Implementierung eines interaktiven Charakters im Rahmen einer Multi-Touch-Anwendung

Fachbereiche IEM und MND der
Fachhochschule Gießen-Friedberg

Bachelorarbeit

vorgelegt von

Hans Christian Arlt
geb. in Frankfurt am Main

Alexander Ehrlich
geb. in Ciili (Kasachstan)

durchgeführt bei

NewMedia Yuppies GmbH
Kreativagentur für Neue Kommunikation

Referent der Arbeit:	Prof. Dr.-Ing. Cornelius Malerczyk
Korreferentin der Arbeit:	Dipl.-Math. (FH) Sabine Langkamm
Firmeninterner Betreuer:	Dipl.-Inform. (FH) Peter Eschler

Fachbereiche

Informationstechnik-Elektrotechnik-Mechatronik IEM
und
Mathematik, Naturwissenschaften und Datenverarbeitung MND

Friedberg, 2010

Für unsere Eltern,
Johanna und Hartmut Arlt sowie Natalja und Alexander Ehrlich.

Danksagung

Verschiedene Personen haben zum Gelingen dieser Arbeit beigetragen, bei denen wir uns an dieser Stelle ganz herzlich für die Unterstützung bedanken wollen. Unser besonderer Dank gilt unseren Eltern, die uns während des Studiums sowie während der Zeit, in der wir diese Arbeit verfasst haben, unterstützt haben. Wir bedanken uns bei den beiden Geschäftsführern der NewMedia Yuppies, Peter Eschler und Sebastian Demmerle, dass wir diese Arbeit bei den NewMedia Yuppies schreiben durften. Besonders wollen wir uns bei Peter Eschler bedanken, der uns von Beginn dieser Arbeit an als Betreuer durchgehend zur Seite stand. Neben der Hilfe bei Programmierproblemen danken wir ihm auch für seine Geduld beim Korrekturlesen dieser Arbeit. Ein Dank geht auch an den Mitarbeiter Wolfram Kresse, für die Unterstützung bei der Programmierung sowie an den Praktikanten Bruno Heller, der uns in der konzeptionellen Phase mit kreativen Ideen unterstützt hat.

Julia Grim, Melanie Schwenk, Johanna und Hartmut Arlt danken wir ganz herzlich für die Zeit, die sie sich genommen haben, um diese Arbeit Korrektur zu lesen. Ein großer Dank geht auch an unsere Referenten Prof. Dr.-Ing. Cornelius Malerczyk und Dipl.-Math. Sabine Langkamm, die schon während unseres Studiums unser Interesse an dem Bereich der 3D-Computergrafik geweckt hatten und uns während dieser Arbeit betreut haben.

Eidesstattliche Erklärung

Hiermit versichern wir, Hans Christian Arlt und Alexander Ehrlich, dass wir die vorliegende Bachelorarbeit selbständig und gemäß der unten aufgeführten Abgrenzung unter ausschließlicher Verwendung der im Quellenverzeichnis angegebenen Hilfsmittel angefertigt haben.

Kapitel	Seite	Verantwortlich
1 Einleitung	1	Hans Christian Arlt Alexander Ehrlich
2.1 Stand der Technik - Einleitung	7	Hans Christian Arlt Alexander Ehrlich
2.2 Virtuelle Charaktere	8	Alexander Ehrlich
2.4 Multi-Touch-Systeme	25	Hans Christian Arlt
2.5 Echtzeit 3D-Entwicklungsumgebungen	37	Hans Christian Arlt Alexander Ehrlich
2.5.1 Anwendungsgebiete von Echtzeit 3D-Computergrafik	38	Alexander Ehrlich
2.5.3 Andere Entwicklungsumgebungen	43	Hans Christian Arlt
2.6 Zusammenfassung und Ausblick	45	Hans Christian Arlt Alexander Ehrlich
3 Konzeptentwicklung	47	Alexander Ehrlich
3.2 Anforderungen	48	Hans Christian Arlt
3.3.3 Aufbau und Spielgeschichte	51	Alexander Ehrlich
3.4 Konzept - Zusammenfassung	62	Hans Christian Arlt Alexander Ehrlich
4 Gegenüberstellung geeigneter Entwicklungsumgebungen	65	Hans Christian Arlt
5 Das Charaktermodell	81	Alexander Ehrlich

6 Animation mit Motion-Capture-Daten	91	Alexander Ehrlich
6.3 Animation in MotionBuilder	92	Hans Christian Arlt
6.4 Nachbearbeitung in MotionBuilder	94	Alexander Ehrlich
6.5 Animation mit Motion-Capture-Daten - Zusammenfassung	94	Hans Christian Arlt Alexander Ehrlich
7 Erstellung der Anwendung	97	Alexander Ehrlich
7.3 Anbindung von Multi-Touch	101	Hans Christian Arlt
7.5 Interaktionsimplementierung	105	Alexander Ehrlich
7.5.2 JetPackController	108	Hans Christian Arlt
7.5.3 Weitere Interaktionen	110	Alexander Ehrlich
7.6 Weitere Asset Implementierungen	112	Hans Christian Arlt
7.7 Erstellung der Anwendung - Zusammenfassung	113	Hans Christian Arlt Alexander Ehrlich
8 Ergebnis	115	Hans Christian Arlt Alexander Ehrlich
9 Zusammenfassung und Ausblick	121	Hans Christian Arlt Alexander Ehrlich

Friedberg, Dezember 2010

Hans Christian Arlt

Alexander Ehrlich

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Mit Multi-Touch wird die Fähigkeit eines berührungsempfindlichen Eingabegeräts bezeichnet, gleichzeitig mehrere Berührungen, meist Finger, zu erkennen. Multi-Touch ist seit einiger Zeit ein verbreitetes Thema in Forschung und Entwicklung. Die momentan gängigen Multi-Touch-Anwendungen beschäftigen sich hauptsächlich mit der Verarbeitung von Objekten, wie zum Beispiel Bildern und Dokumenten und ermöglichen, diese auf dem Bildschirm zu bewegen, zu vergrößern oder zu verschieben (siehe Abbildung 1.1 links). Darüber hinaus existieren Multi-Touch-Anwendungen im Bereich Produktpräsentation, wie das TouchLab Projekt der Firma "New Media Yuppies"[1] und einfache Multi-Touch-Spiele, wie zum Beispiel das Spiel "Touchgrind HD" der Firma "Illusion Labs".[2] Dennoch ist das Angebot an Multi-Touch-Anwendungen, abgesehen von Anwendungen für das "Apple" "iPhone" sowie "iPad", momentan noch sehr begrenzt. [UL09] In dieser Arbeit soll eine Anwendung für einen Multi-Touch-Tisch entwickelt werden, die die Verknüpfung zwischen einem virtuellen Charakter in einer 3D-Welt und einer Multi-Touch-Anwendung darstellt. Der animierte Charakter soll in einer virtuellen Umgebung zur Wissensvermittlung eingesetzt werden. Die Spielindustrie greift seit einiger Zeit, ebenso wie die Filmindustrie, auf Motion-Capturing-Verfahren zurück, um digital erzeugte Charaktere zu animieren. Mit Hilfe von Motion-Capturing ist es möglich einer virtuellen Figur ein sehr natürliches Bewegungsverhalten zu geben (siehe Abbildung 1.1 rechts). Deshalb sollen die Animationen, des in dieser Arbeit verwendeten Charakters, mit Hilfe von Motion-Capture-Daten, erstellt werden.

[1] www.nmy.de
[2] http://www.illusionlabs.com/

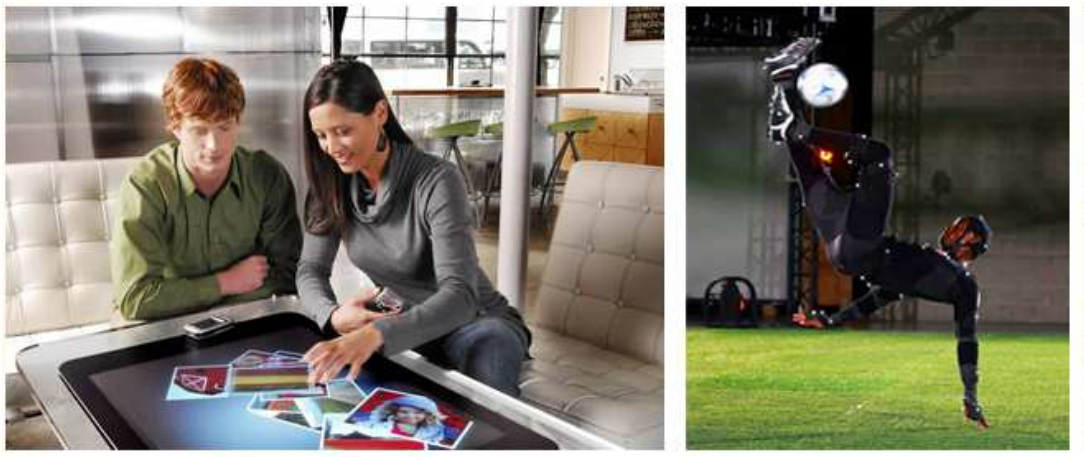

Abbildung 1.1: Links: Verwendung einer Fotoanwendung auf einem Microsoft Surface.[3]Rechts: Maurice Edu vom Toronto FC, beim Durchführen eines Fallrückziehers bei einer Motion-Capture-Session für das Spiel FIFA Soccer 09. Quelle: JEFF VINNICK/AFP/-Getty Images

1.2 Problemstellung und Zielsetzung

Die bisherigen Anwendungsgebiete für Multi-Touch beschränken sich meist noch auf technische Demos und Prototypen. Das hauptsächliche Einsatzgebiet liegt hierbei in der Verwendung von Multi-Touch-Geräten für Produktpräsentationen auf Messen und in Showrooms (siehe Abbildung 1.2). Hierbei sind Softwarepakete bisher an konkrete Hardwareprodukte gekoppelt oder werden nur als Gesamtpaket verkauft. [UL09]

Die Entwicklung von kommerziellen Produkten wird momentan von wenigen, großen Unternehmen vorangetrieben. So gibt es noch keine Normen für die Interaktionsgesten, bei der Bedienung von Multi-Touch-Geräten. Durch die starke Verbreitung der von der Firma Apple produzierten Multi-Touch-Produkte, wie das iPhone und das iPad, haben sich die Grundinteraktionsgesten von Apple für drehen oder zoomen im Konsumentenbereich etabliert. Darüber hinaus haben sich noch keine weiteren Gesten für die Steuerung von Multi-Touch-Geräten durchgesetzt. Das kürzlich erschienene Spiel "R.U.S.E." ist der erste Versuch der Spieleindustrie auch diesen Bereich zu integrieren.[4] In Spielanwendungen für das iPhone bzw. iPad gibt es die ersten Ansätze, zur Multi-Touch-Steuerung von Charakteren in einem Spiellevel, wie zum Beispiel bei dem Spiel "James Cameron's Avatar"[5] von der Firma "Gameloft". Die Firma "Outfit7" hat mit "Talking Tom Cat" eine der wenigen Touch-Anwendung auf den Markt gebracht, in der der Benutzer über einfache Berührungen direkt mit einem virtuellen Charakter interagieren kann. Die Schwierigkeit, bei der Interaktion mit virtuellen Charak-

[3] http://thefutureofthings.com/upload/items_icons/Microsoft-Surface-Computing_large.jpg
[4] http://ruse.de.ubi.com/index.php?page=medias
[5] http://www.avatariphonegame.com

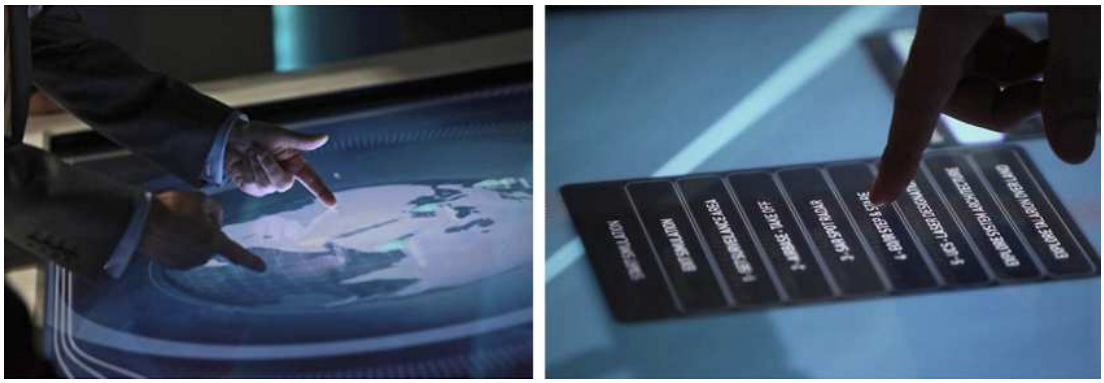

Abbildung 1.2: Bedienung einer Multi-Touch-Demonstrations-Anwendung an einem Messestand der EADS. Quelle: NewMedia Yuppies

teren in einer Echtzeitanwendung, ist die Umsetzung der intuitiven Steuerungsgesten. Der Benutzer einer Anwendung sollte sich keine großen Gedanken machen müssen, wie er die Anwendung zu bedienen hat. Jedoch stellt die Verwendung von Touch- bzw. Multi-Touch-Gesten die Möglichkeit zur Verfügung, zum Beispiel dem Benutzer das Gefühl zu geben, dass auf eine Interaktion mit seinem Finger eine direkte Aktion des Charakters folgt. Hier sind Aktionen wie Kitzlen, Streichen oder schubsen denkbar. Die Verwendung von Multi-Touch-Oberflächen bieten dem Benutzer aber auch noch viele weitere Vorteile. Eine leichte Handhabbarkeit und eine schnelle Erlernbarkeit zeigen, dass Touchscreens in Zukunft andere Eingabegeräte ablösen können. Demgegenüber stehen die Nachteile und Anforderungen an das Eingabegerät. So kann es zu Schwierigkeiten bei der Gestenerkennung kommen, wodurch Fehler entstehen können. Je nach Verfahren der Berührungserkennung (Berührung wird im Folgenden auch als "Touch" bezeichnet) können Finger falsch erkannt oder Probleme durch das Verdecken von Fingern auftreten.

Das Ziel dieser Arbeit ist die Erstellung einer hardwareunabhängigen Multi-Touch-Anwendung, die spielerisch zur Wissensvermittlung dienen soll. Weiter soll die Verbindung zwischen Touch- bzw. Multi-Touch-Gesten und einem virtuellen Charakter in einer 3D-Welt hergestellt werden. Dabei soll die Anwendung als Beispiel dienen und aufzeigen, dass durch Touch- bzw. Multi-Touch-Gesten deutlich mehr sowie intuitivere Interaktionen möglich sind, als bei konventionellen Interaktionen, wie zum Beispiel mit Maus und Tastatur. Hierbei sollen sinnvolle Touch- bzw. Multi-Touch Interaktionsmöglichkeiten zur Bedienung der Anwendung erarbeitet werden. Dazu gehört die Steuerung bzw. Navigation in der 3D-Welt und die Interaktion mit dem virtuellen Charakter. Die Interaktionen mit dem Charakter können in zwei Kategorien zusammengefasst werden:

Indirekte Interaktion

Interaktionen in der Anwendung auf die der Charakter reagiert, wie zum Beispiel eine Positionsänderung eines Objektes in der Spielwelt.

Direkte Interaktion
: Interaktionen die im direkten Bezug zum Charakter stehen. Der Charakter reagiert zum Beispiel auf das Drücken auf bestimmte Körperteile.

Ein Schwerpunkt ist das Ausarbeiten eines Konzepts, in dem grundlegende Fragestellungen im Bezug auf die Anwendung geklärt werden. Dazu gehören u.a. Fragen zu:

- Zielgruppe
- Stil und Aussehen der Anwendung
- Spielgeschichte
- Interaktion
- Spielablauf
- Charakterentwurf

Dabei wird in dieser Arbeit die gesamte Arbeitskette von der Erstellung des Charakters über die interaktive Steuerung bis zur Ausgabe auf einem multi-touch-fähigen Gerät durchlaufen. Zur Modellierung des Charakters wird auf vorhandene Anwendungen zurückgegriffen. Um möglichst natürliche und flüssige Bewegungsabläufe des Charakters zu erreichen kommen Motion-Capture-Daten zum Einsatz. Motion-Capturing ist die Aufzeichnung von Bewegungen. Zur Verknüpfung der verschiedenen Anforderungen der Anwendung muss eine Entwicklungsumgebung gewählt werden, welche die Fähigkeit der Berührungserkennung, die Einbindung und die Echtzeitdarstellung sowie Animation von Charakteren, unterstützt. Weiter soll Wert darauf gelegt werden, dass die Bedienbarkeit der Anwendung möglichst einfach und intuitiv erfolgt und leicht erlernbar ist. So sollen Steuergesten wie Drehen und Zoomen genutzt werden, welche aus anderen Multi-Touch-Anwendungen bekannt sind.

1.3 Organisation der Arbeit

In Kapitel 2 wird zunächst auf die verschiedenen Techniken, die in dieser Arbeit verwendet werden, näher eingegangen, die heute benutzen Verfahren vorgestellt und deren Unterschiede aufgezeigt. Zu den verwendeten Techniken und Verfahren gehören:

- der Einsatz von virtuellen Charakteren
- die Animation von Charakteren
- die verschiedenen Multi-Touch-Verfahren
- die Echtzeit-3D-Entwicklungsumgebungen

Danach wird in Kapitel 3 die Anwendungsidee entwickelt und ein für die Anwendung benötigter Charakter entworfen. Aus den gewonnenen Erkenntnissen der vorigen Kapitel wird zunächst eine Softwareauswahl getroffen, die im nachfolgenden Kapitel 4 gegenübergestellt wird. Die Komponenten und die verschiedenen Ansätze der ausgewählten Software werden auf ihre Brauchbarkeit, in Hinblick auf Funktionalität und Qualität des Ergebnisses, untersucht. Dabei wird für die zwei erfolgversprechendsten Programme eine kleine Test-Anwendung programmiert, die den "Workflow" vergleichbar machen soll. Aus den daraus resultierenden Erkenntnissen erfolgt die Entscheidung für eines der Programme. In Kapitel 5 wird der Charakter, der in Kapitel 3 entworfen wurde, in ein 3D-Modell umgesetzt. Das nachfolgende Kapitel 6 beschäftigt sich mit der Animation des Charakters. Darin wird beschrieben, wie Motion-Capture-Daten bereinigt und auf den Charakter gelegt werden. Danach wird in Kapitel 7 die Anwendung umgesetzt und der zuvor erstellte und animierte Charakter implementiert. Ein weiterer wichtiger Bestandteil dieses Kapitels ist die Programmierung und Umsetzung der Multi-Touch-Steuerungsgesten. In Kapitel 8 wird das Ergebnis der Arbeit präsentiert und bewertet. Abschließend folgt in Kapitel 9 die Zusammenfassung der gesamten Arbeit sowie der Ausblick.

1.4 Was nicht behandelt wird

Die Bachelorarbeit umfasst viele unterschiedliche Gebiete, die für sich gesehen äußerst komplex sind. Viele Bereiche werden deshalb nur angerissen und nicht erschöpfend behandelt. Hauptaugenmerk liegt auf dem Zusammenspiel der verwendeten Technologien. Diese Arbeit veranschaulicht zunächst das Grundkonzept und beschreitet anschließend einen möglichen Weg zur Erstellung einer multi-touch-fähigen 3D-Anwendung. Wenn nötig wird auf andere Möglichkeiten hingewiesen, jedoch ohne näher darauf einzugehen. Der praktische Teil der Arbeit beschäftigt sich mit der Umsetzung der Anwendung. In einigen Bereichen wird JavaScript-, C#- sowie X3D-Quelltext eingebunden. Tiefergehende Erläuterungen des Quelltextes werden in der Bachelorarbeit nicht getätigt, sondern auch hier lediglich auf die Funktionalität und die Zusammenhänge eingegangen.

1.5 Zusammenfassung der wichtigsten Ergebnisse

Durch die zunehmende Verbreitung von Multi-Touch-Geräten wird die Nachfrage nach Multi-Touch-Anwendungen immer größer. Zurzeit überwiegen noch 2D-Anwendungen auf dem Markt. Zwar gibt es auch 3D-Anwendungen, jedoch ist die Verwendung von 3D-Charakteren ausgesprochen selten. Aus diesem Grund soll in dieser Arbeit eine Multi-Touch 3D-Anwendung entwickelt werden, in der gezeigt werden soll, dass die Interaktion über Multi-Touch mit einem Charakter in einer 3D-Welt möglich ist. Zuerst wird ein Konzept ausgearbeitet, in dem die Grundidee der Anwendung festgelegt wird. Die Anwendung wird genutzt, um dem Benutzter spielerisch Informationen über das Sonnensystem und seine Planeten zur Verfügung zu stellen. Hierbei wurde in der Konzeptentwicklung die Zielgruppe Kind festgelegt. Die ausgearbeitete Idee ist ein Weltraumszenario, in dem der Charakter auf einem Planeten steht. Nach bestimmten Touch-Gesten werden verschiedene Aktionen ausgeführt, wie bei-

spielsweise das Drehen der Szene, das Anzeigen von Informationen oder das Auslösen einer Charakteranimation. Als nächstes wird das Aussehen des Charakters festgelegt, welches von einem astronautenähnlichen Anzug geprägt wird, der jedoch einen Fernseher statt eines Helms besitzt. Im nächsten Schritt wird die Wahl zwischen den in Frage kommenden 3D-Entwicklungsumgebungen, "instantreality" und "Unity3D", diskutiert. Nach dem Erstellen einer Testanwendung, in je einer der Entwicklungsumgebungen, ergibt sich ein eindeutiger Favorit. Auf Grund einer existierenden GUI, der einfacheren Importmöglichkeit von Modellen und der besseren Überblendungen der Animationen, wird Unity3D für die Erstellung der Anwendung ausgewählt. Zur Erstellung des Charakters wird die Software "Maya" der Firma "Autodesk" genutzt. Um den Charakter zu animieren, werden zuerst Motion-Capture-Daten aufbereitet und die aufgezeichneten Daten mit der Software "MotionBuilder" auf den Charakter gelegt. Nachdem alle benötigten Daten für die Anwendung vorhanden sind, werden sie in Unity3D importiert und die Szene wird erstellt. Danach werden die Skripte geschrieben, welche die Anwendung steuern. Das Programmieren kann in JavaScript, Boo oder in C# erfolgen. Jedoch ist es sinnvoll sich auf eine Sprache festzulegen. Aufgrund der vielfältigeren Einsatzmöglichkeit von C# wird in dieser Arbeit C# verwendet. Zur Lösung der Probleme kann auf die verschiedensten Funktionen und die umfangreiche Dokumentation von Unity3D zugegriffen werden. Vereinzelt können Skripte, die von Unity3D zur Verfügung gestellt werden, für die Anwendung genutzt bzw. angepasst werden. Nach der Fertigstellung der Anwendung kann diese, über eine Kommunikationsschnittstelle, auf Multi-Touch-Geräten ausgeführt werden. In dieser Arbeit wird deutlich, dass die Navigation in einer komplexen Echtzeit 3D-Welt und die Interaktion mit einem Charakter mittels Multi-Touch möglich ist. Bei der Entwicklung der Anwendung stellt sich aber heraus, dass für Interaktionen mit dem Charakter "Single"-Touches meist intuitiver sind.

Kapitel 2

Stand der Technik

Dieses Kapitel beschäftigt sich mit virtuellen Charakteren sowie ihren Einsatzgebieten. Zudem wird ein Überblick der in Computerspielen bzw. computeranimierten Filmen angewandten Techniken zur Animation von Charakteren gegeben. Besonderes Augenmerk wird dabei auf die Verwendung von Motion-Capturing-Verfahren gelegt. Neben der Aufführung der verfügbaren Methoden, wird auf die Nachbearbeitung von Daten eines optischen Motion-Capturing-Systems eingegangen. Weiter beschäftigt sich dieses Kapitel mit den momentan bekannten Multi-Touch-Verfahren. Die einzelnen Technologien werden beschrieben und die Vor- und Nachteile aufgezeigt. Danach werden verfügbare Hardware- und Softwarelösungen vorgestellt. Der letzte Abschnitt handelt von Echtzeit 3D-Entwicklungsumgebungen. Dabei wird näher auf die Funktionsweise von Game-Engines eingegangen. Auch andere Entwicklungsumgebungen zur Erstellung von Echtzeit 3D-Anwendungen werden vorgestellt.

2.1 Einleitung

Bei der Vielzahl von virtuellen Charakteren die es zurzeit gibt, sind auch ihre Einsatzgebiete recht unterschiedlich. Virtuelle Charaktere können reale Personen repräsentieren oder durch den Computer gesteuert werden. Manche repräsentieren eine Firma oder sind Ersatz für einen menschlichen Assistenten. Es gibt Charaktere, die eine weiterentwickelte Hilfe-Funktion darstellen, Benutzern Tipps und Hinweise geben und mit ihnen kommunizieren. Möglich ist schon heute, dass der Benutzer sich mit dem Charakter in Grenzen unterhalten kann.
Sobald ein virtueller Charakter mit einem Benutzer kommunizieren und interagieren kann, ist er interaktiv. Dabei ist es gleich, auf welche Weise die Kommunikation stattfindet. Wird dabei das Gefühl erweckt, dass der Charakter ein Individuum bzw. eine eigenständige Persönlichkeit ist, wird dieser verstärkt wahrgenommen.

Die Animation eines Charakters ist eine schwierige Aufgabe. Es müssen Methoden gefunden werden, das geometrische Modell des virtuellen Charakters zu verändern. Dazu gehören nicht nur die Verformung des Modells, sondern auch Methoden, um Bewegungen zu generieren und wiederzugeben. Es gibt viel zu beachten und ein Auge für Details ist bei der Animation

wichtig. Wenn animierte Charaktere in Computerspielen oder Filmen möglichst natürliche Bewegungsabläufe haben sollen, greifen die Produktionsfirmen heutzutage immer wieder auf Motion-Capturing zurück, da dieses Verfahren bei komplexen aber auch bei einfachen Animationen zeitsparender ist.

Die Forschung an Multi-Touch-Anwendungen begann bereits in den frühen 1990er-Jahren. Seit Apple im Jahre 2007 mit dem iPhone das erste Endgerät mit Multi-Touch-Funktion auf den Markt brachte, nahm die Popularität von Multi-Touch rapide zu. Seit dem Jahr 2009 sind Multi-Touch-Geräte, vor allem im Bereich der Mobiltelefone, aus dem alltäglichen Gebrauch nicht mehr wegzudenken. Durch die Einführung des Betriebssystems Microsoft Windows 7 sowie das Anfang 2010 erschienene Apple iPad steigt die Nachfrage und das Interesse an Multi-Touch-Anwendungen enorm.

Echtzeit 3D-Anwendungen zeichnen sich dadurch aus, dass Bilder zur Echtzeit auf einem Computer gerendert werden. Es ist der höchste interaktive Bereich der Computergrafik. Die nachfolgend generierten Bilder hängen von der Reaktion und dem Handeln des Benutzers ab. Dieser Zyklus von Reaktion und Rendering (berechnen/generieren von Bildern in einer 3D-Szene) geschieht so schnell, dass der Betrachter die einzelnen Bilder als einen dynamischen Prozess wahrnimmt. [TAM08]

2.2 Virtuelle Charaktere

Virtuelle Charaktere sind fiktive Figuren, beispielsweise aus Zeichentrick- und Animationsfilmen oder Computerspielen. In dieser Arbeit wird der Begriff Charakter aus dem Englischen "Character" abgeleitet und bezieht sich nicht nur auf die Eigenschaften und Persönlichkeit einer fiktiven Figur, sondern impliziert auch die äußere Darstellung. [Hel09] Da virtuelle Charaktere synthetisch entstehen, sind der Gestaltung ihres Aussehens keine Grenzen gesetzt. Sie können in Form von Monstern, Fabelwesen oder anderen Kreaturen auftreten.

In nahezu allen Computerspielen existieren virtuelle Charaktere. Ein Spieler schlüpft in die Rolle des Charakters und erlebt die Welt aus dessen Sicht. Auch in Computerspielen aus der "Ich"-Perspektive (Ego-Perspektive) wird von einem virtuellen Charakter gesprochen. Obwohl der Charakter nicht sichtbar ist, identifiziert sich der Spieler mit der Figur und überträgt sein Verhalten auf sie.

Virtuelle Charaktere müssen nicht mit einem Benutzer interagieren oder ihn repräsentieren. Sie können beispielsweise auch in Computeranimationen als Schauspieler eingesetzt werden. [Ebn09]

2.2.1 Charaktere in Computeranimationen

Bei Computeranimationen werden generell Einzelbilder mit Hilfe von Computergrafikprogrammen gerendert und abgespeichert. Werden diese dann schnell genug abgespielt, wird von einer Animation gesprochen. Bekannte Computeranimationen sind beispielsweise Filme

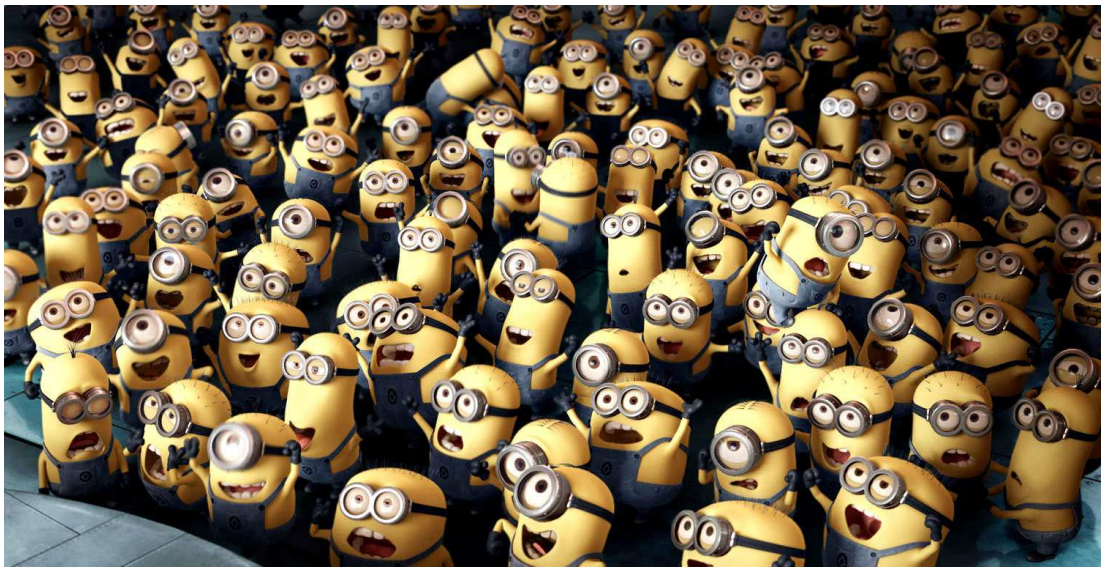

Abbildung 2.1: Szene aus dem Computeranimationsfilm "Ich - Einfach Unverbesserlich" mit comichaften Charakteren, den sog. "Minions".[1]

der "Pixar Animation Studios" wie "Toy Story", "Findet Nemo" oder "WALL-E". Da solche Filme ohne real gedrehte Szenen auskommen, zeichnen sie sich durch ihre Innovationen bei der Entwicklung von Simulations- und Animationstechniken aus.

Dabei setzen die meisten Computeranimationen auf comichafte Charaktere, wie im neusten "Universal Studios" Film "Ich - Einfach Unverbesserlich" (siehe Abbildung 2.1). Grund dafür ist, dass der Betrachter ab einem gewissen Punkt der Realitätsnähe die künstliche Figur immer weniger akzeptiert. Erst bei einem sehr hohen Grad an Realismus steigt die Akzeptanz für eine Figur wieder. Erhebt ein Charakter den Anspruch menschlich sein zu wollen, wird er automatisch mit einem Menschen verglichen. Dabei fallen dem menschlichen Gehirn selbst kleinste Fehler im Aussehen oder bei Bewegungen auf. Versucht ein Charakter aber nicht menschlich zu wirken, sucht der Betrachter nach menschenähnlichen Verhalten, wie bekannte Laute oder Bewegungen. Diese Eigenschaften werden ihm zugute geschrieben und somit erscheint er akzeptabler. [JFP06]

Der erste Computeranimationsfilm, der auf realistisch wirkende menschliche Charaktere setzt, ist "Final Fantasy - Die Mächte in dir" (siehe Abbildung 2.2), der im Jahr 2001 erschien. Nicht nur das Aussehen der Charaktere wurde möglichst realistisch gestaltet, sondern auch ihre Bewegungen, die mit Hilfe von Motion-Capturing aufgenommen und auf die Charaktere übertragen wurden. Dennoch ist es nicht gelungen, einen realen Eindruck der Charaktere zu erwecken. [Ric08]

[1] http://screenrant.com/despicable-me-trailer-ross-50468/

Abbildung 2.2: Nahaufnahme von der Hauptfigur Aki Ross aus dem Animationsfilm "Final Fantasy - Die Mächte in dir".[2]

Zwar gibt es kaum Computeranimationsfilme mit realistisch aussehenden Charakteren, jedoch werden diese immer häufiger in Realfilmen eingesetzt. Hier ist es wichtig, dass der Beobachter den Unterschied zwischen der Real- und Animationsfigur möglichst nicht wahrnimmt. So werden virtuelle Charaktere für Massensimulationen oder auch als virtuelle Stuntmen eingesetzt.

Der Film "Avatar - Aufbruch nach Pandora" (siehe Abbildung 2.3) zeigt, dass eine Vermischung von Realfilm und Computeranimation mittlerweile möglich ist. So entstanden etwa 60 Prozent des Films im Computer. Ziel war auch hier, dass die Charaktere realistisch wirken mussten. Durch Motion-Capturing und zusätzliche Techniken konnten 95 Prozent eines Schauspielers auf den virtuellen Charakter übertragen werden. Nicht nur die Bewegungen sondern auch das Aussehen der virtuellen Charaktere sind täuschend echt gelungen.[3]

2.2.2 Interaktive Charaktere

Das wechelseitige Aufeinanderwirken von Akteuren oder Systemen wird als Interaktion bezeichnet.[4] So ist es nicht überraschend, dass interaktive Charaktere in Computerspielen zu finden sind. Nicht-Spieler-Charaktere (NSC) oder "Non Player Character" (NPC) werden vom Computer gesteuert und haben bestimmte Routinen, die Interaktionen mit dem Spieler erlauben. Je nach Spiel und Charakter gibt es unterschiedliche Reaktionen der NSC.[5]

[2] http://www.rpgamer.com/games/ff/affw/ffmov/screens/aki_face.bmp
[3] http://de.james-camerons-avatar.wikia.com/wiki/Avatar_%E2%80%93_Aufbruch_nach_Pandora
[4] http://de.wikipedia.org/wiki/Interaktion
[5] http://de.wikipedia.org/wiki/Nicht-Spieler-Charakter

Abbildung 2.3: Vergleich von realem Bild und künstlich erstelltem Charakter aus dem Film "Avatar - Aufbruch nach Pandora".[6]

Bei Multi-Touch-Spielanwendungen für mobile Endgeräte, wie zum Beispiel das iPhone, wird bei der Umsetzung der Steuerung einer Spielfigur in einem Spiellevel meist auf Konzepte eines "Gamepads" zurückgegriffen. Dabei wird eine Touch-Fläche benutzt, um die Funktion eines Joysticks zu imitieren. Dabei kann mit einem Finger die Blick bzw. Laufrichtung durch Bewegungen auf der Horizontalen- sowie auf der Vertikalen-Achse gesteuert werden. Zusätzlich werden weitere Buttons benutzt, um Zusatzaktionen auszuführen (siehe Abbildung 2.4, links). Es gibt vereinzelt auch Spielanwendungen, bei denen mit einzelnen Touches direkt mit einem Charakter interagiert werden kann. Am Beispiel "Talking Tom Cat" kann indirekte mit ihm Interagiert werden, in dem ihm ein Glas Milch gegeben wird (siehe Abbildung 2.4, rechts). Eine direkte Interaktion ist das Streichen über seinen Bauch bzw. Kopf worauf er zu "schnurren" anfängt.

Neben Charakteren in Computerspielen und Animationsfilmen, gibt es auch immer mehr Charaktere in Software- oder Online-Anwendungen. Im Gegensatz zu Computerspielen, in denen der Nutzer in eine andere Rolle schlüpft und Charaktere steuern kann, bleibt er in diesen Anwendungen er selbst und interagiert mit dem Charakter. Die Charaktere werden an die verschiedenen Zielgruppen angepasst, um ihre Sympathien zu erwerben. Dabei wird nicht nur auf das Aussehen geachtet, sondern auch auf immer glaubwürdigere Emotionen, natürlichere Sprach- und Bewegungsabläufe hingearbeitet.

So haben interaktive Charaktere viele Einsatzgebiete, wie beispielsweise als Assistenten, Trainer, Werbe- und Verkaufspersonal oder Nicht-Spieler-Charaktere in Computerspielen. Zudem können sie die Hemmschwelle des Benutzers senken und die Motivation und Aufmerksamkeit

[6] http://www.cgnews.com/2010/02/autodesk-played-pivotal-role-in-making-of-avatar/

Abbildung 2.4: Links: "Real Football 2011" Steuerung der Spieler über das "Gamepad"-Konzept. Rechts: "Talking Tom Cat" Interaktion durch einfache Touches möglich.[7]

steigern. Sie können Körpersprache einsetzen, körperliche Handlungen vorführen und sich sozial und emotional verhalten. Weiter können sie auch als Bezugsperson angesehen werden und Akzeptanz, beispielsweise für ein Produkt oder eine Firma, beim Benutzer schaffen. Durch interaktive Charaktere werden Benutzer animiert, sich verstärkt mit ihnen auseinander zu setzen. Inhalte können von ihnen schnell und einfach präsentiert werden. [Kra01] [JG02]

Beispiele dafür sind "Eve", Kundenberaterin der Firma "Yello Strom" oder "der Fuchs" von der Firma "Schwäbisch Hall" (siehe Abbildung 2.5, links). Über ein Eingabefeld können Fragen gestellt werden, die der Charakter dann beantwortet. Auch auf Begrüßungen, Beleidigungen oder Komplimente reagieren die Charaktere. Jedoch sind bei Online-Anwendungen die Möglichkeiten der Interaktion mit dem Benutzer eher beschränkt. Hier kann nur mit der Maus oder schriftlich mit Charakteren interagiert werden.

Ein anderes Beispiel für interaktive Charaktere sind die Avatar basierten Navigationssysteme (siehe Abbildung 2.5, rechts) oder auch die vermenschlichte Repräsentation eines Fahrzeugs, die von Volkswagen, Audi und Mercedes verwendet werden. Hier kann der Benutzer über Tasten oder auch über Sprache mit dem Charakter kommunizieren. Der Charakter wird nicht nur als Stimme wahrgenommen sondern erscheint auf einem Display und kann beispielsweise auch Funktionen von Knöpfen in einem Fahrzeug erklären.[8] Jedoch wirken diese Charaktere noch sehr künstlich.

[7] Links: http://www.touchgen.net/wp-content/uploads/2010/09/rf2011_iphone_screen_-4.jpg
Rechts: http://outfit7.com/wp-content/uploads/2010/06/talking_tom_cat_2.jpg
[8] http://www.charamel.com/show_room/automotive_show_cases.html

Abbildung 2.5: Links: Schwäbisch-Hall-Fuchs, der auf der Website Fragen beantwortet. Rechts: Avatar basiertes Navigationssystem von Volkswagen.[9]

An diesen Beispielen wird deutlich, dass Charaktere eingesetzt werden, um Benutzern leichteren Zugang zu Techniken, Informationen, Dienstleistungen oder Produkten zu ermöglichen. Die Charaktere können aber manchmal durch ablenkende Animationen oder ständige Einblendungen auch als störend empfunden werden.

2.3 Animation von Charakteren

> "To animate means to give life to an inanimate object, image or drawing: anima means soul in Latin. Animation is the art of movement expressed with images that are not taken directly from reality. In animation, the illusion of movement is achieved by rapidly displaying still images–or frames–in sequence." [Ker04]

Das Ziel einer Animation ist also, eine Abfolge von Bildern zu produzieren, so dass der Eindruck einer Bewegung entsteht.
Es gibt viele verschiedene Arten von Animationen, wie beispielsweise die "Stop-Motion"-Animation, die "Hand-Drawn"-Animation, "Visual-Effects"-Animation oder auch die Charakteranimation. Jedoch ist nur letztere wichtig für diese Arbeit. In Abbildung 2.6 ist ein Beispiel "Walk-Cycle" (Zyklus einer Gehbewegung) eines animierten Charakters zu sehen. Dabei sind die einzelnen Posen während der Gehanimation dargestellt.

Um virtuelle Charaktere zum Leben zu erwecken, muss das Aussehen festgelegt, die Figur erstellt und die Bewegungen auf den Charakter übertragen werden. Diese künstlerische Freiheit stellt zugleich eine anspruchsvolle und kunstfertige Aufgabe dar. Wenn es darum geht, realistische Animationen von Charakteren zu erschaffen, wird die Aufgabe erschwert,

[9] Links: http://www.schwaebisch-hall.de
Rechts: http://www.charamel.com/solutions/human_machine_interface/automotive.html

Abbildung 2.6: Bildliche Darstellung eines typischen "Walk-Cycle" eines animierten Charakters.[11]

da die Körper von Menschen oder anderen Charakteren meist aus vielen Gelenken bestehen und sich ständig bewegen. [Ker04]
Das menschliche Gehirn kann zudem sehr genau zwischen computergenerierten und echten Bewegungen unterscheiden. Auch wenn Animationen flüssig wirken, sehen animierte Bewegungen trotzdem oft nicht natürlich aus.[10]

2.3.1 Methoden der Charakteranimation

Ein Charakter kann auf verschiedene Arten animiert werden. Neben der freien Animation der Knochen oder einzelner Vertices (Eckpunkte eines Polygonmodells) mit "Key-Frames", gibt es noch die Methoden der vorwärts gerichteten und der inversen Kinematik. Beide Methoden können auch parallel verwendet werden. Immer häufiger wird zur Charakteranimation auch Motion-Capturing verwendet.

