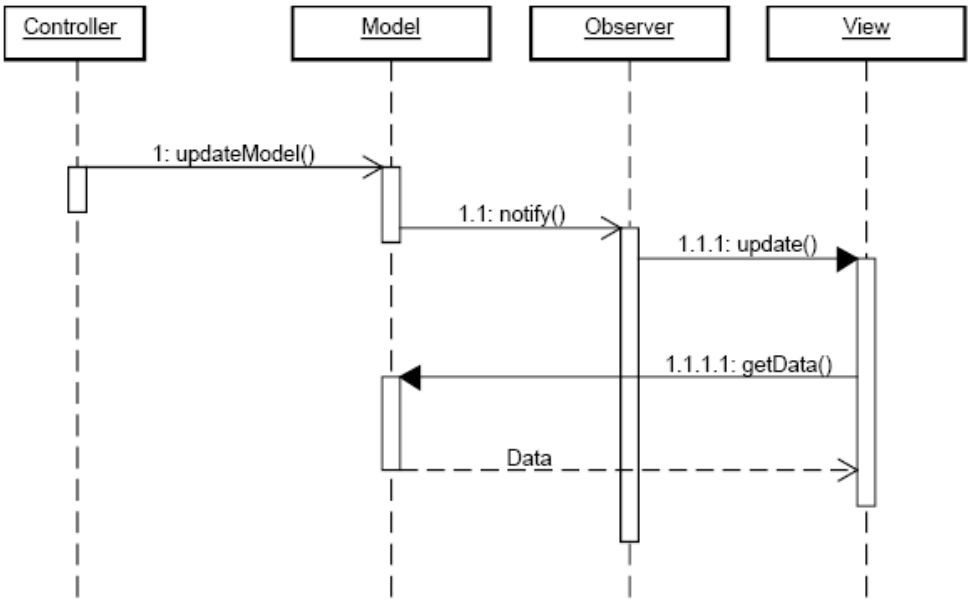

Abbildung 4.12: Sequenzdiagramm MVC-Entwurfsmuster mit aktivem Model

4.2.2 Programmstruktur

Aufgrund der eingenständigen, vom Controller unabhängigen, Manipulation der Daten innerhalb des Models durch die eingesetzte Physik-Engine, wird hier das MVC-Muster mit aktivem Model zugrunde gelegt. Es wird dem MVC-Muster noch die Observer-Komponente hinzugefügt.
Der Aufbau der Software ist an das eben beschriebene MVC-Muster lediglich angelehnt, es entspricht nicht komplett diesem klassischen MVC-Muster. Die Abweichung liegt darin, dass die Daten, die der Controller verarbeitet, nicht von der View stammen, sondern dem Controller vom Motiontracker übergeben werden. Die Informationen zur Manipulation der Daten werden somit unabhänig von der View ermittelt. Die View selbst zeigt die Daten nur an, kann aber nicht zu ihrer Manipulation via Controller beitragen.

Model Die im Model enthaltenen Anwendungsdaten beinhalten die realen Positionen der Handschuhe. Darüber hinaus befindet sich hier die Physik-Engine. Die Physik-Engine hält zusätzlich die Daten der sich in der Simulation befindlichen Objekte vor, wie die Aktoren und den Block. Diese Daten umfassen die konkreten physikalischen Eigenschaften und die Position und Lage im Raum. Die Physik-Engine manipuliert selbstständig diese Daten, die Teil des Models sind. Nach jedem Rechenzyklus der Physik-Engine findet hier auf Grundlage der Daten die Gestenerkennung statt.

View Die View-Komponente ist physikalisch zweigeteilt, ein kleiner Teil befindet sich in dem System, das die Simulation ausführt. Dieser Teil entnimmt dem Model die für die Darstellung relevanten Daten und sendet sie zur eigentlichen View-Komponente, der Powerwall. Die Powerwall dient lediglich der Präsentation der Daten, das heißt über die Powerwall selbst finden keine Benutzereingaben statt. Mit dieser Einschränkung wird vom eigentlichen MVC-Muster abgewichen.

Controller Als Benutzereingaben dienen die Daten des ARTtrackers. Der ARTtracker ermittelt die Positionen der Marker auf den Handschuhen und überträgt diese an den Controller. Im Controller werden diese Daten aufbereitet, die entstandenen Punktwolken von den Markern werden zu einem Objekt zusammengefasst und dieses dann an die Simulation übergeben.

Observer Der Observer prüft das Model auf Veränderung. Sind Veränderungen aufgetreten, werden alle für die View relevanten Daten der einzelnen Entitäten, wie Position und grafische Darstellung dem Model entnommen und in diesem Fall der View gesendet.

Bei der Entwicklung der Software wurde speziell darauf geachtet, die einzelnen Teilgebiete der Software modular und möglichst mit loser Kopplung zu errichten. Das Modul der Punktaufbereitung, um ein Beispiel zu nennen, arbeitet unabhänig vom Datenempfänger, der den Datenstrom vom ARTtracker ausliest. Auch werden die Daten nach der Aufbereitung unabhängig von ihrer späteren Verwendung in der Physik-Engine vorgehalten. Die spezifischen Entitäten der Physik-Engine werden lediglich innerhalb der Engine selbst benutzt. Dies soll einen weitestgehend einfachen Austausch der Physik-Engine ermöglichen, wie auch eine Erweiterung für andere Eingabeverfahren sicherstellen. Der komponentenbasierte Aufbau der Software ist in Abbildung 4.13 dargestellt.
Auch die Trennung zwischen Physiksimulation und der grafischen Darstellung ermöglicht hier einen Austausch, bzw. eine anderweitige Verwendung dieser Module. Eine detailliertere Erläuterung über die Kommunikation folgt in Abschnitt 4.2.3. Der Ablauf der Verarbeitung innerhalb der Simulation soll unter Verwendung eines Sequenzdiagramms in Abbildung 4.14 veranschaulicht werden.