2.3.1.1 Key-Frame

Key-Frame bedeutet Schlüsselbild. Die Key-Frame-Animation ist eine elementare und weit verbreitete Methode der Charakteranimation. Ein Key-Frame definiert den Zustand eines Objektes zu einer bestimmten Zeit. Ein Zustand kann dabei die Translation, Rotation oder eine andere Eigenschaft des Objektes sein. Zwischen zwei Key-Frames interpoliert im Normalfall der Computer, d.h. er verändert den Zustand des Objektes über die Zeit bzw. über den Abstand zwischen den Key-Frames. Üblicherweise werden zuerst die Key-Frames für die Hauptpose des Charakters gesetzt und anschließend überarbeitet, um natürlichere Bewegungen zu erhalten. Abhängig von den Fähigkeiten des Animators können bei dieser Methode gute Ergebnisse erzielt werden. Jedoch ist diese Vorgehensweise arbeitsintensiv und zeitaufwendig.[12] [Mah08]

[10] http://www.4.am/Computer/Computer/Human_Motion_Capturing_20050818983.html
[11] http://www.iamthomasvogel.de
[12] http://characteranimation.art-worxx.com, April 2010

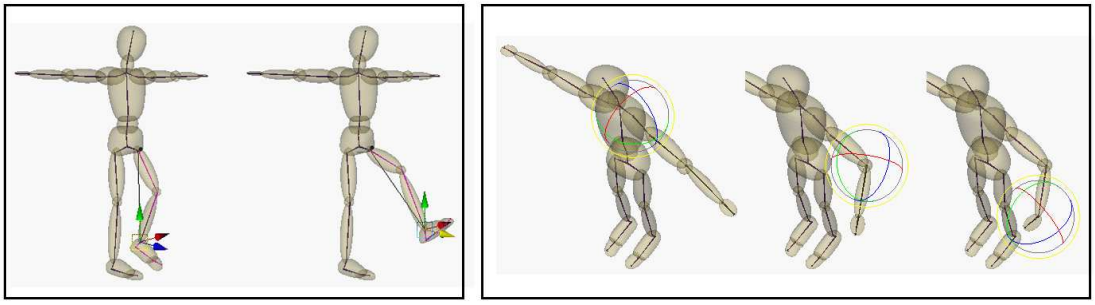

Abbildung 2.7: Links: Darstellung der Funktionsweise der Inversen Kinematik. Rechts: Darstellung der Funktionsweise der Vorwärts gerichteten Kinematik.[13]

2.3.1.2 Vorwärtsgerichtete Kinematik

Die vorwärtsgerichtete Kinematik ist eines der geläufigsten Verfahren zur Manipulation und Steuerung von Bewegungen eines Skelettes. Dabei werden Eigenschaften vererbt. Einzelne Objekte werden miteinander verknüpft. Diese sind bei Charakteren im Normalfall die Knochen und Gelenke. So ergibt sich eine hierarchische Struktur von Knochen und Gelenken, deren Position und Rotation voneinander abhängig ist. Am Beispiel eines Armes kann das Prinzip verdeutlicht werden. Das oberste Glied in der Hierarchie ist in diesem Fall der Oberarm bzw. das Schultergelenk. Untergeordnet sind alle weiteren Gelenke und Knochen des Arms, wie der Ellbogen, das Handgelenk und die Finger. Wird nun das Schultergelenk rotiert, so wirkt sich diese Bewegung auf alle untergeordneten Knochen und Gelenke aus. Die Gelenke und Knochen erben also die Rotation der Schulter und vererben diese, an die in der Hierarchie tiefer stehenden Elemente (siehe Abbildung 2.7, rechts). [Ker04]

2.3.1.3 Inverse Kinematik

> "Inverse kinematics techniques are useful for animating complex models and motion rigs with a large number of joins." [Ker04]

Die inverse Kinematik vereinfacht die Animation von komplexen Modellen, da nicht jedes Gelenk einzeln ausgerichtet werden muss. Bei der inversen Kinematik ist die Funktionsweise entgegengesetzt zu der vorwärts gerichteten Kinematik. So wirkt die Zustandsänderung eines in der Hierarchie untergeordneten Knochens oder Gelenks bis zum obersten Glied der Hierarchie. Wird nun beispielsweise das Fußgelenk ausgewählt und bewegt, so wird nicht nur der Fuß selbst, sondern auch das Knie und die Hüfte beeinflusst. Der "end effector" bezeichnet das unterste Element, das ausgewählte Element, der Hierarchie. Das oberste Element der Hierarchie, in diesem Fall die Hüfte, bleibt während der Bewegung am ursprünglichen Ort. Die Winkel der Elemente werden automatisch berechnet (siehe Abbildung 2.7, links). [DJ06]

[13] http://caad.arch.ethz.ch/info/maya/manual

2.3.1.4 Motion-Dynamics

"Motion-Dynamics" generieren realistische Bewegungen von Objekten, indem die physikalischen Eigenschaften des Objektes und natürlichen Gesetze der physikalischen Bewegung simuliert werden. Ein für den Charakter wichtiger Teil dabei sind die "rigid bodies". Diese werden zur Simulation von Eigenschaften, wie Masse des Objekts und einstellbare Kräfte, wie Wind und Schwerkraft, für eine realistische Animation genutzt. So können Kollisionen oder Stürze eines Charakters gut nachgestellt werden. [Ker04]

2.3.1.5 Prozedurale-Animation

In der prozeduralen Animation werden Algorithmen benutzt, die eine Bewegung errechnen bzw. erzeugen. Die Bewegung wird nur durch die Parameter des Algorithmus beeinflusst. Die Parameter können zum Beispiel durch "User Interfaces" (Benutzerschnittstellen) festgelegt werden, so dass ein Programm alle weiteren Daten für die Animation generiert. Dabei sind die Parameter die Richtlinien, an welche sich die Animation hält. Ein Vorteil dieser Animationsmethode ist, dass die Anzahl der Parameter meist geringer ist als die Informationsmenge einer aufgezeichneten Animation. So sind die mit diesem Verfahren erstellten Animationsdateien oft kleiner als zum Beispiel Dateien einer Motion-Capture-Aufnahme.
Mit diesem Verfahren können komplexe physikalische Prozesse simuliert werden, die nicht durch Key-Framing erzeugt werden könnten. Außerdem kann eine ganze Gruppe von Bewegungen in einem Algorithmus zusammengefasst werden. [DJ06]

2.3.1.6 Performance-Animation

Die Performance-Animation nutzt vorliegende Animationsdaten, welche mit verschiedenen Geräten, wie Datenhandschuhen, Motion-Capture-Systemen oder "Face-Tracking"-Systemen aufgezeichnet werden. Aus diesen Daten wird die Animation des virtuellen Charakters berechnet.
Durch Abtastung der Gliedmaßenbewegung werden überzeugende Daten zur Gliedmaßenanimation gewonnen. Mit Methoden der digitalen Videoverarbeitung wird die Position von Markierungspunkten im Gesicht analysiert, so dass Face-Tracking-Systeme geeignete Animationsdaten für die Gesichtsanimation liefern.[14] Menschliche Bewegungen werden umgesetzt, indem reale Personen mit Sensoren am Körper versehen und mit Spezialkameras aufgezeichnet werden. Die dabei gesammelten Daten können dann auf virtuelle Charaktere übertragen werden, um so möglichst realistische Bewegungen zu erhalten. Zu beachten ist, dass der Begriff Motion-Capturing nur für die Erfassung von Bewegungen und nicht für die Darstellung eines animierten 3D-Modells auf dem Bildschirm, steht. Der gesamte Prozess aus Motion-Capturing und anschließender Animation wird als Performance Animation bezeichnet. Das in der Charakteranimation verbreitete Motion-Capturing ist die schnellste Möglichkeit, um Bewegungen eines Menschen oder Tieres nachzubilden. [Men00]

[14] http://elefant.khm.de/~actor/projects/making_of/making_of_frame.html

Abbildung 2.8: Links: bei den Dreharbeiten zu Herr der Ringe. Mitte: MoCap-Aufnahmen von Gollum im Studio. Rechts: Filmszene mit Gollum aus dem Film "Herr der Ringe".[15]

Die zuvor beschriebenen Animationsmethoden können kombiniert werden, um bessere Ergebnisse zu erzielen. Ein Beispiel dafür ist "Gollum" (siehe Abbildung 2.8, rechts) aus dem Film "Der Herr der Ringe", der aus einer Mischung von Motion-Capturing und Key-Frame-Animation zum Leben erweckt wurde.

2.3.2 Motion-Capturing

Der Begriff "Motion-Capturing" (im folgenden Text auch MoCap genannt) bedeutet soviel wie "Bewegungen einfangen bzw. erfassen". Der Grundstein für MoCap wurde schon im späten 18. Jahrhundert gelegt. Dabei haben die Erfinder, Etienne Jules Marey und Eadweard Muybridge, ein Verfahren entwickelt, in dem sie mit bis zu 24 Kameras Bilder in hoher Frequenz fotografierten.
Ein weiterer wichtiger Schritt war das vom Cartoonisten Max Fleischer 1915 entwickelte Verfahren der Rotoskopie. Hier werden Filmaufnahmen Bild für Bild auf einen Leuchttisch projiziert und dann von Animationszeichnern abgepaust bzw. nachgezeichnet.
Erst 1985 kam der erste vollständig im Computer generierte Werbespot namens "Brilliance", der zudem auch komplett auf MoCap basierte, ins Fernsehen. [TF07]

Seither nimmt die Verbreitung und Entwicklung von MoCap-Systemen (auch Trackingsysteme genannt) und MoCap-Verfahren zu und ist aus vielen Bereichen, wie zum Beispiel Filmen und Computerspielen, nicht mehr wegzudenken. Gollum ist wohl eine der bekanntesten mit MoCap animierten Filmfiguren. Doch für MoCap gibt es nicht nur in der Unterhaltungsindustrie Einsatzmöglichkeiten. Auch in der Biomechanik und Sportmedizin werden diese Verfahren genutzt, um neue Erkenntnisse zu erlangen. In diesen Bereichen werden die Daten jedoch eher zur Analyse von Bewegungen eingesetzt. [Wei04]

Beim Motion-Capturing für einen humanoiden Charakter wird eine reelle Person mit Sensoren versehen. Über die Sensoren werden die Bewegungsdaten aufgezeichnet und anschließend in eine computerlesbare Form überführt. Das verbreitetste Format für solche Daten ist das "Biovision"-Format (BVH).

[15] http://www.herr-der-ringe-film.de

Abbildung 2.9: Ansichten eines mechanischen MoCap-Anzugs mit Exoskelett.[16]

Zur Bewegungserfassung können unterschiedliche Verfahren, wie zum Beispiel optische, magnetische oder mechanische Systeme, eingesetzt werden. Neben diesen werden auch noch andere Verfahren im nachfolgenden Abschnitt vorgestellt.

2.3.2.1 Mechanische Systeme

Bei mechanischen Systemen werden Bewegungen über ein sogenanntes "Exoskelett" aufgenommen. Das Exoskelett ist ein aus starren Segmenten und beweglichen Elementen bestehendes Außenskelett, welches am Körper des Akteurs befestigt wird (siehe Abbildung 2.9). Zu beachten ist, dass das Exoskelett präzise an den Körper des Trägers angepasst werden muss. Dabei soll die Bewegungsfreiheit des Akteurs möglichst wenig eingeschränkt werden. Außerdem sollte der Anzug auf verschiedene Körpergrößen einstellbar und stabil sein, aber auch möglichst wenig wiegen.
Bewegt sich nun der Akteur, so wird auch das Exoskelett mitbewegt. Die an den Gelenken des Exoskeletts befindlichen Sensoren, entweder Potentiometer oder Winkelmessgeräte, liefern dann die Bewegungsdaten. Bei einem Potentiometer wird die Bewegung einer Stange in Spannungswerte umgewandelt, welche anschließend in digitale Signale zur Interpretierung für die Software umgewandelt werden. Winkelmessgeräte liefern direkt digitale Messergebnisse. Zur Orientierungsbestimmung der Hüfte und für die Erfassung der Bewegungen eines Akteurs im Raum, wird ein Gyroskop (Kreiselinstrument) an der Hüfte angebracht.[17] So können alle Körperbewegungen rekonstruiert werden.
Um Daten über das Exoskelett zu erhalten und damit indirekt auf den Träger zu schließen, erfordert das mechanische Trackingsystem eine Kalibrierung. Dabei müssen die Abstände zwischen den Gelenken bzw. den Rotationsmittelpunkten der Sensoren vermessen werden.

[16] http://www.inition.co.uk/inition/product.php?URL_=product_mocaptrack_animazoo_gypsy
[17] http://www.metamotion.com/gypsy/gypsy-motion-capture-system-mocap.htm

Das bedeutet, dass für jeden Akteur eine individuelle Kalibrierung notwendig ist. Eine solche Kalibrierung nimmt ca. eine halbe Stunde in Anspruch. Sind jedoch bereits Daten eines Akteurs für die Einstellung des Systems vorhanden, kann schnell und unkompliziert gearbeitet werden.

Da die meisten Produkte dieses Systems schnurlos sind und Daten per Funk an die Rechner übertragen werden, ist die Reichweite ein Vorteil von mechanischen MoCap-Systemen. Es existieren mechanische MoCap-Systeme mit Funkeinheiten, die Reichweiten für Echtzeitaufnahmen von bis zu einem Kilometer erreichen. Dies kann noch erweitert werden, indem ein Notebook auf dem Rücken des Akteurs befestigt wird, der dadurch ortsunabhängig agieren kann. Das gesamte System kann einfach transportiert werden, da es aus nur wenigen Komponenten besteht. Zudem können solche Systeme auch für Echtzeitanwendungen eingesetzt werden und auch beim Interagieren mehrerer Akteure gibt es keine Probleme, wie das Verdecken oder das Stören von Sensoren.

Der wohl größte Nachteil ist die durch das Exoskelett eingeschränkte Bewegungsfreiheit, so dass kompliziertere bzw. akrobatische Bewegungen, wie beispielsweise Stürze oder Kampfsequenzen, praktisch nicht umsetzbar sind. Allerdings sind die geringeren Kosten ein weiterer Vorteil eines mechanischen Systems. [Hor02]

2.3.2.2 Magnetische Systeme

Bei magnetischen Systemen messen die am Körper des Akteurs angebrachten Sensoren ein niederfrequentes magnetisches Feld, welches von einer Transmitter-Einheit (Sender) ausgesendet wird. Wie auch bei anderen Systemen werden die Sensoren hier an den Gelenken und wichtigen Körperstellen angebracht. Die Transmitter-Einheit ist fest im Raum installiert und dient als Referenzpunkt. Über ein geschirmtes Kabel, werden die gemessenen Daten an eine elektronische Kontrolleinheit übertragen. Diese berechnet dann die Ausrichtung und Position im dreidimensionalen Raum. Ein Nachteil dabei ist, dass von jedem Sensor ein Kabel zur Kontrolleinheit führt und somit die Bewegungsfreiheit des Akteurs eingeschränkt wird. Das wiederum hat zur Folge, dass nicht beliebig viele Sensoren angebracht und auch komplexere Bewegungen nicht ausgeführt werden können. Doch es gibt auch Produkte bei denen die Kabel am Rücken des Akteurs zusammenlaufen, wo sich eine kleine Kontrolleinheit befindet. Von dort aus werden die Daten per Funk an einen Rechner übertragen. Damit kann die Bewegungsfreiheit verbessert werden, aber ist dennoch durch den "Rucksack" am Rücken eingeschränkt.

Zwei Faktoren spielen bei der Reichweite eines solchen Systems eine Rolle. Zum Ersten wird sie durch die Kabelverbindung zwischen Akteur und Transmitter bestimmt. Der andere Faktor ist die Leistungsfähigkeit des Transmitters, der Magnetfelder nur über relativ geringe Distanzen erzeugen kann. Oft beträgt ein solcher Bereich ca. acht mal acht Meter.
Auch das gleichzeitige Aufnehmen von Bewegungen mehrerer Akteure stellt ein Problem dar. Zwar gibt es keine Verdeckungsprobleme der Marker wie bei optischen Systemen, doch es kann zu Interferenzen zwischen den Sensoren führen, wenn sie sich näher kommen. Dabei

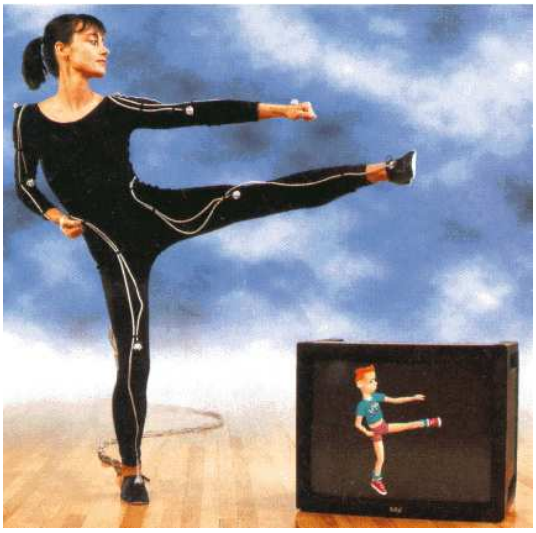

Abbildung 2.10: Ansicht eines magnetischen MoCap-Anzugs bei einer Echtzeitanimation.[18]

kann die Qualität der Daten beeinflusst werden. Aber nicht nur die Sensoren untereinander können Interferenzen hervorrufen. Vor allem elektrische Geräte, metallische Gegenstände bzw. Metall ganz allgemein, beispielsweise in Böden oder Wänden, können dazu führen, dass Interferenzen entstehen. Da Magnetfelder von vornherein schlecht auszumachen sind, müssen diese Störsignale aus der Umgebung gefiltert werden. Jedoch bringt dies den Nachteil mit sich, dass die Abtastrate nicht so hoch wie bei optischen Systemen ist. [Hor02]

Bei der Kalibrierung magnetischer Systeme wird der Akteur, wie auch bei mechanischen Systemen, vermessen und die Sensoren werden den Gelenken zugeordnet.
Vorteile solcher Systeme sind, der geringere Preis im Vergleich zu optischen Systemen, die Möglichkeit in Echtzeit zu "capturen" (siehe Abbildung 2.10) sowie das zeitgleiche Agieren mehrerer Akteure, ohne das Marker verdeckt werden.[19]

2.3.2.3 Optische Systeme

Die wohl bekanntesten MoCap-Systeme sind die optischen, bei denen der Darsteller einen hautengen Anzug trägt, auf dem Marker angebracht sind. Marker sind reflektierende Punkte oder Kugeln. Eine weniger verbreitete Art von Markern sind pulsierende "LEDs" ("Light Emitting Diode" bzw. Leuchtdiode), welche selbst Licht ausstrahlen. Eine wichtige Rolle spielt die Positionierung der Marker, um Bewegungen möglichst gut zu erkennen. Vor allem an Stellen wie Nacken, Schultern, Ellenbogen, Handgelenken, Beckenknochen, Knien und Fußgelenken und auch am Kopf werden diese typischerweise angebracht. An Armen, Händen, Rücken, Ober- und Unterschenkeln und Füßen werden meist zusätzliche Marker

[18] http://www.answers.com/topic/motion-capture#Magnetic_systems
[19] http://www.ascension-tech.com & http://www.polhemus.com

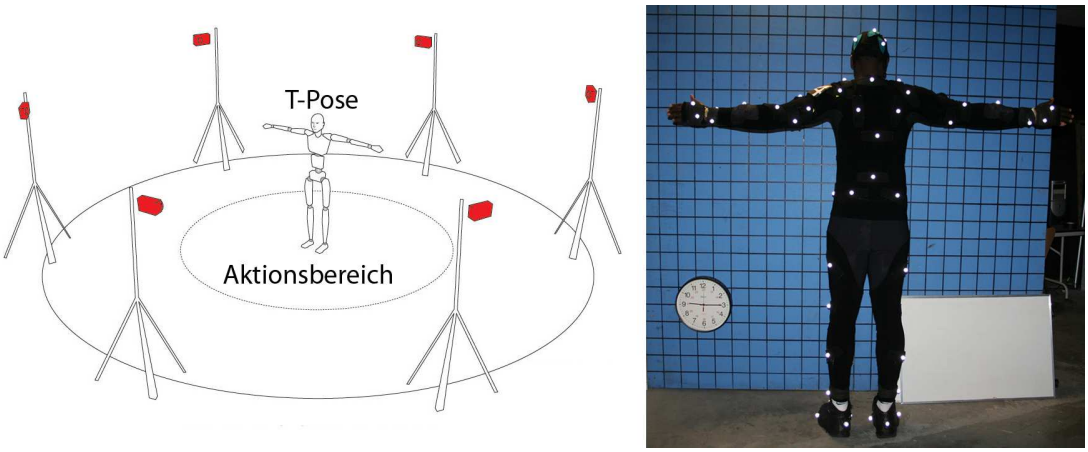

Abbildung 2.11: Links: schematischer Aufbau eines optischen Motion-Capturing-Systems mit sechs Kameras. Rechts: mit Markern besetzter Anzug für Motion-Capturing.[20]

angebracht, denn mit zunehmender Anzahl der Marker wird eine genauere Rekonstruktion der Bewegungen möglich. Allerdings sollte dabei beachtet werden, dass dadurch auch die zu verarbeitende Datenmenge steigt. Vor Beginn der Aufnahmen müssen die Kameras kalibriert werden. Zu diesem Zweck wird ein Objekt mit bekannten Maßen, zum Beispiel ein Würfel, von allen Kameras gefilmt. So kann eine Ausrichtung anhand der bekannten Werte vorgenommen werden. [Hor02]

Beim "ARTtrack"- und "Dtrack"-System der Firma "A.R.T. GmbH" wird zur Raumkalibrierung ein Winkel in das Volumen (Aktionsbereich) gelegt, auf dem Marker in festen Abständen angebracht sind. Dabei muss der Winkel von mindestens zwei Kameras gesehen werden, damit die Richtung der Koordinatenachsen des Weltkoordinatensystems festgelegt wird. Ein mit zwei Markern besetzter Stab wird dann so bewegt, dass ein großer Bereich im Messvolumen abgedeckt wird. Dabei sollte er möglichst im Sichtbereich aller Kameras bleiben. [Ste04]

Die Kameras nehmen das reflektierte Infrarotlicht der Marker auf und geben es an die Rechner weiter. Hierbei gilt, je höher Auflösung und Bildwiederholungsrate der Kameras ist, desto genauer können auch schnelle Bewegungen aufgenommen werden. Es gibt Systeme, die mit einer maximalen Bildwiederholungsrate von bis zu 250 "Frames per Second" (fps) (auch Bilder pro Sekunde genannt) bei einer Messungenauigkeit von +/- 0,2 mm arbeiten. Mit der Anzahl der Kameras wird der Aktionsbereich des Akteurs abgesteckt (siehe Abbildung 2.11, links). Mehr Kameras bedeuten einen größeren Aktionsbereich.

[20] Links: http://www.2connect.eu/optitrack/index.php/bodymotioncapture
Rechts: http://www.easports.com/news/item/file/NBA-LIVE-Motion-Capture

Vor jeder Aufnahme muss der Akteur in der Mitte des Aktionsbereichs die sogenannte "T-Pose" einnehmen. Dabei stellt er sich mit zur Seite ausgestreckten Armen hin (siehe Abbildung 2.11, rechts). Dies ist erforderlich, da zu Beginn der Aufnahme alle Marker zu sehen sein müssen. Außerdem kann die "T-Pose" als Referenz für Anfang und Ende einer Bewegung dienen.

Die Vorteile optischer Systeme sind die hohe Messgenauigkeit, ein großes Arbeitsvolumen, große Bewegungsfreiheit und das gleichzeitige Aufzeichnen von mehreren Akteuren. Vorteil ist auch, dass Gegenstände, wie beispielsweise Stühle oder Schwerter, mit eingebunden werden können. Ein Nachteil optischer Systeme ist, dass nur die wenigsten, wie zum Beispiel Systeme der Firma "Vicon", Daten in Echtzeit erzeugen können, da fast immer Nachbearbeitungen erforderlich sind. Bei der Nachbearbeitung müssen Unsauberkeiten, die bei der Aufnahme entstehen können, beseitigt werden. Dazu gehört unter anderem das Glätten, wenn Marker zittern. Werden Marker verdeckt, können diese in der späteren Erkennung durch die Software nicht wiedergefunden werden und müssen manuell zugewiesen und identifiziert werden. Somit sind auch die längeren Arbeitszeiten und auch die höheren Kosten der Kameras, im Vergleich zu anderen Systemen, ein Nachteil. [Hor02]

2.3.2.4 Sonstige Systeme

Neben den bisher vorgestellten Systemen gibt es auch noch die akustischen Motion-Capturing-Systeme. Diese messen die Laufzeit eines Ultraschallsignals, welches von Lautsprechern an ein Mikrofon gesendet wird. Dadurch erhalten sie Werte die zur Positionsbestimmung dienen. Diese Methode ist im Vergleich zu den anderen Systemen wesentlich ungenauer. Dafür kann ein solches System aber an außergewöhnlichen Orten, wie beispielsweise unter Wasser, eingesetzt werden. Ähnliche Systeme gibt es auch mit Funksignalen. [DJ06]

Ein Verfahren, ähnlich den mechanischen Systemen, ist das Biegesensor System. Dabei werden am Körper des Akteurs flexible Bänder angebracht. Zur Positionsbestimmung wird dann durch verschiedene Methoden die Biegung der Bänder gemessen. Aus dem Wissen über die Körperstruktur und der Position der einzelnen Bänder kann so eine Haltung des Akteurs rekonstruiert werden. Allerdings wird, wie auch beim mechanischen System, ein zusätzliches Verfahren, um die Position im Raum zu bestimmen, benötigt.[21]

Unter den Motion-Capturing-Systemen gibt es auch Systeme, die sich auf bestimmte Bereiche spezialisiert haben. "Face-Tracker" zum Beispiel sind kleine optische Systeme, die sich auf Gesichtsanimationen konzentrieren. Dabei werden an wichtigen Stellen wie Augenbrauen, Mund, Wangen und Kinn kleine Marker platziert und von einer oder mehreren Kameras aufgenommen. Die Schwierigkeit bei diesen Systemen ist, dass bei nur geringfügiger Abweichung eines Muskels eine gegenteilige Mimik entstehen kann. Durch sehr gute Genauigkeit bei der Markererkennung, die für das komplizierte Zusammenspiel von Gesichtsmuskeln notwendig ist, sind optische Systeme als Face-Tracker prädestiniert.

[21] http://www.motion-capture-system.com/

Abbildung 2.12: Links: Dataglove bzw. Datenhandschuh zur Emittlung von Handbewegungen. Rechts: Markerbesetztes Gesicht für Cative-Motion-Aufnahmen.[24]

Eines der momentan besten Systeme dafür wird von der Firma "Captive-Motion" genutzt (siehe Abbildung 2.12, rechts). Im Gegensatz zum früheren MoCap-Verfahren, bei denen nur wenige Punkte gesetzt und dann viel vom Computer interpoliert wurde, kann jetzt mit bis zu 1400 Punkten im Gesicht "gecaptured" werden.[22]

Ein anderes spezialisiertes System ist unter dem Begriff "Dataglove" (Datenhandschuh) bekannt (siehe Abbildung 2.12, links). Hier werden die Bewegungen der Hand, beispielsweise für die Navigation in virtuellen Welten, verarbeitet. Dabei gibt es Systeme die mit Plastik oder Metallstreifen in den Handschuhen arbeiten. Pro Finger wird ein Streifen eingesetzt oder Sensoren werden angebracht, welche die Krümmung und Spreizung der Finger und die Bewegung des Handgelenks messen. Die Daten werden dann von einer am Rücken befindlichen Einheit an den Rechner übermittelt.[23]

Das wohl erste markerlose Motion-Capture-System wurde von der Firma "organic motion" entwickelt.[25] Dabei nehmen zehn Kameras eine Akteur in einem Volumen von 4x4x2,5m auf. Die menschliche Anatomie wird erkannt und kann in "MotionBuilder", eine Animationssoftware der Firma "Autodesk", in Echtzeit auf einen virtuellen Charakter übertragen werden.[26] Ein weiteres Beispiel für markerlose Motion-Capture-Systeme ist "Contour" der Firma "Mova". Dieses beschränkt sich jedoch nur auf Geschichtsanimation. Das Gesicht wird dabei mit einem phosphoreszierenden Make-Up bedeckt, welches für das menschliche Auge aber

[22] http://www.cgheute.de/2009/08/metricminds-exclusiv-mit-captivemotion

[23] http://www.x-ist.de/product_info.php?info=p22_X-IST-DataGlove-SP1-System.html

[24] Links: http://www.inition.co.uk/inition/product.php?URL_=product_glove_5dt_datagloves&SubCatID_=26
Rechts: http://www.cgheute.de/wp-content/uploads/2009/08/p_002_metric_011.jpg

[25] http://www.heise.de/newsticker/meldung/SIGGRAPH-Motion-Capture-guenstig-oder-markerlos-161043.html

[26] www.organicmotion.com

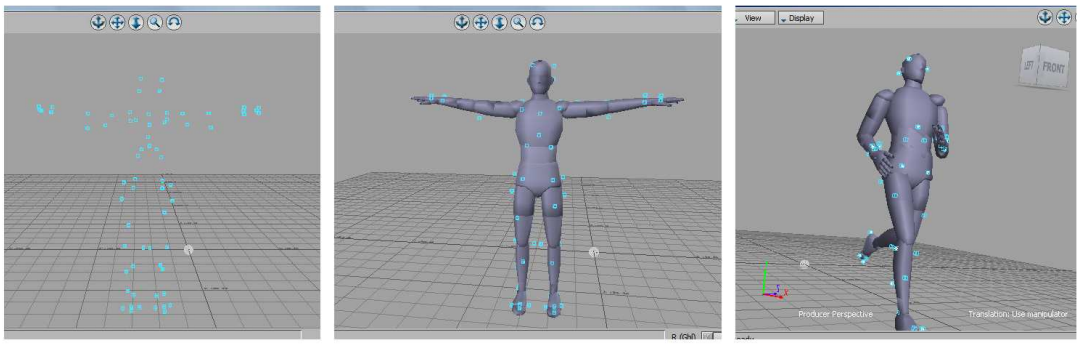

Abbildung 2.13: Links: Ansicht der Marker in MotionBuilder. Mitte: Ausrichten des "Actros" an den Markern. Rechts: Laufanimation des "Actors".

unsichtbar ist. Zwei Sätze an Kameras und Beleuchtung werden mit hoher Frequenz umgeschaltet und erfassen parallel Farbe und Form eines Gesichts.[27] [Lov07]

Die Kombination mehrerer MoCap-Systeme wird als hybrides System bezeichnet. Hier sind alle denkbaren Kombinationen vorstellbar, um die Vorteile der einzelnen Systeme zu nutzten. So kann je nach Bedarf zum Beispiel ein mechanisches System mit einem Face-Tracking System verbunden werden, um eine große Reichweite zu ermöglichen und gleichzeitig auch die Gesichtsmimik des Akteurs einzufangen. [Hor02]

2.3.2.5 Nachbearbeitung der Daten

Bei optischen Motion-Capturing-Systemen werden Marker, die am Körper angebracht sind, zur Bewegungserfassung verwendet. Der dabei resultierende Abstand zwischen dem echten Skelett und den Markern, besonders an Muskeln und Gelenken, muss berücksichtigt werden. Bei der Verdeckung von Markern müssen diese neu zugeordnet und identifiziert werden. Aus diesen Gründen ist ein Nachbearbeiten der aufgezeichneten Daten notwendig.

Ziel der Nachbearbeitung ist, die Rohdaten in ein für 3D-Software lesbares Format, beispielsweise das "C3D"-Format, umzuwandeln. Dazu gehört neben der Fehlerbeseitigung auch die Datenbereinigung und Anpassung der Daten an den Charakter. Der Arbeitsablauf sieht wie folgt aus:
Die Daten werden in eine spezielle Software geladen, wo sie visualisiert werden können. Ein solcher Datensatz besteht aus sich im Raum bewegenden Punkten bzw. Markern. Einige Hardwaresysteme, wie beispielsweise das Vicon-System, beinhalten eine eigene Software zur Nachbearbeitung der aufgenommenen Daten. Hier werden die einzelnen Marker benannt und die aufgezeichneten Daten auf Lücken und Unsauberkeiten überprüft und bereinigt. Anschließend können die fertigen Daten in eine andere Software, wie beispielsweise MotionBuilder, übertragen werden. In dieser Software können die Daten nun ohne weiteres an

[27] http://www.mova.com/

einen Charakter gebunden werden. Zu beachten ist, dass das Skelett des Charakters an die MoCap-Daten angepasst werden sollte. Dieser Vorgang wird als "retargeting"[28] bezeichnet. Damit werden Fehler in der Animation verhindert, wie zum Beispiel das Schweben kleinerer Charaktere über dem Boden oder Proportionsänderungen des Charakters. Als letzter Schritt können nun Feinheiten an der Animation ausgearbeitet und verbessert werden. [DJ06]

2.4 Multi-Touch-Systeme

Im Vergleich zu herkömmlichen Single-Touch-Displays, wie zum Beispiel die Informationsterminals für die Fahrplanauskunft der Deutschen Bahn, bieten Multi-Touch-Systeme viele neue Interaktionsmöglichkeiten, von denen bisher nur wenige etabliert sind.

2.4.1 Technologien

Mittlerweile existieren verschiedene Technologien auf dem Markt, die es ermöglichen Berührungen auf Oberflächen zu erkennen. Jede Einzelne hat unterschiedliche Vor- bzw. Nachteile und ist in ihrem Einsatzgebiet durch ihre Form und Größe beschränkt. Der Preis variiert stark zwischen günstigen Lösungen der Marke "Eigenbau" bis zu technisch aufwendigen, professionellen Technologien. In dieser Arbeit wird lediglich eine Auswahl von Verfahren vorgestellt.

Im Ganzen betrachtet können diese in zwei Kategorien unterteilt werden. [UL09]

- Visuelle Erkennung der Berührungspunkte auf der Oberfläche
- Erkennung durch seitlich zur Oberfläche angebrachter Sensorik bzw. in der Fläche integrierte Sensorik

2.4.1.1 Diffused Illumination (DI)

Die einfachste sowie kostengünstigste Lösung ist das sogenannte "Diffused Illumination"-Verfahren. Bei diesem Verfahren sind meist ein Beamer und eine Kamera die teuersten Bauteile. Der Beamer projiziert sein Bild von unten, wie in Abbildung 2.14 beschrieben, auf eine Acrylplatte. Bei Bedarf kann die Projektion des Beamers mittels eines Spiegels umgelenkt werden, um so die Bauhöhe zu verringern. Mit Infrarotstrahlern wird die Acrylplatte gleichmäßig ausgeleuchtet. Die verwendete Kamera wird mit einem Filter ausgestattet, der das sichtbare Licht filtert und nur das reflektierte Infrarotlicht von den "IR"-Strahlern (Infrarotstrahlern) durchlässt. Aus Sicht des Beamers wird die Kamera auf die Acrylplatte ausgerichtet.

Wenn sich ein Finger der Platte nähert, wird immer mehr infrarotes Licht reflektiert. Die Kamera nimmt so bei einer Berührung einen hellen Fleck auf.
Die Qualität der Berührungserkennung ist von der verwendeten Kamera abhängig. Hier

[28] http://de.wikipedia.org/wiki/Motion_Capture

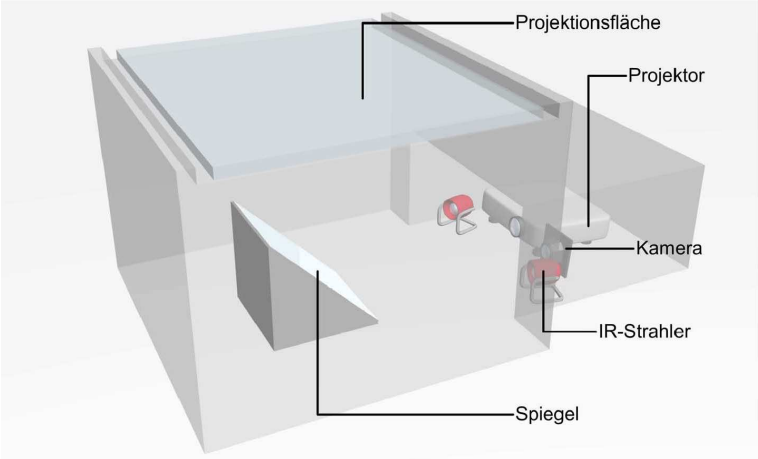

Abbildung 2.14: Schematische Darstellung eines "Diffused Illumination"-Aufbaus. [Bad08]

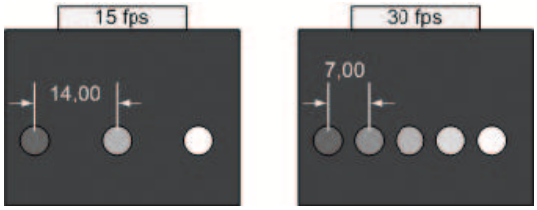

Abbildung 2.15: Abstand der Punkte bei verschiedenen Frameraten und gleicher Bewegungsgeschwindigkeit. [Bad08]

setzten die Auflösung sowie die Bildfrequenz Grenzen. Bei einer zu geringen Auflösung wird die Zuordnung der exakten Position einer Berührung erschwert. Jedoch führt eine hohe Auflösung bei den meisten Kameras zu einer niedrigen Bildwiederholungsrate. Gleichzeitig steigt der Zeitversatz bei der Signalübertragung.
Da die Kameras Einzelbilder und keine Bewegungen übertragen, spielt die Bildfrequenz eine wichtige Rolle. Das System muss entscheiden können, ob eine Bewegung eines bereits erkannten Punktes vorliegt, ob es sich um einen neuen Punkt handelt oder ob ein bereits erkannter Punkt entfernt wurde. Dies wird über einen Schwellwert des Abstandes erkannter Punkte berechnet, der bei einer höheren Bildfrequenz niedriger gewählt werden kann, da die Bilder in zeitlich geringerem Abstand folgen (siehe Abbildung 2.15). Somit ist der Weg, der bei einer Bewegung zwischen zwei Bildern zurückgelegt wird, bei gleicher Bewegungsgeschwindigkeit kleiner. Wird der Schwellwert überschritten, wird der Punkt als ein neuer erkannt. [Bad08][Mul08][Aut09]

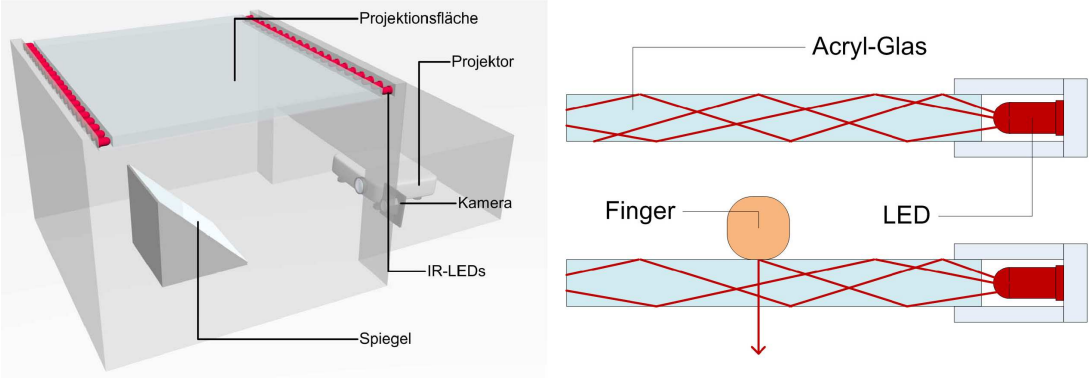

Abbildung 2.16: Links: Schematische Darstellung eines "Frustrated Total Internal Reflection"-Aufbaus. Rechts: Schematische Darstellung des FTIR-Effekts. Oben werden die Lichtstrahlen "rot" eingezeichnet an den Grenzflächen der Scheibe vollständig reflektiert. Unten berührt ein Finger die Projektionsfläche. In diesem Bereich findet keine Totalreflexion statt. [Bad08]

2.4.1.2 Frustrated Total Internal Reflection (FTIR)

Der Aufbau des "Frustrated Total Internal Reflection"-Verfahrens ähnelt dem DI-Aufbau, der im Abschnitt 2.4.1.1 beschrieben wurde. Der Unterschied liegt bei dem Beleuchtungsverfahren und somit auch bei der Beschaffenheit der Projektionsfläche. Beim FTIR-Verfahren wird das Display in den meisten Fällen von Infrarot-LEDs von den Rändern seitlich gleichmäßig beleuchtet. Die Scheibe ist anders als beim DI-Verfahren eine klare durchsichtige Acrylglasplatte. Zur Erkennung von Berührungen nutzt das FTIR-Verfahren den physikalischen Effekt der Totalreflexion.(siehe Abbildung 2.16 links). [Fri10]

> "Der Effekt der Totalreflexion beruht auf der Tatsache, dass Licht am Übergang zwischen zwei optisch unterschiedlich dichten Medien abhängig vom Winkel in dem es auftrifft, gebrochen wird." [UL09]

Dieser Effekt tritt auf, wenn ein Lichtstrahl in einem bestimmten Winkel auf die Grenzfläche von einem optisch dichteren Medium zu einem optisch dünneren Medium trifft. Der Lichtstrahl wird an der Grenzfläche vollständig reflektiert.

Durch die Verwendung und Beleuchtung der klaren Scheibe ist eine zusätzliche Scheibe für die Projektion von Nöten. Wenn die Projektionsscheibe unter der anderen eingebaut wird, entsteht ein unerwünschter Versatz zwischen dem Bild und der Ebene die berührt wird. Die bessere Variante ist, die Scheibe über der Berührungsschicht anzubringen. Dabei ist die Auswahl des Materials für die Projektionsschicht entscheidend. Da bei einer Berührung, zum Beispiel mit einem Finger, die obere Schicht die untere berühren muss. Hierbei ist es wichtig, dass sie nach einer Berührung oder bei einer Verschiebung des Druckpunktes nicht auf der anderen Schicht kleben bleiben darf.

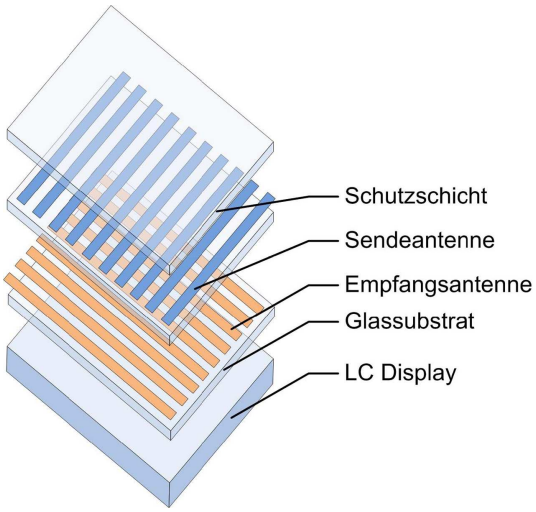

Abbildung 2.17: Schematische Darstellung eines Aufbaus des projektiv-kapazitiven Verfahrens. [Bad08]

Eine Acrylplatte ist im Vergleich zu Luft ein optisch dichteres Medium, d.h. es hat einen höheren Brechungsindex als Luft. So findet das Prinzip der Totalreflexion auch hier Anwendung. Wenn nun zum Beispiel durch eine Berührung mit einem Finger ein Kontakt zur oberen Scheibe hergestellt wird und sich zwischen Finger und Scheibe keine Luft mehr befindet, besteht auch kein Übergang zu einem optischen dünneren Medium. Daher findet auch keine Totalreflexion mehr statt und der Lichtstrahl trifft auf den Finger. Ein Teil des Lichtes wird senkrecht vom Finger reflektiert und trifft auf die Kamera (siehe Abbildung 2.16 rechts). Der große Vorteil dieses Verfahrens ist, dass nur Objekte erkannt werden, die tatsächlich in Berührung mit der Scheibe sind. [Bad08][Aut09]

2.4.1.3 Projektiv-Kapazitive Verfahren

Seit der Einführung des iPhones von Apple im Jahre 2007, werden in mobilen Multi-Touch-Geräten häufig kapazitive Verfahren verwendet. Dieses Verfahren ermöglicht eine gute Berührungserkennung bei Tageslicht und ist auch in kleinen Geräten verbaubar.