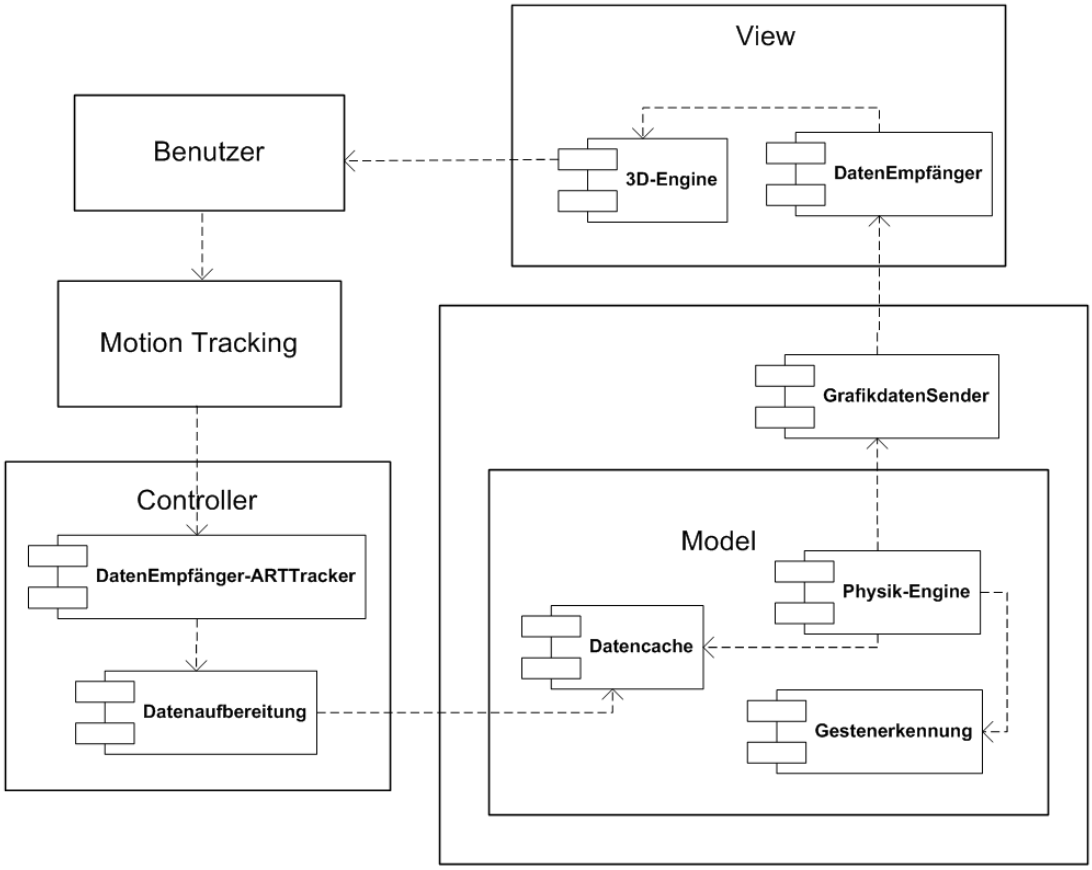

Abbildung 4.13: Komponentendiagramm

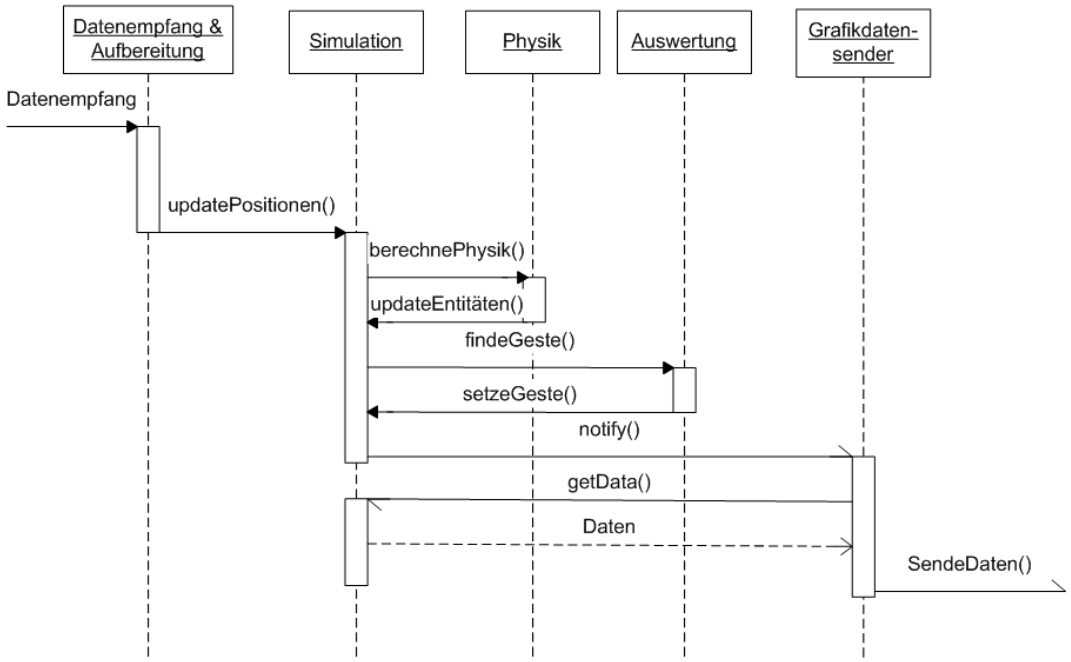

Abbildung 4.14: Sequenzdiagramm

4.2.3 Kommunikation

Wie in Abbildung 4.13 zu sehen ist, besteht das System aus drei getrennten Teilsystemen. Es handelt sich hier also um ein verteiltes System. Die Kommunikation vom ARTtracker erfolgt über ein Netzwerk, siehe (2.2.1) für weitere Informationen. Auch für die Kommunikation zwischen Physiksimulation, dem Model und der grafischen Ausgabe, der View, erfolgt über das Netzwerk.
Die Kommunikation ist in beiden Fällen einseitig. Der Sender sendet ausschließlich Daten, der Empfänger empfängt nur Daten. Für die Art des Datensendens gibt es zwei grundlegende Möglichkeiten, die Push- und die Pull-Variante. Bei der Push-Variante werden die Daten ohne Aufforderung vom Sender versandt. Bei der Pull-Variante meldet sich der Empfänger beim Sender und fordert die Daten an. Da vom ARTtracker die Daten kontinuierlich ohne vorherige Anfrage ins Netz gestellt werden, liegt hier die Pushvariante vor.
Auch für die Umsetzung der Kommunikation zwischen Model und View wird dieses Verfahren gewählt. Grund hierfür ist wie auch in Abschnitt 4.2.2 beschrieben die möglichst lose Kopplung. Ein weiterer Vorteil dieser Wahl ist die Möglichkeit der View ohne viel Aufwand die Daten des Moitiontrackers zusätzlich zu denen des Models zu empfangen und auch anzu-

zeigen. Dies kommt vor allem bei der Entwicklung und Erprobung des Systems zum Tragen. Der einzige Unterschied zur Variante des ARTtrackers besteht darin, dass vom Model nur dann Daten versendet werden, wenn sich etwas in der Physik-Engine geändert hat.
Es werden jeweils UDP-Pakete versandt. Um die Daten abzugreifen, muss man nur die passende UDP IP-Adresse abhören (näheres in Kapitel 2.2.1). Das Eingabegerät und das Model sollten jeweils auf unterschiedlichen IP-Adressen senden. Die Formatierung der Daten kann für die Pakete des ARTtracker aus dem technischen Anhang der D-Track Software [DTrack-Anhang (2007)] entnommen werden. Im Anhang A befindet sich eine Darstellung des Datenformats für die Kommunikation zwischen Model und View. Beide Formate sind, vom Menschen lesbar, im ASCII-Zeichensatz codiert.

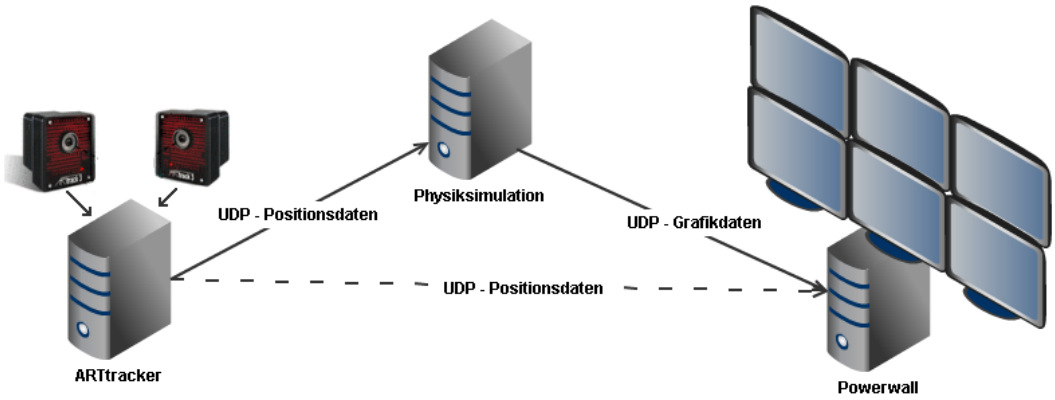

Abbildung 4.15: Schematischer Aufbau, Teile der Grafik stammen von [A.R.T-GmbH2011]

4.3 Zusammenfassung

Im ersten Abschnitt des Kapitels wurde genauer auf die Funktionalität eingegangen. Es wurden verschiedene Möglichkeiten zur Aufarbeitung der empfangenen Daten vorgestellt. Der Bereich der Simulation wurde erläutert, besonderes Augenmerk lag hier auf der Physik-Simulation und der daran anschließenden Auswertung der Zustandsänderungen der Objekte innerhalb dieser Simulation. Darauf aufbauend wurden die sich ergebenden Möglichkeiten der Gestenerkennung vorgestellt. Es folgte eine Beschreibung der Trajektorien als weiterführendes Verfahren zur Erkennung komplexer Bewegungsabläufe und daraus resultierender Gesten.
Der zweite Teil dieses Kapitel widmete sich der Architektur des Systems. Es wurde das MVC-Entwurfsmuster vorgestellt. Das MVC-Muster mit aktivem Model (4.11) diente als Grundlage für die Architektur. Es wurde unter Verwendung eines Sequenz- sowie eines Komponentendiagramms der Aufbau und die Funktionsweise der Software beschrieben. Am Ende wurde der Blick noch auf die Kommunikation zwischen den einzelnen Komponenten der Software gerichtet, da diese verteilt auf mehreren getrennten Computern ausgeführt wird.

Kapitel 5

Realisierung

Kapitel 5 befasst sich mit der Realisierung der Software unter Berücksichtigung der im vorherigen Kapitel ermittelten Anforderungen. Die Umsetzung der verschiedenen Module wird dort aufgeführt. Es werden verschiedene Physik-Engine-Frameworks vorgestellt, die zu Anfang für die Umsetzung infrage kamen und mit denen die Realisierung versucht wurde. Die Brauchbarkeit dieser Physik-Engines wird anhand der für die Umsetzung benötigten Funktionen bewertet. Anschließend wird die Wahl der Physik-Engine, die am Ende genutzt wurde, erläutert.
Außerdem werden einige Entwicklungsetappen der Software aufgeführt. Bei einer kurzen Evaluation wird geschaut, inwieweit die Software die in Kapitel 3 vorgestellten Gesten erkennen kann.
Zum Schluss des Kapitels findet eine Bewertung der technischen Umsetzung statt und es wird ein Ausblick gewährt, welche Erweiterungen noch denkbar wären.

5.1 Realisierung der Software

In dem folgenden Unterkapitel ist die Umsetzung der in (4.1) aufgeführten Verfahren erläutert. Es wird kurz beschrieben, welche Verfahren implimentiert werden und zum Einsatz kommen.

5.1.1 Datenaufbereitung und Auswertung

Für das Motion-Tracking kommt der ARTtracker zum Einsatz. Die empfangenen Daten vom ARTtracker werden aufbereitet. Eine Kombination aus räumlicher Begrenzung (4.1.3.1), der Hinzunahme von Punktdaten aus dem vorherigen Frame (4.1.3.2) und die Begrenzung der Wolke auf den maximalen Abstand der Finger (4.1.3.2) wird verwendet.
Aus den so entstandenen Punktwolken wird der Schwerpunkt ermittelt, wie in (4.1.3.2) beschrieben. Diese Schwerpunkte sind die neuen Positionen für die Aktoren. Die Positionsda-

ten der zwei Aktoren liegen nun beim nächsten Durchlauf der Simulation bereit, um in deren Berechnung mit einzufließen.

5.1.2 Positionsübertragung in der Physiksimulation

Die Positionen des Aktors werden unter Verwendung von Joints in die Physiksimulation übertragen. Die eigentliche Position der Hand ist somit innerhalb des Würfels, die des Aktors außerhalb. Wie in (4.1.4) beschrieben, wirkt somit eine Kraft auf den Block. Wie in Abbildung 5.1b und 5.1c sichtbar, wird die auftretende Kraft hier als gelber Pfeil[10] dargestellt. Beim Einsatz von nur einem Aktor führt die Kraft dazu, dass der Block über den Table geschoben wird, unter der Bedingung, dass diese Kraft größer als die Reibungskraft zwischen Block und Table ist.
Liegen wiederum links und rechts je ein Aktor an, wird der Block gegriffen. Bei ausreichender Kraftausübung kann der Block mit den Aktoren angehoben werden, wenn die so entstehenden Reibungskräfte zwischen dem Block und den Aktoren größer werden als die Schwerkraft, die auf den Block selbst wirkt.

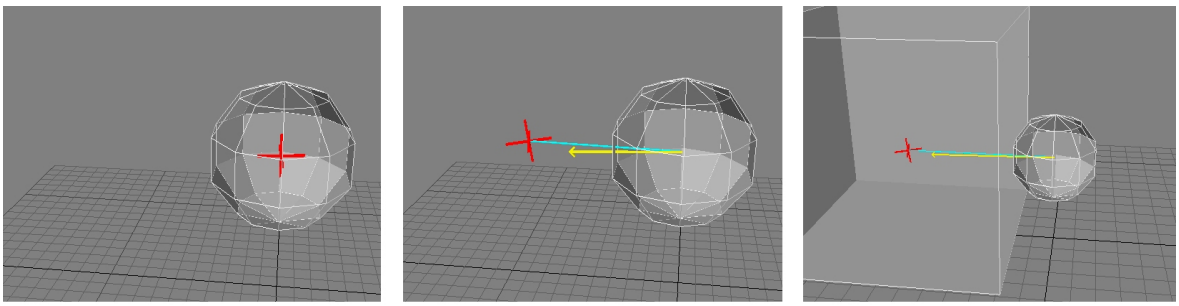

Abbildung 5.1: Positionsübertragung in die Physiksimulation

5.1.3 Gesten

Für die softwaretechnische Umsetzung wird die Anzahl der Gesten, die erkannt werden sollen, eingeschränkt. Die folgende Umsetzung wird nur die vier bereits in Kapitel 3 vorgestellten und näher erläuterten Gesten umfassen. Es handelt sich dabei um die Schiebe-, Greif-, Hebe- und Loslassgeste.
Die am einfachsten umzusetzende Geste ist die Schiebegeste, sie wird als Erstes realisiert. Als zweite Geste wird die Greifgeste implementiert, da sie Voraussetzung für die Hebegeste

[10] Die ODE Physik-Engine ermöglicht zur Fehlersuche die auftretenden Kräfte innerhalb der Physik mittels Pfeilen darzustellen. Je länger ein Pfeil ist, desto größer ist die wirkende Kraft.

ist. Anschließend wird die Hebegeste realisiert. Die Loslassgeste kann auf die Greif- wie auf die Hebegeste folgen und steht somit am Ende der Umsetzung.

Realisierung

Um die vier genannten Gesten zu erkennen, werden einige in (4.1.5) aufgeführten Verfahren nicht benötigt. Auf eine Implementierung, die eine mögliche Rotation des Assemblys (4.1.5) einbezieht, wird verzichtet. Darüber hinaus wird keine Gestenerkennung unter Berücksichtigung von Trajektorien (4.1.7) vorgenommen.

Schieben Das Assembly der Schiebegeste besteht aus dem Block und einem Aktor. Ausgangspunkt ist die Kollision des Blocks mit dem Table. Kommt jetzt ein Aktor hinzu und erzeugt eine Kollision mit dem Block, wird das Assembly gebildet. Das Assembly berührt weiterhin noch den Table. Durch die Bewegung des Aktors in Richtung des Blocks wird das Assembly geschoben. Genau dieses Schieben wird vom System als Schiebegeste gewertet.

Greifen Das Assembly für die Greifgeste besteht aus dem Block und zwei Aktoren. Als Ausgangslage liegt der Block wieder auf dem Table. Jeweils von zwei sich gegenüberliegenden Seiten kommt nun je ein Aktor dazu und diese erzeugen somit eine Kollision mit dem Block. Die Erzeugung dieses Assemblys wird als Greifen gewertet.

Heben Das für das Heben benötigte Assembly ist das gleiche Assembly wie beim Greifen, es besteht aus zwei Aktoren und dem Block. Eine Kollision des Assemblys mit dem Table muss nicht vorliegen. Mithilfe der Aktoren kann nun das Assembly bewegt werden. Wenn sich dann die Höhe des Assemblys ändert, wird das als Hebegeste ausgegeben.

Loslassen Ausgangsposition für die Loslassgeste ist das Assembly gleich der Greif- oder Hebegeste. Der Block wird von zwei Aktoren berührt. Beendet nun einer oder beide Aktoren die Kollision mit dem Block, wird dies als Loslassen gewertet. Berührt der Block nicht den Table, weil er angehoben war, fällt er auf diesen zurück.

Nr.	Geste	Beteiligte Objekte	Mögliche nächste Gesten	Vorherige Geste
1	Schieben	Table, Block, Aktor	Greifen	-
2	Greifen	Table, Block, zwei Aktoren	Loslassen, Heben	-
3	Heben	Block, zwei Aktoren	Loslassen	Greifen
4	Loslassen	Block, zwei Aktoren	Greifen, Heben	Greifen, Heben

Tabelle 5.1: Erkennung der Gesten

Zustandsauswertung

Da bei der ersten Umsetzung auf Assemblys verzichtet wurde, bleiben für die Auswertung erst einmal nur die Entscheidungsmatrix (4.1.6) oder das Verfahren mit dem Automaten (4.1.6). Es wurde sich für das Verfahren der Entscheidungsmatrix entschieden, da es einfacher darstellbar ist und auch schneller erweiterbar scheint, sollten neue Gesten hinzukommen.
Die hier zu erkennenden Gesten werden bei Zustandsübergängen wahrgenommen. Um dies über eine Entscheidungsmatrix abzubilden, werden die Zustände vor und nach einem Berechnungsdurchlauf der Physik-Engine betrachtet. Die Tabelle 5.2 zeigt die Entscheidungsmatrix für die vier Gesten. Die Felder in der Matrix können drei Werte annehmen. Es gibt „ja", „nein" und leere Felder. Zustände, die zwingend erfüllt sein müssen für eine Geste, werden mit 'ja' beschrieben, Felder die zwingend nicht erfüllt sein dürfen mit 'nein' und die Zustände, bei denen es unerheblich ist, ob sie zutreffen oder nicht, werden leer gelassen.

Zustände vor Berechnung	Schieben	Greifen	Heben	Loslassen
Kollision Block - Table	ja			
Kollision Block - ein Aktor			nein	nein
Kollision Block - zwei Aktoren	nein	nein	ja	ja
Zustände nach Berechnung				
Translation Block	ja		ja	
Kollision Block - Table	ja		nein	
Kollision Block - ein Aktor	ja	nein	nein	
Kollision Block - zwei Aktoren	nein	ja	ja	nein

Tabelle 5.2: Entscheidungsmatrix - Gestenerkennung

5.1.4 Wahl der Physik-Engine

Unter dem Blickwinkel einer möglichst plattformunabhänigen Implementierung wurde sich frühzeitig für die Programmiersprache Java[11] entschieden.
Es sollte ermittelt werden, inwieweit die für Java angebotenen Grafik- und Physik-Engines den Anforderungen genügen können.

Wahl der 3D-Engine Die modernen Java-Implementierungen für eine 3D-Engine basieren meist auf der Verwendung eines Wrappers, der die eigentliche, systemnähere Implementierung der 3D-Engine umhüllt und so die Funktionen für Java zugänglich macht. Einer dieser

[11] Java - www.oracle.com/technetwork/java/index.html

Wrapper ist LWJGL[12].
Eine solche Java 3D-Engine ist die JMonkey Engine[13]. Die JMonkey Engine verwendet unter anderem LWJGL zur Grafikdarstellung, es sollen aber auch andere Wrapper möglich sein.
JMonkey Engine liefert bereits mehrere verschiedene Anbindungen an Physik-Engines mit. Aufgrund dieser Tatsache und der umfangreichen mitgelieferten, sowohl rein grafischen wie auch mit Physik-Anbindung versehenen Beispiel-Implementierungen (siehe [JMonkeyTutorials]) wurde die JMonkey Engine zur Grafikdarstellung ausgewählt.

Wahl der Physik-Engine Die JMonkey Engine bietet in der Version 2.0.1 mit dem 'jME Physics 2' Package die Möglichkeit via JBullet [JBullet] die Bullet Physik-Engine [BulletPhysics] oder aber via ODEJava [ODE4J] die ODE Physik-Engine [ODE] einzubinden. Darüber hinaus gibt es mit dem JME Package noch eine weitere Anbindungsweise, um JBullet und somit die Bullet Physik-Engine zu benutzen.
Da keines dieser Projekte über eine aussagekräftige oder hinreichende Dokumentation verfügt, kann zu diesem Zeitpunkt hinsichtlich der in (4.1.4) aufgestellten Anforderungen keine qualifizierte Aussage getroffen werden. Die Punkte sind im Folgenden noch einmal aufgeführt.

- Kollisionserkennung
- Masse
- Schwerkraft
- Joints
- Reibung
- Entität ohne Körper

Alle zur Verfügung stehenden Physik-Engine Varianten, die JMonkey Engine unterstützt, werden auf den folgenden Seiten auf diese sechs Punkte hin überprüft, indem versucht wird, die Umsetzung der Anwendung damit zu verwirklichen.
Die kleinen mitgelieferten Beispielprogramme der jeweiligen Implementierungen, die einem Neueinsteiger eine Hilfe sein sollen, bilden hier den Ansatzpunkt für die Entwicklung.

[12] OpenGL Wrapper: LWJGL - www.lwjgl.org
[13] JMonkey Engine: www.jmonkeyengine.com & www.jmonkeyengine.org

5.1.4.1 jME Physics 2

Dies ist das ältere der beiden Projekte, die Weiterentwicklung wurde 2008 eingestellt. Ziel des Projekts war eine Entkopplung zwischen der Grafik-Engine und der verwendeten Physik-Engine. Vor dem Start einer Anwendung kann man festlegen, ob ODE oder Bullet zum Einsatz kommen soll.
Im Gegensatz zu JME gibt es hier zwei unterschiedliche Gruppen von Physikobjekten, statische und dynamische Objekte.
Dynamische Objekte sind der Standard. Sie haben ihre Masse, die Ausdehung, interagieren mit anderen physikalischen Objekten innerhalb der Simulation und sind auch im Normalfall der Schwerkraft unterworfen.
Statische Objekte hingegen werden nicht von physikalischen Kräften beeinflusst. Sie sind weder der Schwerkraft unterworfen noch lassen sie sich durch andere physikalische Kräfte, die auf sie einwirken, bewegen. Sie eignen sich dazu, eine Umwelt zu erschaffen, in der dynamische Physikobjekte dann miteinander interagieren können.
Die Möglichkeit auf eine weitere Engine auszuweichen und die wesentlich zahlreicheren Beispielanwendungen gaben den Ausschlag dafür, zuerst mit jME Physics 2 den Versuchsaufbau durchzuführen.