> "Die Berührungserkennung basiert auf dem Prinzip der kapazitiven Kopplung zwischen einem Sender und einer Antenne." [Trü07]

Dabei werden die Spannungen gemessen, die bei der kapazitiven Kopplung auftreten. Die kapazitive Kopplung bezeichnet die Übertragung von Energie von einem Schaltkreis zu einem anderen. Wie in Abbildung 2.17 dargestellt, werden beim projektiv-kapazitiven Verfahren über die eigentliche Displayschicht, bestehend aus einem regulären Flüssigkristalldisplay, die Sende- sowie die Empfangsantennen positioniert. Die Empfangsantennen, die auch für die Messung der Spannungsänderungen zuständig sind, verlaufen in vertikaler Richtung. An den horizontal verlaufenden Sendeantennen liegt eine Spannung an, die zur Erzeugung

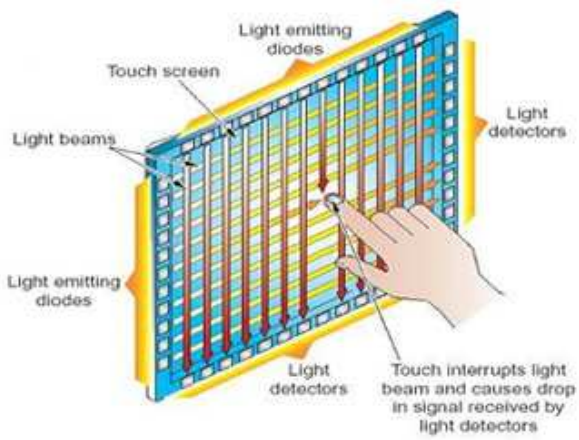

Abbildung 2.18: Prinzip des IR-Lichtvorhang-Verfahrens.[30]

eines elektrischen Feldes dient. Bei der Berührung des Bildschirms mit einem Finger lässt in den umliegenden Bereichen die kapazitive Kopplung nach. Der Druckpunkt kann mittels des Spannungsabfalls exakt ermittelt werden. Mit Hilfe bikubischer Interpolation und der Bestimmung der gemessenen Spannungen kann die Position des Fingers genau berechnet werden.[29] [Trü07][Bad08]

2.4.1.4 Infrarot (IR) - Lichtvorhang

Die Technik des IR-Lichtvorhangs ermöglicht einen sehr flachen Aufbau von ca. 5 - 20mm, da bei diesem Verfahren weder ein Projektor noch Kameras verwendet werden.
Bei dieser Technik werden an zwei nicht gegenüberliegenden Seiten des Displays bzw. Rahmen zwei IR-Leuchtdioden angebracht. Jeder IR-Leuchtdiode ist auf der gegenüberliegenden Seite eine Empfangseinheit (IR-Licht-Detektor) zugeordnet.
Bei einer Berührung der Displayscheibe werden bestimmte IR-Leuchtdioden verdeckt, so dass die Verbindung zu ihren Detektoren gestört wird (siehe Abbildung 2.18). Bei jedem Berührungspunkt werden so zwei IR-Leuchtdioden in X- und Y-Richtung verdeckt und der Punkt kann bestimmt werden. Bei diesem Verfahren ist es daher möglich, die Größe des berührenden Objektes zu ermitteln. Da bei kleinen Objekten, wie zum Beispiel einer Stiftspitze, wenige bis nur eine Lichtquelle, aber bei größeren Objekten, wie zum Beispiel eine Hand, eine ganze Reihe, verdeckt wird.
"Ghost-Touches" (falsch erkannte Punkte) sowie Abschattungseffekte sind bei dieser Technik ein großes Problem. Werden zwei oder mehr Punkte "zeitgleich" auf einer Achse ermittelt, ist das System nicht in der Lage, zu entscheiden, welcher der X-Werte zu welchem Y-Wert gehört.

[29] http://electronics.howstuffworks.com/iphone2.htm
[30] http://web.tradekorea.com/upload_file/emp/200903/sub/453981_sub1.jpg

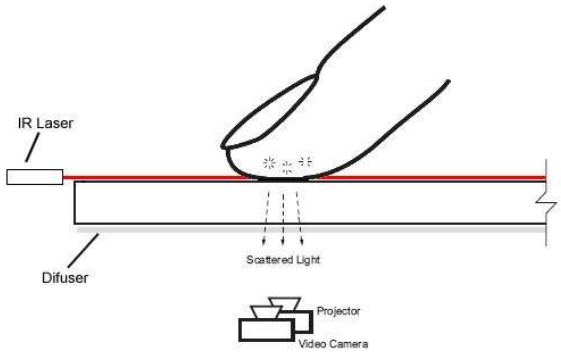

Abbildung 2.19: Schematische Darstellung eines Laser Light Plane Aufbaus.[31]

> "Das Resultat bei zwei Punkten sind beispielsweise 2 korrekt erkannte Punkte und zwei "Ghost-Touches", also insgesamt 4 Punkte." [UL09]

Wenn sich zwei oder mehr bewegende Punkte auf der X- und/oder Y-Achse kreuzen, verlieren sie ihre Zuordnung. So kann dem kreuzenden Punkt ein X- oder Y-Wert des anderen Punktes zugeteilt werden. Die Behebung dieses Problems ist aber zum Teil softwareseitig möglich. [UL09]

2.4.1.5 Laser Light Plane (LLP)

Das LLP-Verfahren ist von der Funktionsweise und dem Aufbau mit den Prinzipien DI sowie FTIR identisch. Die Unterscheidung liegt lediglich in der Abtastung einer Berührung, die bei LLP mit einem Infrarotlaser durchgeführt wird.
Bei dieser Technik werden ein oder mehrere Infrarot-Lasermodule mit einer speziellen Linienlinse eingebaut. Dadurch wird eine Infrarotlicht-Fläche auf der Oberfläche der Scheibe aufgespannt. Wenn jetzt ein Gegenstand oder ein Finger diese Fläche durchdringt erkennt die Infrarotkamera dies und der Punkt kann berechnet werden (siehe Abbildung 2.19). [Aut09][UL09]

2.4.1.6 IR-Light Plane (IRLP)

"IR Light Plane" ist ein weiteres Verfahren, dass sich technisch und in der Funktionsweise dem des IR-Lichtvorhangs sehr ähnelt. Der Unterschied liegt im Aufbau des Verfahrens.
In zwei gegenüberliegenden Ecken der Oberfläche befinden sich Sensoren/Emitter-Module. Die Emitter spannen einen Lichtvorhang über das Display auf und die Sensoren erkennen anhand komplexer Algorithmen Unterbrechungen der Lichtfläche (siehe Abbildung 2.20). [UL09]

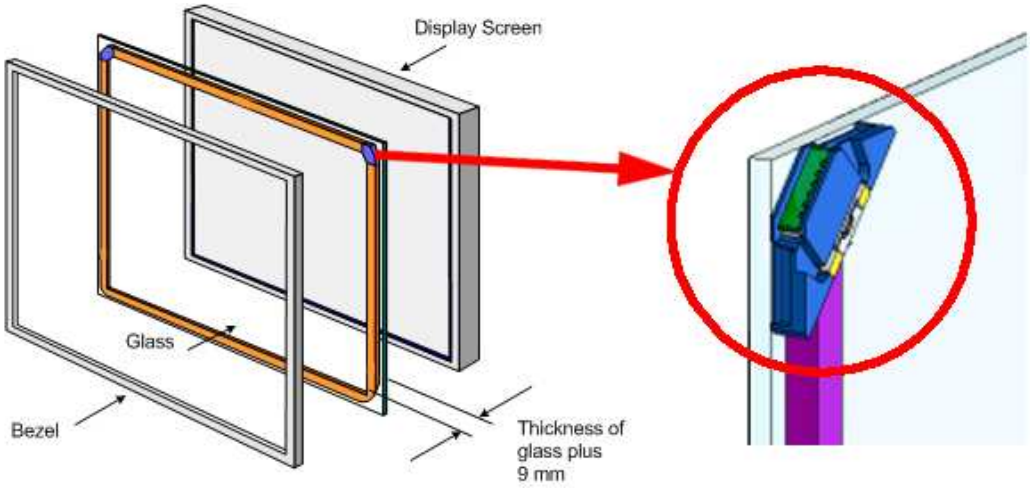

Abbildung 2.20: Einbau eines IR-Light Plane Rahmens in ein Standard LCD-Display sowie eine Vergrößerung eines IRLP-Infrarot-Emitters und IRLP-Infrarot-Sensors.[32]

2.4.2 Hardware

Zu den Technologien, die im Abschnitt 2.4.1 aufgeführt wurden, werden im nachfolgenden Kapitel einige Hardwarelösungen kurz vorgestellt. Prinzipiell lassen sich die Hardwareangebote in zwei Bereiche aufteilen. Es gibt "offen" verfügbare oder "geschlossen" in einem Gesamtsystem integrierte Lösungen. Ein offenes System besteht in den meisten Fällen lediglich aus einem Multi-Touch-Display oder einem Multi-Touch-Rahmen, der auf ein Display gesetzt werden kann. Eine weitere benötigte Komponente ist zum Beispiel ein Rechner. Ein sogenanntes "geschlossenes" System stellt eine Gesamtlösung dar und benötigt, um in Betrieb genommen zu werden, keine weiteren Komponenten. Ein geschlossenes System kann in der Regel aber nicht beliebig erweitert werden. [UL09]

2.4.2.1 Microsoft - Surface

Der von Microsoft 2009 auf den Markt gebrachte Multi-Touch-Tisch "Surface" basiert auf dem DI-Prinzip. Der Tisch besitzt eine Bilddiagonale von 30 Zoll und ermöglicht so eine Interaktion mit mehreren Nutzern gleichzeitig (siehe Abbildung 2.21 rechts). Die Technologie DI unterstützt die Erkennung von Gegenständen, die auf der Projektionsfläche platziert werden, mittels visueller Objekterkennung.
Die sogenannten "Fiducial" haben an der Unterseite angebrachte Tags (eindeutige Markierungen). Mit den verwendeten "Domino-Tags" (siehe Abbildung 2.22) werden standardmäßig 256 Objekte erkannt und können auch verfolgt werden.

[31] http://wiki.nuigroup.com/images/5/57/Llp1.jpg
[32] http://www.nextwindow.com

Abbildung 2.21: Links: Aufbau eines Microsoft Surface. Rechts: Routenplanung am Surface.[33]

Abbildung 2.22: Optisch lokalisierbare Tags, die an den sogenannten Fiducials angebracht sind.[34]

Auf der linken Seite von Abbildung 2.21 wird der Aufbau eines Microsoft-Surface-Tisches gezeigt.

> "Der Index (1) zeigt die horizontale Projektions- und Interaktionsfläche, die von einem herkömmlichen Beamer (4) bestrahlt wird. (2) stellt eine Infrarot-Lichtquelle, mit 850nm Wellenlänge die Projektionsfläche gleichmäßig ausleuchtend, dar. Der Index (3) zeigt 4 Sensoren (Kameras mit einer Auflösung von 1280x960), die von Finger oder Objekt reflektiertes infrarotes Licht aufnehmen." [UL09]

[33] Links: http://www.surfacehq.com/images/content/microsoft_surface_inside.jpg
Rechts: http://www.mr-gadget.de/wp-content/uploads/2009/03/microsoft_surface.jpg
[34] http://www.oekotouch.de/wp-content/uploads/fiducials.gif

Microsoft stellt mit dem Surface ein Gesamtsystem aus integriertem Rechner und Projektionseinheit zur Verfügung. Der Computer überträgt das zu projizierende Bild an den Beamer und ist für die Verarbeitung der Daten der IR-Sensoren zuständig.[35]

2.4.2.2 Apple iPhone - iPad

Das wohl bislang bekannteste Multi-Touch-Gerät ist das von Apple vertriebene iPhone. Im Gegensatz zu herkömmlichen Mobiltelefonen besteht das iPhone lediglich aus einem großen kapazitiven Multi-Touch-Display. Die Interaktion mit dem Telefon erfolgt hauptsächlich mit Single- und Multi-Touch-Gesten.
Es besteht die Möglichkeit eigene Anwendungen für das iPhone zu programmieren, da ein "Software Developer Kit" (SDK) veröffentlicht wurde. Hierbei ist es möglich direkt auf die Daten der Touch-Punkte auf dem Display sowie auf die Neigungssensoren zuzugreifen.[36]
Das im Jahre 2010 auf dem Markt gekommene iPad nutzt die Technologien des iPhones und bietet so die Funktionen in Form eines Tablet-PCs.[37]

2.4.2.3 Nexio, Touch Screen Frames

Produkte, welche die Technologie des "Infrarot-Lichtvorhangs" nutzen, sind die multi-touch-fähigen Rahmen der Firma "Nexio". Die Rahmen sind in den Größen 12.1 Zoll bis 100 Zoll erhältlich. Sie bestehen aus einer Glasscheibe und Hardwarekomponenten, die im Rahmen eingebaut sind und können über eine Projektionsfläche oder vor einem Display montiert werden. Durch das geringe Bauvolumen von ca. 10mm können die Rahmen gut und flexibel montiert werden. Hierbei ist jedoch zu beachten, dass die berührungsempfindliche Oberfläche je nach verwendetem Display räumlich von der Bildfläche getrennt sein muss. Je größer der Abstand zur Bildfläche wird, umso schwieriger ist es für den Benutzer in den Eckpunkten des Rahmens einen präzisen Touch auszuführen. So sollte bei der Installation eines Touch-Rahmens darauf geachtet werden, dass er möglichst nahe sowie parallel zum Display montiert wird.[38]

2.4.2.4 NextWindow, Multi-Touch-Displays und -Rahmen

"NextWindow" vertreibt ebenfalls Multi-Touch-Displays und -Rahmen, welche die Technologie "IR-Light Plane" verwenden. Dabei wird wie bei Nexio eine transparente Glasscheibe als Träger benutzt. In den beiden oberen Ecken der Rahmen wird mit zwei IR-Sensoren eine Lichtfläche aufgespannt. Die Sensoren erkennen Berührungen als Störung der Lichtfläche und können so die Position des Fingers oder des Objektes bestimmen.
Die Displays sowie die Rahmen von "NextWindow" werden in Größen von 30 bis 120 Zoll vertrieben.[39]

[35] http://www.microsoft.com/surface/en/us/Pages/Product/WhatIs.aspx
[36] www.apple.com/de/iphone
[37] http://www.apple.com/de/ipad/
[38] http://www.nexiotouch.com
[39] http://www.nextwindow.com/products/index.html

2.4.3 Software

Neben der in Kapitel 2.4.2 beschriebenen Hardware ist das Vorhandensein von multi-touch-fähiger Software ein wichtiger Punkt bei der Verarbeitung von Multi-Touch-Technologien. Das standardisierte "TUIO"-Protokoll ist eine der verbreitetsten Möglichkeiten, um Multi-Touch-Daten zu verarbeiten.[40] Um die Entwicklung von hardwareunabhängiger Multi-Touch-Anwendungssoftware zu ermöglichen, ist die Unterstützung von Multi-Touch-Technologien durch Betriebssysteme notwendig. [UK08]

2.4.3.1 Betriebssysteme

> "Multi-Touch-Betriebssysteme sind eine wesentliche Grundlage für die weitere Verbreitung von Multi-Touch-Technologie." [UL09]

Damit Anwendungen mit beliebiger Multi-Touch-Hardware zusammenarbeiten, ist eine standardisierte Schnittstelle notwendig. Die Hardwarehersteller stellen hierfür Treiber für das jeweilige Betriebssystem bereit. So können Softwareanwendungen indirekt über das Betriebssystem auf die Multi-Touch-Hardware zugreifen. Durch diese Standardisierung wird Software auf allen Endgeräten lauffähig, die das Betriebssystem unterstützen, sofern die notwendigen Treiber vorhanden sind.

Im nächsten Abschnitt wird das erste verfügbare multi-touch-fähige Betriebssystem MPX Linux sowie das im Jahre 2009 erschienene Betriebssystem von Microsoft "Windows 7" beschrieben.

MPX Linux

"Multi-Pointer X-Server" (MPX) ist eine "X-Server"-Modifikation, die in der Lage ist mehrere Mauszeiger oder Eingabegeräte mit mehreren Bewegungsquellen, wie zum Beispiel Multi-Touch-Screens, zu verarbeiten.[41] Somit stellen MPX-Linux-Varianten eine Plattform dar, auf der Multi-Touch-Anwendungen betrieben werden können. Eine MPX-Modifikation kann mit der Linux-Distribution "Ubuntu" aus dem Web heruntergeladen werden. [UL09]

Microsoft Windows 7

Das neue Betriebssystem von Microsoft, "Windows 7", verfügt ebenfalls über Multi-Touch-Unterstützung. [Aut08] Durch die Marktstellung von Microsoft Windows wird sich die Multi-Touch-Technologie im Konsumentenbereich in nächster Zeit stark verbreiten. [UL09]

[40] http://tuio.org/
[41] http://wearables.unisa.edu.au/projects/mpx/

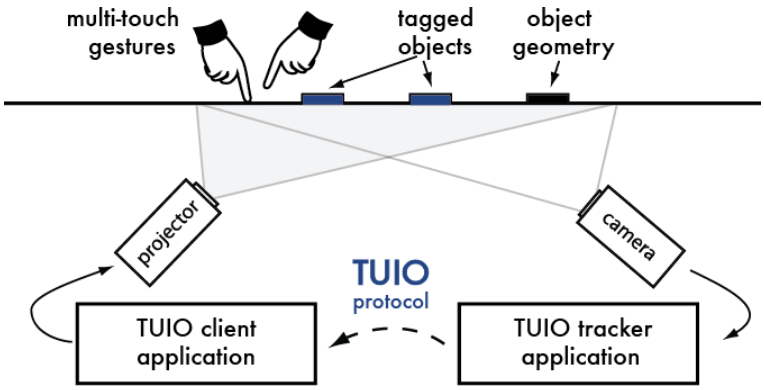

Abbildung 2.23: Schematische Darstellung des TUIO-Prinzips bei optischen Multi-Touch-Techniken.[42]

2.4.3.2 Software Schnittstellen

Neben den Betriebssystemen existiert die Schnittstelle TUIO. Die Treiber von Multi-Touch-Rahmen können die Berührungspunkte meistens direkt mittels des TUIO-Protokolls übermitteln. Bei optischen Verfahren müssen die von der Kamera erkannten Berührungen in das TUIO-Protokoll umgewandelt werden. Dafür stehen verschiedene Softwarelösungen wie zum Beispiel "CCV" und die "Touchlib"-Bibliothek zur Auswahl.

TUIO Protokoll

TUIO (Tangible User Interface Objects) ist ein offenes Framework, welches ein Protokoll und eine API (Programmschnittstelle) für Multi-Touch-Oberflächen definiert. Das TUIO-Protokoll ermöglicht die Übertragung einer abstrakten Beschreibung von interaktiven Oberflächen, einschließlich Touch-Ereignissen und Zuständen von Objekten (siehe Abbildung 2.23). Dabei werden die Daten verschlüsselt und an eine Client-Anwendung gesendet, die zur Dekodierung des Protokolls im Stande ist.
Da die Anzahl an existierenden TUIO-fähigen Anwendungen und Client-Bibliotheken für verschiedene Entwicklungsumgebungen wächst, ist eine zügige Weiterentwicklung für die Multi-Touch-Schnittstelle gegeben.
Technisch basiert TUIO auf "Open Sound Control" (OSC) und kann daher problemlos auf jeder Plattform, die OSC unterstützt, umgesetzt werden. OSC ist ein Standard für interaktive Umgebungen, der nicht nur auf die Kontrolle von Musikinstrumenten beschränkt ist. Die Standardmethode zur Beförderung von TUIO ist die Lieferung von binären OSC-Daten per UDP-Pakete über Port 3333. Diese Methode gewährleistet eine kompakte und plattformunabhängige Nachrichtenübermittlung mit niedriger Latenz. Zusätzlich gibt es auch eine TUIO / TCP Transportmethode.[43]

[42] http://www.tuio.org/images/diagram.png
[43] http://iki.nuigroup.com/TUIO

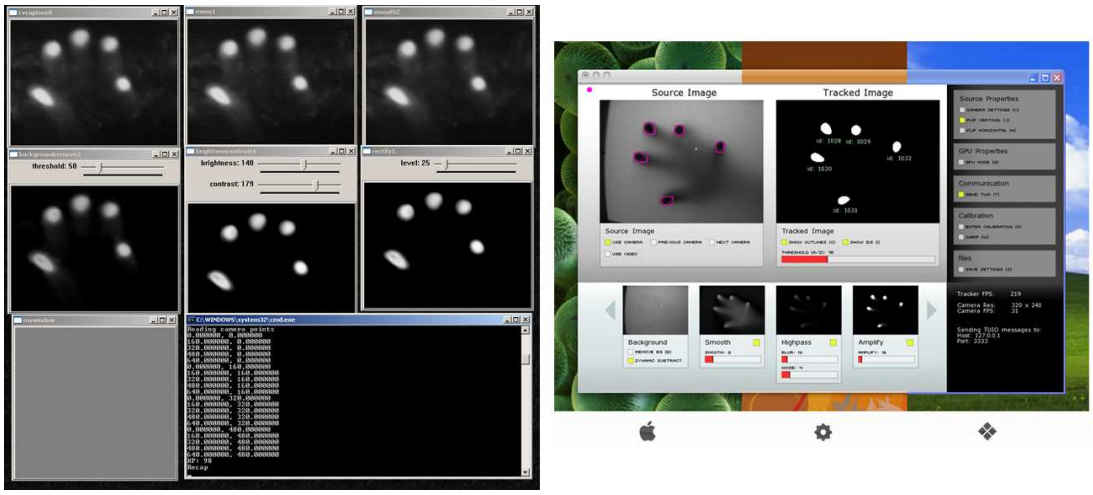

Abbildung 2.24: Links: Kalibrierung von Kamera und Projektor bei der Verwendung von Touchlib. Rechts: Fingererfassung mit Community Core Vision.[45]

Die allgemeine Syntax einer TUIO-Nachricht leitet sich aus dem OSC-Protokoll ab. Eine TUIO-Client-Implementierung muss daher eine geeignete OSC-Bibliothek wie "oscpack" oder "liblo" verwenden und die folgenden Nachrichtentypen und Strukturen, die im kommenden Beispiel ein zweidimensionales "Cursor Profile" darstellen, erkennen.[44]

```
/tuio/2Dcur source application\MVAt address
/tuio/2Dcur alive s\_id0 ... s\_idN
/tuio/2Dcur set s\_id x\_pos y\_pos x\_vel y\_vel m\_accel
/tuio/2Dcur fseq f\_id
```

Touchlib

Touchlib ist eine "Open-Source"-Programmbibliothek für Multi-Touch-Systeme. Darin enthalten sind Funktionen zur Erkennung und zur Verfolgung (auch "Tracking" genannt) mehrerer Punkte (siehe Abbildung 2.24). Bei den Punkten handelt es sich im Normalfall um Finger, die auf einer Oberfläche bewegt und als Berührung erkannt werden. Dabei werden Multi-Touch-Ereignisse, wie beispielsweise "finger down", "finger moved" und "finger released", erzeugt. Diese werden im Format des TUIO-Protokolls an das Anwendungsprogramm gesendet und ausgewertet.[46]

[44] http://www.tuio.org/?specification

[45] Links: http://wiki.nuigroup.com/Touchlib_Screenshots
Rechts: http://ccv.nuigroup.com/#screens

[46] http://wiki.ohm-hochschule.de/toolboxwiki/index.php/Touchlib

Zusätzlich enthält die Bibliothek eine Konfigurationsdatei, in welcher die Einstellungen an die unterschiedlichen Anwendungen angepasst werden können. Außerdem sind einige Beispielapplikationen vorhanden, die zum Testen der Einstellungsänderung verwendbar sind.

Community Core Vision (CCV)

CCV oder auch häufig "tbeta" genannt, ist eine Open-Source-Software für das Betreiben von Multi-Touch-Geräten. CCV verwendet das Bild einer Kamera als Eingangssignal und kann Berührungen in Form sogenannter "Blobs" erkennen.
Es werden alle gängigen Verfahren wie FTIR, DI und LLP unterstützt. Die Software stellt verschiedene Schnittstellen bereit, um Multi-Touch-Anwendungen zu steuern und ist so in der Lage, eine Verbindung zu verschiedenen TUIO- bzw. OSC-fähigen Anwendungen herzustellen. CCV ist für alle gängigen Betriebssysteme (Linux, Windows, Mac OS) erhältlich.[47] Die Software arbeitet mit vielen gängigen Softwarebibliotheken wie der "touchlib" zusammen und ist somit anpassbar.[48]

2.5 Echtzeit 3D-Entwicklungsumgebungen

In Echtzeit 3D-Anwendungen werden computergenerierte Szenarien (dreidimensionale Welten) dargestellt. Hier hat der Anwender die Möglichkeit sich frei zu bewegen, d.h. zu gehen, zu fliegen oder einfach nur zu navigieren. Damit der Benutzer das Gefühl hat, die Anwendung reagiere in Echtzeit, ist eine Mindestbildwiederholrate notwendig. Ab einer Rate von 15 Bildern pro Sekunde ("Frames per second" im Folgenden "fps" genannt) wird von Echtzeit gesprochen. Um dies zu realisieren müssen Echtzeit 3D-Anwendungen Abstriche bei der Detailtiefe und der Realitätsnähe machen. [TAM08]

Die Einsatzgebiete neben der Entwicklung von 3D-Spielen sind auch die Erstellung virtueller Prototypen, Produkt- und Architekturvisualisierung oder die Umsetzung von Besucherinformationssystemen wie zum Beispiel in einem Museum. Zur Erstellung solcher Anwendungen stehen neben zahlreichen Entwicklungsumgebungen für Spiele ("Game-Engines") auch andere, meist spezielle Entwicklungstools, zur Verfügung.

Echtzeit 3D-Computergrafik und "Virtuelle Realität" (VR) werden häufig im selben Zusammenhang genannt. Hierbei sollte beachtet werden, dass Echtzeit 3D-Computergrafik nur ein Bestandteil der VR ist. VR zeichnet sich durch eine realitätsnahe dreidimensionale Darstellung aus. Der hauptsächliche Unterschied liegt im Grad der Immersion sowie der benutzten Ein- und Ausgabegeräte. [Hei04] Der Begriff Immersion beschreibt hierbei die Tiefe des Eintauchens in eine Virtuelle Welt. Während VR hauptsächlich in der Forschung und in großen Industrieunternehmen eingesetzt wird, versuchen die Spielehersteller sich immer mehr an die Möglichkeiten der VR anzunähern.

[47] http://nuicode.com/projects/tbeta/
[48] http://nuigroup.com/touchlib/

3D-Welten werden mit Hilfe von 3D-Modellierungssoftware oder CAD-Systemen konstruiert. Die erstellten Geometrien werden mit Texturen bzw. Materialien versehen. Dann können bewegte Objekte und Animationen sowie Sounds und Videos eingebunden werden.
Es gibt eine Vielzahl von Möglichkeiten eine Echtzeit 3D-Anwendung zu erstellen. Im folgenden Abschnitt werden einige Anwendungsbeispiele für Echtzeit 3D-Computergrafik aufgeführt sowie Game-Engines und andere Entwicklungsumgebungen vorgestellt.[49]

2.5.1 Anwendungsgebiete von Echtzeit 3D-Computergrafik

Die Echtzeit 3D-Computergrafik, auch Interaktive 3D-Computergrafik genannt, hat in nahezu allen Anwendungsgebieten von Computern Einzug gehalten. Durch den enormen Anstieg der Leistung von Grafikkarten mit gleichzeitigem Preisverfall lassen sich immer hochwertigere Bilder in Echtzeit produzieren. Hier werden nun einige Beispiele der Anwendungsgebiete aufgeführt.

2.5.1.1 Ausbildungs-Simulation

Vom Militär vorangetrieben, wurden die ersten interaktiven 3D-Simulatoren für Flugzeugpiloten entwickelt. Dadurch war die Aus- und Weiterbildung der Piloten kostengünstiger und ungefährlicher. Mittlerweile werden Simulatoren nicht nur vom Militär, sondern auch von allen größeren Fluggesellschaft eingesetzt. Inzwischen werden Simulatoren für eine Vielzahl von Anwendungen genutzt. So werden sie in Luft- und Raumfahrt für Raketen und andere Flugobjekte verwendet. Durch die immer bessere und günstigere Grafikhardware wird der Einsatz von Fahrsimulatoren für Schiffe oder Schienenfahrzeuge rentabel. Im Schienenverkehr bietet dies Entlastung auf stark befahrenen Schienennetzen und ermöglicht zusätzlich ein flexibles Training von Gefahrensituationen. Heutzutage werden Simulatoren auch zur Ausbildung an speziellen Straßenfahrzeugen, wie beispielsweise LKW, Polizeiwagen (siehe Abbildung 2.25, rechts) und Notdienstfahrzeugen, eingesetzt. [AN07]

2.5.1.2 Entwicklungs-Simulation

Die Entwicklung von Produkten wird durch die 3D-Computergrafik schneller, flexibler und kostengünstiger gestaltet. Komplexe technische Prototypen werden in verschiedenen Stufen entworfen. Zuerst werden Produktideen in einer Vorsimulation ausgearbeitet, die in ein Grobkonzept münden. Komponenten des Produktes werden als virtuelle Prototypen entwickelt und in verschiedenen Simulationen und Visualisierungen überprüft. Dabei können Krafteinwirkung und andere Einflüsse simuliert und erste Erkenntnis gewonnen werden (siehe Abbildung 2.25, links). Nach dem Anfertigen der ersten Komponenten werden Optimierungen immer erst in der Simulation getestet. Immer häufiger wird Echtzeit 3D-Computergrafik als Nachweis für die Leistungsfähigkeit von Produkten eingesetzt. [AN07]

[49] http://de.wikipedia.org/wiki/3D-Echtzeit

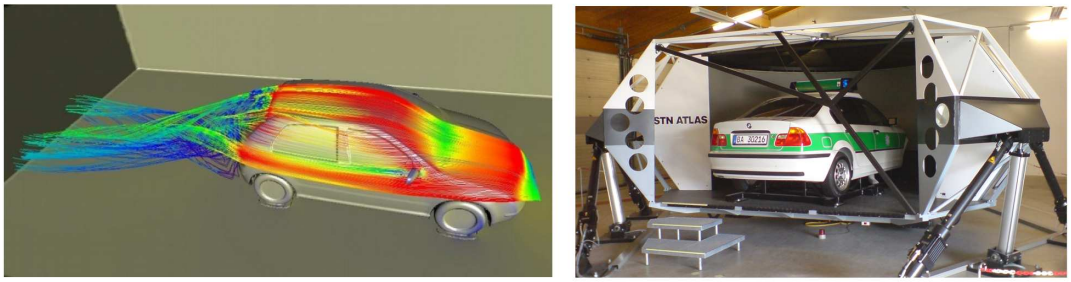

Abbildung 2.25: Beispiele für interaktive 3D-Computergrafik. Links: Virtuelles Simulationsmodell, welches es ermöglicht das Materialverhalten vorherzusagen. Rechts: Fahrsimulator der Bereitschaftspolizei zum Training von Einsatzfahrten.[50]

2.5.1.3 Daten-Visualisierung

In vielen Bereichen wie Wissenschaft, Forschung und Entwicklung sowie Finanz- und Wirtschaftswesen hat sich die interaktive 3D-Visualisierung als Standardmethode etabliert. So wird sie beispielsweise in der Architektur für die Planung von Stadtvierteln, Gebäuden oder auch für die Inneneinrichtung verwendet. Für Laien ist es oft schwer sich bei zweidimensionalen Ansichten das Ergebnis vorzustellen. Mit Hilfe der interaktiven 3D-Computergrafik kann dieses Problem jedoch beseitigt werden. [AN07]

2.5.1.4 Augmented Reality

"Augmented Reality" (AR - Erweiterte Realität) ist die Überlagerung von realen Szenen mit computergestützter Grafik oder Bildern. Zwar ist damit meist die visuelle Darstellung gemeint, jedoch können auch alle anderen Sinne, wie Hören, Fühlen, Schmecken oder Riechen, angesprochen werden. Ein Anwendungsbeispiel ist das Anzeigen von Informationen auf der Frontscheibe eines Fahrzeugs. Dies können für den Fahrer relevante Daten, wie beispielsweise Geschwindigkeitswerte oder Navigationsinformationen, sein. Dadurch kann der Blick auf die Straße gerichtet bleiben. Heutzutage ist es bereits möglich Zusatzinformationen zu Sehenswürdigkeiten oder Orten auf einem Handy oder Pocket PC anzeigen zu lassen. Dazu werden nur eine Kamera und Navigationsdaten benötigt. [AN07]

2.5.1.5 Unterhaltung

Die Leistungssteigerung der 3D-Computergrafik wird durch die Unterhaltungs- und Spieleindustrie vorangetrieben. Computerspiele am PC, auf Spielkonsolen und immer mehr auch auf Handys tragen zur Weiterentwicklung in diesem Bereich bei. Dabei weisen Computerspiele einen hohen Grad an Interaktivität auf. So ist im Moment ein Stand erreicht, der vor kurzer Zeit nur "High-End-Grafikworkstations" vorbehalten war.

[50] Links: http://static.materialsgate.de/thumb/f/67kz.jpg
Rechts: http://de.academic.ru/dic.nsf/dewiki/114912

Abbildung 2.26: Links: Ansicht eines Terrains mit der CryEngine 3. Rechts: Ansicht eines Terrains mit der Unreal Engine 3.[51]

Die meisten verkauften Heim-PCs sind in der Lage Polygonmodelle beschleunigt und qualitativ hochwertig darzustellen. Damit wird es möglich, immer komplexere und realistischere 3D-Welten in Echtzeit darzustellen. [AN07]

2.5.2 Game-Engines

> "Wenn wir das Wort Engine ins Deutsche übersetzen, heißt es so viel wie Antrieb oder Motor. Das trifft den Nagel auf den Kopf - eine Engine ist nichts anderes als der Motor für das Spiel und damit essenziell!" [Sch04]

Um ein eigenes Spiel oder eine Anwendung zu erstellen können Game-Engines genutzt werden. Durch die große Vielfalt verschiedenster Engines kann im Groben zwischen kommerziellen und nicht kommerziellen (Open-Source-Engines) unterschieden werden. Die bekanntesten kommerziellen Game-Engines sind unter anderem die "Unreal-Engine" oder die "CryEngine" für Ego-Shooter (siehe Abbildung 2.26). Bei den Open-Source-Engines sind "Blender", "OGRE", "Crystal Space" oder die "Irrlicht-Engine" verbreitet.
Im Anwendungsbereich der Game-Engines kann zwischen zwei Grundtypen unterschieden werden. Zum einen gibt es "spezielle Engines". Diese sind zum Beispiel auf Ego-Shooter spezialisiert und können nur schwer für andere Spieltypen genutzt werden. Zum anderen existieren "universelle Engines", die theoretisch für alle Spieltypen geeignet sind. [Sch04]

[51] http://www.psu.com/features/6835

Der Anspruch an eine Game-Engine ist, dass wichtige Bestandteile, die zur Spielentwicklung benötigt werden, miteinander verknüpft und zur Verfügung gestellt werden. Neben der Steuerung des Spielverlaufs ist sie auch für die visuelle Darstellung zuständig. Das Verwalten einer Szene unter Ausnutzung moderner Grafikhardware ist ebenso erforderlich.
Durch den modularen Aufbau einer Game-Engine ist sie erweiterbar, übersichtlich und bietet eine leichte Modifizierung der einzelnen Module. [Fug04] Die möglichen Bestandteile einer Engine werden im Folgenden erläutert.

2.5.2.1 Grafik-Engine

Die Grafik-Engine wird meist in zwei Aufgabenbereiche unterteilt. Einen Teil dient zur Echtzeitberechnung und -darstellung der Spieldaten (wie Level, Charaktere, etc.) auf dem Bildschirm. Der andere Teil stellt die grafische Benutzerschnittstelle dar, das sogenannte "Graphical User Interface" (GUI). Die Grafik-Engine nutzt Funktionalitäten zum Laden, Verwalten und Anzeigen von Oberflächen und Texturen. Außerdem ist sie verantwortlich für Licht und Schatten und ihre Wirkung auf verschiedene Materialien sowie das Darstellen unterschiedlicher Effekte, wie beispielsweise Feuer, Wasser oder Nebel. Im Kern ist eine Grafik-Engine für das Erzeugen und die Ausgabe von grafischen Inhalten da. [Kra07]

2.5.2.2 Sound-Engine

In der Sound-Engine sind, ähnlich wie bei der Grafik-Engine, Funktionen und Effekte enthalten, die jedoch für die akustische Ausgabe gedacht sind. Darunter fallen zum Beispiel das Einbinden von Hintergrundmusik und das Abspielen erstellter Soundtracks. Das Modifizieren und Nachbearbeiten dieser Sounds mit Echos oder Rauschen gehört ebenso dazu. In dreidimensionalen Umgebungen muss auch dafür gesorgt werden, dass Sounds räumlich positionierbar sind, damit sie zum Beispiel mit zunehmender Entfernung zum "Hörer" leiser werden.[52]

2.5.2.3 Steuerung

Ein weiterer Bestandteil einer Game-Engine ist die Steuerung. Damit ein Spiel interaktiv ist, muss gewährleistet sein, dass alle gängigen Eingabegeräte erkannt und zur Laufzeit ständig ausgewertet werden können. So kann der Spieler jederzeit Einfluss auf das Geschehen ausüben. In Spielen hat sich speziell die Steuerung mit Maus, Tastatur oder Gamepad etabliert. Mit dem Spiel R.U.S.E., welches das Steuern mit Multi-Touch-Gesten unterstützt, wird nun ein noch nicht so verbreitetes Eingabegerät für Spiele eingeführt. [Kra07]

2.5.2.4 Physik-Engine

Mit einer Physik-Engine können physikalische Abläufe nachgestellt werden. Dies kann zum Beispiel das Werfen eines Objektes oder ein Zusammenprall von Körpern sein. Wichtig wird eine solche Engine bei der Kollisionserkennung, wo überprüft wird, ob eine Figur im Spiel

[52] http://de.wikipedia.org/wiki/Spiel-Engine

eine Wand oder einen Gegenstand schneidet, um dann ein Hindernis zu simulieren. Damit wird gewährleistet, dass die Figur nicht durch Wände oder Gegenstände gehen kann. Es kann zwischen verschiedenen Arten von "Physik" unterschieden werden. Bei der "Ragdoll"-Physik, einem Teilaspekt der "Rigid-Body"-Physik, wird Wert darauf gelegt, dass ein Charakter möglichst realistisch fällt. Auch zur Animation von Fahnen, Kleidung oder Wasser kann eine Physik-Engine eingesetzt werden. Physik in Anwendungen geht mittlerweile so weit, dass dafür eigens Physikbeschleuniger "Physics Processing Unit" (PPU) verfügbar sind. Diese sollen in Form von optimierten Hardwarelösungen die Berechnung der physikalischen Abläufe übernehmen und so den Hauptprozessor entlasten. [Kra07]

2.5.2.5 Skript-Engine

Eine Skript-Engine stellt eine Schnittstelle für Entwickler einer Anwendung dar, mit der die Anwendung nach eigenen Vorstellungen gestaltet bzw. verändert werden kann. Skript-Engines sind zur Programmierung von Spielabläufen und Spiellogiken vorgesehen. Dabei werden zur Laufzeit die definierten Aktionen und Handlungen ausgewertet und ausgelöst. Bei der Entwicklung einer Anwendung bzw. eines Spiels hat sich etabliert, dass das Spiel in einer Skriptsprache entwickelt wird, welche von der Game-Engine zur Verfügung gestellt wird. Zwar ist es auch möglich, die Abläufe in der Entwicklungssprache zu programmieren, Skriptsprachen bieten aber folgende Vorteile:

- Sie sind für nicht professionelle Programmierer schneller und einfacher zu erlernen.
- Änderungen zur Laufzeit sind möglich.
- Nutzer können die Game-Engine erweitern.

Teilweise greifen Game-Engines dabei auf vorhanden Skriptsprachen, wie zum Beispiel "Lua", zurück. Bei besonderen Anforderungen werden jedoch auch eigene Skriptsprachen, zum Beispiel "UnrealScript" in der Unreal-Engine, eingesetzt.[53]

2.5.2.6 Funktionsweise einer Game-Engine

Zusätzlich zu den bereits erwähnten Bestandteilen einer Game-Engine können noch weitere Engines, zum Beispiel zur Zustandsspeicherung, Steuerung der "KI" (Künstliche Intelligenz), Kommunikation im Netzwerk und zum Datenmanagement, eingebunden werden. Die Bestandteile werden im Engine-Kern verbunden, der mit den anderen Teilen der Engine kommuniziert, um Veränderungen darzustellen und auf Interaktionen des Nutzers einzugehen. Ein Teil des Engine-Kerns ist die sogenannte Hauptschleife. Hier werden alle Aktionen zur Aktualisierung und Darstellung des Spielinhaltes zusammengefasst. Bei jedem Durchlauf der Schleife werden folgende Aktionen ausgeführt:

1. Spielereingaben werden eingelesen und interpretiert. Daraus ergeben sich Befehle für die Spielwelt.

[53] http://de.wikipedia.org/wiki/Spiel-Engine

2. Die KI-Engine berechnet aus der aktuellen Lage im Spiel die Änderungen für die Figuren und Objekte.

3. Wegen der Änderungen des Spielers und der KI werden Figuren und Gegenstände animiert bzw. bewegt.

4. Damit alle Bewegungen nach Vorgabe der Physik ablaufen, überprüft die Physik-Engine, ob zum Beispiel Kollisionen entstanden sind, um diese aufzulösen.

5. Die Sound-Engine ist dafür verantwortlich zu überprüfen, ob die Veränderung der Spielwelt neue Geräusche oder Soundtracks erfordert.

6. Die Grafik-Engine berechnet aus der Sicht des Spielers ein neues Bild auf die Spielwelt.

Dabei werden die Punkte 1. und 2. in "Eingabe", 3. und 4. in "Simulation" und 5. und 6. in "Ausgabe" unterteilt. Wird die Hauptschleife schnell wiederholt, bekommt der Spieler den Eindruck, dass die Spielwelt auf seine Handlungen reagiert. [Kna06][Kra07]

2.5.3 Andere Entwicklungsumgebungen

Neben den im Kapitel 2.5.2 beschriebenen Game-Engines gibt es eine Vielzahl von anderen Entwicklungsumgebungen, in denen Echtzeit 3D-Anwendungen erstellt werden können. Hierbei ist zu beachten, dass viele von ihnen spezielle Anwendungsgebiete haben.