ODE unter jME Physics 2 Für ODE müssen plattformabhängige Binaries eingebunden werden (siehe [ODE-Manual]), was die gewünschte Plattformunabhängigkeit einschränkt. Es liegen Binaries für Linux, Linux 64-Bit, MacOs X und Windows vor, allerdings fehlt für die hier eingebunde ODE Version die Windows 64 Bit Version. Das Einbinden einer neueren ODE Version mit entsprechenden Binaries für Windows 64 Bit führte beim Start von jME Physics 2 zu einem Fehler. Die Physik-Simulation wäre somit nicht direkt auf der Powerwall (4.5) ausführbar.
Die Tests der unterschiedlichen benötigten Funktionen der Engine verliefen weitestgehend erfolgreich, lediglich die Kollisionserkennung arbeitet nicht immer zuverlässig. Wirkt genügend Kraft auf einen physikalischen Körper ein, kann er einen anderen Körper durchdringen.

Bullet unter jME Physics 2 Die zweite Engine, die man mittels jME Physics 2 einbinden kann, ist Bullet unter Zuhilfenahme von JBullet. Nach ersten Versuchen stellte sich heraus, dass die Kollisionserkennung zwischen dynamischen Physikobjekten nicht funktioniert. Die betreffenden Funktionen wurden im Quellcode der Engine auskommentiert. Die Versuche mit Bullet unter jME Physics 2 wurden daraufhin beendet.

5.1.4.2 Bullet mit JME

JME ist der dritte Anlauf für JMonkey eine Anbindung an eine Physik-Engine zu implementieren. Das Projekt befindet sich aber noch auf einem recht frühen Entwicklungsstand.

Auf den ersten Blick funktioniert die Kollisionserkennung zuverlässig. Bei jedem Rechendurchlauf werden die Kollisionen neu berechnet und können ausgegeben werden. Es stellte sich aber heraus, dass eine Kollisionsmeldung durch zwei Objekte nur dann ausgeliefert wird, wenn sich mindestens eins der Objekte bewegt. Sobald sich beide Objekte nicht mehr bewegen, wird keine Kollisionsmeldung zur Weiterverarbeitung zurückgeliefert. Erkannt wird die Kollision intern wohl dennoch, da die Objekte sich nicht durchdringen können. Folglich ist eine zuverlässige Auswertung anhand der Kollision, wie benötigt, nicht mehr gegeben.
Die Joints (4.1.4) funktionieren nicht konstant. Die Manipulation des Blocks, wie in Abbildung 5.1 zu sehen ist, setzt ohne erkennbare Fehlermeldung aus.
Darüber hinaus verfügt JME über keine Unterstützung der Materialeigenschaften. Die Simulation von Reibung ist somit nicht möglich und es müsste das Anheben des Blocks durch die Aktoren anderweitig gelöst werden.

In den neueren Releaseversionen von JMonkey ist JME die einzige noch angebotene Variante zur Einbindung einer Physik-Engine.

5.1.4.3 Fazit Physik-Engine-Wahl

Die wichtigsten Punkte der einzelen Varianten sind in Tabelle (5.3) zusammengefasst und gegenübergestellt. Wie man gut erkennen kann, ist die Variante mit ODE jME Physics 2 (5.1.4.1) unter den getesteten die brauchbarste Lösung.

	JBullet JME	JBullet jME Physics 2	ODE jME Physics 2
Kollisionserkennung	bedingt	nein	bedingt
Masse	ja	ja	ja
Schwerkraft	ja	ja	ja
Joints	unzuverlässig	-	ja
Reibung	nein	-	ja
Entität ohne Körper	nein	-	ja

Tabelle 5.3: Übersicht Funktionsumfang

5.1.5 Entwicklungsetappen

Bei der Vorstellung einzelner Entwicklungsetappen liegt das Augenmerk auf der Physiksimulation, da die Umsetzung dort die meisten Änderungen und Entwicklungsvorgänge besaß. Bei der Entwicklung stellte sich recht schnell heraus, dass die Physik-Engine doch nicht so zuverlässig arbeitet wie gedacht. Vor allem die Kollisionserkennung versagte häufig bei zu viel Kraftübertragung durch den Joint an den Aktor (5.1). Der eigentliche Aktor wurde als Kugel beschrieben. Es stellte sich aber als praktikabler heraus, wenn man bei der Umsetzung in

der Physik-Engine eine Kugel mit relativ wenig Flächen verwendete. Wie in Abbildung 5.2 zu sehen, wurde die Anzahl der Flächen halbiert. Die einzelnen Flächen werden somit größer, was eine größere Reibungsfläche zwischen Aktor und Block zur Folge hatte.

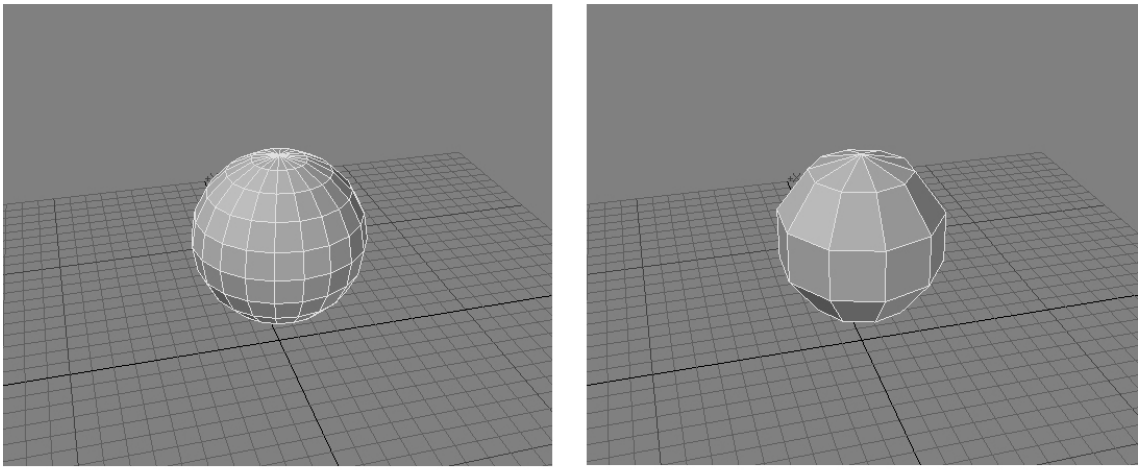

Abbildung 5.2: Entwicklungsetappen Aktoren

Ein weiteres Entwicklungsfeld betraf den sogenannten Table, die Fläche, auf dem der Block liegt. Anfänglich war der Table eine einfache Platte, die als starres Objekt in der Simulation vorhanden war und nicht von der Schwerkraft nach unten gezogen wurde.
Bereits bei den ersten Versuchen mit dem System stellte sich heraus, dass eine Platte allein etwas unhandlich war, da sie dem Block keine Grenzen bot. Der Block rutschte über den Table und fiel von ihm herunter. Auch das Ausweiten des Tables, um ein Runterfallen zu verhindern, half nur wenig, weil der Block dann so weit rutschte, dass man ihn mit den Aktoren aufgrund des begrenzten Kamerablickfeldes und auch des begrenzten Messbereichs des ARTtrackers nicht wieder erreichen konnte. Das Begrenzen des Tables und somit der Aktionsfläche für den Block wurde notwendig.
Aufgrund von fehlenden Möglichkeiten, Wände für den Block als undurchdringlich, aber für die Aktoren als passierbar zu kennzeichnen, wurde im ersten Schritt eine Umrandung der Fläche, wie in Abbildung 5.2a erkennbar hinzugefügt. Für diese Umrandung musste eine Höhe gewählt werden, die den Block stoppte, aber auch noch den Aktoren ermöglichte den Block wieder vom Rand wegzuschieben.
Für die Realisierung der Hebegeste war eine Umrandung zur Begrenzung selbst nicht ausreichend. Es wird eine Art Käfig aus Gitterstäben an die Ränder des Tables gesetzt. Der Abstand der Gitterstäbe ist groß genug, sodass ohne größere Probleme die Aktoren passieren können, der Block aber selbst den Table nicht verlassen kann. Die Höhe der Stäbe ist variable einstellbar, die Höhe geht im Aufbau bis an den Rand des Messbereichs des Moti-

ontrackers. Das Gebilde wird nach oben hin auch durch eine Platte, gleich der Bodenplatte abgedeckt, damit der Block auch hier nicht den Bereich verlassen kann. In Abbildung 5.3b wurde auf die Abdeckung verzichtet um einen besseren Blick auf den Aufbau zu ermöglichen.