2.5.3.1 vvvv-System

Die grafische Entwicklungsumgebung "vvvv" erzeugt und manipuliert Video-, Grafik- und Datenströme in Echtzeit. Es macht keinen Unterschied, ob das Programm erstellt oder ob es gerade ausgeführt wird, da sich das Programm grundsätzlich in Echtzeit befindet. Änderungen am grafischen Programmcode werden direkt übernommen und ausgeführt.
Die Software-Engine "vvvv" dient zur Handhabung großer medialer Umgebungen mit physikalischen Schnittstellen, Echtzeit "Motion-Graphics", Audio- und Videoelementen.[54]

2.5.3.2 Mixed-Reality-System instantreality

Das System "instantreality" ist ein komplexes "Mixed-Reality"-System. Es verknüpft "Virtual Reality"- und "Augmented Reality"-Funktionalitäten. So können Telepräsenzaspekte in VR-Anwendungen einfließen oder Echtzeitsimulationen in AR-Applikationen implementiert werden.

Das Mixed-Reality-System orientiert sich an Standards in der 3D-Computergrafik und basiert auf dem X3D-Standard. Das System instantreality übernimmt die Organisation des Szenegraphen (Datenstruktur einer 3D-Szene), die Kontrolle spezifischer Ein- und Ausgabegeräte sowie die Unterstützung von verschiedenen Echtzeitsimulationen. Dieses System ist durch die folgenden Eigenschaften definiert:

[54] http://wiki.ohm-hochschule.de/toolboxwiki/index.php?title=Vvvv

Modularität
Es ist modular aufgebaut und wurde in der Programmiersprache C++ entwickelt. So besteht die Möglichkeit der Anpassung an applikationsspezifische Anforderungen von Kunden.

Unabhängigkeit vom Betriebssystem
Das Einsatzgebiet von instantreality erstreckt sich neben PC- und Apple-Systemen auch über Grafikworkstations sowie PDAs, da es unter MS-Windows, Mac-OS, Linux und anderen Unix Lösungen läuft.

Applikationsunabhängigkeit
Durch die strikte Trennung von Applikationen und VR-Systemen können neue Applikationen ohne komplexe Programmierung oder Kompilierung des VR-System erstellt werden, da Applikationen durch einfache Skriptdateien konfiguriert werden.

Standardkonformität
Das X3D-Standardformat bildet die Basis von instantreality. So werden alle Funktionalitäten von X3D unterstützt.[55]

2.5.3.3 Multi-Touch for Java (MT4j)

"Multi-Touch for Java", kurz "MT4j" ist eine auf Open-Source-Komponenten basierende Plattform, die ebenfalls frei verfügbar ist. MT4j dient zur Unterstützung von Multi-Touch-Anwendungsentwicklungen für gängige PC-Hardware und PC-Betriebssysteme. Es ist möglich zwei- sowie dreidimensionale Computergrafik einzusetzen. MT4j verwendet die Programmiersprache Java und stellt so eine nutzbare Plattform für eine große Entwicklungsgemeinde dar. Die Plattform stellt die Auswertungen sowie die Steuerungsgesten, die bei Multi-Touch-Anwendungen häufig benötigt werden, zur Verfügung. Da MT4j modular aufgebaut ist, kann die Plattform einfach erweitert werden.[56]

2.5.3.4 Visualization Toolkit (VTK)

VTK ist eine Open-Source-C++-Klassenbibliothek, die besonders für die 3D-Computergrafik mit einem Schwerpunkt auf wissenschaftlicher Visualisierung geeignet ist. Weitere Einsatzgebiete sind 3D-Computergrafik, Modellierung, Bildverarbeitung, Volumen-Rendering und Informationsvisualisierung.[57]

[55] http://www.vivera.org/Basistechnologie_IGD_InstantReality.htm
[56] http://www.iao.fraunhofer.de/lang-de/geschaeftsfelder/informations-und-kommunikationstechnik/404-moulti-touch-softwaretechnik.html
[57] http://www.vtk.org/

2.6 Zusammenfassung und Ausblick

Im ersten Teil des Kapitels wurde auf virtuelle Charaktere eingegangen, um einen Überblick über ihre Einsatzgebiete zu erlangen. Virtuelle Charaktere finden durch ihre Vielfältigkeit immer mehr Verbreitung. Neben Computeranimationen werden sie in Filmen, Spielen, Software- oder Online-Anwendungen eingesetzt. Durch ihr variables Äußeres können sie verschiedenste Zielgruppen ansprechen und unterhalten. Ebenfalls wurde auf die Interaktions- bzw. Steuerungsmöglichkeiten mit Charakteren in Multi-Touch-Spielen eingegangen und unterschiedliche Ansätze vorgestellt. In Animationsfilmen sind meist comichafte Charaktere zu finden. Reale Filme hingegen nutzen virtuelle Charaktere, die als solche kaum mehr erkennbar sind. Interaktive Charaktere werden beispielsweise online als Berater und Ansprechpartner für einen Kunden zur Verfügung gestellt. Selbst in Navigationssystemen werden solche Charaktere eingesetzt, um die Bedienung zu erleichtern. Dadurch ist abzusehen, dass sich virtuelle Charaktere auch weiterhin in neuen Einsatzgebieten etablieren.

Weiter wurde in diesem Kapitel gezeigt, dass es viele Möglichkeiten gibt, einen virtuellen Charakter zu animieren. Einzelne Verfahren, wie Key-Framing und vorwärtsgerichtete Kinematik, wurden vorgestellt. Die jedoch schnellste und realistischste Art, einem Charakter Leben einzuhauchen, ist das Motion-Capturing. Dadurch wird klar, warum Motion-Capturing eine immer wichtigere Rolle im Unterhaltungsbereich einnimmt. Aber auch in weiteren Einsatzgebieten hält MoCap Einzug. Zu beachten ist dabei, dass jedes MoCap-System seine Vor- und Nachteile hat, die je nach Anforderung der Bewegungsaufnahme abgewogen werden müssen. Durch Weiterentwicklung von Hardware und Verfahren, die von vielen Firmen vorangetrieben und immer weiter verbessert werden, kann immer günstiger und effizienter gearbeitet werden. Ein Beispiel dafür ist das neue Face-Tracking-System von "Captive-Motion" mit dem Gesichtsanimationen noch realistischer werden und damit die Grenzen zwischen der virtuellen und der realen Welt noch mehr verschwimmen. Die Technologien gewinnen auch zunehmend an Bedeutung im Konsumentenbereich. Der Spielekonsolenentwickler "Nintendo" verwendet bereits Techniken aus dem Bereich Motion-Capturing in seinen Controllern.

Die in diesem Kapitel beschriebenen Multi-Touch-Technologien liefern einen Überblick über die am meisten verbreiteten Verfahren. Weiter wurde exemplarisch verfügbare Hardware vorgestellt sowie auf die Notwendigkeit von Softwarelösungen im Blick auf Hardwareunabhängigkeit eingegangen. Die Multi-Touch-Technologien befinden sich immer mehr auf dem Vormarsch und unterliegen einer ständigen Weiterentwicklung. Bestehende Verfahren werden günstiger und besser. Die optischen Systeme befinden sich in einem frühen Entwicklungsstadium und werden vermutlich in nächster Zeit einen deutlichen Entwicklungsschub erhalten. Durch die direkte Manipulierbarkeit von Inhalten auf einem Multi-Touch-Gerät können intuitive Benutzungsschnittstellen entwickelt werden. Intuitive Eingabesysteme, das bedeutet auf den Menschen abgestimmte Eingabegeräte, werden in naher Zukunft eine wichtige Rolle spielen. Bei Anwendungsszenarien, bei denen die Benutzer wenig oder kein Erfahrungswissen im Umgang mit den Systemen haben, wie zum Beispiel Automaten oder Kiosksysteme, bieten sich intuitive Benutzerschnittstellen an. Touch-Screens sind nicht nur intuitiv, sondern auch schnell in der Bedienung. Im Bereich der mobilen Geräte und bei fest

stationierten Workstations wird sich die Technik als Standard etablieren. Trotz des starken Vormarschs dieser Technologie zeichnen sich Defizite in der Benutzerfreundlichkeit durch das Fehlen visueller und haptischer Wahrnehmung ab. Beim Bedienen virtueller Tastaturen auf einer Touch-Oberfläche fehlt die gefühlte Rückmeldung beim Betätigen einer Taste, so ist eine "blinde" Bedienung solcher Geräte eigentlich nicht möglich. Aus diesem Grund werden traditionelle Eingabegeräte, wie beispielsweise die Tastatur, nicht ganz verschwinden.

Im Bereich Echtzeit 3D-Anwendungen wurden verschiedene Entwicklungsumgebungen vorgestellt. Die verbreitetste Form von Echtzeit 3D-Anwendungen ist bisher das Computerspiel. Zur Erstellung eines Spiels werden Game-Engines eingesetzt. Dabei werden verschiedene Anforderungen an eine Game-Engine gestellt. Je nach Anforderung sind die Bestandteile einer Game-Engine unterschiedlich. So ist zum Beispiel die Grafik-Engine für die Visualisierung der virtuellen Welt verantwortlich. Zusätzlich beinhalten Game-Engines eine Vielzahl weiterer Techniken, die das Erstellen eines Spiels erleichtern.
Aber auch andere Entwicklungsumgebungen, neben den Game-Engines, eignen sich zur Entwicklung von Echtzeit 3D-Anwendungen. Ein Beispiel dafür ist instantreality, ein Mixed-Reality-System, welches Virtual Reality und Augmented Reality verknüpft. Der grundlegende Unterschied zwischen Game-Engines und anderen Entwicklungsumgebungen ist, dass Game-Engines auf die Erstellung von Spielen spezialisiert sind und auch extra dafür entwickelte Funktionen und Methoden zur Verfügung stellen. Andere Entwicklungsumgebungen sind für die speziellen Anforderungen, je nach Einsatzgebiet, aufgebaut. Durch die jahrelange Weiterentwicklung von Game-Engines und dem Zuwachs der Leistungsfähigkeit der Hardware greifen Entwickler aus anderen Anwendungsgebieten immer häufiger auf Game-Engines zurück. Der technische Fortschritt bringt die Echtzeit 3D-Anwendungen immer näher an die virtuelle Realität und es ist absehbar, dass die genutzten Engines und Technologien verschmelzen.

Die Verwendung von Motion-Capturing, gerade in der Spielentwicklung, ist ein nützliches und schon sehr verbreitetes Verfahren, um interaktive Charaktere zu animieren. Die Unterhaltungsindustrie wird in naher Zukunft auf die Technologie von Multi-Touch als Steuermedium zurückgreifen. Aus diesem Grund werden die beiden Techniken in den kommenden Jahren eine große Rolle bei der Entwicklung neuer Echtzeit 3D-Anwendungen spielen. Im nachfolgenden Kapitel wird ein Konzept für eine Multi-Touch-Anwendung entwickelt. Hierbei wird neben der Festlegung der Spielegeschichte sowie der Zielgruppe auch der Ablauf der Anwendung beschrieben.

Kapitel 3

Konzeptentwicklung

Im letzten Kapitel wurden virtuelle Charaktere, verschiedene Arten der Animation, Verfahren für Multi-Touch-Oberflächen sowie Echtzeit 3D-Entwicklungsumgebungen vorgestellt. Die verschiedenen Einsatzgebiete und Einsatzformen von virtuellen Charakteren wurden erklärt und mit Beispielen unterlegt. Bei der Animation wurde besonders auf die Verfahren von Motion-Capturing eingegangen. Im Abschnitt Multi-Touch wurden die Schnittstellen zur Verarbeitung der Touch-Gesten, Beispiele von Multi-Touch-Hardwareherstellern und Betriebssysteme, die Multi-Touch unterstützen, aufgeführt. Bei den 3D-Echtzeitentwicklungsumgebungen wurden der Aufbau von Game-Engines und die Besonderheiten von anderen Entwicklungsumgebungen, wie zum Beispiel instantreality, dargestellt.

Im folgenden Kapitel werden, nach der Erarbeitung der Anwendungsanforderungen, die Ideen und inhaltlichen Grundlagen der Anwendung entwickelt. Danach wird die Anwendungsidee konkretisiert und das Design des Charakters sowie der Anwendung festlegt.

3.1 Einleitung

Auf Grundlage der Entscheidungen, die im Konzept erarbeitet werden, müssen anschließend alle Arbeitsschritte festgelegt und die entsprechenden Modelle erstellt werden. Im Konzept werden wichtige Fragen zur Anwendung geklärt, wie beispielsweise:

- Mit welchen Methoden und Vorgehensweisen soll gearbeitet werden?
- Was ist Ziel der Anwendung?
- Wer ist die Zielgruppe?
- Welche Interaktionen mit dem Charakter sind sinnvoll?
- Welche Touch- bzw. Multi-Touch-Gesten werden verwendet?

Auf der Basis der erstellten Anforderungen und der Idee wird die Anwendung beschrieben, um später umgesetzt zu werden. Dazu werden die Arbeitsschritte, die während dieser Phase der Konzeptentwicklung durchlaufen werden, möglichst anschaulich dargelegt.

3.2 Anforderungen

Bevor die Anwendung entstehen kann, müssen Ziele und Anforderungen, die aus der Aufgabenstellung dieser Arbeit hervorgehen, festgelegt werden. Ziel ist die Erstellung einer Spiel- und Spaßanwendung zur Wissensvermittlung mit Hilfe eines virtuellen Charakters. Dem Nutzer muss die Möglichkeit gegeben werden mit dem virtuellen Charakter in Echtzeit interagieren zu können. Das Überblenden bzw. Austauschen von Animationen muss zur Laufzeit der Anwendung möglich sein. Hierbei müssen die Benutzung des Spiels sowie die Interaktion mit dem Charakter über Multi-Touch-Gesten erfolgen. Die einzelnen Animationen des Charakters müssen möglichst realistisch sein. Aus der Aufgabenstellung ergeben sich folgende Anforderungen an die Anwendung:

Verknüpfung von Technologien

Das Spiel muss folgende Technologien verknüpfen:

- Multi-Touch
- Motion-Capturing
- Virtuelle Charaktere
- Echtzeit 3D-Anwendung

Aus diesem Grund wird besonders die Entwicklungsumgebung eine wichtige Rolle spielen. Diese muss dazu in der Lage sein Charaktere mit Animationen einzubinden, eine Möglichkeit geben eine virtuelle Welt aufzubauen und zusätzlich die von Multi-Touch-Oberflächen gesendeten Daten zu verarbeiten.

Hardwareunabhängigkeit

Die Spiel-Anwendung muss möglichst Hardware unabhängig umgesetzt werden. Hierbei bietet sich das im Stand der Technik 2.4.3.2 vorgestellte TUIO-Protokoll an. Es definiert eine API für Multi-Touch-Oberflächen und liefert eine generelle Beschreibung von Multi-Touch-Eingaben. So kann das Spiel auf unterschiedlichen Multi-Touch-Geräten, unabhängig von der zugrunde liegenden Technologie, genutzt werden.

Echtzeitdarstellung

Die Darstellung des Spiels muss in Echtzeit geschehen, damit die Interaktion des Nutzers nicht eingeschränkt wird. Im Idealfall erfolgt die Reaktion auf Benutzereingaben so schnell, dass das menschliche Auge die Verzögerung, welche durch die notwendige Berechnung entsteht, nicht bemerkt. [AN07]

Animationsüberblendung
Neben der Echtzeit Darstellung des Spiels ist auch die Überblendung der Animationen zur Echtzeit wichtig. Die Reaktion des Charakters muss direkt nach der Aktion des Benutzers erfolgen. Dabei muss Wert darauf gelegt werden, dass die Überblendung auch "sauber" d.h. ohne Sprünge oder Ruckler ausgeführt wird. Wenn dies nicht gewährleistet werden kann, ist ein natürlicher Bewegungsablauf des Charakters nicht gegeben.

Zielgruppenorientiert
Das Spieldesign und das Aussehen des Charakters müssen an die Bedürfnisse der ausgewählten Zielgruppe angepasst werden, damit eine persönliche Bindung aufgebaut und dem virtuellen Charakter Sympathie entgegengebracht werden kann. Weiter ist zu beachten, dass das zu vermittelnde Wissen für die entsprechende Zielgruppe aufbereitet wird.

Sinnvolle Nutzung von Multi-Touch-Gesten
Eine weitere Anforderung ist, dass die Multi-Touch-Gesten einen sinnvollen Zweck in der Anwendung erfüllen. Bereits etablierte Multi-Touch-Gesten, wie das Skalieren bzw. Zoomen, werden dabei nicht verändert sondern beibehalten. Da sich bisher aber nur wenige Gesten etabliert haben, müssen andere zur Steuerung der Anwendung entwickelt werden. Zudem muss die Bedienung der Spiel-Anwendung möglichst intuitiv erfolgen und ohne lange Einweisung funktionieren.

Die benötigten technischen Einsatzmittel zur Erstellung einer Multi-Touch-Anwendung, wie ein Multi-Touch-Tisch und eine Auswahl an kommerziellen bzw. nicht kommerziellen Entwicklungsumgebungen, sind vorhanden und können genutzt werden.

3.3 Entwicklung der Spielidee

Die Spielentwicklung läuft in mehreren Schritten bzw. Phasen ab. Meist werden die einzelnen Phasen einer Entwicklung mehr als einmal durchlaufen, um auf aktuelle Ereignisse bei der Entwicklung zu reagieren. An erster Stelle steht die Entwicklung der Spielidee. Eine gute Spielidee reift mit der Zeit. Details entwickeln sich im Laufe der Konzeptionsphase fast von alleine, wenn die Grundidee verinnerlicht wird. Die beste Möglichkeit, um den Anfang zu machen, ist sicher das Abhalten von "Brainstorming-Sessions" und das Erstellen von "Mindmaps". Die Handlungsstränge müssen entwickelt werden, sobald das Grobkonzept ausgearbeitet ist.[1] [2]

> "Das Allerwichtigste bei der Spielentwicklung ist die möglichst neuartige Idee - an Brain-Storming-Sessions führt deshalb in den meisten Fällen kein Weg vorbei."[3]

[1] http://entwickler.de/zonen/portale/psecom,id,101,online,1125,p,0.html
[2] http://www.spieleentwickler.org/ressourcen/faq_einsteiger.htm
[3] http://entwickler.de/zonen/portale/psecom,id,101,online,1125,p,0.html

Es ist ratsam jede Kleinigkeit der Handlung sowie die Ausgestaltung des Spieles im Konzept festzuhalten. Bevor der Stil und die Spielgeschichte festgelegt werden und entstehen können, muss sich auf die Zielgruppe geeinigt werden.

Bei der Entwicklung des Konzepts werden Recherchen zu Lernanwendungen, Charakteren und Zielgruppen durchgeführt, um einen Überblick zu erhalten. Anschließend werden Brainstormings zur Spielidee und zum Charakteraussehen abgehalten. Skizzen werden erstellt, überarbeitet und weiterentwickelt. Neben bereits bekannten Interaktionsmöglichkeiten, sollen auch weitere intuitive Interaktionsmöglichkeiten für die direkte und indirekte Interaktion mit dem Charakter ausgearbeitet. Zudem werden die Navigation in der 3D-Spielwelt sowie die Animationen und Bewegungen des Charakters festgelegt. Dabei werden nicht alle Ideen und Ergebnisse der Ausarbeitung vorgestellt, sondern exemplarisch wichtige Bestandteile, die in der Anwendung umgesetzt wurden, aufgezeigt.

3.3.1 Zielgruppenbestimmung

Anhand unterschiedlicher Erfahrungen, Wissen, Bildung, Interessen sowie Differenzen in der Wahrnehmung ist es wichtig die Zielgruppe vor Beginn der Arbeit zu definieren. [Wir04] Da in dieser Arbeit eine Spiel- und Lernanwendung entstehen soll, werden als potentielle Zielgruppe Kinder im Alter von sechs bis zehn Jahren gewählt. In diesem Alter benötigen Kinder eine pädagogische Begleitung bei der Nutzung von Medien.
In der heutigen Gesellschaft haben sich die Medien derart etabliert, dass sie aus unserem Leben nicht mehr wegzudenken sind. So ist es auch kein Wunder, dass die heutige Kindheit stark von den Medien beeinflusst wird und der Markt sich durch die Ausbreitung der neuen Medien mehr auf die Zielgruppe “Kind” ausrichtet. [Bic01][TE09]

Die Bemühungen von vielen Erwachsenen, Kinder von Medien fern zu halten, ist in der heutigen Mediengesellschaft nicht mehr der richtige Weg. Vielmehr sollten Kinder schon im frühen Alter den richtigen Umgang mit den Medien erlernen. Hier spielt auch der Gewinn an Qualifikationen eine wichtige Rolle. So ist es eine pädagogische Aufgabe, Kinder auf die Medienwelt vorzubereiten und ihnen bei der Orientierung zu helfen. [TE09]

Allein durch den Besitz von Medien lernen Kinder noch nicht den richtigen und sinnvollen Umgang mit ihnen. Durch das zur Verfügung stellen von Lernprogrammen oder Lernspielen, sogenannten Edutainment-Programmen, können sich Kinder mit neuen Medien auseinandersetzen. Gleichzeitig haben sie die Möglichkeit, sich mit neuen Themen auseinander zu setzen und ihr Wissen zu erweitern. [TE09]
Mit der im Rahmen dieser Arbeit entwickelten Anwendung soll Kindern die Möglichkeit gegeben werden, sich mit Hilfe neuer innovativer Techniken, Wissen spielerisch anzueignen.

3.3.2 Stilfestlegung

Nachdem die Zielgruppe festgelegt ist, wird recherchiert, wo und wie in anderen Anwendungen virtuelle Charaktere auftreten. Es ist sehr auffällig, dass eine Vielzahl von virtuellen Charakteren ein comichaftes Äußeres haben. Dieser Stil hat sich etabliert, da viele Benutzer, darunter besonders Kinder, solche Charaktere ansprechend und sympathisch finden. So wird auch für dieses Spiel der Comicstil gewählt und festgelegt, dass bei der Erstellung des Charakters die Merkmale des Kindchenschemas berücksichtigt werden sollen.

Das Kindchenschema bezeichnet die Merkmale eines Kleinkindgesichtes. Dazu zählen neben einem großen Kopf, einer großen Stirnregion, rundlichen Wangen auch große runde Augen, eine kleine Nase sowie ein kleines Kinn. Der kindliche Kopf ist, anders als beim erwachsenen Menschen, im Vergleich zum restlichen Körper größer und Gliedmaßen, wie Arme, Beine und Finger, sind kürzer. Wenn diese Merkmale in den Comicstil integriert werden, werden Assoziationen, wie zum Beispiel "süß" und "niedlich", beim Betrachter erzeugt.[Tin09][Wir04]

Die Umgebung des Spieles muss an den Charakter und auch an die Zielgruppe angepasst werden. Es sollte keine gruselige und furchterregende Stimmung erzeugt werden. Die Farben müssen möglichst freundlich gewählt werden und sollten dabei die vermittelte Atmosphäre nicht stören.

3.3.3 Aufbau und Spielgeschichte

Bei der Ausarbeitung der Spielidee soll eine Anwendung für Kinder entstehen, welche Wissen spielerisch und unterhaltsam vermittelt. Möglichkeiten bzw. Ideen werden in einem Brainstorming gesammelt.
Dabei setzt sich die Idee eines Weltraumszenarios durch, in der ein virtueller Charakter einzelne Planeten bereisen kann und auf Wunsch des Benutzers Informationen zum aktuellen Planeten vermittelt. Die Spielwelt ist das Weltall bzw. das Sonnensystem, in dem sich die Erde befindet. Es werden Informationen zum Sonnensystem und den einzelnen Planeten zur Verfügung gestellt.

Die Spielgeschichte

> Ein neugieriger Weltraumreisender trifft auf unser Sonnensystem und möchte mehr darüber erfahren. Sein Name ist "Newtrix". Er beschließt die Planeten näher zu untersuchen. Dabei hat er auf jedem Planeten besondere Erlebnisse. Nachdem alle Planeten besucht wurden, verlässt Newtrix das Sonnensystem und macht sich wieder auf die Reise zurück zu seinem Heimatplaneten.

Nachdem die ersten groben Züge der Spiel-Anwendung feststehen, können die Einzelheiten und Details ausgearbeitet und das Konzept weiterentwickelt werden. Dabei müssen die verschiedenen Aspekte, wie das Design der Welt und des Charakters sowie der Ablauf des Spiels, konkretisiert werden.

Zur besseren Visualisierung und zum besseren Verständnis werden die Maßstäbe der Planeten und ihre Entfernungen zueinander nicht realitätsgetreu nachgebildet. Jedoch wird die Reihenfolge der Planeten im Sonnensystem eingehalten. Der Charakter wird möglichst groß gewählt, um in der Umgebung nicht "unterzugehen". Dadurch wird gewährleistet, dass der Charakter im Mittelpunkt der Anwendung steht.

Besucht Newtrix einen Planeten, so stellt er sich oben auf die Planetenkugel und scannt den Planeten ab. Während Newtrix auf dem Planeten steht, bewegen sich um ihn herum die anderen Planeten langsam auf ihrer Kreisbahn um die Sonne. So sieht der Benutzer das Sonnensystem je nach Planet aus einem anderen Blickwinkel.

Ist das Scannen abgeschlossen, kann auf die gesammelten Informationen zugegriffen werden. Um den Charakter weiterhin im Fokus des Benutzers zu belassen und ihn in alle Aktionen und Funktionen der Anwendung einzubeziehen, wird entschieden, dass Newtrix einen fernseher- bzw. monitorähnlichen Kopf besitzt. Darauf können die Informationen über die Planeten abgerufen bzw. angezeigt werden. Dies geschieht durch Betätigung des "Info-Buttons" am unteren linken Fensterrand. Ein Kameraflug auf den Kopf zu wird ausgeführt und ein Video zum jeweiligen Planeten wird auf dem "Fernsehkopf" abgespielt. Zusätzlich ermöglicht der Fernsehkopf des Charakters noch weitere Funktionen. So wird beschlossen, dass nun die gesamte Gesichtsanimation und Mimik mit dem Einsatz von Videos im Kopf von Newtrix umgesetzt wird.
Daraus ergibt sich, dass Newtrix einem Astronauten ähneln, jedoch einen Fernseher als Kopf besitzen soll. Weitere Details zum Charakter und seinem Aussehen werden in Kapitel 3.3.6 ausgearbeitet.

Auf jedem Planeten ist ein besonderes Ereignis möglich. Ein Beispiel hierfür ist, dass ein Satellit um die Erde kreist. Erreicht dieser eine bestimmte Position über dem Planeten, werden Strahlen sichtbar, die den Charakter durcheinander bringen und Störsignale auf seinem Fernsehkopf verursachen. Auf dem Mars erscheint nach einer gewissen Zeit ein "Marsmännchen" und greift Newtrix mit einer fliegenden Untertasse an (siehe Abbildung 3.1). Durch die geringere Schwerkraft wird auf dem Mond eine andere Laufanimation abgespielt. Der Laufstil des Charakters wird dabei eher hüpfend sein, wie es von Aufnahmen auf dem Mond bekannt ist.

Um Newtrix neugierig wirken zu lassen, werden weitere Funktionen ausgearbeitet. Der Benutzer erscheint dem Charakter als spannend und deshalb schaut Newtrix die meiste Zeit in die "Kamera" bzw. den Benutzer an. Selbst bei Änderungen der Kameraposition bleiben die Augen stets auf den Benutzer gerichtet. Passiert aber etwas in der Umgebung des Charakters, so schaut er dorthin. Bespiel dafür ist ein Komet, der von Zeit zu Zeit auftaucht und an ihm vorbeifliegt.

Wird eine Weile die Touch-Oberfläche nicht bedient, werden verschiedenen Warteanimationen vom Charakter ausgeführt, zum Beispiel dass Newtrix nach einer Weile mit Winken auf sich aufmerksam macht oder sich gelangweilt in der Gegend umschaut.

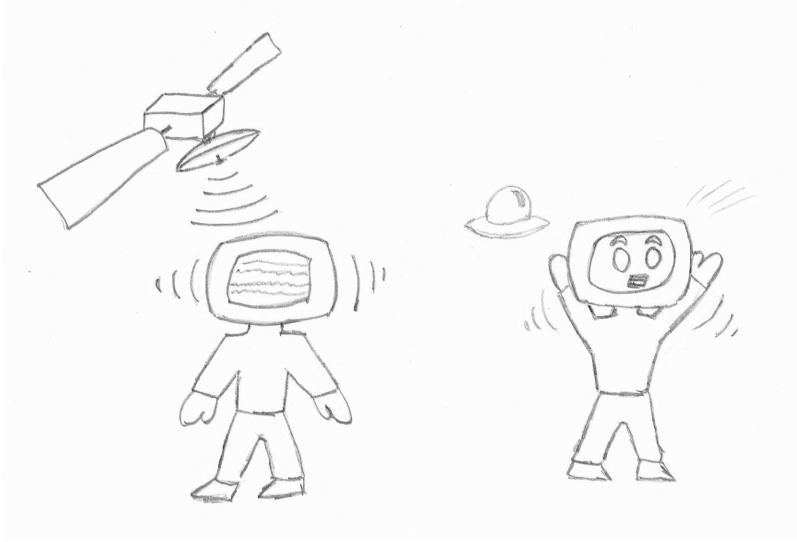

Abbildung 3.1: Skizzen zur Entwicklung von Animationen auf verschiedenen Planeten. Links: Satellit, dessen Strahlen den Charakter verwirren. Rechts: Angriff des Marsmännchens auf den Charakter.

Der Planetenwechsel wird durch ein Menü im unteren rechten Bildrand realisiert. Der Benutzer kann durch Drücken und Halten des Sonnen-Buttons ein Menü mit den Planeten des Sonnensystems zum Vorschein bringen. Durch Druck auf einen der Planeten wird ein neues Level geladen. Dabei ist ein Video zu sehen, wie Newtrix sich dem gewählten Planeten nähert.

Um im Weltall von Planet zu Planet reisen zu können, ist ein Fortbewegungsmittel notwendig. Hierfür erhält der Charakter ein Jetpack. Hat der Charakter einen Planten besucht, wird dieser im Menü deaktiviert.

Aufbau eines Spiellevels

Den Aufbau eines Levels zeigt Abbildung 3.2. Zu sehen ist eine Charakterskizze in der Mitte des Bildes. Im Hintergrund ist der Weltraum angedeutet. Im unteren rechten Bereich des Bildes ist das Planetenwechselmenü. Ein Info-Button (zum Informationsabruf) ist im unteren linken Bereich positioniert.

Hat der Benutzer alle Planeten besucht, erscheint der Heimatplanet von Newtrix im Menü, wobei nun nur noch dieser ausgewählt werden kann. Wird er ausgewählt, startet das “End”-Video und Newtrix fliegt nach Hause.

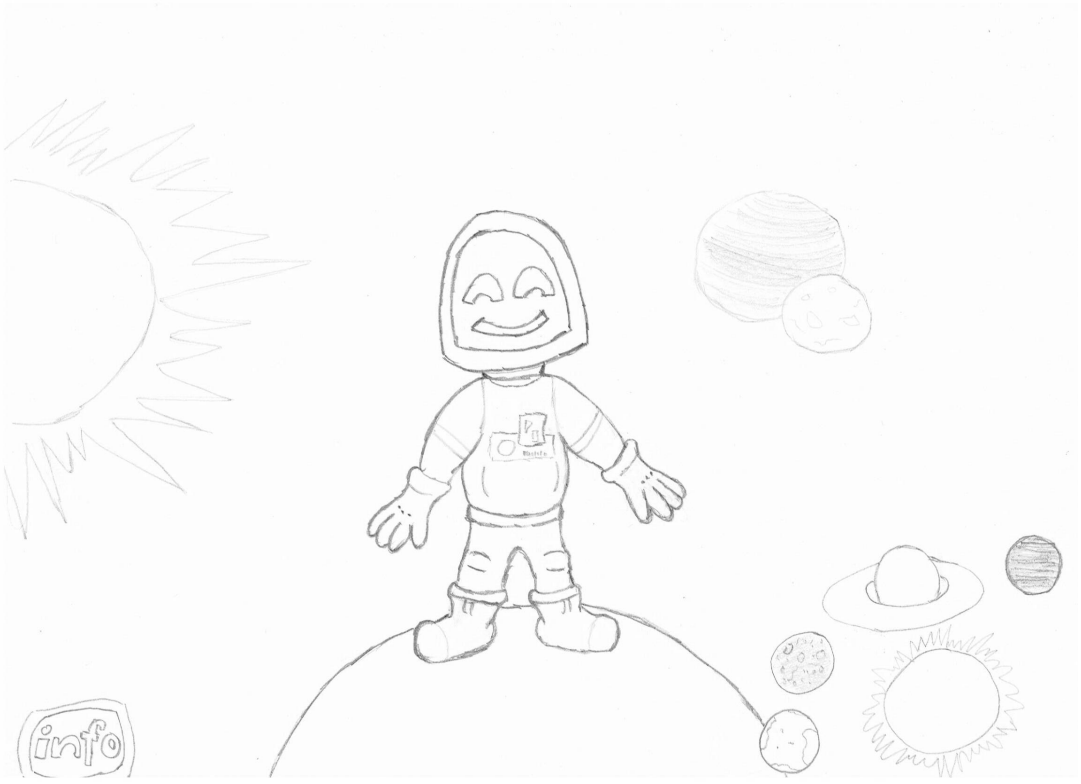

Abbildung 3.2: Skizze, welche die erste grobe Übersicht der Spielwelt zeigt.

3.3.4 Interaktion

Nachdem Aufbau und Spielgeschichte der Anwendung feststehen, werden die Interaktionsmöglichkeiten und die Navigation in der Spielwelt ausgearbeitet. Durch die Verwendung von Fingern zur Interaktion mit einem Charakter ergeben sich ganz neue Möglichkeiten, die über das "normale" Interagieren mit der Maus bzw. Tastatur hinausgehen. So ist ein intuitiveres Interagieren mit dem Charakter möglich. Dabei kann der Benutzer Touch-Gesten, wie beispielsweise das Streicheln, Kratzen, Anstoßen oder Schubsen ohne Einweisung oder Erklärung einsetzen.

Die einfachste Form der Interaktion mit dem Charakter ist das Drücken auf bestimmte Auslöser am Körper oder Kopf von Newtrix. Ein offensichtlicher Auslöser ist der Startknopf, der den Jetpack aktiviert und den Charakter fliegen lässt. Nachdem Newtrix abgehoben ist, erscheinen zwei Anzeigen links und rechts vom Charakter. Nach Halten des rechten "Schub"-Buttons, kann auf der linken Anzeige die Intensität und die Richtung des Schubs bestimmt werden. Der Einfachheit halber ist nur ein Auf- oder Abwärtsfliegen möglich.

Weiter kann auf die Füße des Charakters gedrückt werden. Beim mehrmaligen Drücken fängt Newtrix an auf einem Bein zu hüpfen. Durch Druck auf das Display des Charakters entsteht dort ein Fingerabdruck, welcher vom Charakter weggewischt wird.

Grundlegend kann mit bereits bekannten Touch-Gesten in der Spielwelt navigiert werden. Mit einfachem Ziehen des Fingers über die Touch-Oberfläche erfolgt die Drehung der Spielwelt. Auch ein Zoomen wird ermöglicht. Dazu müssen zwei Finger voneinander wegbewegt werden, um hinein zu zoomen und aufeinander zubewegt, um raus zu zoomen. Da der Charakter der Mittelpunkt der Anwendung ist, wird die Kamera immer auf ihn gerichtet. So rotiert die Kamera um den Charakter und zoomt auch an ihn heran oder von ihm weg. Eingeschränkt sind hierbei nur der Winkel für die Drehung um die horizontale Achse sowie die minimale und maximale Entfernung zum Charakter. Damit wird eine Drehung über den Charakter hinweg sowie unter ihm hindurch verhindert. Die Einschränkung beim Zoomen hindert die Kamera daran durch den Charakter zu fliegen.

Eine der grundlegenden Interaktionen ist das Laufen von Newtrix auf den Planeten. Dazu muss der Benutzer den jeweiligen Planeten durch Ziehen eines Fingers auf der Planetenoberfläche drehen. Dabei ist vorwärts und rückwärts Laufen möglich. Der Charakter passt sein Lauftempo an die Drehgeschwindigkeit an. Wird das Tempo jedoch zu hoch für ihn, so fällt er vom Planeten.

Durch Einsatz von zwei Fingern, verbunden mit dem Ziehen über den Charakter, ist ein "schubsen" möglich. Dabei verliert Newtrix kurz sein Gleichgewicht, rudert mit den Armen und fängt sich aber wieder. Dies gilt sowohl für das Schubsen von links, als auch für das Schubsen von rechts.

Eine Auflistung der direkten und indirekten Interaktionen und die dazugehörigen Auslöser werden im Folgenden erläutert.

Direkte Interaktion

Interaktion, welche direkt am Charakter selbst ausgeführt werden:

- Doppelter Touch auf einen der Füße.
  Charakter beginnt auf einem Bein zu Hüpfen.
- Einzelner Touch auf den Startknopf auf der Konsole.
  Charakter startet den Jetpack und hebt ab.
- Einzelner Touch auf den Kopf bzw. das Display.
  Charakter wischt über das Display.
- Ziehen von zwei Fingern über den Charakter.
  Charakter verliert das Gleichgewicht und Wackelt.

Indirekte Interaktion

Interaktion, welche nicht am Charakter ausgeführt werden, aber ihn dennoch beeinflussen:

- Drehen des Planten, auf dem der Charakter steht. Charakter beginnt zu Laufen.
- Halten des Schub-Buttons und bewegen über den Geschwindigkeits-Button. Charakter verändert seine Flughöhe.
- Drehen in der 3D-Szene durch Ziehen des Fingers über die Touch-Oberfläche. Charakter sieht den Benutzer an und dreht gegebenenfalls den Kopf.
- Ein Zeit lang keine Berührung der Touch-Oberfläche.
  Charakter beginnt Animationen wie: Winken, sich Umschauen, Gähnen. Auf dem Mars erscheint ein Ufo.
- Betätigen des Info-Buttons.
  Charakter spielt ein Info-Video über den aktuellen Planeten ab.
- Auswahl eines neuen Planeten über das Planetenwechselmenü.
  Charakter beginnt eine Scan-Animation auf dem neuen Planeten.

3.3.5 Spielablauf

Der Spielablauf wird im Zustandsdiagramm 3.3 dargestellt. In den Kreisen ist der Zustand bzw. die jeweilig ausgeführte Aktion zu sehen. Die Kreise werden durch Pfeile verbunden. Die Pfeile zeigen die Verbindungen der Zustände und somit auch aus welchem Zustand ein Anderer erreicht werden kann. Der Text an den Pfeilen beschreibt den Auslöser, um von einem Zustand in den anderen zu wechseln. Pfeile, die kein Auslöseereignis haben, werden automatisch nach Beendigung des Zustandes, meist nach Ende einer Animation oder Ablauf einer bestimmten Zeit, ausgeführt.

Beim Start der Anwendung wird vorab ein "Intro"-Video abgespielt, in dem zu sehen ist wie Newtrix auf den Planeten zufliegt. Ist dieses beendet, startet eine Animation, in der Newtrix den Planeten scannt. Diese Animation wird immer aufgerufen, wenn ein Planetenwechsel über das Auswahlmenü ausgelöst wird. Ein Planetenwechsel kann jederzeit erfolgen, egal welcher Zustand in der Szene aktiv ist. Beim Planetenwechsel wird der ausgewählte Planet im Menü gesperrt, so dass er nicht ein zweites Mal ausgewählt werden kann.

Im Zentrum des Ablaufs stehen die "Idle"-Animationen. Diese sind minimale Animationen eines Charakters, die abgespielt werden, wenn gerade keine Aktion ausgeführt wird. Dabei bewegen sich die Körperteile leicht, so dass der Charakter lebendig wirkt. Der Charakter ist dabei nicht auf bestimmte Auslöser angewiesen. Vielmehr ist eine Idle-Animation der Grundzustand, zu dem er immer wieder zurückkehrt, nachdem eine Aktion abgeschlossen ist. Alle anderen Aktionen sind mit einem Auslöser verbunden.

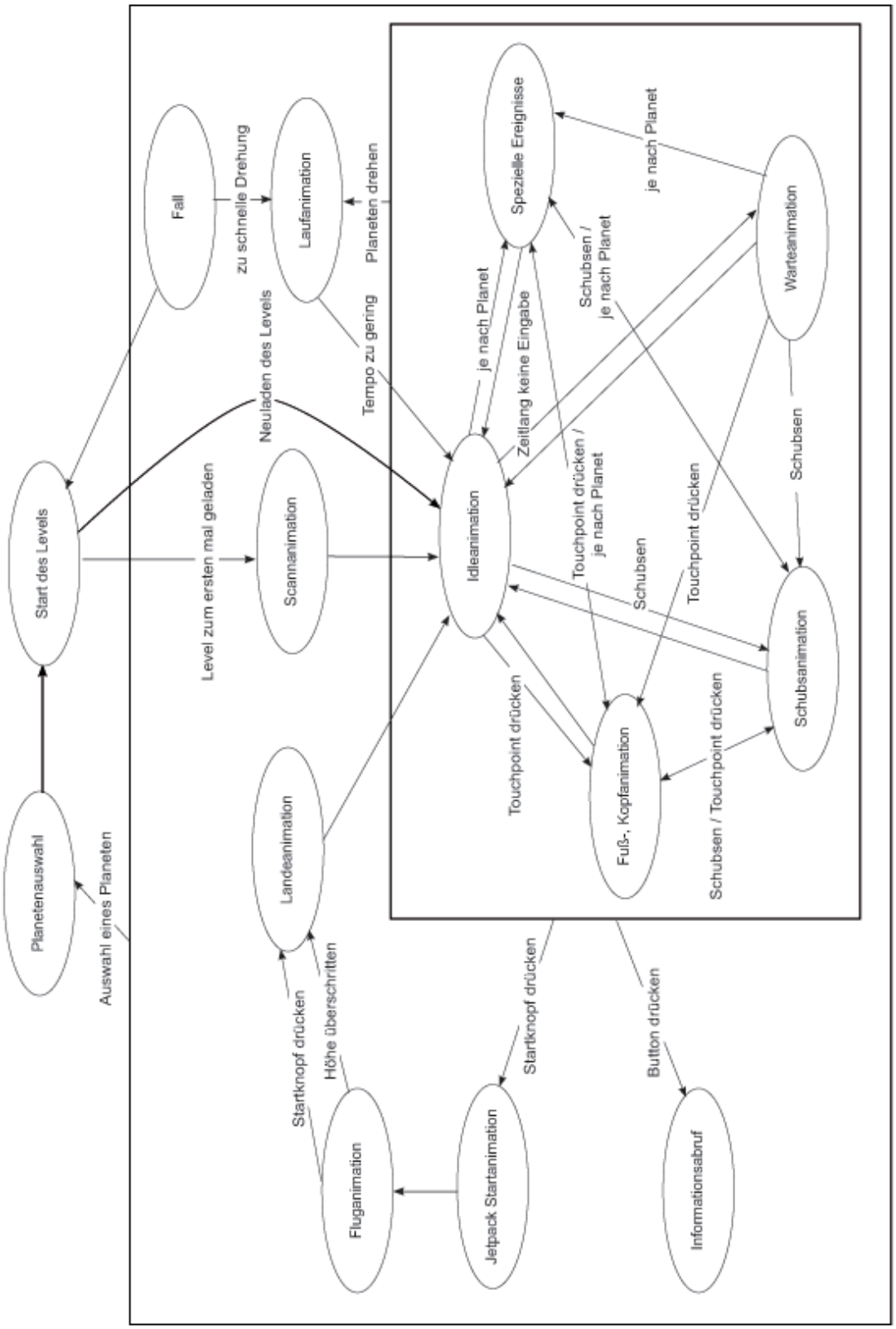

Abbildung 3.3: Das Zustandsdiagramm zeigt die Verknüpfungen im Ablauf des Spiels. Es wird gezeigt welche Aktionen zu welchem Zeitpunkt möglich sind.

Das Abrufen der Planeteninformationen über den Info-Button, ist nach dem Scannen des Planeten möglich. Auch während laufender Warte-, Schubs-, Kopf- und Fußanimationen kann der Info-Button benutzt werden. Er wird jedoch ausgeblendet oder gesperrt, sobald der Charakter die Fluganimation startet.

Durch bestimmte Touch-Punkte, beispielsweise an den Füßen, am Kopf oder dem Startknopf am Bauch, wird eine Animation aufgerufen. Bei Berührung des Kopfes wird auf seinem Bildschirm zusätzlich ein Fingerabdruck eingeblendet. Der Startknopf am Bauch löst neben einer Animation noch ein Video am Kopf des Charakters und Flammen am Jetpack aus. Nachdem das Video beendet ist, steigt Newtrix ein Stück auf und beginnt mit einer Fluganimation. Zudem erscheinen zwei weitere Buttons neben Newtrix. In diesem Zustand sind weitere Aktionen möglich, die Grundaktionen hingegen sind gesperrt. So ist beispielsweise das Drehen des Planeten nicht mehr möglich. Durch Benutzen der Buttons kann Newtrix bis zu einer festgelegten Höhe aufsteigen. Ein Signal und eine rote Lampe warnen den Benutzer, sobald diese Höhe erreicht wird. Überschreitet er diese, werden die zum Fliegen benötigten Buttons blockiert. Der Charakter sinkt daraufhin ab und startet die Landeanimation.

Die Drehgeschwindigkeit des Planeten sowie die Laufgeschwindigkeit werden nach dem Anstoß des Planeten gedämpft. Dies führt dazu, dass die Laufanimation verlangsamt. Ist ein Grenzwert erreicht, wird zur Idle-Animation überblendet. Bei einer zu starken Drehung des Planeten, bei Überschreiten einer bestimmten Geschwindigkeit, fällt Newtrix vom Planeten und schwebt im Weltraum davon. Beim Schweben rudert er mit Armen und Beinen, bis das aktuelle Spiellevel neu gestartet wird. Dabei werden jedoch das Startvideo und die Scan-Animation übersprungen. Während dieses Zustands können keine anderen Aktionen, wie beispielsweise Starten des Jetpacks oder Schubsen, vom Benutzer ausgelöst werden.

Je nach Planet können verschiedene Ereignisse auftreten. Auf der Erde wird Newtrix kurzzeitig von einem Satellitenstrahl verwirrt. Diese Aktion hat aber keinen Einfluss auf andere Zustände. Auf dem Mars hingegen wird eine Animation mit einem Ufo abgespielt. Währenddessen kann der Benutzer keine anderen Aktionen starten oder ausführen. Ausgenommen davon sind nur der Planetenwechsel und der Info-Button. Das spezielle Ereignis auf dem Mond ist eine veränderte Laufanimation. Beim Drehen des Mondes ist das Laufen des Charakters eher ein Hüpfen, was auf die geringe Schwerkraft bezogen ist.

Sind alle Planeten besucht, erscheint der Heimatplanet von Newtrix im Auswahlmenü. Wird dieser ausgewählt, ist das Spiel zu Ende. Ein abschließendes Video und Credits werden abgespielt.

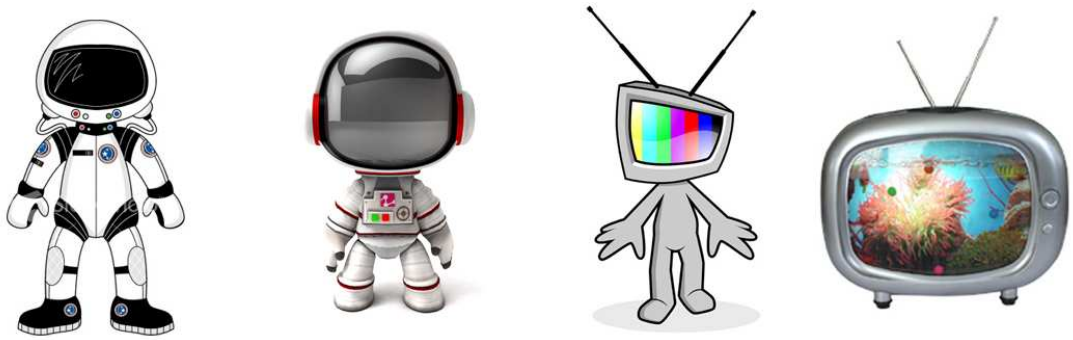

Abbildung 3.4: Verschiedene Ideenvorschläge für das Aussehen des Charakters.[4]

3.3.6 Charakterentwurf

Der Charakter in einer Anwendung wird stark durch seine Umwelt beeinflusst. Je nach Stil und Aussehen der Umgebung muss auch der Charakter in diese hineinpassen und das Gesamtbild muss stimmig sein. So wird Newtrix auf Grundlage der in Kapitel 3.3.2 ausgearbeiteten Vorstellungen ein cartoonhaftes Aussehen erhalten.

Cartoon Stil

Cartoon-Charaktere sind unrealistische Abbildungen, oft Karikaturen, von Menschen oder Tieren. Bei der Erstellung werden Details reduziert und bestimmte Züge einer Figur werden in den Proportionen verändert. So wird meist der Kopf solcher Charaktere vergrößert dargestellt. Aber auch Hände und Füße werden oft größer abgebildet. Zusätzlich wird häufig auch auf das Kindchenschema zurückgegriffen.

Da die 3D-Welt ein Weltraumszenario darstellt, folgt die Entscheidung, das Newtrix vom äußeren Erscheinungsbild Züge eines Astronauten besitzen muss. Zudem ist es notwendig, dass er einen Fernsehkopf bzw. einen Bildschirm, auf dem Videos abgespielt werden können, besitzt. Diese beide Merkmale müssen in die Charakterentwicklung mit einfließen. So entsteht eine Mischung aus Astronaut und einem Charakter mit Fernseherkopf.
Um Ideen und Inspirationen zum Charakteraussehen zu erhalten, werden Recherchen durchgeführt. Abbildung 3.4 zeigt einen Überblick über die verschiedenen Charaktere, die als Beispiele dienen.

Zur Fortbewegung des Charakters ist ein Jetpack vorgesehen. Ein Entwurf des Jetpacks ist in Abbildung 3.5 zu sehen. Der Jetpack wird am Rücken des Charakters platziert. Als Haltevorrichtung erhält der Charakter eine Weste. Ein Knopf zum Starten des Jetpacks wird

[4] 1.http://www.istockphoto.com/file_thumbview_approve/5808677/2/istockphoto_5808677-cute-spaceman-character.jpg
2.http://www.play3.de/wp-content/gallery/little-big-planet-220908/Spaceman_render.jpg
3.http://www.bytedust.nl/wp-content/uploads/2009/11/escapation_tvhead_basic-500x300.jpg
4.http://vulcanstev.files.wordpress.com/2009/03/cartoon_tv_lamp.jpg

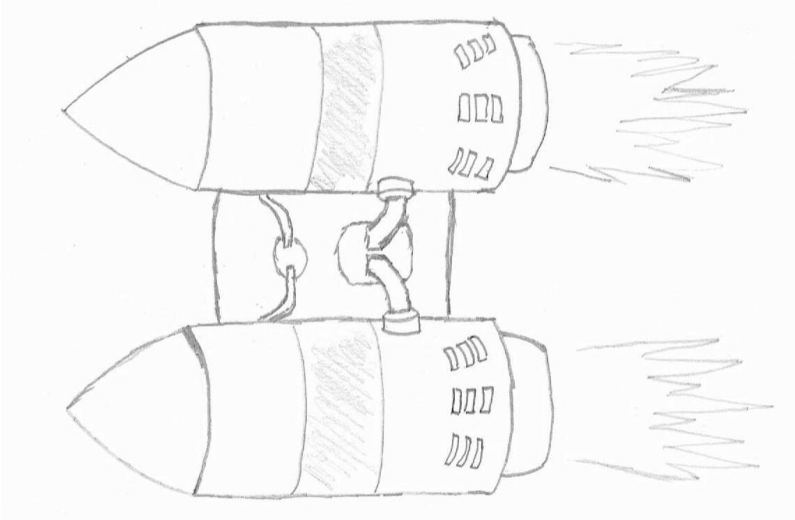

Abbildung 3.5: Entwurf des Jetpacks für den Charakter.

auch an der Weste auf einer Konsole angebracht. Die Konsole befindet sich in der Körpermitte von Newtrix. Neben dem Startknopf an der Konsole sind auch eine grüne und eine rote Lampe angebracht. Diese geben dem Benutzer eine Rückmeldung zum Status des Jetpacks. Dabei steht ein aufleuchtendes Grün für "Alles in Ordnung" und ein aufleuchtendes Rot für "Vorsicht".

Abbildung 3.6 zeigt die erstellten Skizzen, an denen die Entwicklung des Aussehens von Newtrix abzulesen ist.

- Das erste Bild zeigt eine der frühsten Charakterskizzen. Sie dient als erster Anhaltspunkt für die weitere Charakterentwicklung. Dabei wird das Kindchenschema nicht so stark berücksichtigt.
- Im zweiten Bild ist ein cartoonhafter Astronaut zu sehen. Die Körperproportionen und der Kopf entsprechen eher den gesuchten Charaktermerkmalen. Jedoch fehlt das Display für die Videos.
- Das dritte Bild zeigt eine Mischung aus Bild eins und zwei. Hier erfüllt der Charakter die Vorgaben des Stils und des Charakteraussehens, ist jedoch noch nicht detailliert ausgearbeitet. Er bildet aber eine gute Grundlage für die weitere Entwicklung von Newtrix.
- Im vierten Bild ist die Vorlage für Newtrix zu sehen. Hier wird die Grundform des vorherigen Entwurfs beibehalten, jedoch fließen weitere Details wie Konsole und die Weste ein.

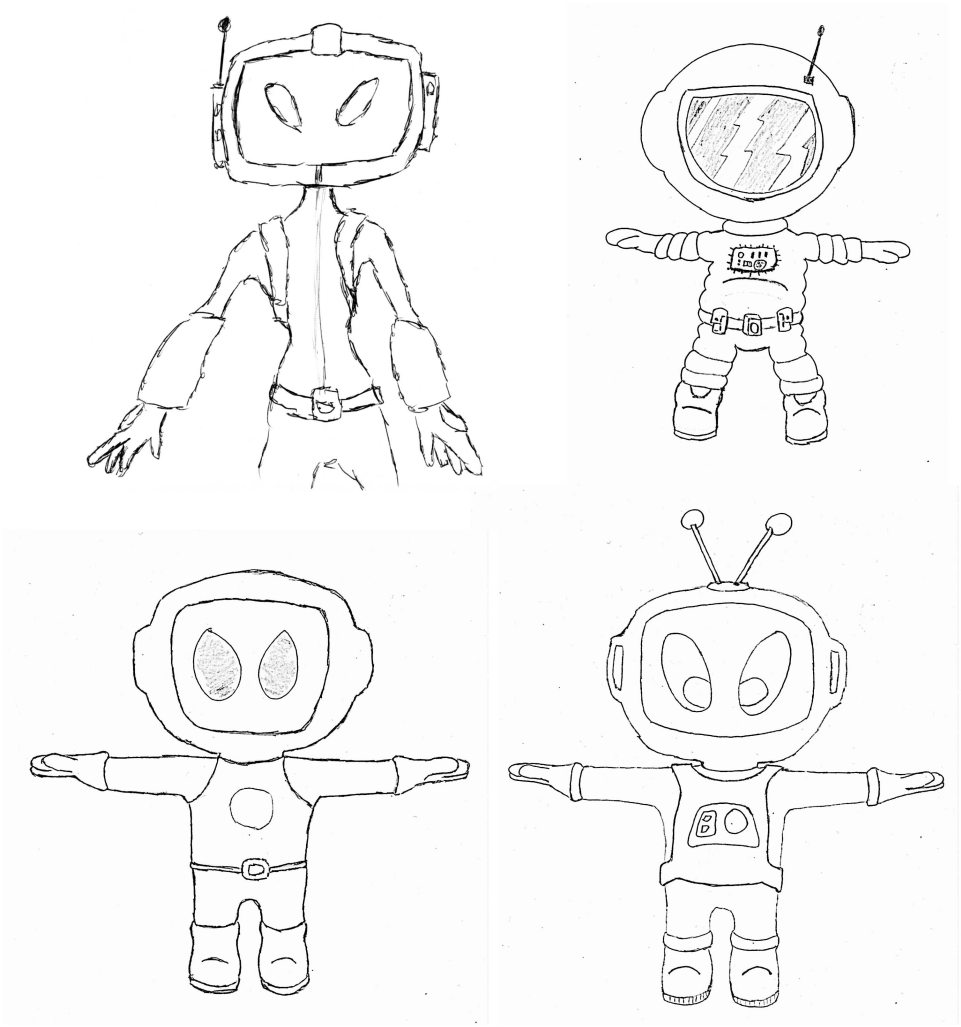

Abbildung 3.6: Vier Skizzen für den Charakterentwurf

Die entstandene Vorlage muss nun noch farbig gestaltet werden, wobei die endgültige Farbgebung und Design auch in einem Entwicklungsprozess gefunden wurden. Die Fläche am Hinterkopf wird dazu genutzt, sie mit einem Logo zu verzieren. In Kapitel 5 wird die Umsetzung (Modellierung, Texturierung, Rigging, Skinning) des Charakters ausführlich beschrieben.

3.3.7 Verwendung von Sounds

Ein wichtiges Element in Spielen ist der Sound bzw. sind die Geräusche in einer Spielwelt. Obwohl die meisten Geräusche von Menschen nicht bewusst wahrgenommen werden, fällt das Fehlen wiederum sofort auf. Besonders die Hintergrundmusik spielt für die Stimmung eine große Rolle. [Ste08] Zudem sind Geräusche gut geeignet, dem Benutzer eine Rückmeldung zu seinen Aktionen zu geben. Wie ein bestimmtes Geräusch klingen muss, ist nicht

genau definiert. Wichtig ist nur, dass es mit den Erwartungen des Benutzers übereinstimmt um Mehrdeutigkeiten zu vermeiden. Um die Kommunikation zwischen Spieler und Spiel zu erleichtern, wird der Sound in Verbindung mit einem visuellen Eindruck vermittelt. So wird dieser verstärkt und es wird gewährleistet, dass die Informationen beim Benutzer ankommen. Über Sounds können Informationen bereitgestellt werden, die außerhalb des Sichtfeldes des Benutzers liegen. [Gru08]

Bei der Musikauswahl wird darauf geachtet, dass die Musik in der Anwendung nicht bedrohlich wirkt. Für die Grundstimmung im Weltall wird ein mystisch wirkendes Musikstück ausgewählt, welches in einer Endlosschleife in jedem Level laufen wird.

Um dem Benutzer eine Rückmeldung zu verschiedenen Aktionen des Charakters zu geben, werden beispielsweise beim Laufen über den Planeten Geräusche von Schritten abgespielt. Auch die Wischanimation, in der Newtrix den Fingerabdruck von seinem Display abwischt, wird mit Geräuschen unterlegt. Weiterhin strahlt der Komet Geräusche aus, während er seiner Umlaufbahn folgt. Dabei werden die Geräusche lauter, wenn er sich dem Charakter nähert und leiser wenn sich der Komet entfernt. Der Satellit, der um die Erde kreist, gibt ein Piepsen und ein Geräusch, welches das Aussenden von Strahlen untermalt, von sich, sobald die Strahlen auf den Charakter abgefeuert werden. Auch das Ufo, welches den Charakter auf dem Mars angreift, hat einen Flugsound. Beim Info-Button und dem Planetenauswahlmenü werden Sounds eingebunden, um dem Benutzer eine Rückmeldung zu geben, wann und ob er einen dieser Buttons gedrückt hat.

3.4 Zusammenfassung

In diesem Kapitel wurde das Konzept für die Anwendung ausgearbeitet. Zuerst wurden verschiedene Anforderungen definiert. Dazu gehört neben der Verknüpfung von Technologien, Hardwareunabhängigkeit, Echtzeitdarstellung, Animationsüberblendung und Zielgruppenorientierung auch die sinnvolle Nutzung von Multi-Touch-Gesten. Auf Grundlage dieser Anforderungen entstand das Anwendungskonzept.

Ziel der Anwendung soll es sein, Kindern spielerisch Wissen zu vermitteln und sie an neue Medien heranzuführen. Im Zentrum der Anwendung wird ein Charakter stehen, mit dem die Kinder interagieren können. Nachdem die Zielgruppe bekannt war, wurde der Stil der Lern- und Spielanwendung festgelegt. Der Charakter wird dazu ein comichaftes Aussehen erhalten und auch die Umgebung muss kindgerecht aufgebaut werden. Es wurde beschlossen, dass die Anwendung Wissen über die Planeten im Sonnensystem zur Verfügung stellt. Die Spielgeschichte beschreibt die Reise eines Weltraumreisenden namens Newtrix, der das Sonnensystem besucht und alles darüber lernen will. Dabei erlebt er verschiedene Ereignisse auf den unterschiedlichen Planeten. Nach dem Besuch aller Planeten kehrt Newtrix zu seinem Heimatplaneten zurück.

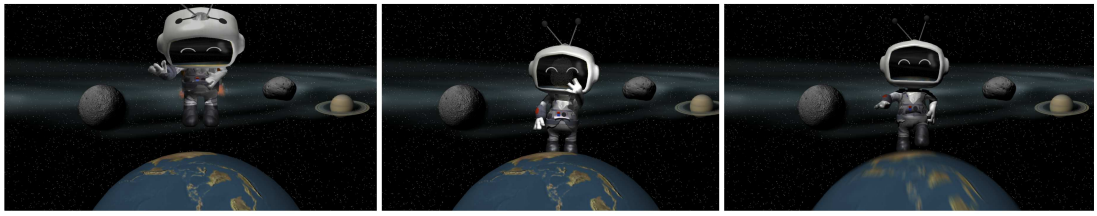

Abbildung 3.7: Testrenderings für den Ausblick auf das Aussehen der Anwendung.

Wenn der Charakter einen Planeten besucht, stellt er sich auf ihn. Die Verhältnisse von Größe und Entfernung der Planeten werden nicht maßstabsgetreu übernommen. Newtrix wird möglichst groß dargestellt. Für die Interaktion mit dem Charakter über Touch bzw. Multi-Touch wurden direkte und indirekte Interaktionsmöglichkeiten entwickelt. So steht Newtrix zur Laufzeit der Anwendung auf einem Planeten und interagiert mit dem Benutzer. Neben dem Wechsel der Planeten werden auch die zugehörigen Informationen dazu bereitgestellt. Die Informationen werden auf einem Monitor, der dem Charakter als Kopf dient, angezeigt. Weiter gibt es noch andere Interaktionsmöglichkeiten für den Benutzer. So kann er, zum Beispiel durch Drehen der Planeten, Newtrix zum Laufen bringen. Weitere Funktionen sind beispielsweise der Einsatz eines Jetpacks und das Zeigen von Videos oder Animationen.

Ein weiterer Punkt im Konzept war das Ausarbeiten des Spielablaufs. Hier wurde festgehalten, welche Animationen zu welcher Zeit möglich sind. Dies wurde mit Hilfe des Zustandsdiagramms in Abbildung 3.3 erklärt.

Ebenfalls wurde der Charakter entworfen. Dazu wurden Ideen gesammelt und Skizzen erstellt, um die Anforderungen, ausgehend von der Anwendungsidee, umzusetzen. Das Ergebnis ist ein astronautenähnlicher, im Cartoonstil gehaltener Charakter mit Fernsehkopf und Jetpack am Rücken.

Weiter wurden Überlegungen zum Einsatz von Musik und Geräuschen angestellt und Aktionen, bei denen Sounds eingebunden werden, festgelegt. Entscheidend dabei war, dass der Benutzer zusätzlich zum visuellen auch ein akustisches Feedback zu seinen Aktionen erhält. Hintergrundmusik und Umgebungsgeräusche sind dabei ebenfalls wichtig für die Stimmung in der Anwendung.

Nachdem die Entwurfsphase abgeschlossen war, wurden Testrenderings zur Anwendung erstellt. Dazu wurde ein Prototyp von Newtrix modelliert und durch Bildbearbeitung entstanden Entwürfe (siehe Abbildung 3.7), die einen konkreteren Einblick in die fertige Spiel-Anwendung geben sollten. Im nachfolgenden Kapitel 4 werden Entwicklungsumgebungen verglichen und erste Testanwendungen erstellt. Dies dient dazu, Stärken und Schwächen einer Entwicklungsumgebung zu erkennen und daraus Rückschlüsse für die Anwendungsentwicklung zu ziehen.

Kapitel 4

Gegenüberstellung geeigneter Entwicklungsumgebungen

Im Kapitel 2.5 wurde auf die wichtigsten Merkmale einer Echtzeit 3D-Entwicklungsumgebung sowie die Unterschiede zwischen Game-Engines und anderen speziellen Entwicklungsumgebungen eingegangen. Es wurde festgestellt, dass die meist verbreiteten Echtzeit Entwicklungsumgebungen sogenannte Game-Engines sind. Diese stellen eine Sammlung von Technologien und Hilfsmitteln zur Verfügung, welche das Erstellen von Spielen bzw. Echtzeitanwendungen in großem Maße unterstützt und erleichtert. Darunter fallen unter anderem folgende Engines: Grafik-Engine, Sound-Engine und Physik-Engine. Ebenso ist auf Basis einer Game-Engine neben Spielen auch eine Vielzahl weiterer interaktiver Anwendungen möglich. So werden zum Beispiel bei Virtual Reality die gleichen Techniken wie bei einem Spiel genutzt und unter anderem wissenschaftliche Inhalte vermittelt. Entwicklungsumgebungen wie das Mixed-Reality-System instantreality, welche bei der Erstellung von Anwendungen im Bereich Virtual Reality und Augmented Reality eingesetzt werden, erweitern die Funktionalität der meisten Game-Engines, gerade was die Ein- bzw. Ausgabegeräte betrifft.

Mit den in Kapitel 3 gewonnenen Anforderungen an die Echtzeit 3D-Entwicklungsumgebung soll in diesem Kapitel aus der Vielzahl der vorhandenen Softwarepakete eine potentielle Auswahl geeigneter Entwicklungsumgebungen getroffen werden. Mit den zwei am besten geeigneten Entwicklungsumgebungen wird in diesem Kapitel jeweils eine schematische Testanwendung umgesetzt, um im direkten Vergleich die endgültige Entwicklungsumgebung festzulegen.

4.1 Einleitung

Bei der Erstellung einer Echtzeit 3D-Anwendung ist die Wahl der richtigen, d.h. der effizientesten, Entwicklungsumgebung ein wichtiger Bestandteil. Es gibt viele Entwicklungsumgebungen mit denen Echtzeit 3D-Anwendungen erstellt werden können. Sobald aber spezielle Anforderungen an die Anwendung gestellt werden, ist schnell ersichtlich, dass die Auswahl eher begrenzt ist. Daher soll zunächst ein Überblick geschaffen werden, der die verschiede-

nen Komponenten sowie Funktionen der einzelnen Entwicklungsumgebungen aufgezeigt und analysiert. Auf Basis dieser Auflistung kann dann die engere Auswahl getroffen werden, die im Laufe dieses Kapitels weiter ausgearbeitet wird. Zum Schluss wird entschieden, welche Anwendung für die Umsetzung des Projekts am besten geeignet ist.

4.2 Mindestanforderungen an die Entwicklungssoftware

Neben den Anforderungen aus der Aufgabenstellung sowie der im Kapitel 3 (Konzeptentwicklung) festgelegten Anforderungen an die Anwendung ergeben sich Anforderungen an die Entwicklungsumgebung. Die vier wichtigsten dabei sind:

Multi-Touch-Anbindung
: Die Entwicklungsumgebung muss in der Lage sein, Multi-Touch-Technologien als Eingabegerät zu nutzen. Da das TUIO-Protokoll momentan die beste Möglichkeit für plattformunabhängige Anwendungen darstellt, muss die Entwicklungsumgebung dieses unterstützen.

Charakter-Implementierung
: Ein weiterer Bestandteil in der Anwendung ist die Implementierung eines animierten Charakters. Diesbezüglich muss die Entwicklungsumgebung in der Lage sein, einen Charakter zu importieren, auf die einzelnen Komponenten zuzugreifen sowie die einzelnen Animationen zu manipulieren.

Animations-Überblendung
: Damit die Animationen in der Anwendung ohne weiteres kombiniert werden können, muss die Entwicklungsumgebung Methoden zur Überblendung von Animationen zur Verfügung stellen.

Einbinden von Video & Sound
: Wie in Kapitel 3 beschrieben, muss das Gesicht des Charakters mittels eines Videos angezeigt werden. So ist die Implementierung von Videos ein wichtiger Bestandteil bei der Umsetzung der Anwendung. Für die Atmosphäre der Anwendung müssen auch Sounds verwendet werden können.

Um einen guten Überblick über die Funktionalität der verschiedenen Softwarepakete zu erhalten, wird eine Auflistung der Entwicklungsumgebungen erstellt (siehe Tabelle 4.1[1,2,3,4,5,6,7]). Aus Gründen der Effizienz und Übersichtlichkeit wird in diese Auflistung nur ein kleiner Teil aller Entwicklungsumgebungen aufgenommen, der stellvertretend für das große Spektrum der Entwicklungsumgebungen steht.

[1] http://www.blender.org/
[2] http://crytek.com/cryengine/cryengine3/overview
[3] http://www.instantreality.org/
[4] http://irrlicht.sourceforge.net/
[5] http://www.ogre3d.org/
[6] http://unity3d.com/
[7] http://www.unrealtechnology.com/

Name	Sound-Engine	Physik-Engine	Multi-Touch	Grafik-Engine	Skript-sprachen	Plattform
Blender	Ja	Ja	Ja	OpenGL	C, C++, Python	Windows, Linux, Mac OS, Solaris
CryEngine	Ja	Ja	Nein	DirectX	C++	Windows, PlayStation 3, Xbox 360
instantreality	Ja	Ja	Ja	OpenGL	X3D/VRML, JavaScript, Java	Windows, Linux, Mac OS, SunOS, IRIX
Irrlicht	Ja	Ja	Nein	OpenGL, DirectX, OpenGL ES	C++, Python, Java, Perl, Ruby, Delphi	Windows, Linux, Mac OS, Sun Solaris
OGRE	Nein	Nein	Nein	LGPL, OUL	C++, Python, Java, C#, .NEZ	Windows, Linux
Unity3D	Ja	Ja	Ja	OpenGL, DirectX	JavaScript, C#, Boo	Windows, Mac OS, Wii, iOS
Unreal Engine	Ja	Ja	Ja	OpenGL, DirectX	C++	Dreamcast, XBox360, iOS, PlayStation 3

Tabelle 4.1: Auflistung verschiedener Entwicklungsumgebungen und ihre Bestandteile sowie ihre Kompatibilitäten.

4.3 Analyse der ausgewählten Software

Wie aus Tabelle 4.1 zu entnehmen ist, bleiben vier Entwicklungsumgebungen zur Auswahl übrig. Neben dem Mixed-Reality-System instantreality und der Game-Engine Unity3D kann auch die Game-Engine von Blender das TUIO-Protokoll und somit Multi-Touch unterstützen. Das französische Entwicklerteam schreibt auf seiner Webseite aber, dass es sich um eine inoffizielle Version von Blender handelt.[8] Die Unreal Engine ermöglicht die Erstellung von Multi-Touch-Spielen für das "Apple iOS". Das "iOS" ist ein Betriebssystem der Firma Apple, welches bei den Mobilen Geräten iPhone, iPod touch sowie iPad verwendet wird.[9] Somit ist bei der Nutzung der Unreal Engine die Hardwareunabhängigkeit nicht gewährleistet. Aus diesem Grund wird sie nicht weiter berücksichtigt. Bei näherem Vergleich von Unity3D und Blender wird festgestellt, dass die Dokumentation von Unity3D umfangreicher ist als die von Blender. Aus diesen Gründen wird Blender aus der engeren Auswahl entfernt.

[8] http://forge.lifl.fr/PIRVI/wiki/MTUtils/blenderTUIO

[9] http://www.gamingbits.com/general-gaming-news-bits/epic-games-unreal-engine-3-enters-iphone-ipod-touch-and-ipad-arena-with-stunning-epic-citadel-and-project-sword/

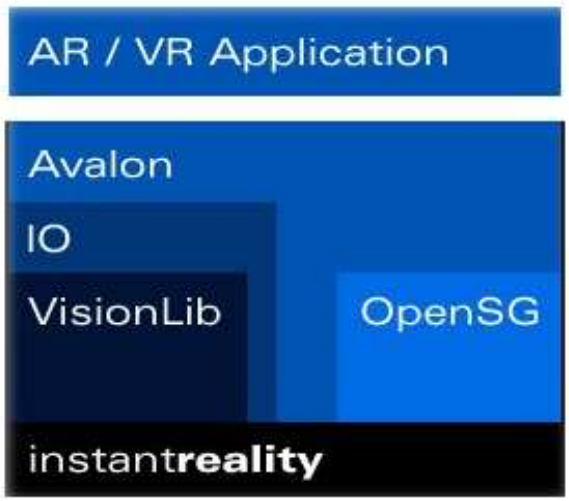

Abbildung 4.1: Aufbau des instantreality-Systems.[10]

Zur weiteren Analyse bleiben so das Mixed-Reality-System instantreality und die Game-Engine Unity3D. Im folgenden Abschnitt wird in jeder der beiden Entwicklungsumgebungen jeweils eine vergleichbare Testszene erstellt, in der die Unterschiede sowie Vor- und Nachteile der einzelnen Umgebungen aufgezeigt werden sollen.

Die Testszenen sind eine vereinfachte Form der im Kapitel 3 entstandenen Spielidee. Ein Testcharakter steht auf einer Weltkugel und soll durch Drehen dieser anfangen zu laufen. Durch Drehbewegungen außerhalb der Weltkugel soll sich die Kamera in der Szene um den Charakter drehen. Zum Testen werden ein Charakter und ein Walk-Cycle von Autodesk MotionBuilder benutzt.

4.3.1 Mixed-Reality-System instantreality

Das instantreality-Framework ist ein Szenengraph basiertes "Mixed-Reality"-System, welches zur Beschreibung und Erstellung von komplexen interaktiven 3D-Szene bzw. virtuellen 3D-Welten geeignet ist. Hierbei unterstützt es die Programmiersprachen "VRML" sowie "X3D". Es kombiniert verschiedene Bestandteile, um eine einzelne und gleichbleibende Schnittstelle für AR/VR-Entwickler zur Verfügung zu stellen (siehe Abbildung 4.1). Die Szenen werden entweder mit der Sprache "VRML97" (nachfolgend kurz als VRML bezeichnet) - VRML steht hierbei für "Virtual Reality Modeling Language" - oder mit X3D ("Extensible 3D"), dem Nachfolger des VRML-Formats, beschrieben. Der Funktionsumfang von VRML wurde in X3D deutlich erweitert, zum Beispiel um die Möglichkeiten der Definition von Shadern. Es ist möglich, VRML-Quelltext in X3D-Quelltext zu konvertieren.[11] [YJ07][YJ09]

[10] http://www.instantreality.org/wp-content/uploads/whatisit.jpg
[11] http://www.instantreality.org/

Primitive 3D-Objekte können direkt in X3D erzeugt werden, komplexe 3D-Inhalte, die aus einer 3D-Modellierungsanwendungen als X3D exportiert wurden, können eingebunden werden. Dabei werden Objekte, die in einer mathematischen Beschreibungsform, zum Beispiel als NURBS oder Bézier-Kurven vorliegen, in ein echtzeitfähiges Format (d.h. eine polygonale Darstellung) konvertiert. Es ist ebenfalls möglich, fertige Szenen mit integrierter Beleuchtung und Kamerainformationen in das X3D-Format zu exportieren. Animationen der 3D-Objekte können in den 3D-Modellierungsumgebungen definiert werden oder später in der X3D-Szene programmiert werden. [Vit08]

Nach dem Exportieren der 3D-Modelle, muss der Szenengraph des 3D-Modells mit einem Autorenwerkzeug angepasst werden, damit das Verhalten der 3D-Objekte manipuliert werden kann und die restliche 3D-Szene ausgearbeitet wird. [Vit08] Raimund Dachselt [Dac04] definiert in seiner Arbeit den Szenengraph wie folgt:

> "Ein Szenengraph ist ein gerichteter azyklischer Graph (Directed Acyclic Graph - DAG) zur effizienten hierarchischen Beschreibung einer dreidimensionalen Szene." [Dac04]

Dabei werden anhand von Knoten im Graph unter anderem Gruppierungen von 3D-Objekten oder der Spezifizierung von Geometrien, Materialien und Transformationen beschrieben. Hierfür stellt instantreality eine Vielzahl von Knoten zur Verfügung und erweitert die im X3D-Standard festgelegten Knoten um ein Vielfaches. Knoten dienen zur Beschreibung:

- des Szenengraphen (Geometrie, Transformationen usw.)
- des Verhaltensgraphen (Verhalten der Objekte sowie Reaktion bei Benutzereingabe)

Mit Hilfe des `Script`-Knoten in VRML bzw. X3D können Skriptsprachen, wie zum Beispiel JavaScript, integriert werden. Dadurch ist es möglich, 3D-Objekte zu animieren und ihnen Verhalten zuzuordnen, welches beispielweise ereignisbasiert gesteuert werden kann.
Es ist möglich, einzelne 3D-Modelle als Teilgraphen auszugliedern und diese in eigenständige Szenengraphen zu schreiben. Durch das Referenzieren der ausgegliederten Teilgraphen im Hauptszenengraphen ergibt sich der Vorteil, dass der Szenengraph übersichtlich bleibt. Eine Route (`ROUTE`) verknüpft die Knoten, die Ereignisse auslösen, mit den Knoten, die diese empfangen können.[12]

4.3.1.1 Umsetzung in instantreality

Als erstes wird eine X3D-Szene erzeugt, die den standardmäßigen "Header" (Kopfbereich) beinhaltet.

```
<?xml version='1.0' encoding='UTF-8'?> <!-- XML file declaration -->
<!DOCTYPE X3D PUBLIC 'ISO//Web3D//DTD X3D 3.0//EN' 'http://www.web3d.org/specifications/x3d
    -3.0.dtd'>
<X3D xmlns:xsd='http://www.w3.org/2001/XMLSchema-instance' profile='Full' version='3.0'
    xsd:noNamespaceSchemaLocation='http://www.web3d.org/specifications/x3d-3.0.xsd'>
  <Scene DEF='scene'> <!-- Scene-Tag, entspricht dem Wurzel-Knoten -->
```

[12] http://www.web3d.org/x3d/wiki/index.php/About_the_X3D/VRML_ROUTE_statement

Danach werden externe Prototypen, die für die Objektsteuerung zuständig sind, eingebunden. Prototypen sind selbst erstellte Knoten, die wie interne Knotentypen verwendet werden können. Die in dieser Arbeit verwendeten Prototypen werden von der Firma "NewMedia Yuppies" zur Verfügung gestellt. Mit dem `ProductHypersurfaceSensor` können Objekte skaliert oder gedreht werden. Der `NextWindowMultiTouch`-Prototyp dient zur Ansteuerung eines Multi-Touch-Rahmens.

```
<ExternProtoDeclare name="ProductHypersurfaceSensor" url="protos/ProductHypersurfaceSensor/
    ProductHypersurfaceSensor_PROTO.x3d#ProductHypersurfaceSensor" />

<ExternProtoDeclare name="NextWindowMultiTouch" url="protos/NextWindowMultiTouch/
    NextWindowMultiTouch_PROTO2.x3d#NextWindowMultiTouch" />
```

Nach dem Einbinden des Prototypen wird der Inhalt der Szene beschrieben. Hier wird eine 3D-Kugel mit einer Erdtextur, die einen Touch-Sensor enthält, der an den Prototypen `ProductHypersurfaceSensor` Bewegungen der X-Achse übergibt.

```
  <Background skyColor='0.000 0.000 0.000'/>
    <Transform DEF="globeTrans" translation="-1.40346 -9.97 3.65884">
      <ProductHypersurfaceSensor DEF="globeSensor" minAccel="0.07" maxAccel="4"
          initial_accel="0.1 0 0" rotSpeed="0.003"  minScale="1 1 1" maxScale="1 1 1"
          rotLimitUp="0.01" rotLimitDown="0.01" />
      <Transform>
        <TouchSensor DEF='Planet'/>
        <Transform DEF='pSphere1' translation='0.000 0.000 0.000' scale='10 10 10' >
          <Shape>
            <Appearance>
              <Material diffuseColor='0.800 0.800 0.800' specularColor='0.000 0.000 0.000'
                  transparency='0.000' shininess='0.050'/>
              <ImageTexture DEF='file156' url='Texture\earthmap1k.jpg' />
              <TextureTransform translation='-0.000 -0.000' rotation='0.000' scale='1.000
                  1.000' center='-0.500 -0.500' />
            </Appearance>
            <IndexedFaceSet solid='false' creaseAngle="3" coordIndex='0 1 40 -1 40 1 41
                -1 1 2 41 -1 41 2 42 -1 ... '>
              <Coordinate DEF='pSphere1-COORD' point='0.0775 -0.0123 0.9969, 0.0746 -0.0242
                  0.9969, 0.0699, ... '/>
              <TextureCoordinate point='1.000 0.975, 0.975 0.975, 0.950 0.975, 0.925 0.975,
                  0.900 0.975, 0.875 0.975, 0.850 0.975, 0.825 0.975, ... '/>
            </IndexedFaceSet>
          </Shape>
        </Transform>
      </Transform>
```

Mit Hilfe einer Route wird die Rotation aus der Berechnung des Knotens, der für die Verarbeitung der Touch-Ereignisse verantwortlich ist, mit der Rotation der Weltkugel verknüpft. Abschließend wird die Szene geschlossen.

```
    <ROUTE fromNode="globeSensor" fromField="xRotation_changed" toNode="globeTrans" toField
        ="set_rotation" /> <!-- Eine Route verknüpft Knoten -->
  </Transform>
 </Scene>
</X3D>
```

Um in instantreality einen Charakter einzubinden wird ein Exporter, der das 3D-Modell in ein VRML- bzw. X3D-Format schreibt, benötigt. Das Fraunhofer Institut für Graphische Datenverarbeitung (IGD) stellt hierfür einen Exporter für die Software Autodesk 3dsMax zur Verfügung. Dieser wandelt den Charakter bzw. andere 3D-Modelle in eine VRML- oder X3D-Datei um. (siehe Abbildung 4.2) Die Besonderheit dieses Exporters ist, dass bei Charakteren die Knochen sowie die Animationen mit exportiert werden können.
Damit in der späteren Anwendung mehrere Animationen eingebunden werden können, wird

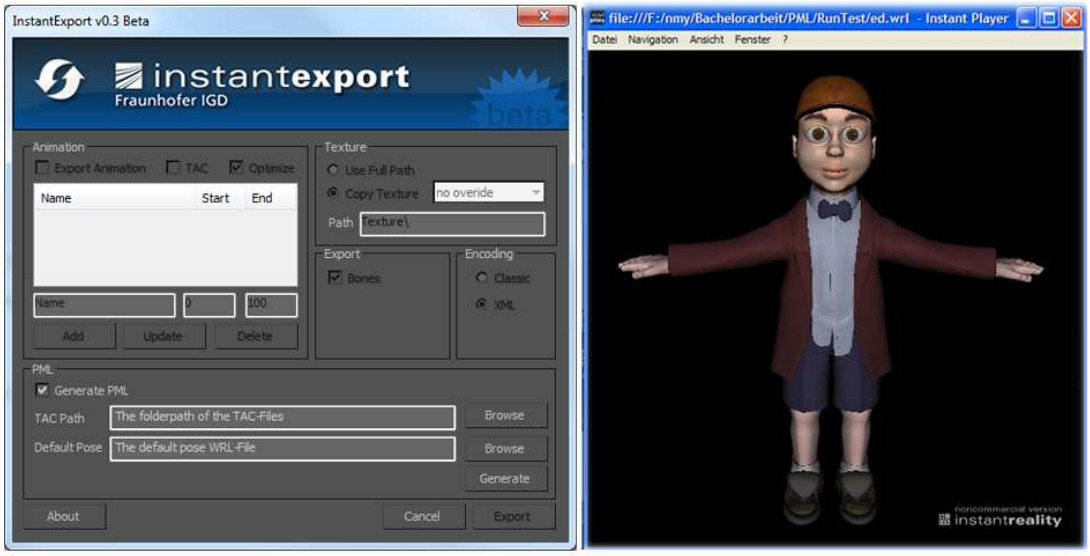

Abbildung 4.2: Links: Fraunhofer Exporter für Autodesk 3dsMax, um Objekte und Charaktere in eine VRML- oder X3D-Datei zu schreiben. Rechts: Der exportierte Test-Charakter in einer VRML-Datei, angezeigt mit dem "instantplayer".

die instantreality Erweiterung "Player Markup Language" (PML) benötigt. PML ist eine auf "Extensible Markup Language" (XML) basierende Skriptsprache, welche die Kontrolle interaktiver Szenen erleichtert. PML beschreibt die Handlungsszene eines virtuellen Charakters, der im Skript durch eine `id` referenziert wird. Das Skript beschreibt die Bewegungen des Charakters, welche gleichzeitig oder nacheinander ausgeführt werden können.

PML unterscheidet zwischen einem Definitions-Skript und mehreren Aktions-Skripten. Das Definitions-Skript beschreibt die Animationen, die für einen Charakter verfügbar sind. Die Aktions-Skripte definieren die zeitliche Abfolge der Animationen. Die Animationen werden in einer VRML-Datei beschrieben und mit `_TAC` gekennzeichnet.