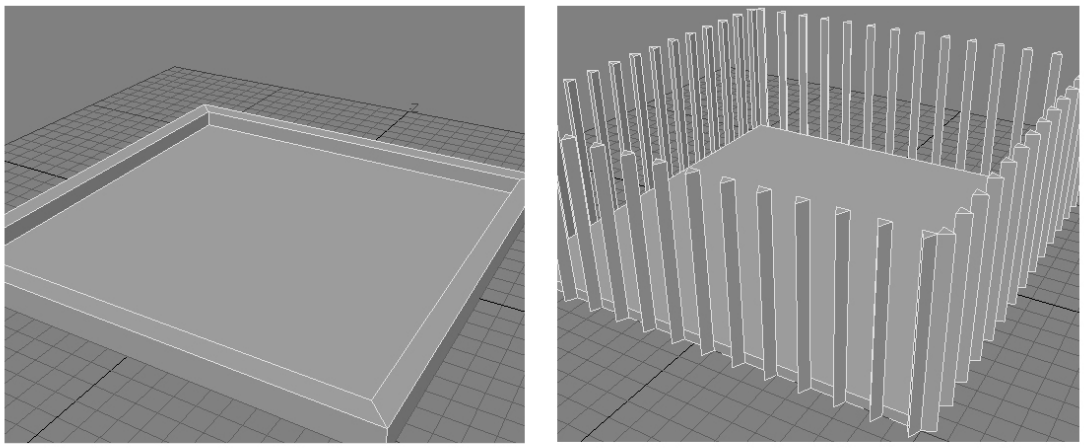

Abbildung 5.3: Entwicklungsetappen Table

5.1.6 Evaluation

In der Evaluation werden erste Tests mit der Software vollzogen und die vier Gesten werden nacheinander ausprobiert.
Gestartet wird zunächst mit einem Aktor. In der Abbildung B.1 kann man auf der Powerwall den Block und einen Aktor erkennen. Rechts im Bild ist der Handschuh, der vom ARTtracker erfasst wird, zu sehen.
In Abbildung B.2 berührt der Aktor den Block von der rechten Seite. Nun wird die Hand weiter nach links bewegt, was dazu führt, dass der Aktor den Block auch nach links schiebt. Das System erkennt die Schiebegeste (3.3.1), zu sehen in Abbildung B.3.
Es wird jetzt auch der zweite Handschuh hinzugenommen, ein zweiter Aktor erscheint in der Simulation. Mit der linken Hand wird nun der Block wieder in die Mitte geschoben.
Als nächstes wird die Greifgeste erprobt. Dazu wird der Block links und rechts jeweils von einem Aktor berührt, wie in Abbildung B.4 zu sehen ist, wird die Greifgeste (3.3.2) erkannt.
Im Anschluss werden die Hände noch ein Stück dichter zueinander bewegt, die Reibung, die so zwischen Block und den Aktoren entsteht, reicht aus, um beim Heben der Aktoren den Block mit anzuheben. Das System ermittelt die Hebegeste (3.3.3). Das Ergebnis ist in B.5 abgebildet.
Zum Schluss wird der Abstand der Hände weiter vergrößert, die Berührungen der Aktoren

mit dem Block enden. Der Block fällt zu Boden, das System erkennt die Loslassgeste (3.3.4) (siehe Abbildung B.6).
Das System konnte exemplarisch alle vier Gesten erkennen.

5.2 Fazit

Es konnten folgende Punkte in der Software umgesetzt werden:

1. Auswertung und Aufarbeitung der Daten vom ARTtracker
2. Umsetzung einer physikbasierten Simulationsumgebung
3. Erkennen der vier in Kapitel 3 aufgeführten Gesten
4. Ausgabe der Simulation, wie auch der erkannten Gesten auf der Powerwall

Physik-Engine Die Ergebnisse der Physik-Engine sind häufig nicht zufriedenstellend. Das Verhalten der Objekte innerhalb der Simulation ist nicht immer nachvollziehbar. Der Verlust an Kontrolle, Funktionalität und Fehlererkennung durch das mehrfache Wrappen der eigentlichen Physik-Engine ist erheblich. Nicht zuletzt, weil die Weiterentwicklung des verwendeten Frameworks JM2 eingestellt wurde, muss über einen Umstieg auf eine andere Engine nachgedacht werden.
Da dies wohl aber nicht die Probleme mit dem nur indirekten Zugriff auf die Physik-Engines löst, empfiehlt sich auch gleichzeitig ein Umstieg auf eine Programmiersprache, in der selbst Physik-Engines geschrieben wurden, um so einen direkteren Zugriff auf diese zu erhalten.

5.2.1 Erweiterungen des Systems

Im Laufe der Arbeit haben sich verschiedene Punkte ergeben, in denen eine Weiterentwicklung des Systems denkbar wäre.

Erweiterung des Eingabegerätes

Das Tracking der sich dynamisch zueinander verhaltenden Marker der Handschuhe ermöglicht nicht immer eine zufriedenstellende Repräsentation in der Simulation. Besonders die Lage der Hand im Raum kann so nicht ermittelt werden. Versuche mit dem von der A.R.T. GmbH entwickelten Handschuh (2.3) mit starren Komponenten wären sicherlich eine Möglichkeit, die Eingabe zu verbessern. Stehen erst einmal genauere Eingabedaten zur Verfügung, kann von der Form einer Kugel als Repräsentant des Aktors in der Simulation Abstand

genommen werden. Es könnte hier eine wesentlich handähnlichere Repräsentation stattfinden. Als langfristiges Ziel könnte man hier das Greifen mittels einer Hand setzen, wenn der Block mithilfe des Daumens und der Finger gegriffen wird.
Eine weitere Möglichkeit das Eingabegerätes zu erweitern, wäre die Verwendung einer Feedback-Komponente. Diese weitläufig als Force-Feedback beschriebene Komponente wäre eine zusätzliche Möglichkeit der Ausgabe vom System. Der Benutzer würde nicht nur via Bildschirm das Berühren des Blocks sehen, sondern könnte dieses auch durch einen entsprechenden Vibrationsimpuls spüren.

Unterstützung weiterer Eingabeverfahren

Wie im Abschnitt 5.1.1 beschrieben, wurde die Umsetzung der Eingabe durch den ARTtracker realisiert.
Es wäre durchaus denkbar, dass auch Time of Flight Kameras (2.2.2) oder insbesondere auch die Kinect (2.2.3) als weitere Möglichkeit in Betracht gezogen werden, um die Aufgabe zu erfüllen.

Erweiterung der Simulation

Um dem Benutzer eine möglichst sinnbildliche Interaktion mit dem Block zu ermöglichen, gibt es außerdem die Möglichkeit die Form des Blocks bei Berührung passend zu verändern. Der Block ließe sich so durch die Aktoren beispielsweise zusammendrücken (siehe Abbildung 5.4).
Diese Erweiterung lässt sich in zwei Teile unterscheiden.
Bei der ersten Variante der Umsetzung würde sich die Deformationen des Blocks rein auf die grafische Darstellung beschränken. Einige 3D-Engines bieten eine Deformation von Objekten mittels Bones an. Die physikalische Repräsentation bliebe hierbei unberührt.
Die zweite Variante dieser Umsetzung ist eine Deformation, die durch die Physik-Engine stattfindet. Hier bedarf es einer Physik-Engine, die das Deformieren von Objekten unterstützt. Jegliche Deformation, die durch die Physik-Engine errechnet wird, muss dann an die entsprechende grafische Entität weitergereicht werden.

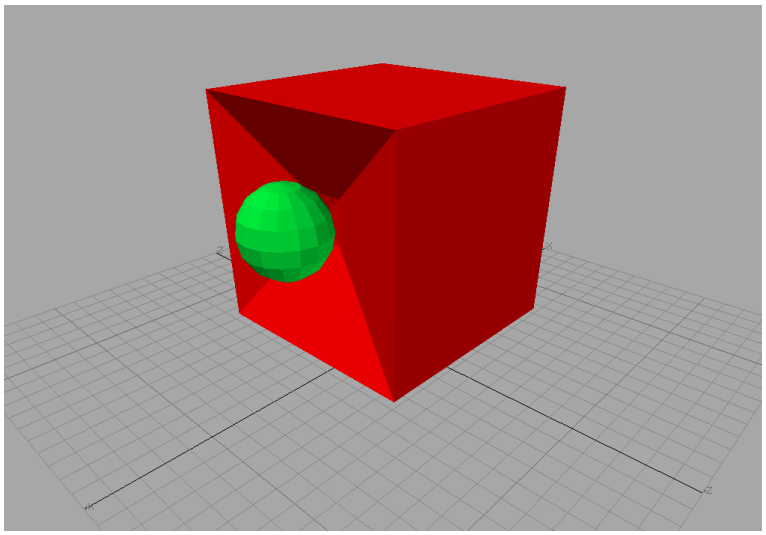

Abbildung 5.4: Deformation des Blocks durch einen Aktor

5.3 Zusammenfassung

Dieses Kapitel befasste sich mit der Realisierung der Software. Es wurde beschrieben, mit welchen Verfahren aus Kapitel 4.1 die einzelnen Komponenten umgesetzt wurden. Ein Schwerpunkt dieses Kapitels lag auf der Wahl der Physik-Engine. Es wurden die Physik-Engines und die dazugehörigen Wrapper vorgestellt. Die Ergebnisse der Versuche mit diesen Engines wurden gegenübergestellt. Aus den Ergebnissen wurde die daraus als brauchbarste Lösung resultierende Physik-Engine ausgewählt, um in der endgültigen Umsetzung verwendet zu werden.
Danach folgte die Beschreibung einiger Entwicklungsetappen und daraufhin wurde die entwickelte Software evaluiert. Der Ablauf einer tatsächlichen Gestenerkennung wurde festgehalten. Das System konnte alle vier geforderten Gesten ermitteln.
Im zweiten Teil dieses Kapitels wurde ein Fazit über die technische Umsetzung gezogen. Einzelne Umsetzungspunkte wurden bewertet. Es wird sich für einen Umstieg auf eine andere Physik-Engine, verbunden mit dem Wechsel zu einer systemnäheren Programmiersprache ausgesprochen.
Im Anschluss wurden noch einige Erweiterungsmöglichkeiten für das System aufgeführt, die sowohl die Eingabegeräte wie auch die Simulation betreffen.

Kapitel 6

Schluss

6.1 Zusammenfassung

Ziel dieser Arbeit war die Entwicklung eines Systems zur Gestenerkennung im Raum unter Verwendung einer Physiksimulation.
Im Zuge dieser Arbeit wurden einige andere Arten der Gestenerkennungen betrachtet und unterschiedliche Motiontracking- Verfahren vorgestellt. Außerdem erfolgte noch eine kurze Betrachtung der Aufgaben und Funktionsweisen von 3D- wie auch Physik-Engines.
Mit Bezugnahme auf Projekte im Rahmen der Media Facaden wurde ein Szenario entworfen, das als Grundlage für die zu entwerfende Gestenerkennung dienen sollte.
Anhand des Szenarios wurden Gesten ermittelt. Diese Gesten wurden genauer analysiert und beschrieben. Es wurden die funktionalen Anforderungen herausgefunden, die ein System erfüllen muss, um diese Gesten mittels einer räumlichen Simulation erkennen zu können.
Im ersten Teil des Designkapitels wurde auf die nötigen Funktionalitäten eingegangen. Mögliche Lösungen zur Aufarbeitung der vom Motion-Tracker gesandten Daten wurden erläutert.
Im Bereich der Simulation wurde vor allem die Physik-Simulation beschrieben, aber auch die Anforderungen an eine zum Einsatz kommende Physik-Engine formuliert. Von der Erfüllung dieser Anforderungen ausgehend, wurden theoretische Möglichkeiten der Gestenerkennung erläutert. Es wurde verdeutlicht, wie die Gesten selbst durch Zustandsänderungen der einzelnen Objekte in der Simulation zur Interpretation von Gesten herangezogen werden können.
Im Anschluss wurde die dem System zugrundeliegende Architektur entwickelt. Es handelt sich um ein verteiltes System, welches auf dem MVC-Entwurfsmuster basiert. Simulation und Präsentation sind physikalisch getrennt, die Kommunikation findet über einer Netzwerkverbindung statt, über die einfache UDP-Pakete im ASCII-Format versandt werden.
Im nächsten Abschnitt der Arbeit wurde auf die Realisierung eingegangen, das beinhaltet eine Beschreibung der Verfahren, die zur Umsetzung kamen. Da zu Anfang der Umsetzung

noch nicht klar war, welche der zur Verfügung stehenden Physik-Engines die praktikabelste Lösung bot, wurde der Aufbau der Simulation mit allen versucht. Anschließend erfolgte die Auswahl der brauchbarsten Engine einerseits anhand der vorher festgelegten Eigenschaften, die für eine Umsetzung nötig waren und andererseits aufgrund der Bewertung, wie gut diese Eigenschaften durch die verschiedenen Engines im praktischen Gebrauch erfüllt wurden.
Einige signifikante Entwicklungsetappen wurden aufgeführt, danach wurde das System evaluiert. Dieses System erfüllte die aufgestellten Anforderungen und die Gesten wurden erkannt. Am Ende wurde außerdem ein Fazit über die technische Umsetzung gezogen.
Den Abschluss der Realisierung bildeten verschiedene Erweiterungsmöglichkeiten sowohl für Eingabegeräte als auch für die Simulation selbst.

6.2 Fazit

Das hier entworfene Programm ist ein Beispiel für eine physikbasierte Anwendung zur Gestenerkennung mittels Manipulation im Raum. Die bei der Umsetzung aufgetretenen Grenzen liegen hier in den technischen Bereichen. Einmal ist hier das Trackingverfahren zu nennen, was die Benutzer-Interaktion in das System überführt und zum Zweiten die Physik-Engines, die zur Verfügung standen. Da Physik-Engines immer mehr zur Grundausstattung gehören, wenn es sich um Computerspiele handelt, werden diese auch stetig weiterentwickelt. Somit bestehen gute Chancen, dass die hier bei der Umsetzung aufgetretenen Probleme nicht von Dauer sein werden. Das Verfahren selbst, ein System zur physikbasierten Interpretation von Gesten im Raum, ist funktionstüchtig.

6.3 Ausblick

Durch die Weiterentwicklungen des Motiontracking und dort insbesondere die Entwicklungen, die auf Hilfsmittel verzichten können, wird sich ein immer breiteres Anwendungsspektrum für diese Form der Kommunikation und Steuerung von Computersystemen eröffnen.
Was heute schon im Bereich der Computerspiele nahezu alltäglich geworden ist, nämlich das Steuern durch einfache Gesten, wird sich auch in anderen Bereichen etablieren. Eine Form der controller- und berührungslosen Kommunikation mit Computersystemen ist vor allem im öffentlichen Raum denkbar. Diese Art von System ist überall dort vorstellbar, wo Kommunikationsmittel wie Touchscreens, aus Gründen des Schutzes vor Staub, Schmutz, zu schneller Abnutzung oder Vandalismus nicht praktikabel erscheinen.

Eingaben durch simulationsuntypische Controller im Bereich der virtuellen Realität können mithilfe solcher Kommunikationsformen vermieden oder zumindest verringert werden. Dies würde zu einer Steigerung des Grades der Immersion führen.
Großen Chancen zur Ausbreitung solcher Systeme bestehen auch im Umfeld der „Smart Environment" Entwicklung, da bei entsprechend großem Einzugsbereich des Motions Trackings die Interaktion mit dem System nicht mehr ortsgebunden sein muss. Die HAW Hamburg befasst sich seit 2009 mit der Entwicklung eines „Smart Home", dem sogenannten living place[14]. Hier wäre die Installation eines solchen Systems durchaus vorstellbar.
Lässt man einmal die Computersysteme, bei denen die Steuerung exakt und zeitnah erfolgen muss und bei denen eine missglückte Geste oder misslungene Gestenerkennung kritische Folgen hätte, außen vor, wäre der Einsatz dreidimensionaler Gestenerkennung zur Computersteuerung nahezu überall denkbar.
Auf diesem Gebiet sind in den kommenden Jahren noch viele Fortschritte zu erwarten.

[14] www.livingplace.org

Literaturverzeichnis

[Aarhus1] : *Aarhus by light, letzter Abruf: 7.12.2010*. – URL http://www.aarhusbylight.dk/images/aarhus-by-light-01.jpg

[Aarhus2] : *Aarhus by light, letzter Abruf: 7.12.2010*. – URL http://www.aarhusbylight.dk/images/aarhus-by-light-04.jpg

[A.R.T-GmbH2011] : *A.R.T-GMBH: Advanced Realtime Tracking GmbH., letzter Abruf: 10.04.2011*. – URL http://www.ar-tracking.de/

[BulletPhysics] : *BulletPhysics, letzter Abruf: 10.01.2010*. – URL www.bulletphysics.org

[GeoLab] : *Geometry Lab, letzter Abruf: 10.02.2011*. – URL http://www.geometrylab.de/RandomPolygon/index.html

[Kinect] : *install kinect for windows pc, letzter Abruf 18.4.2011*. – URL http://www.kinect-hacks.com/sites/kinect-hacks.com/files/install-kinect-for-windows-pc.png

[JBullet] : *JBullet, letzter Abruf: 12.01.2010*. – URL www.jbullet.advel.cz/

[JMonkeyTutorials] : *jMonkeyEngine.org, letzter Abruf: 5.1.2011*. – URL http://jmonkeyengine.org/wiki/doku.php/jme3#tutorials_for_beginners

[MediaFacadesFestival] : *Media Facades Festival, letzter Abruf: 7.12.2010*. – URL http://www.mediafacades.eu/node/206

[NvidiaPhysX] : *Nvidia PhysX, letzter Abruf: 19.04.2011*. – URL http://www.nvidia.de/object/nvidia_physx_de.html

[ODE-Manual] : *ODE-Manual, letzter Abruf: 20.02.2010*. – URL http://opende.sourceforge.net/wiki/index.php/Manual

[ODE4J] : *ode4j - A Java 3D Physics Engine and Library, letzter Abruf: 25.02.2011*. – URL `http://ode4j.sourceforge.net/demo_crash3.png`

[ODE] : *Open Dynamics Engine, letzter Abruf: 18.02.2010*. – URL `www.ode.org/`

[OpenNIUserGuide] : *OpenNI User Guide*. – URL `http://www.openni.org/images/stories/pdf/OpenNI_UserGuide.pdf`

[Automaten] : *Äquivalenz von Automaten, letzter Abruf: 03.03.2011*. – URL `http://www.iris.uni-stuttgart.de/lehre/eggenberger/ksn/13_Reduktion/Reduktion.htm`

[Smslingshot] : *SMSlingshot, letzter Abruf: 8.12.2010*. – URL `http://www.vrurban.org/smslingshot.html`

[zwille2010] : *SMSLINGSHOT, letzter Abruf 8.12.2010*. – URL `http://www.vrurban.org/res/doku_zwille_juli_2010_low.pdf`

[vrurbanSchema] : *VR URBAN Schema, letzter Abruf 9.12.2010*. – URL `http://www.vrurban.org/res/smsl_schema.jpg`

[vrurbanZwille] : *VR URBAN Zwille, letzer Abruf 9.12.2010*. – URL `http://www.vrurban.org/res/zwilli.jpg`

[spreadgun2010] : *VR/URBAN reclaim the screens, letzter Abruf 8.12.2010*. – URL `http://www.vrurban.org/spreadgun.html`

[XBox360] : *XBox 360 Webside, letzter Abruf: 28.11.2010*. – URL `http://www.xbox.com/`

[Fingertracking 2006] *ART Fingertracking v1.0.1 - A.R.T. GmbH*, 2006

[ARTtracker-Anleitung 2007] *ARTtrack und DTrack Benutzerhandbuch v1.24.3 - A.R.T. GmbH*, 2007

[DTrack-Anhang 2007] *DTrack Technischer Anhang v1.24.3 - A.R.T. GmbH*, 2007

[Boetzer 2008] BOETZER, Joachim: *Bewegungs- und gestenbasierte Applikationssteuerung auf Basis eines Motion Trackers*. 2008. – URL `http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/boetzer.pdf`

[Breuer 2005] BREUER, Pia: *Entwicklung einer prototypischen Gestenerkennung in Echtzeit unter Verwendung einer IR-Tiefenkamera*. 2005. – URL `http://www.uni-koblenz.de/~cg/Diplomarbeiten/DABreuer.pdf`

[Erich Gamma 2001] ERICH GAMMA, Ralph Johnson John V.: *Entwurfsmuster*. Bonn : Addision-Wesley, 2001

[Fischer 2007] FISCHER, Christian: *Entwicklung eines multimodalen Interaktionssystems für computergestützte Umgebungen*. 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/fischer.pdf