Der 3dsMax-Exporter vom Fraunhofer IGD ist ebenfalls in der Lage, das Definitions-Skript, die Aktions-Skripte sowie die Animations-Skripte zu erstellen (siehe Abbildung 4.2). Beim Exportieren wird folgende Datenstruktur angelegt:

* `/PML`
    - `Hips_def.pml`
    - `Hips_act.pml`
    - `Hips_act1.pml`
    - `Hips_act2.pml`

* /TAC
    - ed_pause_TAC.wrl
    - ed_run_TAC.wrl
    - ed_run2_TAC.wrl
* main.x3d

In der Definitions-PML wird der Charakter referenziert und einer `id` zugewiesen. Danach werden die einzelnen Animationen zugewiesen. Pausenanimationen, die sogenannten Idle-Animationen, werden separat referenziert.

```
<?xml version='1.0' encoding='utf-8'?>
<definitions id='DEF_Hips'>

  <character id='Ed' src='ed.wrl' root='HUMANOID_Hips_TRANS'>
    <multiPoses id='ed_pause' src='TAC/ed_pause_TAC.wrl' dur='1667'/>
    <multiPoses id='ed_run2' src='TAC/ed_run2_TAC.wrl' dur='967'/>

    <idlePoses id="idP1" random="true">
      <multiPoses refId='ed_pause' dur='1667'/>
    </idlePoses>
  </character>

</definitions>
```

Die in den Aktions-Skripten aufgeführten Animationen, müssen ebenfalls im Definitions-Skript referenziert werden. Im Aktions-Skript wird der Charakter anhand seiner `id` angesteuert. Mit Hilfe des `animate`-Knotens, werden die einzelnen Animationen, hier ist es nur eine, definiert, die dann im "Sequenz"-Knoten (im Quelltext `seq`) ausgeführt werden. Die Reihenfolge im Sequenz-Knoten gibt die Abspielreihenfolge an.

```
<?xml version='1.0' encoding='utf-8'?>
<actions id='ACT_Hips' start='true'>

  <character refId='Ed'>
    <animate id='anim-ed_run2'>
      <multiPoses refId='ed_run2' />
    </animate>
  </character>

  <schedule>
    <seq>
      <action refId='anim-ed_run2' begin='0' dur='967' />
    </seq>
  </schedule>

</actions>
```

In der "main"-Datei einer Anwendung (Quelltext-Datei, die die Hauptanwendung repräsentiert) muss zuerst das Definitions-Skripte geladen werden.

```
<![CDATA[javascript:
  function initialize(){
    start = new MFString('PML/Hips_def.pml');
  }
]]>
```

In den Animations-Skripten, werden mit Hilfe von Positions- sowie Orientierungsinterpolatoren die einzelnen Knochen des Charakters, über eine feste Zeit, animiert.

```
#VRML V2.0 utf8
#MultiPose

DEF ed_pause TimedAnimationContainer {
  name "ed_pause"
  duration 1.667
  fieldnames [ "translation", "rotation", ...]
  targetnames [ "Hips", "Hips" ,...]
  interpolators [
    PositionInterpolator {
      key [ 0.000, 0.020, 0.040, ...]
      keyValue [ -1.421 2.139 3.414, ...]
    },
    OrientationInterpolator {
      key [ 0.000, 0.020, 0.040, ...]
      keyValue [ -0.617 -0.535 -0.578 ...]
    }
  ]
}
```

Um die Hauptanwendung übersichtlicher zu gestalten, wird die Weltkugel in eine externe X3D-Datei geschrieben und per `Inline`-Knoten in die Szene eingebunden. Damit sie weiter touch-fähig bleibt, wird ein `TouchSensor`-Knoten sowie der `Inline`-Knoten von einem `Transform`-Knoten umschlossen.

```
<Transform>
  <TouchSensor DEF='Planet'/>
  <Inline DEF="modelInline" url="erde.x3d" load="true" containerField="children" />
</Transform>
```

Den Austausch der Aktions-Skripte übernimmt ein JavaScript, welches die Rotationsänderung des `globeSensors` überprüft.

```
if(globeSensor.xDeltaRotation_changed != 0){ <!-- bei != 0 dreht sich der Globus -->
  if (!isrunning){ <!-- wenn Charakter nicht läuft, wird die PML gestartet -->

    if((Math.abs(globeSensor.xDeltaRotation_changed) > 1) && (Math.abs(globeSensor.
        xDeltaRotation_changed) < 1.5)){
      start = new MFString('PML/Hips_act.pml');
    }

  }
}else{
  if (!isrunning){
    start = new MFString('PML/Ed_pause.pml');
  }
}
```

Im nächsten Schritt wird die Drehung der Weltkugel mit der Laufanimation des Charakters verknüpft. Dazu wird der `TouchSensor` per `ROUTE` mit dem JavaScript verbunden.

```
  <ROUTE fromNode='Planet' fromField='touchTime' toNode='script' toField='touchedPlanet'/>
```

Die Umsetzung der Kamerasteuerung wird mit einer ähnlichen Route umgesetzt. Eine unsichtbare Kugel, die die Szene ausfüllt und ebenfalls einen `TouchSensor` sowie einen `ProductHypersurfaceSensor` besitzt, wird in der Szene angelegt. Anhand der Parameter `rotLimit` kann die Rotation eingeschränkt werden. Für die Rotation um die X-Achse ist es sinnvoll, eine Einschränkung festzulegen, damit die Kamera sich später nicht unterhalb der Erdkugel befinden kann. Bei der Kamerasteuerung können auch die Parameter `minScale` und `maxScale` für eine Zoom-Funktion genutzt werden.

Abbildung 4.3: Die beiden Ergebnisse der Testszene. Links: Die Testanwendung im "instantplayer". Rechts: Ein Screenshot aus der Unity3D-Szene. In beiden Anwendungen läuft ein Testcharakter auf einer Weltkugel, wenn der Benutzer diese per Touch-Geste dreht.

```
<Transform DEF="camTrans" center="-1.40346 2.63187 3.65884">
  <Viewpoint description='ViewPoint01' zNear='0.0001' zFar='8000' position='-1.5 3 20'/>
</Transform>
```

Damit das Verhalten der Kamera manipuliert werden kann, wird ein `Transform`-Knoten um den `Viewpoint`-Knoten gelegt, der dann mit zwei Routen die Rotation und die Skalierung des `ProductHypersurfaceSensor` übergeben bekommt.

```
<ROUTE fromNode="boxSensor" fromField="xRotation_changed" toNode="camTrans" toField="
    set_rotation" />
<ROUTE fromNode="boxSensor" fromField="scale_changed" toNode="camTrans" toField="set_scale" /
    >
```

Das Ergebnis der Testszene ist in der Abbildung 4.3 in der linken Bildhälfte zu sehen.

4.3.2 Game-Engine Unity3D

Unity3D ist eine Entwicklungsumgebung für 3D-Spiele. Sie stellt als Game-Engine zahlreiche Funktionen, wie zum Beispiel zur Simulation von Echtzeitschatten, Wind, Schwerkraft und Kollisionen, zur Verfügung. Zudem hat Unity3D eine hochoptimierte Grafik-Pipeline für DirectX und OpenGL (siehe Abbildung 4.4). Die Programmoberfläche stellt eine grafische Benutzeroberfläche (kurz GUI) zur Verfügung (siehe Abbildung 4.5), die die Konfiguration von einzelnen 3D-Objekten in einer Szene erheblich erleichtert. Mit Hilfe des Terrain-Editors stellt Unity3D dem Benutzer ein mächtiges Werkzeug zur Erstellung von Landschaften zur

[13] http://download.unity3d.com/unity/features/images/graphics/1-normal.jpg

Abbildung 4.4: Ein Screenshot aus einer Unity3D-Anwendung, der die Grafikqualität von Unity3D gut veranschaulicht.[13]

Verfügung. Durch die Skriptunterstützung können Abläufe, Verhalten und Eigenschaften der 3D-Objekte beeinflusst werden. Skripte können in den Programmiersprachen C#, JavaScript oder auch in Boo erstellt werden.
Unity3D stellt eine Vielzahl von Importmöglichkeiten zur Verfügung. Die importierten Objekte werden im Projektordner gespeichert. Wenn nachträglich Änderungen an den Daten vorgenommen werden, übernimmt Unity3D die Änderungen sofort. Neben der umfangreichen Unterstützung verschiedenster 3D-Formate der gängigsten 3D-Modellierungsanwendungen (siehe Tabelle 4.2)[14] bietet Unity3D auch den Einsatz einer Vielzahl von Bildformaten, wie zum Beispiel "JPEG", "PNG", "BMP" und viele mehr, an. Beim "TIFF"-Format und dem Format "PSD", der Software "Photoshop" von der Firma "Adobe", werden automatisch die einzelnen Ebenen der Datei mit importeiert. Weiter unterstützt Unity3D das Importieren einer Vielzahl von Audio- und Video-Formaten.

Mit der Game-Engine Unity3D ist es möglich, "Standalone"-Anwendungen für Mac OS X und Windows (ab der Version 2000) zu erstellen. Weiter ist es in Unity3D möglich, Spiele für den Unity3D Web-Player zu veröffentlichen. Hierfür wird lediglich ein Plug-In benötigt, was von Unity bereitgestellt wird. Zudem stellt Unity3D auch die Möglichkeit zur Veröffentlichung von Spielen für "Nintendo Wii", iPhone und das iPad von Apple zur Verfügung. Unity3D bietet für seine Nutzer eine umfangreiche Dokumentation sowie eine Skriptreferenz an und liefert neben zahlreichen Tutorials auch eine Vielzahl von Beispielprojekten sowie Programmerweiterungen.

[14] http://unity3d.com/unity/features/asset-importing

3D-Formate	Meshes	Texturen	Animationen	Knochen
Maya .mb & .ma	✓	✓	✓	✓
3D Studio Max .max	✓	✓	✓	✓
Cheetah 3D .jas	✓	✓	✓	✓
Cinema 4D .c4d	✓	✓	✓	✓
Blender .blend	✓	✓	✓	✓
Carrara	✓	✓	✓	✓
Lightwave	✓	✓	✓	✓
XSI 5.x	✓	✓	✓	✓
SketchUp Pro	✓	✓		
Wings 3D	✓	✓		
3D Studio .3ds	✓			
Wavefront .obj	✓			
Drawing Interchange	✓			
Files .dxf	✓			
Autodesk FBX .fbx	✓	✓	✓	✓

Tabelle 4.2: Übersicht über 3D-Formatunterstützung von Unity3D.

Die wichtigsten Fenster der GUI von Unity3D sind in Abbildung 4.5 abgebildet. Im "Szenen"-Fenster lassen sich die einzelnen 3D-Objekte einer Szene beliebig positionieren und in der Größe anpassen. Das "Game"-Fenster ist eine Spielvorschau, in der die Spielabläufe getestet werden können. Mit der "Abspiel"- sowie "Pause"-Taste, lässt sich der Spielmodus zu jeder Zeit anhalten oder wieder starten. Alle Objekte eines Projektes, die sogenannten "Assets", werden im "Projekt"-Fenster angezeigt. Neue Assets können einfach per "drag-and-drop" in das "Projekt"-Fenster gezogen werden, um sie zu importieren. Assets, die in der Szene verwendet werden, benötigen ein `GameObject`. `GameObjecte` in Unity3D sind Container, die verschiedenste Komponenten beinhalten können, wie Skripte, Renderer, Kamera, Licht, Materialien und viele mehr. Importierte 3D-Modelle beinhalten nach dem Import ein `GameObject` mit den zugehörigen Komponenten. Alle `GameObjecte` die in der Szene verwendet werden, befinden sich in der Projekthierarchie. Im "Inspector"-Fenster können die Eigenschaften von Objekten, welche sich in der Hierarchie oder im Projektfenster befinden, angepasst werde.[15]

4.3.2.1 Umsetzung in Unity3D

In Unity3D werden als erstes die gewünschten 3D-Objekte in das Projekt importiert. Dies kann durch einfaches Ziehen in das Projekt-Fenster geschehen. Danach werden die Objekte, wie Erdkugel und Testcharakter, in die Szene gezogen und dort platziert. Wie bei der Umsetzung in instantreality, wird auch hier eine Kugel in den Hintergrund gesetzt, mit der später die Kamerasteuerung realisiert werden soll. Damit in Unity3D Multi-Touch-Abfragen umge-

[15] http://unity3d.com/unity/features/

[16] http://unity3d.com/support/documentation/Images/manual/Learning the Interface-1.jpg

Abbildung 4.5: Übersicht über die grafische Oberfläche von Unity3D.[16]

setzt werden können, wird die "uniTUIO"-Bibliothek benötigt, welche von der "cd-cologne GmbH&Co.KG" zur Verfügung gestellt wird. Die in dieser Skriptbibliothek enthaltenen Skripte ermöglichen die Ansteuerung von Multi-Touch-Geräten mittels des TUIO-Protokolls. Um die Skripte zu benutzen, werden leere `GameObjecte` in der Szene angelegt und die Skripte darauf gezogen. Danach sind die Klassen der Skripte in der kompletten Szene bekannt und können von anderen Skripten referenziert werden.

Als nächstes wird ein neues C#-Script angelegt und mit der Weltkugel verknüpft, in dem es auf das `GameObject` der Weltkugel gezogen wird. Das Skript bekommt den Datentyp `TouchableObject` aus dem TUIO-Skript. Hierbei wird es von der Klasse des `TouchableObject`-Skriptes abgeleitet. Nachdem die benötigten Variablen deklariert sind, wird in der `Awake`-Funktion das `GameObject` des Charakters gesucht und einer Variable zugewiesen, damit aus dem Skript auf Komponenten des Charakters zugegriffen werden kann. Die `Awake`-Funktion wird als erstes nach dem Laden des Skriptes ausgeführt.

```
using UnityEngine;
using System.Collections;

public class earth_touch : TouchableObject {
  public bool allowRotate = true;
  public float speed = 1f;
  public Vector3 movement;
  public float min_speed = 0.05f;
  public float damp = 0.0f;
  private float v = 0.0f;
  private GameObject Ed = null;

  public void Awake(){
    Ed =  GameObject.Find("Ed");
  }
```

Als nächstes wird die `handleSingleTouch`-Funktion aus der abgeleiteten Klasse überschrieben und aus der Touch-Bewegung ein Bewegungsvektor berechnet. Dieser wird mit einer Geschwindigkeitsvariable verrechnet und so ergibt sich die Drehgeschwindigkeit der Kugel. In Unity3D werden Vektoren genutzt, um beispielsweise Bewegungen in X-, Y- und Z-Richtung zu beschreiben. Da die Weltkugel sich nur um eine Achse drehen soll, werden die nicht gewollten Vektorwerte auf Null gesetzt.

```
public override void handleSingleTouch(TouchEvent aTouch){
  if (!allowRotate) return;

  Vector3 objectPosition = this.renderingCamera.WorldToScreenPoint (aTouch.rayCastHitPosition
      );
  Vector3 positionNow = new Vector3(aTouch.screenPosition.y,aTouch.screenPosition.y,
      objectPosition.z);
  Vector3 positionThen = new Vector3(aTouch.lastScreenPosition.y,aTouch.lastScreenPosition.y,
      objectPosition.z);

  movement = this.renderingCamera.ScreenToWorldPoint(positionNow) - this.renderingCamera.
      ScreenToWorldPoint(positionThen);
  movement.y = 0.0f;
  movement.z = 0.0f;
  movement.x = movement.x*speed;
}
```

In der `Update`-Funktion, die für jeden Frame aufgerufen wird, wenn das Skript aktiv ist, wird die Geschwindigkeit gedämpft. Im nächsten Schritt wird überprüft, ob die Geschwindigkeit größer als der `min_speed` ist und die "Lauf"-Animation des Charakters wird überblendet.

```
public void Update (){
  damp = (Mathf.Abs(movement.x)*Time.deltaTime)*0.5f;

  if(movement.x>min_speed){
    movement.x=movement.x-damp;
  }else if(movement.x<-min_speed){
    movement.x=movement.x+damp;
  }else{
    movement.x=0;
  }

  transform.Rotate (movement.x, 0f, 0f);

  if(Mathf.Abs(movement.x)>0){
    Ed.animation["run"].speed = movement.x;
    Ed.animation.CrossFade("run");
  }else{
    Ed.animation["idle"].speed = 1;
    Ed.animation.CrossFade("idle");
  }
 }
```

Die Umsetzung der Kamerasteuerung ist äquivalent zur Steuerung der Weltkugel. Das Skript, welches auf die Kugel im Hintergrund gelegt wird, unterscheidet sich lediglich darin, dass neben der Drehung um die X-Achse auch noch die Drehung um die Y-Achse berechnet wird. Der Geschwindigkeitswert wird an das `GameObject` der Kamera übergeben. Auch hier wird, wie in der Anwendung in instantreality, eine Einschränkung der Kamerarotation um die X-Achse vorgenommen. Das Ergebnis der Testszene ist in der Abbildung 4.3 in der rechten Bildhälfte zu sehen.

4.4 Entscheidung und Zusammenfassung

Für die Umsetzung der Anwendung sind nach Festlegung der Mindestanforderungen, wie zum Beispiel die Anbindung von Multi-Touch-Technologien oder die Implementierung von animierten Charakteren sowie die Überblendung von Animationen des Charakters, zwei Entwicklungsumgebungen zur Auswahl geblieben. Um eine endgültige Entscheidung zu treffen, wurden in den beiden Umgebungen instantreality und Unity3D jeweils eine Testanwendung umgesetzt. Hier wurde eine Beispielinteraktion aus dem Konzept (Kapitel 3.3.4) erstellt. In den Testanwendungen läuft ein Charakter auf einer Weltkugel, wenn der Benutzer diese über Touch-Gesten dreht. Dabei wurden Probleme sowie Vor- und Nachteile der einzelnen Varianten aufgezeigt, die zur endgültigen Entscheidung für eine Entwicklungsumgebung verhalfen. Die endgültige Entscheidung für eine Entwicklungsumgebung wurde anhand folgender Entscheidungskriterien getroffen:

Bedienbarkeit

Die Bedienbarkeit ist bei den beiden Umgebungen sehr unterschiedlich. Da bei instantreality keine grafische Benutzeroberfläche zur Verfügung steht, müssen alle Objekte per Programmierung positioniert und angepasst werden. Diese Arbeitsweise ist jedoch zeitaufwendig. Im Gegensatz dazu erleichtert die GUI von Unity3D die Arbeit erheblich. Die Bedienbarkeit der Software erinnert an die Arbeitsweise einer 3D-Entwicklungsanwendung. So lässt sich das gewünschte Ergebnis, bezüglich der Anpassung sowie Positionierung der einzelnen `GameObjecte` betrifft, schnell erreichen.

Flexibilität

Die Unterstützung der zahlreichen 3D-Formate in Unity3D erhöht die Flexibilität enorm, da die 3D-Objekte nicht wie in instantreality erst in ein kompatibles Format wie VRML oder X3D konvertiert werden müssen. Animationen oder Knochen von Charakteren werden beim Einlesen in Unity3D automatisch angelegt, was die Arbeit erleichtert und die Arbeitszeit verkürzt, da kein externer Exporter benötigt wird. Die Animationen werden in Unity3D separat im Projektordner abgelegt und können in der Szene referenziert werden. Weiter ist es in Unity3D möglich, Animationen nachträglich aufzuteilen. Dies macht eine flexible Anpassung der importierten Animationen innerhalb der Entwicklungsumgebung möglich. Bei der Umsetzung in instantreality wurde festgestellt, dass die Geschwindigkeit von Charakteranimationen zur Laufzeit nicht angepasst werden kann, was bei der Laufanimation auf der Weltkugel zu unschönen Effekten führt, wenn die Kugelrotation gedämpft wird. Um ein akzeptables Ergebnis

zu erhalten, müsste eine Vielzahl von verschiedenen Animationen angelegt werden, die dann ein Skript austauscht und so an die Geschwindigkeit der Kugel angepasst. In Unity3D kann die Geschwindigkeit einer Animation über das Attribut speed beliebig anpasst werden.

Qualität des Ergebnisses

Beim Vergleich der grafischen Qualität der beiden Testanwendungen(siehe Abbildung 4.3) fallen erst keine Unterschiede auf. Höhere Ansprüche an die Grafik könnten im weiteren Verlauf des Projektes in beiden Entwicklungsumgebungen erzielt werden. Jedoch ist dies in instantreality mit mehr Zeit und Arbeitsaufwand verbunden. Ein kritischer Qualitätsverlust, der beim Umsetzen der Testanwendung in instantreality beobachtet wurde ist, dass die Animationen nur überlagert und nicht flüssig überblendet werden können. Dabei kommt es zu Sprüngen, wenn eine Animation beendet wird. Um dieses Problem zu beheben, müssten alle Animationen aus der gleichen Pose starten und enden, was aber einen erheblichen Zeitaufwand bei der Erstellung zur Folge hat.

Anhand der oben aufgeführten Kriterien wird die Game-Engine Unity3D als Entwicklungsumgebung gewählt. In den nächsten zwei Kapiteln werden die Erstellung sowie die Animation des Charakters beschrieben. Danach wird im Kapitel 7 auf die endgültige Umsetzung der Anwendung mit Hilfe von Unity3D eingegangen.

Kapitel 5

Das Charaktermodell

Im vorangegangenen Kapitel wurde eine Entwicklungsumgebung ausgesucht, in der die Anwendung umgesetzt werden soll. Erst wurde dabei eine Auswahl von multi-touch-fähigen Entwicklungsumgebungen anhand ihrer Funktionen verglichen und in den beiden am besten geeignetsten Umgebungen jeweils eine Testanwendung umgesetzt. In den Testanwendungen wurden Stärken und Schwächen der Entwicklungsumgebungen aufgezeigt. Anhand der daraus resultierenden Erkenntnisse wurde die am besten geeignete Entwicklungsumgebung ausgewählt.
In diesem Kapitel wird die Erstellung des 3D-Charakters erläutert. Neben der Modellierung wird auch auf die Texturierung des Charakters eingegangen. Als Vorbereitung für das nächste Kapitel entsteht das für die Animation wichtige "Character-Rig".

5.1 Einleitung

3D-Modellierung bezeichnet die Erstellung, Manipulation und Speicherung von geometrischen Objekten. Zum Erstellen eines 3D-Modells wird eine spezielle Software wie "Maya" oder "Blender" benötigt. Es gibt verschiedene Techniken ein 3D-Modell zu erstellen.

NURBS-Modellierung

NURBS (Non Uniform Rational B-Splines) sind am besten für komplexe Formen und organische Modellierung geeignet. Sie dienen der mathematischen Beschreibung von Oberflächen und Kurven. Dabei wird eine Kurve durch mindestens vier Punkte beschrieben. Im Gegensatz zu Polygonen sind NURBS von Anfang an rund. [MB05]

Polygon-Modellierung

"The most basic definition of a polygon is a shape defined by its corners (vertices) and the straight lines between them (edges)." [Gui05]

Demzufolge ist ein Polygon durch "Vertices" (Eckpunkte) und "Edges" (Kanten) beschrieben. Der Innenraum zwischen den Edges wird als "Face" (Fläche) bezeichnet.

Bei der Polygon-Modellierung werden Modelle in viele Dreiecke zerlegt. Das hat zur Folge, dass je nach Grad der Feinheit oder Rundung eine große Anzahl von Polygonen entstehen kann.

Der Charakter in dieser Arbeit wird als Polygonmodell erstellt. Der wesentliche Vorteil von Polygonmodellen ist, dass sie am flexibelsten und schnellsten für den Computer zu verarbeiten sind. Dies ist wichtig, da die Anwendung in Echtzeit laufen soll und gewährleistet werden muss, dass genügend Einzelbilder gerendert werden.[1] Zudem muss bei der Charaktererstellung auf eine möglichst geringe Anzahl von Polygonen geachtet werden.

5.2 Das Polygonmodell des Charakters

Zu Beginn muss der in Kapitel 3 beschriebene und skizzierte Charakter in ein 3D-Modell umgesetzt werden. Dazu wird die Software Autodesk Maya verwendet. Maya stellt eine Reihe von Werkzeugen für die Modellierung, Texturierung und Animation zur Verfügung. Die Texturen des Charakters werden in Adobe Photoshop erstellt und in Maya eingebunden.

Die im Konzept erstellten Skizzen werden beim Modellieren als Referenz genutzt. Dabei werden sie in die Modellierungssoftware Maya geladen und als Blaupausen verwendet.
In Maya werden Polygone genutzt, um Oberflächen zu füllen, die von den Edges begrenzt werden. Dabei kann ein Polygon aus drei bis "n" Edges bestehen. [MB05]

Das Aussehen des Charakters wird neben den Polygonen, auch vom Material der Oberfläche beeinflusst. In Maya stehen mehrere Materialtypen zur Verfügung, die dafür verantwortlich sind, wie Licht reflektiert wird. Phong wird beispielsweise für Glas und weitere glänzende bzw. reflektierende Flächen genutzt. Lambert ist eine Formel, um diffuse Materialen zu simulieren und ist für weniger spiegelnde Flächen geeignet. [Mah08] Daher wird für den Körper des Charakters das Material Lambert und für die Scheibe seines Monitorkopfes Phong verwendet.

Zur Erstellung der Grundform des Charakters kann ein Primitiv, wie beispielsweise ein Würfel, Zylinder oder eine Kugel eingesetzt werden. In diesem Fall wird der Würfel genutzt, um den Torso des Charakters zu bilden. Es wird lediglich ein grobes Modell erstellt und dann mit immer mehr Details aufgewertet. Dies wird erreicht, indem die Polygone beispielsweise mit der "Smooth"-Funktion aus Maya geteilt werden. Die Smooth-Funktion hat den Vorteil, dass der Körper zusätzlich noch abgerundet wird [Fle99]. In Abbildung 5.1 ist ein "ungesmoothtes" und ein "gesmoothtes" Modell zu sehen.

Zum weiteren Bearbeiten des Charakters gibt es in Maya die Möglichkeit, einzelne Polygone eines Modells zu bearbeiten. Dabei können Edges, Faces oder Vertices verschoben, skalieren oder rotieren werden. Körperteile wie Arme und Beine werden mittels der "Extrude"-Funktion erstellt. Hierbei werden ein oder mehrere Polygone markiert und aus dem Modell gezogen bzw. gedrückt. Auch die Weste am Körper wird mit dieser Funktion erstellt. [Sch06]

[1] http://www.worldlingo.com/ma/enwiki/de/Polygonal_modeling

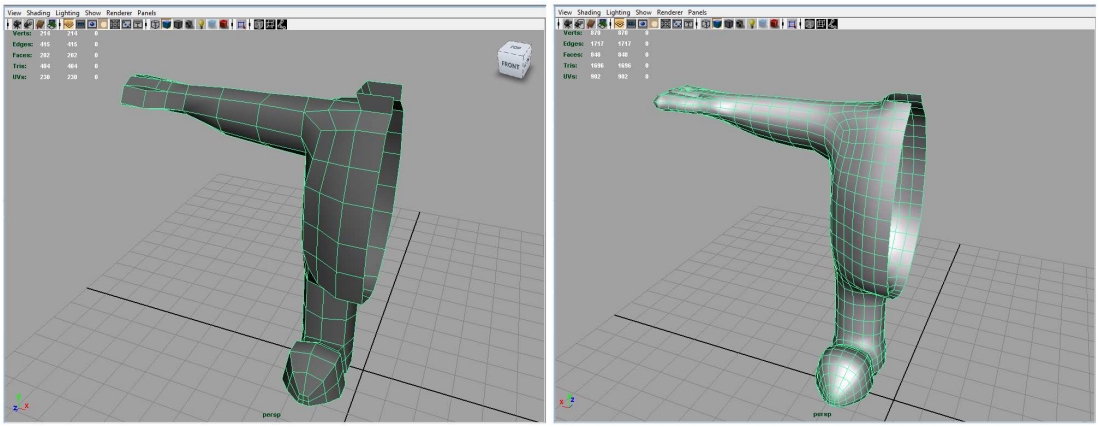

Abbildung 5.1: Links: Grobes Modell des Charakters. Rechts: Etwas detaillierteres Modell des Charakters nach dem "smoothen".

Der Einfachheit halber wird erst eine Hälfte des Charakters modelliert und dann auf die andere Seite gespiegelt. Anschließend werden die Seiten verbunden und ein synchrones Modell entsteht.
Der Kopf des Charakters wird getrennt vom Körper erstellt und erst später mit dem Körper verbunden. Hier wird ein einfacher Fernseher modelliert und etwas abgerundet. Anschließend werden an den entsprechenden Stellen Faces markiert und herausgezogen, um die Ohren zu erstellen. Die Augen des Charakters werden nachträglich als Video im Fernseher angezeigt. Einfache Primitive, wie Kugeln und Zylinder, stellen Antennen auf dem Kopf dar. Dabei ist es wichtig, dass die Antennenkugeln einzelnen Objekte sind, damit sie später in Satelliten transformiert werden können.
Handschuhe und Schuhe des Charakters werden aus den bereits modellierten Händen und Füssen erstellt. Dazu werden diese vom Körper getrennt und dann überarbeitet. Dies hat mehrere Vorteile. Zum einen ist die Größe der Handschuhe und Schuhe genau passend und zum anderen sind sie so leichter zu bearbeiten und können als einzelne Objekte auch auf eine andere Textur als der Körper gelegt werden.
Anschließend müssen noch weitere Gegenstände, wie zum Beispiel ein Jetpack und die Konsole mit dem Startknopf, modelliert werden. Abbildung 5.2 zeigt den Charakter mit und ohne Weste und Jetpack.
Eine Schwierigkeit bei der Charaktererstellung ist eine Balance zwischen einem nicht zu eckigen Charakter und einem Charakter mit zu vielen Polygonen zu finden. Dabei erzielt der Charakter mit einer Gesamtpolyzahl von ca. 7000 Polygonen ein zufriedenstellendes Ergebnis.

Damit das Charaktermodell an bestimmten Stellen rund wirkt, werden die Edges an diesen Stellen in "Soft Edges" umgewandelt. Dabei wird zwischen den angrenzenden Polygonen interpoliert, so dass keine "harten" Kante entsteht. [Sch06] [Mah08]

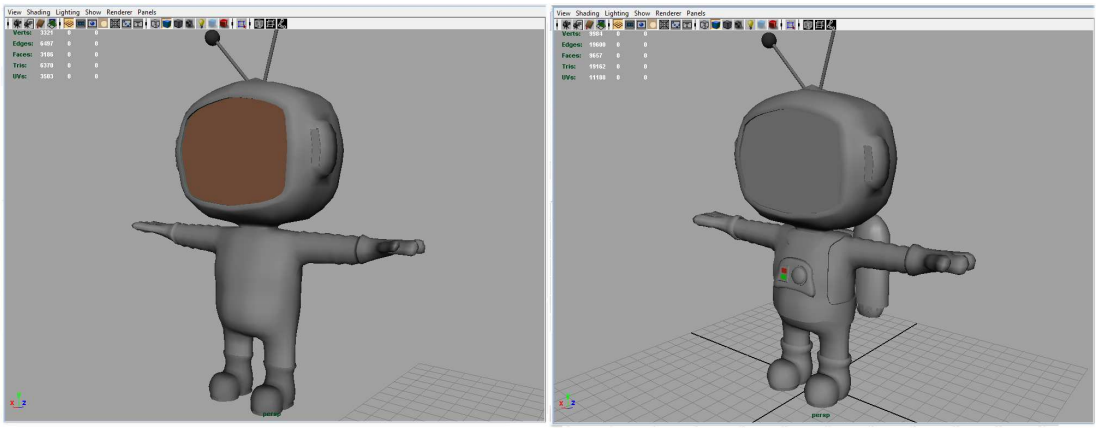

Abbildung 5.2: Links: Modell des Charakters ohne Jetpack und Weste. Rechts: Weiterentwickeltes Modell des Charakters mit Jetpack und Weste.

5.3 Die Charaktertextur

Nachdem der Charakter modelliert ist, muss er im nächsten Schritt texturiert werden. Texturen sind im Allgemeinen Bilddaten, die als Oberfläche für ein 3D-Modell dienen. Sie können Farben und Strukturen auf dem Modell darstellen ohne zusätzliche Geometrien zu gebrauchen. Meist sind Texturen Fotos, erstellte Strukturen oder bemalte Bilder. Da der Charakter nicht den Anspruch erfüllen soll, möglichst realistisch zu sein, sondern eine Comicfigur darstellt, werden nur wenige reale Bilder zur Texturierung verwendet.

Vor der Texturierung wird ein "UV-Layout" benötigt. Dieses enthält die Informationen, an welcher Stelle die Geometrie auf der Textur zu liegen hat und wie die Textur auf den Polygonen eines Objektes abgebildet wird. Um ein UV-Layout anzulegen, gibt es in Maya verschiedene Möglichkeiten. Neben dem "Automatic Mapping", welches automatisch ein Objekt aus verschiedenen Richtungen auf den "UV-Koordinaten" verteilt, gibt es zum Beispiel noch das planare und das zylindrische "Mapping"-Verfahren. Nachteil beim "Automatic-Mapping" ist, dass durch die Unterteilung sehr viel Kanten entstehen. [Ing08] [Oli07] Bei der Erstellung einer Textur sind Übergänge zwischen den Kanten meist leicht sichtbar und störend.

Um diese Probleme zu vermeiden werden Bereiche des Charakters markiert und dann von einer Seite planar "gemappt" (abgebildet). Vorteil hierbei ist, dass Kanten im UV-Layout an schwer einsehbaren Stellen platziert werden können. Beim Charakter werden die Kanten am Übergang von Körper und Weste gesetzt. Die Weste wird jedoch von vorne und hinten gemappt, um die vordere Textur unabhängig von der hinteren bearbeiten zu können. Durch das Anpassen der UV-Koordinaten im "UV-Layout Editor", wird der Übergang zwischen den Kanten der Weste überarbeitet, so dass der Übergang nicht mehr auffällt. Hier können einzelne Vertices, Edges und Polygone verschoben werden, um Verzerrungen, Überlappungen

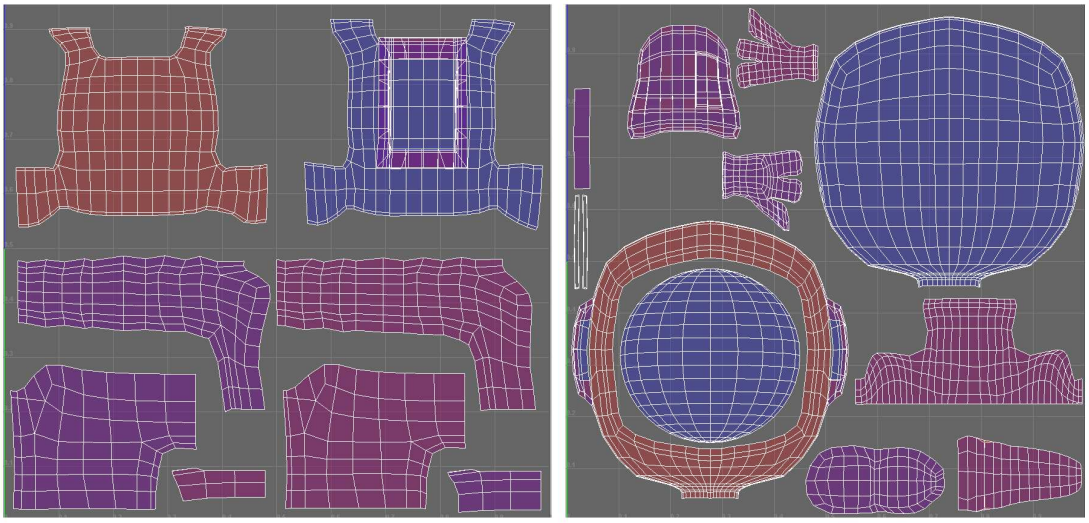

Abbildung 5.3: Links: UV-Layout vom Körper des Charakters. Rechts: UV-Layout von Kopf, Händen, Füßen und Konsole des Charakters.

oder Wiederholungen einer Textur zu vermeiden. Das manuelle Verändern der UVs kostet jedoch viel Zeit.[2]

Überlappungen können jedoch auch beabsichtigt sein, zum Beispiel um möglichst viel Platz der Textur auszunutzen. So können beispielsweise die UV-Koordinaten der Handschuhe gleich bzw. übereinander positioniert werden, da sich der linke nicht vom rechten unterscheidet. Das hat den Vorteil, dass der Bereich der Handschuhe größer auf der Textur und somit auch besser aufgelöst ist. Durch die Platzersparnis kann die Gesamtgröße der Textur dann möglichst klein gewählt werden, was für die Rechenleistung bei Echtzeitanwendungen wichtig ist. Kopf, Schuhe, Handschuhe und die Konsole mit dem Startknopf werden auf einer Textur zusammengefasst. Der Körper wird auf einer anderen Textur platziert, um eine gute Platzausnutzung der Texturen und somit auch eine gute Auflösung der Textur auf dem Charakter zu erreichen. In Abbildung 5.3 sind die beiden UV-Layouts des Charakters zu sehen.

Wie oben beschrieben, ist der Körper in einzelne Bereiche unterteilt. Die Weste besteht aus Vor- und Rückseite. Bei den Armen und Beinen wird nur zwischen links und rechts unterschieden. Somit liegen jeweils Vorder- und Rückseiten im UV-Layout übereinander, was auch für die Handschuhe gilt. Die Schuhe werden in drei verschiedene Bereiche unterteilt, um sie später besser texturieren zu können.
Zu beachten ist, dass die Glasscheibe des Charakterkopfes als einzelnes Objekt und mit einer eigenen Textur erstellt werden muss, um später in Unity3D Videos einbinden zu können.

[2] http://www.jawa9000.com/Technical/UVs/UVs.htm

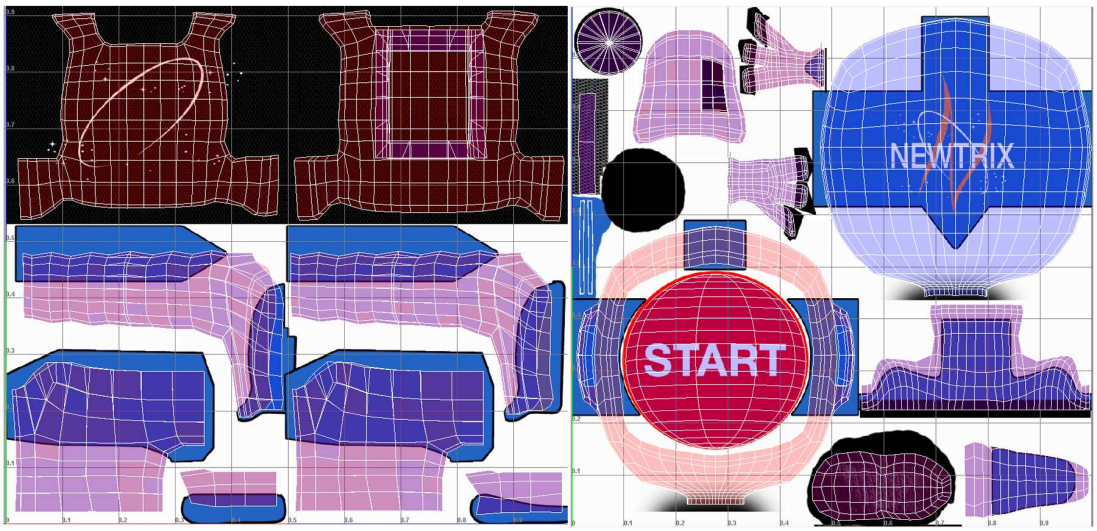

Abbildung 5.4: Links: UV-Layout des Körpers auf der fertigen Textur. Rechts: UV-Layout von Kopf, Händen, Füßen und Konsole auf der fertigen Textur.

Nachdem das UV-Layout angelegt ist, kann sich der Erstellung der Textur gewidmet werden. Wie erwähnt, wird hier auf das Bildbearbeitungsprogramm Photoshop zurückgegriffen. Um das UV-Layout in Photoshop zu bearbeiten, stellt Maya im "UV-Texture-Editor" die Funktion "UV-Snapshot" zur Verfügung.[3] Diese erstellt automatisch ein Bild des UV-Layouts, in einem beliebigen Format und einer beliebigen Größe. Für die weitere Bearbeitung ist es vorteilhaft, das Bild mit transparentem Hintergrund abzuspeichern. Dafür bietet sich das "PNG"-Bildformat an. Das entstandene Bild wird in Photoshop geöffnet und bearbeitet.

Photoshop ist für die Texturierung des Charakters gut geeignet, da es eine Vielzahl von Funktionen und Filtern für die Bildbearbeitung und Bildgestaltung zur Verfügung stellt. Der zuvor erstellte UV-Snapshot wird geöffnet und dient als Anhaltspunkt für die weitergehende Texturierung. Eine Ebene wird in Photoshop angelegt und mit Hilfe des transparenten UV-Snapshots an den verschiedenen Stellen bemalt. Zur Kontrolle wird die Textur abgespeichert und in Maya auf den Charakter gelegt. Danach werden die fehlerhaften Bereiche der Textur in Photoshop nachgebessert. Bei kleineren Ungereimtheiten in der Textur können die einzelnen UVs im UV-Layout in Maya angepasst werden. In Abbildung 5.4 ist das UV-Layout mit der Textur zu sehen. [Wal05]

[3] http://www.paulnaas.com/canada/mart420/handouts/UVmapping.htm

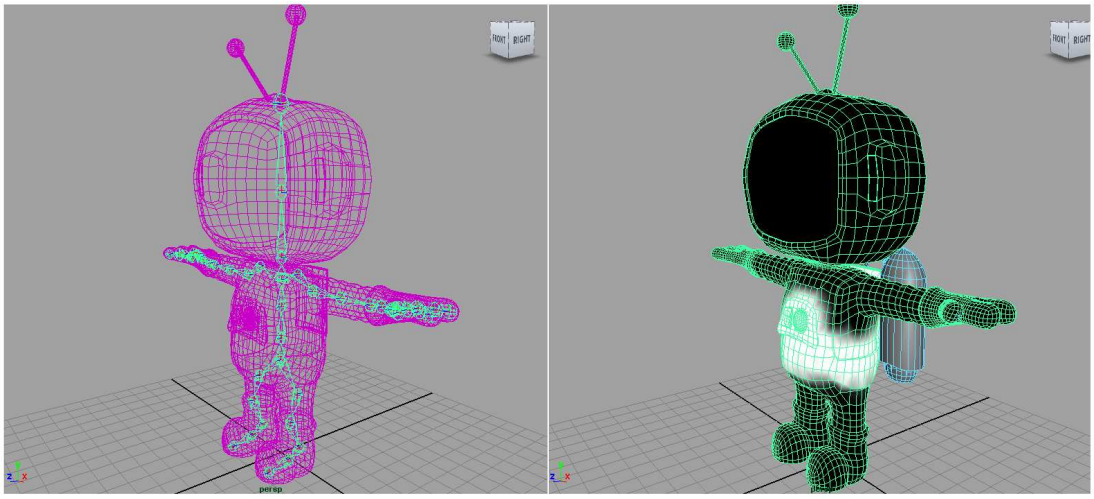

Abbildung 5.5: Links: Das Charakterskelett mit "geskinntem" Charakter. Rechts: Das "Paint Skin Weights Tool" am Charakter, welches den beeinflussten Bereich (weiß) eines Knochen zeigt.

5.4 Das Charakter-Rig

Der Begriff "Charakter-Rig" oder "Rigging" steht für den Aufbau der technischen Grundlagen sowie der individuellen Mechanismen, welche zur Animationskontrolle von 3D-Objekten dienen. Das Ziel ist hierbei, den Charakter auf die späteren Anforderungen der Animation optimal vorzubereiten. In diesem Fall wird ein Knochengerüst benötigt, welches anschließend mit den MoCap-Daten animiert werden kann. Es gibt verschiedene Vorgehensweisen, um ein Charakter zu "riggen". Einer der herkömmlichen und flexibelsten Ansätze ist das Erstellen des Charakterskelettes in Autodesk Maya. Dabei wird mit Hilfe von Knochen, den sogenannten "joints", eine Skelettkonstruktion aufgebaut. [Oli07]

Bei der Erstellung des Skelettes für den Charakter muss auf verschiedene Dinge geachtet werden. Als erstes wird der "root-Joint" mit dem "Joint Tool", in der Mitte des Beckens erstellt. Dieser ist wichtig, da dies der oberste Punkt in der Gelenk- und Knochenhierarchie ist. Zwischen zwei Gelenken entsteht zur besseren Übersicht ein Knochen. Ausgehend vom "root" werden die weiteren Gelenke bis zur Fußspitze gesetzt, ausgerichtet und anschließend gespiegelt. Danach werden die Wirbelsäule, der Hals und der Kopf mit Gelenken versehen. Zum Schluss werden an einem der Arme die Gelenke bis zu den Fingern hin platziert und auch wieder gespiegelt. Das fertige Skelett ist in Abbildung 5.5 auf der linken Bildhälfte zu sehen. Bevor das Skelett an den Körper gebunden wird, muss die Benennung der "joints" vorgenommen werden. Da die nachfolgenden Animationen in MotionBuilder, einer Animationssoftware der Firma Autodesk, auf den Charakter gelegt werden, müssen hier die von MotionBuilder erwarteten Namen vergeben werden.

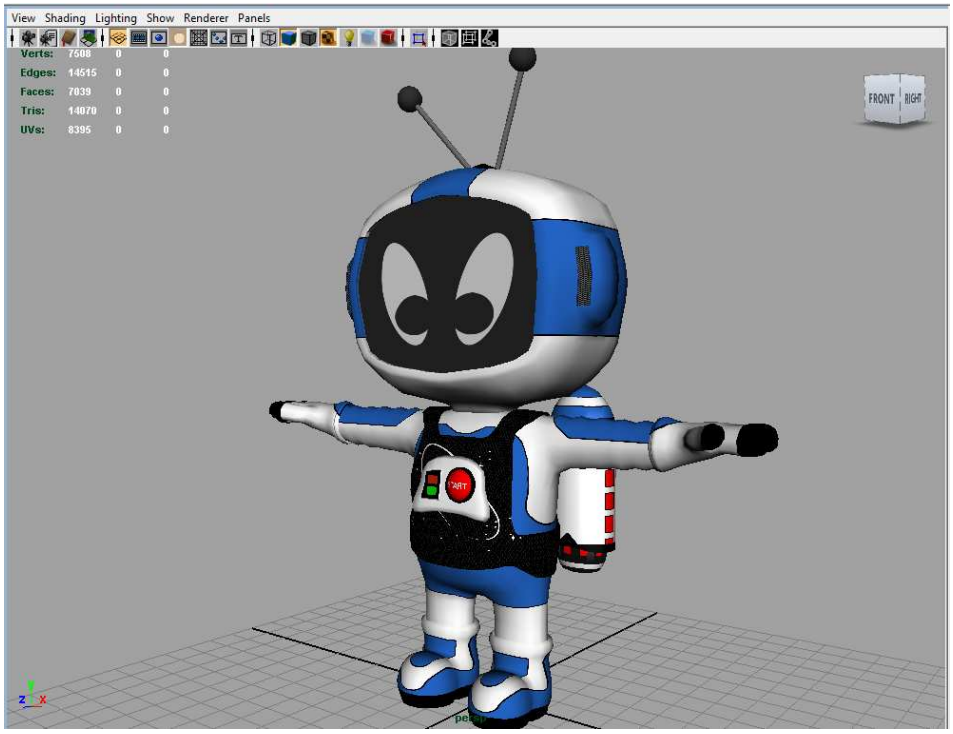

Abbildung 5.6: Fertiges Newtrix Modell mit Textur.

Ist das Skelett fertig, muss es an den Charakter gebunden werden. Dazu wird der Charakter und das Skelett ausgewählt und mit "Smooth Bind" verbunden. Smooth Bind wird benutzt um weiche Verformungen des Polygonmodells an Gelenken zu erzeugen. Zusätzlich wird anschließend mit dem "Paint Skin Weights Tool" die Stärke und der Einflussbereich von Gelenken überarbeitet. Auf der rechten Seite in Abbildung 5.5 ist der Charakter mit aktivem Paint Skin Weights Tool zu sehen. Der weiße Bereich auf dem Mesh des Charakters ist der von einem Gelenk beeinflusste Bereich. Dabei wird sichergestellt, dass beispielsweise ein Fingergelenk nicht die Geometrie des Nachbarfingers beeinflusst. Teilweise werden Bereiche wie Knie, Achsel, und Ellenbogen mit dem Paint Skin Weights Tool überarbeitet, um dort bessere Ergebnisse beim Animieren der Gelenke zu erhalten. [Sch06]

Zum Schluss wird der Jetpack an das Skelett gebunden. Würde der Jetpack, wie auch der Rest des Körpers, an das Skelett geskinnt werden, so könnte er durch Bewegungen des Charakters verformt werden. Da dies aber nicht erwünscht ist, wird der Jetpack an einen Joint am Rücken "geparentet". Dies hat zur Folge, dass er sich an alle Rotationen des Knochens anpasst, ohne sich zu verformen. Ist das Skelett mit dem Charakter verbunden, wird es durch Rotation der Gelenke animiert. Das einzige Gelenk das zusätzlich noch transliert wird, ist das "root"-Gelenk, um eine Bewegung im Raum darzustellen. [Gui05]

5.5 Zusammenfassung

In diesem Kapitel wurde die Erstellung des Charakters beschrieben. Dabei wurde auf die Erstellung des Polygonmodells sowie die Texturierung und das Rigging des Charakters näher eingegangen. Beim Modellieren des Charakters wurde auf eine möglichst geringe Polygonzahl geachtet, um einen "flüssigen" Ablauf der später umgesetzten Anwendung zu erreichen. Die Texturauflösung spielt bei Echtzeitanwendungen eine große Rolle. Sie sollte so klein wie möglich sein aber dennoch nicht "verpixelt" wirken. Mit einem guten UV-Layout ist die optimale Nutzung der Textur möglich. Weiter wurde ein Skelett zur Animation des Charakters erstellt und an das Modell gebunden. Im Anschluss wurde der Charakter "geweightet", um die Anpassung der Knochen an die Geometrie zu verbessern. In Abbildung 5.6 ist das fertige Modell zu sehen.

Im folgenden Kapitel 6 werden die Arbeitsabläufe für die Animation des Charakters beschrieben. Dabei werden MoCap-Daten verwendet, mit denen der Charakter natürliche Bewegungen erhält.

Kapitel 6

Animation mit Motion-Capture-Daten

Im Kapitel 5 wurde Newtrix, die Hauptfigur der Anwendung, erstellt. Neben der Geometrie seines Körpers wurde auch seine Textur gestaltet und eingebunden. Zudem entstand auch das für die Animation wichtige Skelett für den Charakter.

Dieses Kapitel beschreibt die Vorgehensweise der Animation des Charakters mit MoCap-Daten. Hierbei wird nicht weiter auf den Aufnahmeprozess dieser Daten eingegangen. Vielmehr wird die Verarbeitung und Nachbearbeitung der Daten nach einer MoCap-Aufnahme beschrieben. Die Animationsdaten für die Anwendung wurden von der Firma NewMedia Yuppies zur Verfügung gestellt.

6.1 Einleitung

Das zur Erstellung der MoCap-Daten zugrunde liegende System ist ein optisches MoCap-System der Firma Vicon. Bei optischen MoCap-Systemen werden reflektierende oder leuchtende Marker am Akteur befestigt und von Kameras aus verschiedenen Positionen aufgenommen. Bedingt durch das Aufnahmeverfahren können verschiedene Fehler in den Rohdaten auftreten. Aus diesem Grund müssen die Daten nach der Aufnahme überprüft und wenn nötig nachbearbeitet werden. Die Nachbearbeitung findet in der Software "Blade" der Firma Vicon statt. Nach der Datenbereinigung und Fehlerbeseitigung können die Daten exportiert und weiterverarbeitet werden.

6.2 Nachbearbeitung der MoCap-Daten

Zur Nachbearbeitung gehören neben der Erzeugung der 3D-Bewegungsdaten, aus den 2D-Bewegungsdaten der einzelnen Kameras, auch die Bereinigung der MoCap-Daten. Dazu werden die Rohdaten in die Software Blade geladen. Blade stellt eine Vielzahl von Werkzeugen für die Bearbeitung der Daten zur Verfügung. Häufige Fehler in den Datensätzen der MoCap-Systeme sind Laut Dietmar Jackèl [DJ06] folgende:

Verdeckungsprobleme
Hierbei werden die einzelnen Marker vom Akteur selbst oder von Gegenständen verdeckt. Ein Marker muss während der Aufnahme von mindestens drei Kameras zu sehen sein. Ist das nicht der Fall, so verschwindet der Marker in der Aufnahme.

Interferenzen und Störungen
Kommen sich zwei Marker zu nah, kann es dazu kommen das sie von den Kameras nicht mehr unterschieden werden können. Durch Störsignale (Lichtquellen) oder Reflektionen können falsche Marker von den Kameras erkannt oder überblendet werden.

Rauschen der Sensoren
Viele Sensoren verursachen Rauschen und Signalausreißer neben dem Nutzsignal. Dies kann von der Qualität und der Anzahl der Kameras abhängig sein.

Aussetzer
Marker können, durch Verschmutzung oder weil sie fehlerhaft sind, ausfallen und nicht mehr von den Kameras erkannt werden. Zudem können Marker während der Aufnahme abfallen.

Zu diesen Problemen kommt hinzu, dass Akteure bestimmte Animationen nicht durchführen können. Außerdem kann es vorkommen, dass die aufgenommenen Daten nicht den Vorgaben entsprechen oder die Bewegungen aus anderen Gründen korrigiert werden müssen. [DJ06] Diese Fehler müssen nun beseitigt werde. Wichtig für das "cleanen" der Daten ist, dass jeder Marker eine eindeutige Bezeichnung hat. Die Namen werden jedem Marker beim "labeln" vor der Aufnahme zugewiesen. Verschwindet ein Marker, beispielsweise weil er verdeckt wird, kann er in "Blade" wieder regeneriert werden. Anhand von anderen Markern wird dabei die Position des Fehlenden ermittelt. Mit der Funktion "fill gaps" werden dann die Lücken in der Animation gefüllt. Bei jeder Aufnahme ist ein Hintergrundrauschen (leichtes Zittern der Marker) vorhanden. Wird dieses an manchen Stellen der Aufnahme zu stark, kann es mit der "smooth"-Funktion abgeschwächt werden. Die durch Interferenzen entstandenen "falschen" Marker werden gelöscht. Gibt es in der Aufnahme keine Fehler mehr, wird sie als "C3D"-Datei abgespeichert. Die C3D-Datei beinhaltet die einzelnen Markerpunkte, die bei der MoCap-Aufnahmen "getrackt" (erfasst) wurden sowie deren Positionswerte im virtuellen Raum.

6.3 Animation in MotionBuilder

Die Animationsdateien die als C3D-Daten herausgeschrieben wurden, werden in "MotionBuilder", eine Animationsanwendung der Firma Autodesk, importiert. Damit die Animation später auf einen Charakter gelegt werden kann, wird grundsätzlich eine Zwischeninstanz, der sogenannten "Actor", benötigt. Dieser Actor nimmt die Bewegungen auf und überträgt diese dann auf einen Charakter (siehe Abbildung 6.1). Der Actor ist anatomisch geformt und hat menschliche Proportionen.

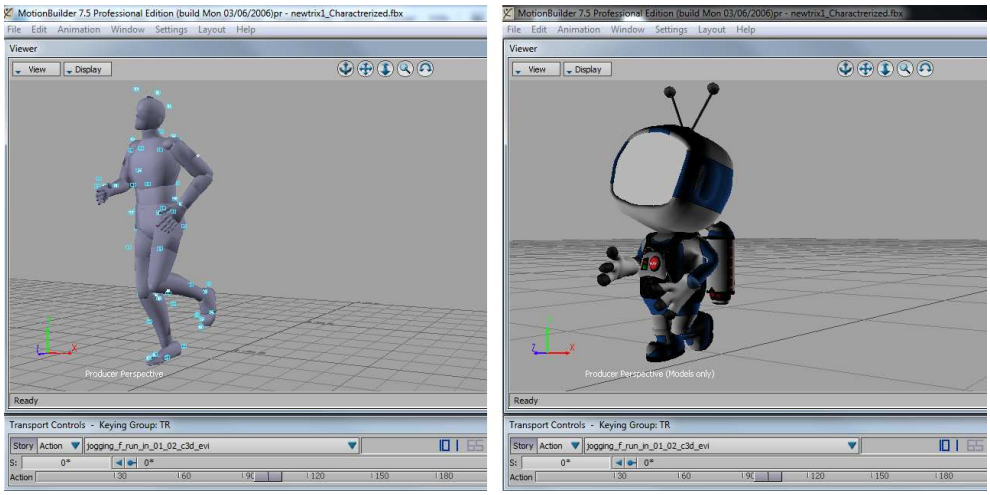

Abbildung 6.1: Animation eines Charakters in MotionBuilder. Über die Hilfsinstanz "Actor" werden die Animationswerte an den Charakter übertragen. Links: der Actor mit dem Markerset. Rechts: der animierte Charakter.

Ein Actor kann aus dem "Asset Browser" heraus in MotionBuilder erzeugt werden, indem er in den 3D-Arbeitsbereich, den sogenannten "3D-Workspace", gezogen wird. Danach müssen die Proportionen des Actors an die Markerpositionen der C3D-Datei angepasst werden. Dies ist später für eine saubere Übertragung der Bewegungen notwendig. Hierbei ist es wichtig, dass die Markerpunkte auf der Actor-Oberfläche positioniert werden.

Wenn alle Markerpunkte an der Actor-Oberfläche liegen, erfolgt das Anbinden dieser an den Actor. Hierfür wird in den Actor-Einstellungen ein "Markerset" angelegt. Das Markerset setzt vordefinierte Stellen, die sogenannten "Cells", auf den Actor. Diese dienen zur Orientierung für den Actor. Die entsprechenden Marker müssen den zugehörigen Cells zugewiesen werden. Einer Cell können bis zu fünf Marker zugeordnet werden. Um den Arbeitsumfang möglichst gering zu halten, sollte das zugewiesene Markerset gespeichert werden, da bei allen Animationsdaten einer Person, die von einer Aufnahmesession stammen, die Marker identisch sind. Abschließend werden die optischen Daten an den Actor gemappt, indem die Funktion "Snap" in den "Actor Settings" aufgerufen wird. Danach kann der Actor als Bewegungsquelle für einen Charakter genutzt werden.

Der Charakter muss mit seinem Skelett in den 3D-Workspace importiert und anschließend "Characterized" werden. Dazu wird die "Character"-Funktion aus dem "Asset Browser" auf die Hüfte der 3D-Figur gezogen. Wenn darauf geachtet wurde, dass die "Joints" des Skelettes der Namenskonventionen von MotionBuilder entsprechen, wird die Zuweisungen von Skelettelementen zu Character-Elementen automatisch vorgenommen. Ansonsten gibt MotionBuilder eine Fehlermeldung aus, in der die nicht zuordnungsfähigen Skelettelemente aufgelistet sind, die dann in den "Character Definitions" zugewiesen werden müssen.

In den "Character Settings" wird der "Input Type" auf "Actor Input" gestellt und mit dem Aktivieren der "Checkbox" "Active" wird die 3D-Figur mit der Bewegung des Actors verknüpft. Bevor der Charakter jetzt weiterverwendet werden kann, muss die Animation noch auf die Knochen des Skelettes übertragen werden, da bisher die Bewegung des Skelettes nur von dem Actor abgeleitet ist und keine wirklichen Bewegungsdaten für die Gelenke und Knochen des Skelettes vorhanden sind. Diese werden mit der Funktion "Plot Character" in den "Character Settings" an das Skelett gebunden. Als nächstes kann der Charakter mit der Animation in das FBX-Format gespeichert und in Unity3D importiert werden. [Kaj07]

6.4 Nachbearbeitung in MotionBuilder

In diesem Abschnitt werden Animationen, die teilweise angepasst oder verändert werden müssen, abschließend überarbeitet. Dazu wird der in Kapitel 6.3 beschrieben Arbeitsablauf durchlaufen. Der Unterschied besteht lediglich in dem abschließenden "Plotten" des Charakters. Hier wird die Animation zur besseren Weiterbearbeitung nicht an das Skelett, sondern auf das "Control-Rig" (Kontrollgerüst), gebunden. Zwar ist das Bearbeiten der Animation auch mit den Daten auf dem Skelett möglich, jedoch ist dies mit dem Control-Rig um einiges leichter und schneller zu bewerkstelligen. Durch die inverse Kinematik des Control-Rigs können Teile, wie beispielsweise Arme und Beine, des Charakters beliebig rotiert und transliert werden, ohne darauf achten zu müssen ob dadurch das Skelett verändert wird. Zusätzlich können "Auxiliary Effectors" (unterstützende Effektoren) angelegt werde, die hilfreich bei der Bearbeitung der Animation sind. Mit einem rechten Mausklick auf den Bereich des entsprechenden Körperteils in den Charakter Controls und dem Auswählen des "Create Aux Effector" wird ein solcher erstellt. Dieser erscheint nach dem Erstellen außerhalb des Charaktermodells und gibt die Position an, zu dem das Körperteil hingezogen bzw. ausgerichtet wird.
Ist die gewünschte "Pose" an einer Stelle der Animation erreicht, wird ein "Key" gesetzt, um diesen Zustand zu speichern. So wird eine Animation durchlaufen, bis die gewünschten Veränderungen vorgenommen sind. Zum Schluss wird die Animation auf das Skelett geplottet. [Kaj07]
Animationen welche nicht "gecaptured" werden können, wie beispielsweise die Fluganimationen mit dem Jetpack, müssen zusätzlich umgesetzt werden. Beim Erstellen der Fluganimation wird eine Animation verwendet, in der die Grundbewegungen, ausgenommen die Beine, zum Großteil übernommen werden kann. Die Animation der Beine wird dann komplett über Key-Frames umgesetzt. Durch die Möglichkeit Key-Frames zu kopieren, auszuschneiden und zu verschieben wird das Arbeiten erleichtert.

6.5 Zusammenfassung

In diesem Kapitel, wurden die Nachbearbeitung sowie die Verarbeitung von Motion-Capture-Daten beschrieben. Es wurde auf das sogenannte "Cleanen" in der Software Blade von Vicon eingegangen, wo mögliche Störungen, die bei einer Aufnahme entstehen können, behoben werden. Dazu gehört unter anderem das Wiederherstellen von verlorenen Markern. Wenn

ein Marker bei der Aufnahme verdeckt und nicht von mindestens drei Kameras erkannt wurde, entsteht in der Aufnahme an der Stelle des Markers eine Lücke. Diese muss dann in Vicon wieder gefüllt werden. Wenn alle Fehler behoben sind, kann in MotionBuilder eine 3D-Figur animiert werden. Dabei wurde auf den Zwischenschritt über den in MotionBuilder zur Verfügung gestellten Actor eingegangen. Dieser dient als Verbindung zwischen den Bewegungsdaten, die in der C3D-Datei enthalten sind und dem Charakter. Nachdem die Bewegungsdaten an den Actor gemappt wurden, können diese bei einer 3D-Figur als Animationsquelle ("Actor Input") genutzt werden. Die fertige Animation kann nach dem "Plotten" auf das Charakterskelett als FBX-Datei gespeichert und in Unity3D als neues Asset importiert werden. Wenn Animationen nach der Animation nicht das gewünschte Ergebnis erbringen oder nicht aufgenommen werden konnten, besteht in MotionBuilder die Möglichkeit eine Aufnahme zu erstellen oder zu verändern. Dabei ist eine der hilfreichsten Methoden, die vorhandene Animation nicht gleich auf das Skelett zu plotten, sondern erst auf ein Control-Rig. Das Control-Rig nutzt inverse Kinematik, so dass bei Bewegungen einzelner Effektoren, die entsprechenden Knochen mitbewegt werden.

Das nachfolgende Kapitel beschäftigt sich mit der Umsetzung der fertigen Spieleanwendung. Hierbei wird neben dem Grundgerüst der Anwendung sowie die Anbindung an den TUIO-Stream, auch mit zwei Skriptbeispielen, die Umsetzung der Interaktion in der Anwendung beschrieben. Alle anderen Interaktionsumsetzungen werden ebenfalls kurz beschrieben und erklärt, ohne weiter auf Programmcode einzugehen.

Kapitel 7

Erstellung der Anwendung

Nachdem die MoCap-Animationsdaten in Kapitel 6, mit der Software Blade, gesäubert wurden, wurde der Charakter in MotionBuilder animiert. Ein weiterer Teil des Kapitels beschäftigte sich mit der Erstellung weiterer Animationen in MotionBuilder, für die keine Daten vorhanden waren oder nicht gecaptured werden konnten.

In diesem Kapitel wird auf die Umsetzung der Anwendung eingegangen. Im ersten Abschnitt wird das Grundgerüst, welches zur Steuerung der gesamten Anwendung dient, erklärt. Anschließend werden Skripte für die Multi-Touch-Steuerung eingebunden. Weiter wird die Kontrolle und der Wechsel der Level erläutert. Danach wird anhand von zwei Beispielen auf die Implementierung der Interaktion eingegangen. Hier wird relevanter Quelltext erläutert und erklärt. Im darauf folgenden Abschnitt werden alle umgesetzten Interaktionen kurz in ihrer Funktionsweise beschrieben. Die Anbindung und Verwaltung von Video- und Audiodaten wird im darauf folgenden Abschnitt beschrieben. In diesem Kapitel wird jedoch nicht der gesamte Quelltext der Anwendung erklärt, sondern nur kurze und relevante Ausschnitte. Eine Übersicht aller geschriebenen und verwendeten Skripte sowie deren Beschreibung sind im Anhang A zu finden.

7.1 Einleitung

Anhand der Vorgaben aus Kapitel 3 wird das Weltraumszenario aufgebaut. Dazu werden die erstellten Planeten in Unity3D importiert und in der Szene positioniert. Der Charakter und andere Objekte, wie beispielsweise der Komet und der Satellit, werden in die Szene eingebunden. Weiter werden Lichter gesetzt, um die Weltraumstimmung zu verstärken.
Zur Steuerung der Anwendung werden Objekte in der Szene mit Skripten versehen. Dazu werden Skripte geschrieben bzw. bereits vorhandene Skripte benutzt. Durch das Zusammenspiel der einzelnen Skripte werden Aktionen gesteuert, die durch verschiedene Eingaben ausgelöst werden können.

7.2 Grundgerüst der Anwendung

Das Grundgerüst der Anwendung bildet das Skript `StateManager`. Der `StateManager` basiert auf dem Softwarekonzept der "state machine" (Zustandsautomat).[1] "States" sind Zustände, die in der Szene vorherrschen und Regeln definieren. Durch die Änderung eines States werden andere Regeln für den Ablauf der Anwendung gültig. Das ist besonders wichtig für die entstehende Anwendung, da viele Absicherungen nötig sind, um zu verhindern, dass Aktionen ausgeführt werden, die in diesem Zustand nicht ausgeführt werden dürfen. Um Inkonsistenz in der Anwendung zu vermeiden, ist es sinnvoll nur ein Skript zur Zustandsänderung zu nutzen. So regelt der `StateManager` zur Laufzeit alle Aktionen, welche die Zustände betreffen.

Ein weiteres wichtiges Skript für die Anwendung ist der `GameManager`. Hier sind alle `GameObjecte` und Komponenten, die für andere Skripte relevant sind, in Variablen referenziert. Da viele Skripte auf Variablen von Komponenten zugreifen, ist ein Einbinden der `GameObjecte` und deren Komponenten notwendig. Um dies zu umgehen, wird der `GameManager` angelegt. So muss immer nur der `GameManager` eingebunden werden, um alle benötigten Variablen in einem Skript nutzen zu können.

7.2.1 GameManager

Die `MonoBehaviour`-Klasse ist in Unity3D die Basisklasse aller C#-Skripte. Deshalb müssen alle C#-Skripte, damit auch der `GameManager`, explizit von dieser Klasse abgeleitet sein. Die Klasse `MonoBehaviour` beinhaltet wichtige Funktionen wie `Awake`, `Start` und `Update`. Die `Awake`-Funktion wird aufgerufen, wenn das Skript geladen wurde. Bei der `Start`-Funktion findet der Aufruf kurz vor dem ersten Durchlauf der `Update`-Funktion statt. Wenn das Skript aktiv ist, wird jeden Frame die `Update`-Funktion einmal ausgeführt.[2]

```
public class NewMonoBehaviour : MonoBehaviour {
  // Use this for initialization
  void Awake(){
  ...
  }

  void Start(){
  ...
  }

  void Update(){
  ...
  }
}
```

Damit die, im `GameManager` referenzierten `GameObject` und Komponenten in anderen Skripten bekannt sind, müssen sie bei der Deklaration mit dem Zugriffsmodifikator `public` angelegt werden. Variablen werden somit im "Inspector" sichtbar. Der Übersichtlichkeit halber können diese, mit dem Attribut `[HideInInspector]` im Inspector unsichtbar gemacht werden. Dadurch wird zudem verhindert, dass der Benutzer diese Variablen im Insepctor ändern

[1] http://www.mikrocontroller.net/articles/Statemachine
[2] http://unity3d.com/support/documentation/ScriptReference/MonoBehaviour.html

kann. Danach wird der Datentyp, hier `GameObject`, angegeben und ein Variablenname deklariert.

```
[HideInInspector] public GameObject mainCamera = null;
```

Da der `GameManager` von `MonoBehaviour` abgeleitet ist, steht ihm unter anderem die Funktion `Awake` zur Verfügung. In dieser Funktion wird das `GameObject MainCamera` der Variablen `mainCamera` zugewiesen.

```
void Awake () {
  mainCamera = GameObject.Find("MainCamera");
  ...
}
```

Ist ein `GameObject` bekannt - hier `mainCamera` - kann auf seine Komponenten zugegriffen werden. So wird das Skript `CameraFad` vom `GameObject MainCamera` in die Variable `cameraFade` gespeichert.

```
public class GameManager : MonoBehaviour {
  [HideInInspector] public GameObject mainCamera = null;
  [HideInInspector] public CameraFade cameraFade;

  void Awake () {
  mainCamera = GameObject.Find("MainCamera");
  cameraFade = mainCamera.GetComponent<CameraFade>();
  ...
  }
}
```

Durch die Zuweisung der Objekte und Komponenten an die Variablen, können die Komponenten und `GameObjecte` über die Variablen beeinflusst werden. Wird der `GameManager` in einem Skript eingebunden, stellt er dem einbindenden Skript alle `public`-Variablen zur Verfügung.

7.2.2 StateManager

Anhand des Zustandsdiagramms in Abbildung 3.3 aus Kapitel 3.3.5 kann ein Überblick der Zuständen in der Anwendung gewonnen werden. Zu beachten ist, dass nicht jeder Zustand des Diagramms auch einem Zustand im `StateManager` entspricht. Im `StateManager` werden einzelne Zustände, wie beispielsweise die Flugzustände, zu einem Zustand zusammengefasst.
Zuerst wird im `StateManager` der `GameManager` und der `LevelController` eingebunden. Der `LevelController` verwaltet, wie der Name schon sagt, die Spielwelten. Eine genauere Beschreibung des `LevelControllers` folgt in Kapitel 7.4.

Weiter wird eine Variable `state` als `string`, ein Zeichen oder eine Zeichenkette, deklariert. Diese enthält den aktuellen Zustand, in dem sich die Anwendung befindet. Zum Ändern des Zustandes wird aus einem beliebigen Skript die `StateChange`-Funktion des `StateManagers` aufgerufen und der neue Zustand in der Variablen `sname` übergeben. Zudem wird die `state`-Variable von anderen Skripten genutzt, um den aktuellen Zustand zu erfahren. Meist dient dies zur Absicherung von Aktionen. Eine `switch case`-Anweisung überprüft die Variable im `StateManager` und vergleicht sie mit den definierten Zuständen. Stimmt der Name mit einer Definition überein, wird die dazugehörige Funktion aufgerufen.

```
public void StateChange(string sname){
  state = sname;
  switch(state){
    case "initial":
      State_Initial();
    break;
    case "levelLoad":
      State_LevelLoad();
    break;
    ...
   break;
  }
}
```

Wird beispielsweise der Zustand `initial` übergeben, so setzt die Funktion `State_Initial` eine Reihe von Variablen von verschiedensten Skripten auf definierte Werte. Zudem werden hier noch Funktionen aufgerufen. Über den `GameManager`, welcher in der `game`-Variablen deklariert ist, wird auf Variablen, `GameObjecte` und Komponenten zugegriffen.

```
 private void State_Initial(){
  print("Statechange to ==> Initial");
  game.newtrix.animation.CrossFade("idle_main");
  game.infoFader.Show();
  game.kometController.kometLook = true;
  game.fingerController.allowFinger = true;
  game.planetController.allowRotate = false;
  game.planetController.allowWalk = true;
  game.camController.allowRotate = true;
  game.infoController.allowTouch = true;
  game.jetPackController.allowStartFly = true;
  game.satellitController.satellitInterference = true;
}
```

Für den fehlerfreien Ablauf der Anwendung werden folgende Zustände definiert:

- `initial` - Ausgangszustand bzw. Grundzustand, der immer wieder aufgerufen wird.
- `walk` - Zustand, der beim Drehen des Planeten bzw. "gehen" oder "laufen" des Charakters aktiv ist.
- `fall` - Bei zu schnellem Drehen des Planeten fällt der Charakter und aktiviert diesen Zustand.
- `fly` - Wird aufgerufen, sobald der Startknopf betätigt wird und der Charakter anfängt zu fliegen.
- `levelLoad` - Bei jedem Laden eines Levels wird dieser Zustand aufgerufen.
- `info` - Dieser Zustand ist solange aktiv, wie der Nutzer sich die Informationen zum Planeten anschaut.

7.3 Anbindung von Multi-Touch

Wie im Kapitel 4.3.2 schon erwähnt, wird von Unity eine Skriptbibliothek[3] zur Anbindung an das TUIO-Protokoll zur Verfügung gestellt. Um die TUIO-Anbindung nutzen zu können, muss das `SceneManager`-Skript in die Unity3D-Szene eingebunden werden. Die Aufgabe des Skriptes ist die Überprüfung, ob ein `TouchEventManager`-Skript in der Szene vorhanden ist. Weiter verhindert es, dass es mehrere `TouchEventManager` gibt. Wenn kein `TouchEventManager` vorhanden ist, legt der `SceneManager` einen an. Der `TouchEventManager` baut eine Verbindung zum "TUIO-Stream" auf, die bei einem Levelwechsel nicht abbrechen darf, da sonst Fehler bei der Touch-Erkennung auftreten können. Damit bei einem Wechsel der Szene oder dem erneuten Laden der Szene die Verbindung nicht abbricht, ist der `TouchEventManager` persistent. Ein persistentes `GameObject` wird bei einem Szenenwechsel nicht zerstört.

Damit ein Objekt in Unity3D auf einen Touch reagieren kann, muss dieses dem Layer `touchableObjects` angehören.

Layer in Unity3D

Mit Layern können in Unity3D Objektgruppen festgelegt werden, die von einer Kamera gerendert oder von einem Licht beeinflusst werden sollen. Aber sie können auch dazu benutzt werden, bei einem Raycast Objektgruppen zu bestimmen. Bei einem Raycast wird in Unity3D ein Strahl in die Szene geschossen und überprüft, ob er auf einen "Collider" stößt. Hierbei ist zu beachten, dass die zu testenden Objekte auch eine "Collider"-Komponente besitzen müssen.

Der `TouchEventManager` parst den TUIO-Input und erhält "Cursor"-Positionen. Diese sind Touch-Positionen auf dem Bildschirm. Der `TouchEventManager` ermittelt anhand der Funktion `Raycast` ob sich an diesen Positionen Objekte befinden, die auf Touches reagieren. Hierbei sucht er nur Objekte, die dem Layer `touchableObjects` angehören.
Wenn bei einem Raycast ein Objekt getroffen wird, schickt der `TouchEventManager` dem Objekt ein Touch-Event. Die Touch-Events sind von der Klasse `TouchEvent` abgeleitet und beinhalten Informationen des Touch-Ereignisses. Dazu gehören Variablen, wie `screenPosition` und `eventState`. Die `screenPostion` gibt die Position des Touches auf dem Bildschirm an. Die Variable `eventState` gibt folgende States vor:

- `Began` - steht für neu erkannten Touch.
- `Moved` - steht für die Bewegung eines erkannten Touches.
- `Ended` - steht für Touch wird nicht mehr erkannt.

Damit ein Objekt das Touch-Event verarbeiten kann, muss es ein Skript besitzen, welches von dem `TouchableObject`-Skript abgeleitet ist.

[3] http://xtuio.com/index.php/projectsmain/utuiomain

```
public class LevelChange : TouchableObject {
...
}
```

Durch die Ableitung einer Skriptklasse, beinhaltet die Klasse Variablen und Funktionen der übergeordneten Klasse. Daher besitzt das abgeleitete Skript auch die Funktion `queueEvent` der `TouchableObject`-Klasse. An diese Funktion sendet der `TouchEventManager` das Touch-Event. Die Funktion fügt beim Event-State `Began` das Event in seine interne Eventliste `touchEvents` hinzu und entfernt das Event bei dem State `Ended`.

```
virtual public void queueEvent(TouchEvent theEvent){
  if (theEvent.eventState == TouchEventState.Began) {
    touchEvents.Add(theEvent);
  }
  if (theEvent.eventState == TouchEventState.Ended) {
    touchEvents.Remove(theEvent);
    if (touchEvents.Count == 0) this.noTouches();
  }
}
```

Anhand der Länge der internen Eventliste wird in der `finishFrame`-Funktion ermittelt, wie viele Touch-Events das Objekt betreffen. Diese Funktion wird nach jedem Raycast, in der `Update`-Funktion des `TouchEventManager` aufgerufen. Hierbei werden anhand der gezählten Events entsprechende Funktionen aufgerufen. Diese erhalten im Übergabewert die interne Eventliste.

```
virtual public void finishFrame(){
  if (touchEvents.Count >= 3) {this.handleTripleTouch(touchEvents);}
  if (touchEvents.Count == 2) {this.handleDoubleTouch(touchEvents);}
  if (touchEvents.Count == 1) {this.handleSingleTouch((TouchEvent)touchEvents[0]);}
  if (touchEvents.Count == 0) this.noTouches();
}
```

Die Funktionen `handleTripleTouch`, `handleDoubleTouch`, `handleSingleTouch` sowie `noTouches` werden in der `TouchableObject` Klasse beim Anlegen mit dem Schlüsselwort `virtual` gekennzeichnet. Dadurch können abgeleitete Klassen diese überschreiben.

```
virtual public void handleDoubleTouch(ArrayList events) {}
```

Je nach Bedürfnis des einzelnen Objektes, können die oben genannten Funktionen in der abgeleiteten Klasse überschrieben werden. Hierbei muss das Schlüsselwort `override` angegeben werden. Der nachfolgende Codeauszug zeigt einen Teil eines einfachen Buttonskriptes, in dem die Funktionen `handleSingleTouch` sowie `noTouches` überschrieben werden. Mit Hilfe der `if`-Anweisung und der Variable `touchDown` wird verhindert, dass der Inhalt der Funktion öfter ausgeführt wird, wenn der Touch noch nicht beendet wurde.

```
public override void handleSingleTouch(TouchEvent aTouch){
  if (touchDown) return;
  ...
  touchDown = true;
}

public override void noTouches(){
  touchDown = false;
}
```

Das `TouchEventManager`-Skript stellt ebenfalls die Möglichkeit zur Verfügung, mit der Maus sogenannte “fake events” zu erzeugen. Hier werden Touch-Events simuliert. Ein einfacher Mausklick ist hierbei ein “Single-Touch”. Durch gleichzeitiges Drücken der Maus sowie

der Windows- oder der Alt-Taste wird ein "Double-Touch" simuliert. Um ein "Triple-Touch" zu erhalten, müssen die beiden Tasten gleichzeitig zu dem Mausklick gedrückt werden. Um eine Rückmeldung über durchgeführte Touches zu erhalten, gibt es das `CrosshairController`-Skript. Dieses erstellt an den Bildschirmkoordinaten, an denen ein Touch ermittelt wird, Testobjekte, die als orangene Punkte dargestellt werden.

7.4 Levelsteuerung

Bei der Levelsteuerung spielen mehrere Skripte eine Rolle. Wie in Abbildung 3.2 im Kapitel 3.3.3 gut zu erkennen ist, muss das Menü zum Planetenwechsel - im folgenden Levelwechsel genannt - am unteren rechten Bildrand platziert werden. Hierfür stellt Unity3D die Möglichkeit zur Verfügung, GUI-Elementen in der Szene einzubinden. Durch GUI-Elemente können Texte, Texturen aber auch Buttons und Textfelder im Vordergrund der Kamera positioniert werden.
Für die Umsetzung des Menüs können diese Elemente aber nicht zur Hilfe gezogen werden, da auf ihnen kein Touch erkannt werden kann. Damit das Menü trotzdem umgesetzt werden kann, wird mit Hilfe des `CamPlaneAdjust`-Skripts eine Ebene im Sichtfeld der Kamera positioniert. Dazu wird das Skript auf ein `GameObject` namens `CameraPlane` in das Kameraobjekt gelegt. Die `GameObjecte` des Menüs müssen als Unterobjekte des `CameraPlane` Objektes angelegt werden, damit sie auf dieser "Plane" positioniert werden. Mit Hilfe des `GUIScreenPositioning`-Skriptes werden die Menüobjekte pixelunabhängig auf der Kameraebene mit normalisierten Koordinaten positioniert. Normalisierte Koordinaten sind hier, die umgerechneten Bildschirmkoordinaten auf Werte zwischen -0,5 und 0,5. Normalisierte Koordinaten werden benötigt, um eine auflösungsunabhängige Positionierung zu ermöglichen. So ergeben sich folgende Positionierungswerte:

- (0,0) = mittig
- (0.5,0.5) = oben/rechts
- (-0.5,-0.5) = unten/links

Die Menüobjekte, die für den jeweiligen Planetenwechsel, also für das Laden eines neuen Levels, zuständig sind, besitzen das vom `TouchableObject` abgeleitete Skript `LevelChange`. Nach der Auswahl eines Planeten im Menü führt die Funktion `handleSingleTouch` die `LevelChange`-Funktion im `LevelController` aus. Hierbei wird der eindeutige Name des zu ladenden Levels als `string` in der Variablen `levelName` übergeben. Der Levelname wird bei der Erstellung eines Levels festgelegt.

```
game.levelController.LevelChange(levelName);
```

Der `LevelController` ist für das Starten sowie das Wechseln der Level verantwortlich. Ein Level wird in Unity3D mit folgender Funktion geladen.

```
Application.LoadLevel(levelName);
```

Bevor die LevelChange-Funktion ein neues Level lädt, wird in die Variable lastLevel der Name des aktuellen Levels geschrieben. Weiter werden das Bild der Kamera sowie die Hintergrundmusik ausgeblendet. Da dies über einen bestimmten Zeitraum geschehen soll, wird eine Coroutine benötigt.

Coroutine

Coroutinen sind Funktionen, in denen Abläufe zeitlich gesteuert werden. Sie werden mit StartCoroutine aufgerufen. Die WaitForSeconds-Funktion ermöglicht das Warten in einer Coroutine für eine bestimmte Zeit. Die Funktion, aus der die Coroutine aufgerufen wird, läuft währenddessen parallel zur Coroutine weiter.

So wird in der LevelChange-Funktion, nachdem das Bild der Kamera sowie die Hintergrundmusik ausgeblendet sind, das gewünschte Level geladen.

```
public void LevelChange(string levelName){
  level.lastLevel = level.currentLevel;
  StartCoroutine(LevelChangeCo(levelName));
  Application.LoadLevel(levelName);
}

IEnumerator LevelChangeCo(string levelName){
  game.cameraFade.FadeOut(2);
  game.mainCamera.GetComponent<AudioController>().FadeOut();
  yield return new WaitForSeconds(3);
}
```

Wenn ein Level gestartet wird, wird im State State_LevelLoad die Coroutine LevelStart im LevelController gestartet. Diese blendet das Kamerabild ein und steuert die im Kapitel 3 erwähnte Scan-Animation.

Wie im Kapitel 3.3.4 schon erklärt, kann Newtrix bei zu starkem Drehen des Planeten von ihm herunterfallen, was ein Neuladen des Levels zur Folge hat. Hierbei ist es aber nicht erwünscht, dass die Scannanimation erneut abgespielt wird. Weiter soll, wie im Kapitel 3.3.5 beschrieben, das aktuelle Level im Auswahlmenü nicht wieder ausgewählt werden können. Aus diesem Grund ist ein LevelManager-Skript nötig. Dieses hat die Aufgabe zu überprüfen, welches Level gerade geladen ist, welches zuletzt geladen war sowie die Verwaltung einer Liste aller Level die schon geladen wurden.

Damit nach einem Levelwechsel die Variablen noch bekannt sind, muss der LevelManager wie auch schon der TouchEventManager persistent sein. Auch hier muss wieder darauf geachtet werden, dass nach einem Levelwechsel nur ein LevelManager-Script in der Szene vorhanden ist. So kann im StateManager beim State State_LevelLoad überprüft werden, ob das eben geladenen Level neu gestartet wurde oder nicht. Neu besuchte Level werden in die Levelliste im LevelManager eingetragen.

```
private void State_LevelLoad(){
  print("Statechange to == LevelLoad");
  level.SetCurrentLevel(Application.loadedLevelName);

  if(level.GetLastLevel() != level.GetCurrentLevel()){
    level.AddToVisitLevel(level.GetCurrentLevel());
  }
  ...
}
```

Weiter können, anhand der Liste der bereits geladenen Level, die Menüobjekte in Erfahrung bringen, welche Level nicht mehr geladen werden dürfen. Hierbei wird ein Touch verboten und das Menüobjekt farblich markiert.

```
foreach (string l in level.visitedLevel){
  if(l == levelName){
    allowTouch = false;
    this.renderer.material.SetColor ("_Emission", new Color(0.7f,0.1f,0.1f));
  }
}
```

7.5 Interaktionsimplementierung

Neben den oben erwähnten Skripten, enthält die Anwendung noch eine Vielzahl weiterer Skripte. Diese übernehmen verschiedene Aufgaben in der Anwendung. So wird die Problemstellung der Touch-Interaktion mit dem Charakter über die Skripte gelöst. Durch die folgenden Interaktionen werden neue Konzepte implementiert, die bisher in anderen interaktiven Anwendungen nicht vorhanden sind. Da sich Programmabschnitte teilweise wiederholen oder ähnlich aufgebaut sind, wird anhand von zwei Beispielen die Umsetzung der Interaktionen in der Anwendung erläutert. Dabei ist zu beachten, dass durch die Verfahren der Touch-Erkennung in der Entwicklungsumgebung nicht alle Interaktionen umgesetzt werden können. Eines dieser Skripte ist der `CamController`, der für die Navigation im Raum verantwortlich ist. Die Flugsteuerung wird über die Skripte `JetPackController`, `JetPackSpeedChanger` und `JetPackSpeedController\verb` geregelt.