[Fowler 2003] FOWLER, Martin: *Patterns für Enterprise Application-Architekturen*. Bonn : mitp-Verlag, Auflage 1, 2003

[Fritz] FRITZ, Susanne: *Medienfassaden , letzter Abruf: 19.04.2011*. – URL http://www.architonic.com/de/ntsht/medienfassaden/7000408

[Hara 2011] HARA, Tenshi C.: *Realisierung der Erkennung menschlicher Bewegungsmuster durch Wireless Sensor Nodes*. 2011. – URL http://hara.tc/publications/CaptureWSN.pdf

[Hein 2005] HEIN, Torsten: *Gestenbasierte Interaktion Gestenbasierte Interaktion - Technologische Grundlagen und Technologische Grundlagen und Toolkits*. 2005. – URL http://ebus.informatik.uni-leipzig.de/www/media/lehre/uiseminar05/uiseminar05-hein-folien.pdf

[Heitsch 2008] HEITSCH, Johann: *Ein Framework zur Erkennung von dreidimensionalen Gesten*. 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/heitsch.pdf

[Mohammadali Rahimi 2008] MOHAMMADALI RAHIMI, Matthias V.: *Gestenbasierte Computerinteraktion auf Basis von Multitouch-Technologie*. 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/rahimi_vogt.pdf

[Roßberger 2008] ROSSBERGER, Philipp: *Physikbasierte Interaktion in kollaborativen computergestützten Umgebungen*. 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/rossberger.pdf

[Smith 2002] SMITH, Russell: *How to make new joints in ODE*. 2002. – URL http://www.ode.org/joints.pdf

[VR/URBAN] VR/URBAN: *SMSlingshot 9.12.2010, Photorights © 2008-2010. VR/Urban ,letzter Abruf 8.12.2010*. – URL http://www.mediafacades.eu/sites/default/files/smslingshot_montage_web_0.jpg

[Winograd 1972] WINOGRAD, Terry: *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language*. 1972. – URL http://hci.stanford.edu/~winograd/shrdlu/AITR-235.pdf

Abbildungsverzeichnis

Glossar

ASCII	ASCII ist eine 7 Bit Zeichenkodierung, sie wird dazu genutzt, Text darzustellen.
Bones	Bones, deutsch: Knochen, sind ein Konstrukt im Bereich der 3D-Modellierung. Angelehnt an die Knochen im Körper dienen diese Bones dazu, das 3D-Modell in eine bestimmte Form zu bringen.
Framework	Ein Framework, deutsch: Rahmenwerk, bestimmt die Architektur der Anwendung. Es stellt dem Softwareentwickler üblicherweise eine Reihe von Grundfunktionalitäten in Form von Klassen und Methoden zur Verfügung, sodass der Entwickler sich auf die spezifischen Details seiner Anwendung konzentrieren kann [Erich Gamma (2001)].
Immersion	Immersion ist ein Bewusstseinszustand. Je höher der Grad der Immersion ist, desto mehr fühlt sich der Anwender als Teil einer simulierten Umgebung.
Media Facade	Media Facade, deutsch: Medienfassaden, sind Gebäudefassaden die mittels großen Bildschirmen oder Leuchtmitteln selbst in der Lage sind Botschaften zu verbreiten und unteranderem als Werbefläche dienen können [Fritz].
UDP	User Datagramm Protocol - UDP ist ein verbindungsloses Netzwerkprotokoll. Es können damit Daten über ein Netzwerk versendet werden, in dem dem Sender der oder die Empfänger nicht bekannt sein müssen.
Wrapper	Als Wrapper wird eine Software bezeichnet, die eine andere Software umgibt. Die umhüllte Software bleibt dem Aufrufer verborgen.

Anhang A

ASCII-Datagramm

Das hier abgebildete ASCII-Datagramm dient zur Kommunikation zwischen dem Physikmodul und der grafischen Ausgabe.
Das Datagramm ist an das Datagramm vom DTracker angelehnt [DTrack-Anhang (2007)]. Es besteht aus mehreren Zeilen, die durch Zeilenumbrüche voneinander getrennt sind. Jede Zeile beginnt mit einer Kennung, die den Typ der folgenden Daten bestimmen.

Beispiel:
fr 12345
text Greifgeste
box 1 [1][0.0 0.0 0.0 0.0 0.0 0.0][255 0 0][8.0 10.0 15.0]
sphere 2 [2][10.0 0.0 0.0 0.0 0.0 0.0][0 0 255][5.0 10 10] [3][-10.0 0.0 0.0 0.0 0.0 0.0][0 0 255][5.0 10 10]

Schema:

FrameNR
Geste
typ anzahl [id][x y z r1 r2 r3][r g b][breite hohe tiefe]
typ anzahl [id][x y z r1 r2 r3][r g b][radius zSamples radialSamples]

Datentypen:

- int
- String
- float

Wertebereich:

Kennung	Beschreibung	Datentyp	Wertebereich
fr	Framenummer	int	1 bis int.max
text	Name der erkannten Geste	String	
typ	Körpertyp	String	'box' oder 'sphere'
anzahl	Anzahl	int	1 bis int.max

Tabelle A.1: Datagramm Head

Daten	Beschreibung	Datentyp	Wertebereich
id	Eindeutige Identifikation	int	1 bis int.max
x	Raumposition auf der X-Achse	float	-float.max bis float.max
y	Raumposition auf der Y-Achse	float	-float.max bis float.max
z	Raumposition auf der Z-Achse	float	-float.max bis float.max
r1	Rotationswert um die X-Achse	float	-float.max bis float.max
r2	Rotationswert um die Y-Achse	float	-float.max bis float.max
r3	Rotationswert um die Z-Achse	float	-float.max - float.max
r	Farbwert rot	int	0 bis 255
g	Farbwert grün	int	0 bis 255
b	Farbwert blau	int	0 bis 255

Tabelle A.2: Datagramm Objekt-Body

Daten	Beschreibung	Datentyp	Wertebereich
radius	Radius der Kugel	float	Wert > 0
zSamples	Anzahl Unterteilungen längs der Z-Achse	int	Wert >= 2
radialSamples	Anzahl Seiten der radialen Grundfläche	int	Wert >= 3

Tabelle A.3: Details Kugel

Daten	Beschreibung	Datentyp	Wertebereich
breite	Breite des Quaders	float	Wert > 0
hoehe	Höhe des Quaders	float	Wert > 0
tiefe	Räumliche Tiefe des Quaders	float	Wert > 0

Tabelle A.4: Details Quader

Anhang B

Fotos der Evaluation

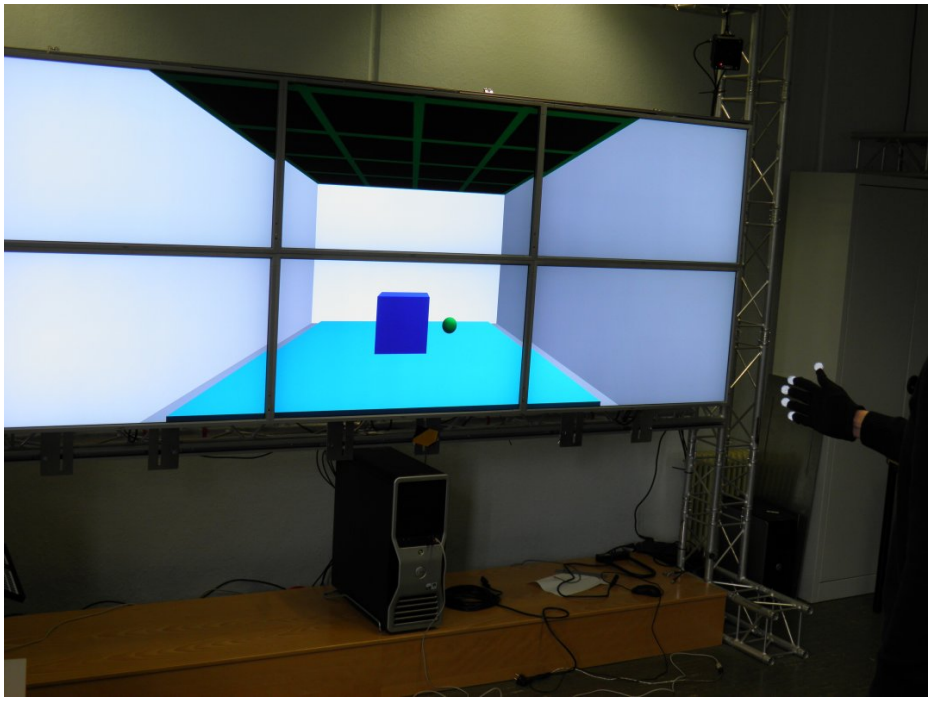

Abbildung B.1: Ausgangsposition mit einem Aktor

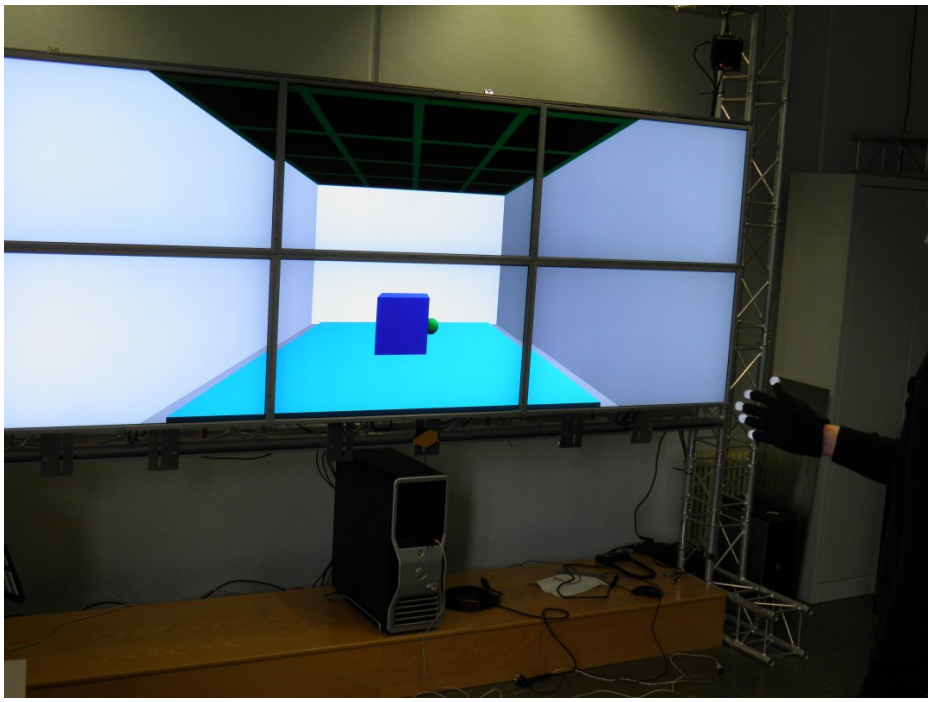

Abbildung B.2: Block wird von einem Aktor berührt

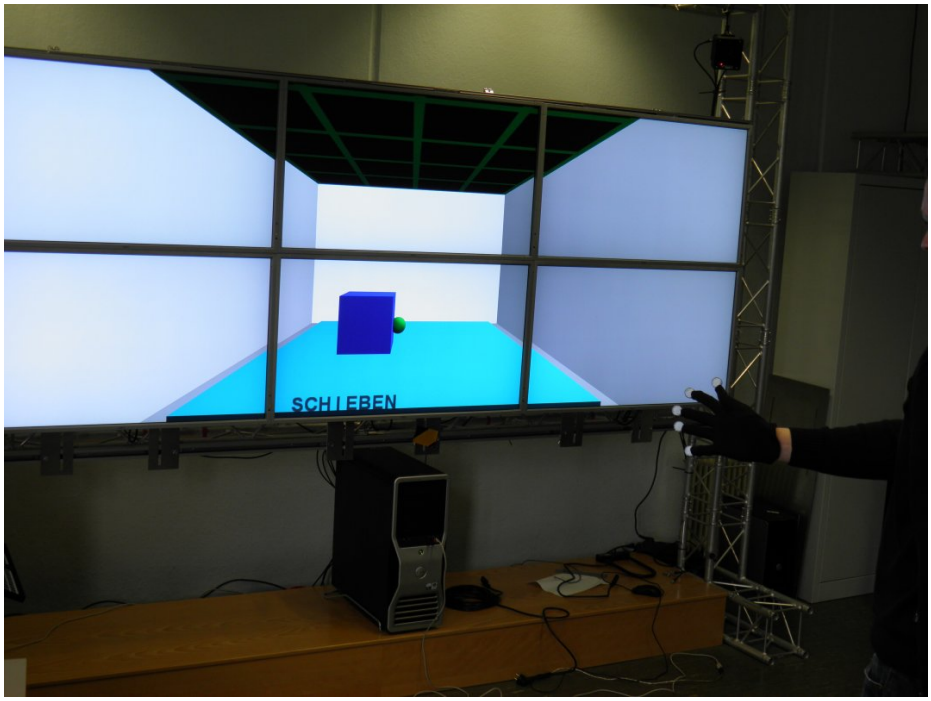

Abbildung B.3: Block wird geschoben

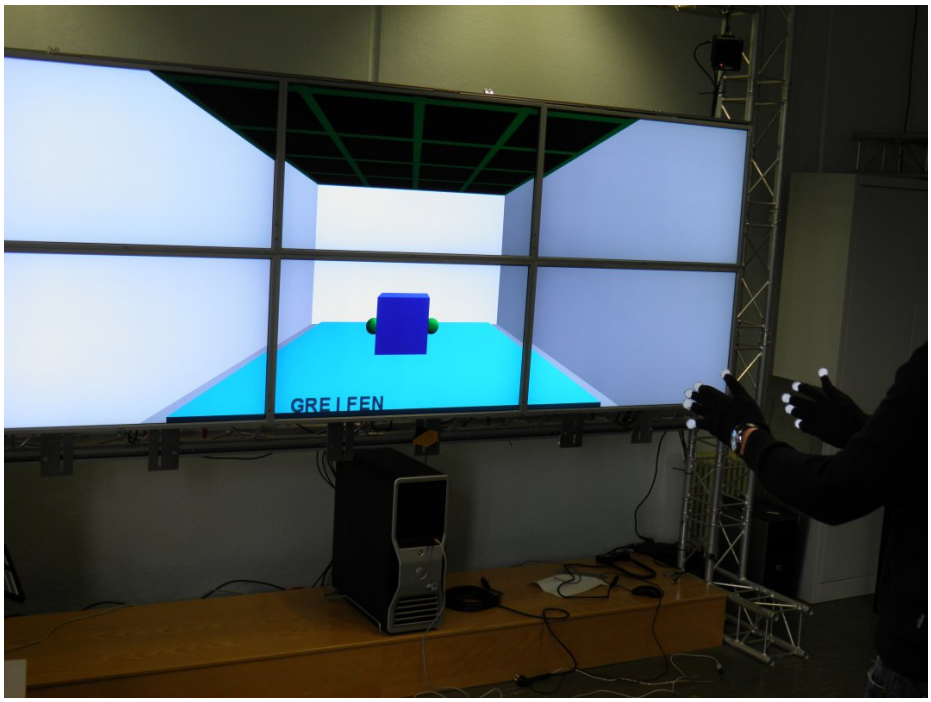

Abbildung B.4: Block wird gegriffen

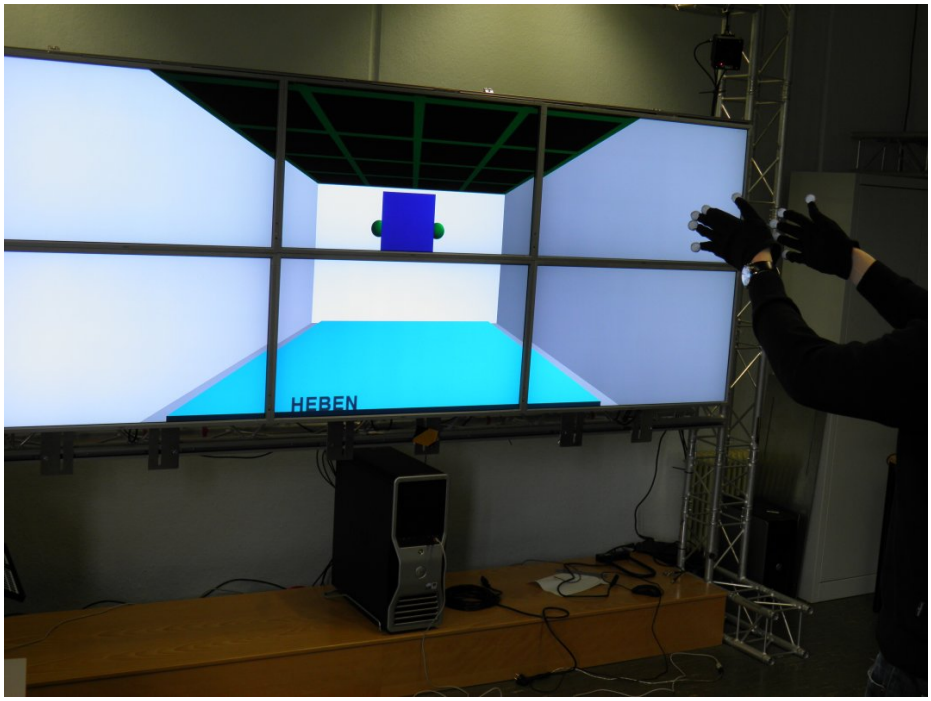

Abbildung B.5: Block wird angehoben

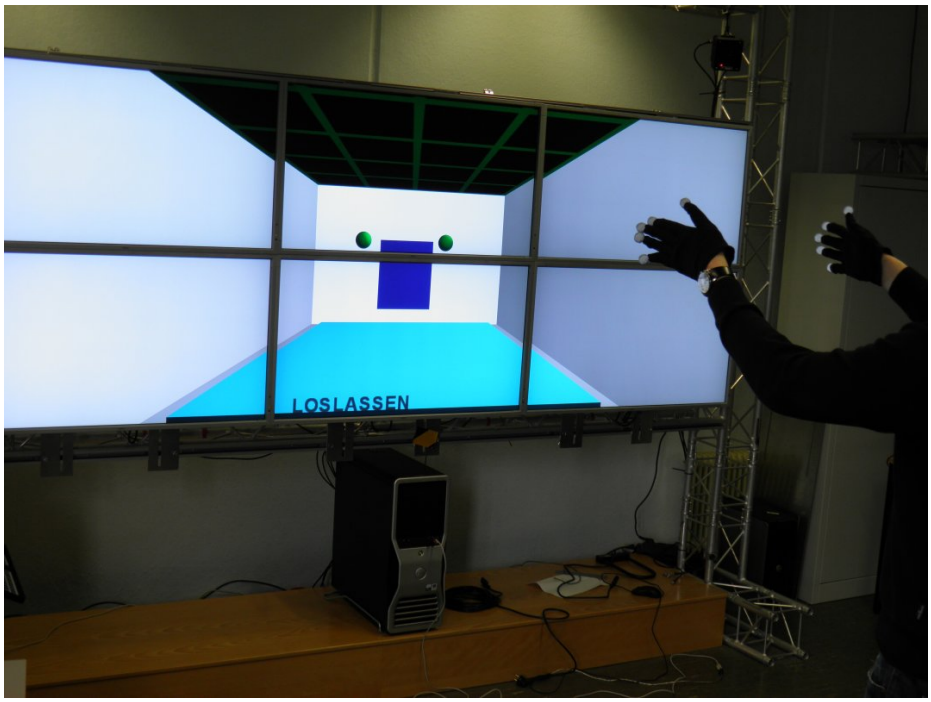

Abbildung B.6: Block wird losgelassen

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 21.04.2011 Olaf Potratz