7.5.1 CamController

Zum Erstellen der Weltraumszene wird eine Kugel im 3D-Raum positioniert. Die Kugel wird so groß gewählt, dass darin alle Objekte der Szene Platz finden. Auf die Kugel wird eine Textur gelegt, um den Weltraum darzustellen. Diese Kugel wird ebenfalls zum Navigieren in der Anwendung genutzt. Damit Touches auf der Innenseite erkannt werden können, wird ein "Collider" auf die Kugel gelegt. Da ein "Collider" Kollisionen nur auf der Oberfläche eines Objektes erkennen kann, muss die Kugel invers sein. Bei einer inversen Kugel ist die Oberfläche nach innen gedreht. So kann durch erkannte Touches in der Kugel die Kamera gesteuert werden. Das `CamController`-Skript befindet sich ebenfalls auf dieser Kugel. Die erkannten Touches werden, wie im Kapitel 7.3 beschrieben, an das Skript übermittelt. Der `CamController` wertet diese aus und wandelt sie in Positionsangaben für die Kamera um. Wie bereits im Kapitel 3 beschrieben, bewegt sich die Kamera dabei aber immer nur um den Charakter.

7.5.1.1 Zoomen in der Szene

Im `CamController` wird zuerst der `GameManager` eingebunden. Danach werden skriptinterne Variablen, wie die Distanzvariablen `distance_new` und `distance_old`, angelegt. Werden zwei Touches auf der Kugel erkannt, so werden die Variablen `distance_new` und `distance_old` errechnet. Diese werden dazu genutzt, den Zoom zu realisieren. Dabei wird

die Distanz zwischen den zwei Touch-Punkten errechnet. In `distance_new` steht die aktuelle Distanz und in `distance_old` die vorherige Distanz der Punkte.

```
public override void handleDoubleTouch(ArrayList events){
  ...
  distance_new = Mathf.Sqrt(Mathf.Pow(touch1.screenPosition.x - touch0.screenPosition.x,2)+
      Mathf.Pow(touch1.screenPosition.y - touch0.screenPosition.y,2));
  distance_old = Mathf.Sqrt(Mathf.Pow(touch1.lastScreenPosition.x - touch0.lastScreenPosition
      .x,2)+Mathf.Pow(touch1.lastScreenPosition.y - touch0.lastScreenPosition.y,2));
}
```

Beide Variablen werden benötigt, um zu erkennen, ob die Distanz größer oder kleiner geworden ist. In die Variable `distance_var` wird die resultierende Distanz geschrieben. Nimmt die Distanz zu, wird reingezoomt, nimmt sie ab, wird rausgezoomt. Damit das fortwährend passiert, wird dies in der `Update`-Funktion geprüft. Die `maxZoom`- und `minZoom`-Variablen sind fest definierte Grenzwerte, die festlegen, wie weit rein- bzw. rausgezoomt werden kann. Die Distanz wird in den Z-Wert (dritte Stelle des Vektors) des Vektors `position` geschrieben. Da nur auf der Z-Achse der Kamera gezoomt werden soll, werden die X- und Y-Werte des Vektors genullt. Zusätzlich wird die Position des `target` auf den Vektor addiert. Im `target` ist der Punkt definiert, um den die Kamera rotieren wird. Zum Schluss überschreibt der Positionsvektor `position` die Position der Kamera.

```
void Update(){
  ...
  distance_var = Mathf.Abs(distance_new - distance_old)/200;
  if(distance_old < distance_new && distance + distance_var < maxZoom ){
    distance += distance_var;
  }else if(distance_old > distance_new && distance + distance_var > minZoom){
    distance -= distance_var;
  ...
  Vector3 position = rotation *new Vector3(0.0f, 0.0f, distance) + target.position;
  ...
  renderingCamera.transform.position = position;
}
}
```

7.5.1.2 Drehen in der Szene

Das Drehen um den Charakter wird mit einzelnen Touches gesteuert. Dies wird umgesetzt indem die Variablen `movementx`, `movementy`, `speedx`, `speedy`, `x`, `y` sowie ein `target` angelegt werden. Erkennt die Kugel einen einzelnen Touch, werden die Variablen `movementx` und `movementy` errechnet. Sie enthalten die Entfernungen, die ein Touch vom Ausgangspunkt bis zum aktuellen Punkt zurückgelegt hat.

```
public override void handleSingleTouch(BBTouchEvent aTouch)
{
  ...
  movementx = (aTouch.lastScreenPosition.x - aTouch.screenPosition.x) / 200;
  movementy = (aTouch.lastScreenPosition.y - aTouch.screenPosition.y) / 200;
}
```

In der `Update`-Funktion erfolgt eine Überprüfung der Werte. Die Variablen `speedx` und `speedy` geben die Geschwindigkeit in X- und Y-Richtung an und haben standardmäßig den Wert `0`. In den `if`-Anweisungen wird mit den `maxSpeed`-Variablen sichergestellt, dass die Geschwindigkeit in `speedx` und `speedy` nicht ununterbrochen zunehmen kann. Damit wird

abgesichert, dass das die Drehung um den Charakter nicht zu schnell wird. Die movement-Variablen werden auf die speed-Variablen addiert. Damit wird erreicht, dass die Drehgeschwindigkeit der Kamera von den jeweils ausgeführten Touch-Bewegungen (die zurückgelegte Strecke und Geschwindigkeit der Touch-Bewegung) abhängig ist.

```
  ...
  if(Mathf.Abs(speedx+movementx) <= maxSpeedX){
    speedx += movementx;
  }

  if(Mathf.Abs(speedy+movementy) <= maxSpeedY){
    speedy += movementy;
  }
  ...
```

Als nächstes werden die speed-Variablen gedämpft, damit die Drehbewegung von selbst aufhört. Dazu stellt Unity3D die Funktion Mathf.Lerp zur Verfügung. Diese interpoliert zwischen den aktuellen speed-Werten und dem Wert 0 über eine festgelegte Zeit. Wird ein definierter minSpeed unterschritten, so wird die zugehörige speed-Variable genullt. Grund dafür ist, dass ab einem bestimmten Bereich zwar noch weiter gedämpft wird, es aber nicht mehr auffällt.

```
  ...
  speedx = Mathf.Lerp(speedx, 0, Time.deltaTime*1.2f);
  speedy = Mathf.Lerp(speedy, 0, Time.deltaTime*1.2f);

  if(Mathf.Abs(speedx) < minSpeed) speedx = 0;

  if(Mathf.Abs(speedy) < minSpeed) speedy = 0;
  ...
```

Die Variablen x und y stehen für die Rotationswinkel um die X- und Y-Achse. So wird die Rotation um die X-Achse zwischen 50 und -10 Grad eingeschränkt. Um die Y-Achse wird keine Einschränkung gesetzt, so dass ein kompletter Rundumblick möglich wird. Grund dafür, dass die Variable x mit speedy verrechnet wird, ist das eine Bewegung auf einem Touch-Monitor in Y-Richtung, eine Rotation in der Szene um die X-Achse zur Folge hat. Demzufolge ist eine Bewegung in X-Richtung mit einer Rotation um die Y-Achse in der Szene umzusetzen.

```
  ...
  if(x+speedy <= 50 && x+speedy >= -10){
    x+= speedy;
  }else if(x+speedy>50){
    x=50;
    speedy=0;
  }else if(x+speedy<-10){
    x=-10;
    speedy=0;
  }

  y-= speedx;
  ...
}
```

Die Funktion Quaternion.Euler gibt eine Rotation zurück, welche die eingegebenen Werte x um die X-Achse und y um die Y-Achse rotiert. Diese Rotation wird in der Variable rotation gespeichert. Damit auch die Kamera diese Rotation ausführt, wird der Rotationswert der Kamera mit dem Wert der Variable rotation ersetzt.

```
  ...
  Quaternion rotation = Quaternion.Euler(x, y, 0);
  ...
  renderingCamera.transform.rotation = rotation;
```

7.5.2 JetPackController

Für die Steuerung der Fluganimation werden drei Skripte eingesetzt. Das `JetPackController`-Skript steuert den Start sowie das Ende des Flugzustandes. Das Skript liegt auf dem Startknopf, der sich auf der Konsole von Newtrix befindet. Beim Drücken des Knopfes wird überprüft, ob Newtrix schon fliegt oder ob er starten muss bzw. darf. Hierbei gibt die Variable `allowTouch` an, ob Newtrix starten darf. Die Variable `allowFly` gib an, ob Newtrix fliegen darf. Wenn Newtrix starten darf, wird beim Drücken des Knopfes zum State `fly` gewechselt und die Coroutine `StartJetPackCo` wird gestartet. Damit bei mehrfachem Drücken nicht mehrere Coroutinen ausgeführt werden, müssen alle bereits laufenden Coroutinen auf dem `GameObject` mit der Funktion `StopAllCoroutines` angehalten werden. Danach werden weitere Interaktionen beschrieben, jedoch ohne auf den Quelltext einzugehen.

```
public override void handleSingleTouch(TouchEvent aTouch){
  if (touchDown) return;

  StopAllCoroutines();

  if(allowFly != true && allowStart == true){
    allowTouch = false;
    game.sm.StateChange("fly");
    StartCoroutine(StartJetPackCo());
  }else{
    ...
  }

  touchDown = true;
}
```

In der Coroutine `StartJetPackCo` wird mit Hilfe vom `VideoController`-Skript ein Video für den Startcountdown abgespielt. Gleichzeitig wird der zugehörige Sound gestartet. Nach zwei Sekunden, wenn das Video zu Ende ist, wird die Variable `allowFly` gesetzt und Newtrix darf abheben.

```
IEnumerator StartJetPackCo() {
  video.PlayMovie(2, false);

  jetPackSound.audio.clip = jetPackSoundFile[0];
  jetPackSound.audio.Play();

  yield return new WaitForSeconds(2);
  allowFly = true;
  ...
```

Weiter wird die Animation von Newtrix mit der Animation für den Start überblendet und das normale Video der Augen wird wieder abgespielt. Bevor zur Fluganimation überblendet wird, wartet das Skript so lange, wie die Animation dauert. Hierbei wird ein Zeitwert abgezogen, der für die Überblendung benötigt wird, da beendete Animationen nicht überblendet werden können.

```
  ...
  game.newtrix.animation.CrossFade("jetpack_start");
  video.PlayMovie(0, true);

  yield return new WaitForSeconds(game.newtrix.animation["jetpack_start"].length-0.2f);
  game.newtrix.animation.CrossFade("jetpack_idle");
  ...
```

Nach dem Überblenden in die Fluganimation werden die Objekte zur Flugsteuerung mit der Funktion `Show()` von dem Skript `FadeObjectByAlpha` eingeblendet. Das grüne Betriebslicht an der Konsole wird ebenfalls eingeschaltet, indem es ein helles Grün als Farbe bekommt.

```
  ...
  jetPackButton.GetComponent<FadeObjectByAlpha>().Show();
  jetPackSpeed.GetComponent<FadeObjectByAlpha>().Show();

  buttonGreen.renderer.material.color = Color.green;
}
```

Da in der Coroutine die Variable `allowFly` auf `true` gesetzt wird, ist in der `Update`-Funktion die Schranke für das Steigen von Newtrix aufgehoben. Die Partikelemitter des Jetpacks sowie ein Licht werden ebenfalls eingeschaltet. Das verwendete Partikelsystem simuliert Feuer, das aus den Jetpackdüsen ausgestoßen wird. Das Licht dient zur Verstärkung des Partikeleffekts. Newtrix wird nun bis zu einem bestimmten Höhenwert transliert. In der Variable `posY` wird die Y-Position von Newtrix vor dem Abheben gespeichert und dient als Höhenreferenz.

```
void Update () {
  if(allowFly == true){
    jetPackParticle.particleEmitter.emit = true;
    jetPackParticle2.particleEmitter.emit = true;
    jetLight.light.enabled = true;

    if(game.newtrix.transform.position.y < posY+0.4f){
      game.newtrix.transform.Translate(Vector3.up * Time.deltaTime*0.5f);
    ...
```

Bei erneutem Drücken des Startknopfes, wird die Variable `allowFly` wieder auf `false` gesetzt sowie die Flugsteuerungsobjekte ausgeblendet. Ebenfalls wird die grüne Lampe ausgeschaltet, indem sie wieder auf einen dunkleren Farbwert gesetzt wird. Durch das Ändern der Variable `allowFly`, wird Newtrix in der `Update`-Funktion wieder auf seine Y-Position transliert. Dabei wird zu einer Landeanimation überblendet. Wenn Newtrix seine Ursprungshöhe erreicht, wird der Jetpack ausgeschaltet, und der State wird zu `initial` geändert.

Die in der Coroutine `StartJetPackCo` aus dem `JetPackController`-Skript eingeblendeten Steuerungsobjekte dienen zur Steuerung der Fluggeschwindigkeit. Das `GameObject jetPackSpeed` enthält das Skript `JetPackSpeedChanger` und dient als Regler für die Fluggeschwindigkeit. In diesem Skript wird anhand der Position des Touches auf dem Regler ein normierter Wert zwischen -0.15 und 0.15 erzeugt. Da beim Fliegen neben der Veränderung der Fluggeschwindigkeit auch der Jetpacksound angepasst wird, wird hier zusätzlich eine `pitch`-Variable erzeugt, die zur Anpassung der Tonhöhe dient.

```
public override void handleSingleTouch(TouchEvent aTouch){
  if(AllowTouch){
    touchDown = true;

    this.renderer.material.SetColor ("_Emission", Color.green);
    y = (aTouch.lastScreenPosition.y/Screen.height)-0.5f;

    if(y <= 0.15 && y> -0.15){
      speedVar= y*12;
      pitchVar = 1+y;
    }
  }
}
```

Das `GameObject jetPackButton` besitzt das Skript `JetPackSpeedController` und dient zur Steuerung des Jetpackschubs. Solange der Knopf gehalten wird, holt sich das Skript in der `Update`-Funktion aus dem Skript `JetPackSpeedChanger` den Geschwindigkeitswert `speedVar` sowie den Wert, der zum "pitchen" des Jetpacksound benötigt wird. Die beiden Werte werden mit "Get"-Funktionen, welche die gewünschten Werte zurückgeben, aufgerufen.

```
void Update () {
  if(istouched == true){
    speedVar = jetPackSpeedChanger.GetSpeedVar();
    pitchVar = jetPackSpeedChanger.GetPitchVar();

    game.newtrix.transform.Translate(Vector3.up * Time.deltaTime*speedVar);
    jetPackSound.audio.pitch = pitchVar;
    ...
```

Weiter regelt das Skript anhand des Geschwindigkeitswertes die Stärke des Ausstoßes der Partikel aus dem Jetpack sowie die maximale Höhe, die Newtrix erreichen darf. Wenn die Höhenschranke überschritten wird, blendet das Skript die Steuerungsobjekte aus und Newtrix sinkt bis zu seiner Startposition.

7.5.3 Weitere Interaktionen

Weitere umgesetzte Interaktionen aus dem Kapitel 3.3.4 (Konzeptentwicklung) sind:

- Das Scannen, bei erstmaligem Besuch eines Planeten.
- Das Hüpfen des Charakters durch "Double-Touch" auf einen seiner Füße.
- Das ständige Anschauen des Benutzers von Newtrix.
- Das Abwischen eines Fingerabdruckes nach einem Touch auf das Display.
- Das Anzeigen der Informationen durch Drücken des Info-Buttons.

In diesem Abschnitt wird jedoch nicht weiter auf den Programmquelltext eingegangen, da das Meiste aus bereits vorgestellten Skripten bekannt ist. Zusätzlich zu den Interaktionen, die der Benutzer auslösen kann, gibt es noch die, bei denen der Charakter Aktionen ohne Einwirken des Benutzers ausführt. Diese werden vom Kometen und Satelliten ausgelöst.

7.5.3.1 Scannen des Planeten

Besucht der Charakter das erste Mal einen Planeten, wird diese Aktion ausgeführt. Ausgelöst wird dies durch einen Planetenwechsel seitens des Benutzers. Während der Scan-Animation sind alle weiteren Funktionen, bis auf das Drehen und das Zoomen der Kamera, deaktiviert. Beim Scannen selbst werden die Antennenkugeln auf dem Kopf des Charakters zu Satellitenschüsseln transformiert. Danach werden Scannstrahlen eingeblendet, die mit pulsierendem Licht unterlegt werden. Dadurch, dass Newtrix seinen Kopf bewegt, scannen die Strahlen den Planeten ab.

7.5.3.2 Hüpfen nach "Double-Touch" auf einen Fuß

Zur Umsetzung der Hüpfinteraktion werden zuallererst "Collider" an den Füßen angelegt. Diese sind in der Lage einen Touch zu erkennen. Nachdem ein Touch erkannt wurde, setzt das `FootController`-Skript eine Zeitvariable. Bei einem weiteren Touch wird die aktuelle Zeit mit der gesetzten Zeitvariable verglichen. Wird dabei die vorgegebene Zeitschranke nicht überschritten, so startet das Skript die Hüpfanimation.

7.5.3.3 Anschauen des Benutzers

Das Anschauen des Benutzers realisiert der `HeadLookController`, der auf dem Charakter liegt. Dieses Skript wird von Unity3D zur Verfügung gestellt.[4] Der HeadLookController wird an die Ansprüche der Anwendung angepasst. Das `HeadLook`-Skript definiert ein Ziel, welches angeschaut wird und bei Bedarf auch ausgetauscht werden kann. Im Normalfall ist das Ziel die aktuelle Position der Kamera. Das Ziel wird an den `HeadLookController` übergeben. Dieser berechnet die Winkel, um die der Charakter den Kopf drehen muss, damit er das Ziel anschaut. Zudem können noch weitere Knochen angegeben werden, die von der Kopfdrehung beeinflusst werden, um eine realistischere Körperhaltung zu erhalten.

7.5.3.4 Abwischen des Fingerabdrucks nach einem Touch

Bei der Umsetzung der Fingerabdruckinteraktion wird ein Collider auf dem Display des Charakters positioniert. Zudem wird eine transparente Fläche ("Plane") kurz vor dem Display platziert. Auf der Plane befindet sich die Textur des Fingerabdrucks. Sobald ein Touch erkannt wird, macht das Skript `FingermarkController` die Fläche mit dem Fingerabdruck sichtbar und startet die Wischanimation. Dabei wird der `HeadLookController` auf ein festes Target vor Newtrix umgeschaltet, damit der Kopf während der Wischanimation nicht beeinflusst werden kann. Zudem wird der Fingerabdruck langsam ausgeblendet.

7.5.3.5 Informationsabruf

Der Info-Button wird, wie das schon erwähnte Planetenauswahlmenü in Kapitel 7.4, vor der Kamera positioniert. Bei Betätigung dieses Buttons wird ein Kameraflug auf den Kopf bzw. das Display des Charakters ausgeführt. Anschließend wird das Video zum aktuellen

[4] http://unity3d.com/support/resources/unity-extensions/head-look-controller

Planeten abgespielt. Durch nochmaliges Betätigen des Buttons wird das Video gestoppt und die Kamera kehrt an die ursprüngliche Position zurück.

7.5.3.6 Interaktion mit einem Komet

Der Komet fliegt auf seiner Umlaufbahn an dem Charakter vorbei. Ab einem bestimmten Punkt wird er vom Charakter erkannt. Dabei wird das Ziel des `HeadLook`-Skriptes auf den Kometen gesetzt, so dass der Charakter nun den Komet anschaut. Ein Sound, der vom Kometen ausgeht, wird abgespielt und Partikel sorgen für einen Feuerschweif. Nachdem der Komet am Charakter vorbei geflogen ist, verschwindet er in den Weiten des Weltalls und der Charakter schaut wieder in die Kamera. Auch der Sound und die Partikel werden ausgeschaltet.

7.5.3.7 Interaktion mit einem Satellit

Der Satellit kreist um die Erde und gibt dabei dauerhaft ein piepsendes Signal von sich. Erreicht er eine definierte Position über dem Charakter, werden Partikel, in Form von Strahlen, ausgesandt. Sobald die Partikel den Charakter erreichen, wird ein Störsignal, in Form eines verrauschten Videos, abgespielt. Nach einiger Zeit hören die Störstrahlen auf und auch das Störvideo wird beendet.

7.6 Weitere Asset-Implementierungen

In diesem Kapitel wird die Erstellung der Videos sowie der Sounds erklärt. Weiter wird auf die Texturierung innerhalb von Unity3D eingegangen.

7.6.1 Videoimplementierung

Für den Fernsehkopf von Newtrix werden, wie im Kapitel 3.3.5 beschrieben, verschiedene Videos benötigt. Mit Hilfe der Programme "Premiere" und "After Effects" der Firma Adobe, werden die einzelnen Videos umgesetzt. In Unity3D besteht die Möglichkeit Videos als "MovieTexture" auf Objekte zu ziehen. Dabei werden Funktionen wie `Play`, `Stop` und `Pause` zur Verfügung gestellt.

Die Augenanimation von Newtrix wird mit einfachen Skalierungen von zwei Ellipsen in After Effects realisiert. Durch setzten von Key-Frames entsteht eine Animation.
Auch die Informationsvideos der Planeten, die beim Drücken des Infoknopfes abgespielt werden, werden in "After Effects" erstellt. Dazu werden Einzelbilder aus Maya rausgerendert und zu einer Sequenz verknüpft. Diese Sequenz zeigt eine komplette Drehung des Planeten. Diese Filmsequenz soll während des gesamten Infovideos im unteren rechten Rand eingebettet sein. Die Informationen werden mit verschiedenen Textebenen zeitlich ein- und ausgeblendet.

In Adobe "Premiere" entstehen die Videos, die beim Starten des Jetpacks sowie bei der Signalstörung durch den Satellit aufgerufen werden. Hier wird neben dem Verwenden des Videos für die Augen noch weitere Filmausschnitte ineinander geschnitten.
Die Videos, die nach einem Planetenwechsel gezeigt werden sollten, wurden aus Zeitgründen nicht umgesetzt. Als Ersatz werden verschiedene Ladebilder verwendet. Diese zeigen Newtrix beim Anflug auf den ausgewählten Planeten.

7.6.2 Audioimplementierung

Für Feedback beim Drücken von Knöpfen, als Hintergrundmusik sowie für die Untermalung von Animationen werden Sounds verwendet. Für diese Arbeit steht eine große Soundbibliothek zur Verfügung. In der Software "Soundbooth" von der Firma Adobe werden die Sounds nachträglich noch mit Effekten versehen und zusammengeschnitten.
Sounds können mit Hilfe der Audiokomponente an jedes beliebige `GameObject` in Unity3D gebunden werden. Unity3D unterstützt Stereosounds und 3D-Sounds. Bei 3D-Sounds kümmert sich die Audio-Engine von Unity3D um die korrekte Berechnung von Lautstärke und Tonhöhe, in Abhängigkeit der Position und Entfernung zum Betrachter.

7.6.3 Texturimplementierung

Unity3D stellt einfache Verfahren zur Verfügung, um Objekten eine Textur zuzuweisen. `GameObjecte`, die eine "Mesh Renderer"-Komponente besitzen, bekommen auch ein Material zugewiesen. Texturen können durch einfaches Ziehen auf das `GameObject`, im "Scene"-Fenster oder auch in der Projekthierarchie, zugewiesen werden. Zur Texturierung von Planeten können einfache Kugeln mit Texturen versehen werden. Dabei wird im Material die Textur zugeteilt. Ebenfalls wird ein Shader einem Material zugeordnet. Die Shader ermöglichen zum Beispiel transparente Texturen oder auch ein "bumpmap"-Effekt. Eine Bump-Map verändert die Oberflächenstruktur eines Objektes, ohne die Geometrie zu verändern. [Bir09]

7.7 Zusammenfassung

Dieses Kapitel beschäftigte sich mit der Implementierung sowie Umsetzung der Anwendung. Zu Beginn wurde das Grundgerüst der Anwendung erklärt. Dazu gehören der `Game`- und der `StateManager`. Der `StateManager` verwaltet die einzelnen Zustände der Applikation und legt Regeln für den Ablauf fest. Der `GameManager` verwaltet alle wichtigen Variablen und stellt sie anderen Skripten zur Verfügung.

Zur Implementierung der Multi-Touch-Gesten wurde die Anbindung an den TUIO-Stream erläutert und die Funktionsweise der einzelnen Skripte näher betrachtet. Hierbei lag der Schwerpunkt auf der Erkennung von "Single"- und "Multi-Touches" auf Objekten in der 3D-Szene. Weiter wurde in diesem Kapitel die Funktionsweise der Levelsteuerung erklärt. Ein wichtiger Punkt dabei war die Persistenz bestimmter Variablen und Objekte.

Im nachfolgenden Teil wurde anhand zweier Beispiele die Umsetzung von Interaktionen in der Anwendung beschrieben. Zum einen wurde die Navigation in der Szene sowie das Fliegen mit dem Jetpack erklärt. Alle weiter umgesetzten Interaktionen wurden anschließend aufgeführt und erläutert.

Danach wurde auf die Verwendung weiterer Assets, wie beispielsweise Video- und Audioclips, eingegangen. Neben der Implementierung wurde auch deren Erstellung näher beleuchtet.

Im nächsten Kapitel werden die Ergebnisse und Erkenntnisse dieser Arbeit beschrieben und analysiert.

Kapitel 8

Ergebnis

In diesem Kapitel werden die Ergebnisse der Arbeit vorgestellt. Dabei wird auf die gewählten Lösungswege und Probleme bei der Umsetzung der Anwendung eingegangen. Das Ergebnis dieser Arbeit wird in folgende Teilbereiche gegliedert:

- Interaktivität des Charakters und Echtzeitfähigkeit
- Multi-Touch-Fähigkeit
- Hardwareunabhängigkeit

Zudem definiert das in Kapitel 3 entwickelte Konzept Anforderungen an die Anwendung, deren Ergebnisse ebenfalls behandelt werden. Bedingt durch den begrenzten Zeitraum, der zur Umsetzung der Anwendung zur Verfügung stand, wurde nicht das gesamte Konzept in der Anwendung umgesetzt.

8.1 Interaktivität

Die wichtigste Anforderung an die Entwicklungsumgebung war, dass sie in der Lage sein musste, Charaktere einzubinden und zu steuern. Zudem musste in ihr die Möglichkeit bestehen, eine echtzeitfähige 3D-Anwedung zu erstellen.
Mit der gewählten Game-Engine Unity3D wurden diese Anforderungen erfüllt. Die implementierten Aktionen des Charakters werden zur Echtzeit ausgeführt. Dafür wurde auf verschiedenen Rechnern, mit verschiedener Leistung, die Echtzeitfähigkeit geprüft und die resultierenden Frameraten verglichen. Die Frameraten der Anwendung bewegen sich dabei in einem Bereich von 60 bis 200 fps. Da aus Kapitel 2.5 bekannt ist, dass bereits bei 15 fps die Rede von Echtzeit ist, wird damit klar, dass die Anwendung zur Echtzeit läuft. Ab einer Framerate von 60 fps ist ein Ablauf der Anwendung ohne Verzögerung gewährleistet. Bei Frameraten von 72 fps und höher ist jedoch kein Unterschied der Ausgabe auf dem Monitor zu erkennen. [TAM08]

Bei höheren Frameraten wird deutlich, dass die Bewegungen auf dem Touch-Display eine höhere Geschwindigkeit, zum Beispiel bei der Navigation, zur Folge haben. Da die `Update`-Funktion, in der die Geschwindigkeit der Bewegung berechnet wird, einmal pro Frame aufgerufen wird, wird diese öfter durchlaufen und ein höherer Geschwindigkeitswert wird schneller erreicht. Um diesen unerwünschten Effekt vorzubeugen, wird die Funktion `targetFrameRate` genutzt. Diese versucht eine eingestellte Framerate zu halten bzw. zu erreichen. Somit wird sichergestellt, dass die Anwendung sich auf allen Geräten gleich verhält. Da ab 72 fps keine Verbesserung mehr sichtbar ist, wird dieser Wert als angestrebte Schranke in der Funktion festgelegt.

8.2 Multi-Touch-Fähigkeit

Damit die Anwendung multi-touch-fähig ist, musste gewährleistet werden, dass mehrere Touches erkannt werden. Zur Umsetzung dieser Anforderung wurden in der Entwicklungsumgebung Unity3D Skripte eingebunden, die eine Verbindung zum TUIO-Stream aufbauen. Ein TUIO-fähiges Multi-Touch-Eingabegerät erkennt die Touches und schickt ein Ereignis mittels des TUIO-Protokolls an den Rechner. Die Anwendung kann die Touches mit Hilfe der Skripte auslesen und verarbeiten. So konnten verschiedene Interaktionen durch verschiedene Single- und Double-Touches in der Anwendung umgesetzt werden.

8.3 Hardwareunabhängigkeit

Hardwareunabhängigkeit ist die Eigenschaft einer Anwendung, unabhängig von der zugrundeliegenden physischen Hardware lauffähig zu sein. Neben einer gewissen Grundrechenleistung, die ein Computersystems dabei erfüllen muss, spielt auch das zu Grunde liegende Verfahren zur Touch-Erkennung eine Rolle. Deshalb kann bei der umgesetzten Anwendung nur in einem bestimmten Rahmen von Hardwareunabhängigkeit gesprochen werde. Die im Moment größtmögliche Freiheit dabei bietet das standardisierte TUIO-Protokoll, welches auf das OSC-Protokoll aufsetzt. Somit wird gewährleistet, dass die Anwendung auf allen Geräten, die dieses Protokoll unterstützen, ausgeführt werden kann.

Ein anderer Lösungsansatz könnte die Umsetzung der Anwendung für Betriebssysteme mit Multi-Touch-Unterstützung sein. Durch die Wahl eines Betriebssystems wird jedoch die Hardwareunabhängigkeit eingeschränkt, da die Anwendung nur noch auf dem jeweiligen Betriebssystem und der Hardware, die dieses unterstützt, ausgeführt werden kann.

8.4 Konzeptanforderungen

Das selbst entwickelte Konzept stellt weitere Anforderungen an die Anwendung, welche wie folgt definiert sind:

- möglichst natürliche Bewegungen des Charakters
- zielgruppenorientierte Erstellung der Anwendung

- sinnvolle Nutzung der Touch-Gesten
- Umsetzung der Konzeptideen

8.4.1 Natürliche Bewegungen

Es gibt viele verschiedene Möglichkeiten einen Charakter zu animieren. Unter dem Gesichtspunkt, dass die Bewegungen möglichst natürlich sein sollen, kam aber nur Motion-Capturing in Frage. Zwar ist es auch möglich, ähnlich gute Ergebnisse mit anderen Animationsmethoden zu erhalten, jedoch wird dafür wesentlich mehr Zeit benötigt. Da Motion-Capture-Daten von realen Darstellern aufgenommen werden, ist ein Höchstmaß an Natürlichkeit der Bewegungen gegeben. Die Bewegungsdaten für die Anwendung wurden von der Firma NewMedia Yuppies zur Verfügung gestellt. Die Daten wurden für den erstellten Charakter in MotionBuilder aufbereitet und überarbeitet. Andere Softwarepakete, die MoCap-Daten verarbeiten können, sind aber ebenso gut zum Bearbeiten der Animationen geeignet. Die einzelnen Animationen wurden abgespeichert und in Unity3D importiert.
Beim Einsatz der Animationen in Unity3D wurde darauf geachtet, dass das Überblenden rechtzeitig stattfindet. Ist eine Animation beendet, kann nicht mehr zwischen den Animationen geblendet werden und es kann zu einem "Sprung" in der Bewegung des Charakters kommen. Animationen, wie beispielsweise das Gehen oder Laufen, wurden von vornherein so erstellt, dass sie "loopbar" sind. Animationen, wie die Startanimation beim Fliegen, sollten zudem nicht zu kurz sein, denn das Blenden würde wichtige Bereiche der Animation überschreiben. Wird all dies beachtet, so bewegt sich der Charakter sehr natürlich.

8.4.2 Zielgruppe

Der Fokus der Anwendungsentwicklung richtet sich an die Zielgruppe Kind. So wurden das Aussehen des Charakters sowie die Gestaltung der Spielwelt darauf ausgerichtet. Hierbei wurde auf ein kindgerechtes Aussehen von Newtrix Wert gelegt. Das zu vermittelnde Wissen wurde auf den Wissensstand der Zielgruppe ausgerichtet. Die Informationen werden in Form von kurzen Info-Videos zur Verfügung gestellt. Darin werden wichtige Fakten zu den Planeten aufgezählt. Zudem besteht die Möglichkeit, durch einfaches Anklicken von Planeten in der Szene, den Namen und weitere Informationen zu erhalten.
Weiter wurden Elemente zur Unterhaltung der Kinder eingebunden. Dazu gehören beispielsweise das Drehen der Planeten, das Nutzen des Jetpacks und verschiedene Warteanimationen, die Abwechslung bieten.

Die Anwendung wurde vereinzelt von Kindern und auch erwachsenen Personen getestet. Die Resonanz war durchweg positiv. Nach einer kurzen Einweisung waren alle Kinder in der Lage die Anwendung zu bedienen, was ihnen sichtlich Spaß gemacht hat. Auch von den Erwachsenen wurden die Anwendung und damit die vorgegebenen Interaktionen intuitiv bedient. Im Rahmen einer Hochschulveranstaltung wurde die Anwendung einer Vielzahl von Studenten vorgestellt. Dabei fand die Anwendung großen Anklang und weckte die Neugier der Studenten, die mit Freude und Begeisterung mit dem Charakter interagiert haben. Ei-

ne genauere Evaluationsstudie über die Zielgruppentauglichkeit ist geplant, wird aber im Rahmen dieser Arbeit nicht durchgeführt.

8.4.3 Sinnvolle Multi-Touch-Gesten

Für die festgelegten Interaktionen wurden Touch- bzw. Multi-Touch-Gesten erarbeitet. Dabei unterteilen sich die Touch-Gesten in direkte und indirekte Interaktionen. Interaktionen die im direkten Bezug zu dem Charakter stehen, wie zum Beispiel das Drücken des Displays, sind direkte Interaktionen. Indirekte Interaktionen sind Interaktionen, die in erster Linie nicht auf den Charakter bezogen sind, aber auf die der Charakter reagiert. Ein Beispiel dafür ist das Drehen des Planeten damit Newtrix läuft. Um zu gewährleisten, dass Multi-Touch-Gesten sinnvoll und intuitiv eingesetzt werden, wurde dies bereits ab dem ersten lauffähigen Stand der Anwendung getestet. Eine bereits bekannte Geste, wie das Zoomen, wurde in der Anwendung erwartungsgemäß implementiert. Hierbei werden zwei Finger aufeinander zu oder voneinander weg bewegt. Das Drehen um den Charakter wurde zuerst mit einer Double-Touch-Geste umgesetzt. Dabei wurden zwei Finger im Kreis- bzw. Halbkreis über das Display bewegt, um die Szene zu drehen. Bei den Tests wurde jedoch schnell klar, dass dies nicht intuitiv für die Nutzer ist. Vielmehr wird ein Drehen mit nur einem Finger erwartet. Da die Multi-Touch-Anwendung für Kinder erstellt wird, steht die intuitive Bedienung mehr im Vordergrund als der Einsatz von Multi-Touch-Gesten. Die Studie "Touch it, move it, scale it - Multitouch" [CH09] belegt, dass für Benutzer Interaktion durch einen Single-Touch in den meisten Fällen intuitiver sind. Dies ist der Anlass warum das Drehen mit einem Single-Touch umgesetzt wurde.
Die Verwendung von Muli-Touch-Gesten zur Interaktion mit dem Charakter gestaltete sich als schwierig. Zur Aktivierung von Funktionen am Charakter sind Single-Touches intuitiver. Jedoch bietet sich, bei der Umsetzung des Planetenauswahlmenüs und der Steuerung des Jetpacks, die Möglichkeit Double-Touches einzusetzen, da es für diese Funktionen keine allgemein gültigen Vorstellungen gibt. Eine weitere Möglichkeit für den Einsatz von Multi-Touch-Gesten bietet die Funktion, in der der Charakter geschubst wird. Diese wurde aber nicht umgesetzt, da Touches nur auf dem getouchten Objekt erkannt werden. Wird also, wie im Konzept beschrieben, neben den Charakter getoucht und die Finger dann über ihn gezogen, so kann nicht erkannt werden, ob die Finger über dem Charakter sind. Ein Lösungsansatz wäre, dass der Charakter mit zwei Fingern getoucht wird und die Finger bewegt werden, um ihn zu schubsen. An diesem Beispiel ist zu erkennen, dass durch die Funktionsweise der Entwicklungsumgebung Grenzen bei der Umsetzung von denkbaren Interaktionen bestehen. Durch die Abbildung einer 3D-Welt auf eine 2D-Ebene, wird ebenfalls der Aktionsumfang des Benutzers auf den Charakter begrenzt. So kann der Benutzer zum Beispiel nicht hinter Newtrix greifen, sondern ist auf die Projektion der 3D-Welt auf dem Display angewiesen.
In dieser Arbeit konnte gezeigt werden, dass neue Interaktionsformen mit einem Charakter mit Touch- bzw. Multi-Touch-Gesten realisiert werden können.

8.5 Erweiterung und Optimierung der Anwendung

Nach den ersten Tests der Anwendung fiel auf, dass viele Benutzer versuchten die Planeten im Hintergrund anzuklicken. Dies berücksichtigend, wurde eine nicht im Konzept enthaltene Funktion hinzugefügt. Durch Drücken der Planeten im Hintergrund wird eine Art "Sprechblase", über dem jeweiligen Planeten, eingeblendet und Kurzinfos zu diesem bereitgestellt. Mit dem Skript `CameraFacingBillboard` werden Objekte, in diesem Fall Planes mit einer Textur, in Richtung der Kamera gedreht. Zudem wird auch die Größe der Objekte angepasst, so dass unabhängig von der Entfernung zur Kamera alle Objekte gleich groß sind. Ein "TextMesh" wurde auf die Plane gelegt, das die Informationen des Planeten enthält. Jeder Planet erhält einen "Collider", um Touches erkennen zu können. Wie schon bei der Interaktion Hüpfen, muss der jeweilige Planet getoucht werden, damit die Informationen erscheinen. Diese werden mit Hilfe des `ObjectFadeOnTouch`-Skriptes ein bzw. ausgeblendet. Das Skript erkennt die Touches und übergibt das zu blendende Label an das `FadeObjectByAlpha`-Skript. Hier wird das Label dann über eine beliebige Zeit ein- oder ausgeblendet. Bei nochmaligem Touchen auf den Planeten verschwinden die Informationen wieder.

Zur weiteren Optimierung der Framerate in der Anwendung wurden die verschiedenen Modelle, wie beispielsweise der Charakter, die Planetenkugeln und der Satellit überarbeitet, um die Polygonzahl in der Szene zu reduzieren. Zudem wurde die Texturauflösung soweit wie möglich verringert, ohne dass große Qualitätsunterschiede auffallen.

Während der Umsetzung der Aktionen in Unity3D musste gewährleistet werden, dass Abläufe und Aktionen ordnungsgemäß ausgeführt werden. Durch die Komplexität der Anwendung sind viele Absicherungen nötig. Deshalb wurde auf das Konzept einer Zustandsmaschine (`StateManager`) zur Anwendungssteuerung zurückgegriffen.

Nach der Implementierung des `HeadLookController`, der die Blickrichtung des Charakters steuert, musste für manche Animationen das Überschreiben der Kopfanimation ausgeschaltet werden. Das Skript stellt eine solche Möglichkeit zur Verfügung, jedoch entstand beim Umschalten zwischen Überschreiben und nicht Überschreiben der Animation eine ruckartige Bewegung des Kopfes. Um diese Problem zu beheben, wurde das Skript an dieser Stelle im Quelltext angepasst, so dass beim Umschalten eine Interpolation ausgeführt wird.

8.6 Wahl der Entwicklungsumgebung

Zusammenfassend kann gesagt werden, dass durch die Wahl der Echtzeitentwicklungsumgebung Unity3D die verschiedenen Technologien verknüpft werden konnten. Unity3D ermöglicht den Einsatz und die Steuerung von Charakteren. Die benötigten Animationen für den Charakter können importiert werden. Animationen können in Unity3D mit verschiedenen Funktionen überlagert oder überblendet werden, was einen natürlichen Bewegungsablauf des Charakters ermöglicht. Durch das Einbinden der Skripte der TUIO-Bibliothek ist auch der Einsatz von Multi-Touch-Geräten möglich. Mit den geschriebenen C#-Skripten wer-

den die Aktionen der Anwendung gesteuert. Das wichtigste Steuerelement hierbei ist der `StateManager`. Durch andere Skripte werden die Interaktionen mit dem Charakter und die Navigation in der 3D-Welt festgelegt. Unity3D stellt auch die Möglichkeit zur Einbindung von Audio- und Videoclips zur Verfügung.

Kapitel 9

Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde die Problemstellung "Interaktion mit einem Charakter über Touch bzw. Multi-Touch-Gesten" untersucht. Dabei wurden neue intuitive Interaktionen ausgearbeitet, exemplarisch implementiert und anschließend getestet bzw. auf ihre Tauglichkeit geprüft. Dazu wurde eine 3D-Anwendung entwickelt, die spielerisch Wissen vermitteln soll. In dieser Anwendung kann der Benutzer mit Hilfe von Multi-Touch-Gesten navigieren und mit einem animierten Charakter interagieren. Hierbei lag das Ziel bei der Umsetzung darin, dass die Anwendung möglichst Hardwareunabhängig ausgeführt werden kann. Weiter wurde im Kapitel 1 festgelegt, dass die entstehende Anwendung echtzeitfähig sein muss. In diesem Kapitel wurde ebenfalls festgelegt, dass die Bewegungsabläufe des Charakters möglichst natürlich und flüssig dargestellt werden sollen. Zur Animation des Charakters sollten Motion-Capture-Daten eingesetzt werden. Generell sind Anwendungen, in denen ein interaktiver Charakter eingebunden ist keine Neuheit, aber die Verbindung mit dem Eingabeverfahren Multi-Touch ist zum momentanen Zeitpunkt noch eine Seltenheit. Das Kapitel 2 "Stand der Technik" gab einen Überblick über die vielen Teilbereiche, der in der Arbeit verwendeten Technologien sowie Verfahren. Zuerst wurde ein Teilbereich der 3D-Computergrafik, die virtuellen Charaktere, näher betrachtet. Hierbei lag der Schwerpunkt auf dem Bereich der Interaktion. Des weiteren wurden die verschiedenen Methoden zur Charakteranimation beschrieben. Dabei wurde das Augenmerk auf optische Motion-Capture-Verfahren gelegt. Im nachfolgenden Abschnitt wurden Technologien zur Verwendung von Multi-Touch-Erkennung vorgestellt. Dabei wurde vor allem auf das hardwareunabhängige Verfahren unter Nutzung des TUIO-Protokolls eingegangen. Danach folgte eine Beschreibung von 3D-Echtzeitanwendungen sowie deren Einsatzgebiete. Hier lag der Schwerpunkt bei den aktuellen "Game-Engins".

Die Herausforderung dieser Bachelorarbeit bestand darin, die entsprechend dem Stand der Technik vorgestellten Verfahren und Technologien (Kapitel 2), in einer 3D-Anwendung zu verknüpfen. In Kapitel 3 (Konzeptentwicklung), wurden Ansätze zur Umsetzung der Anwendung entworfen und konkretisiert. Neben der Festlegung des Spielablaufs, wurden im Konzept die Zielgruppe sowie das Aussehen des Charakters und der Spielwelt erarbeitet. Auch die Umsetzung der Multi-Touch-Gesten zur Interaktion mit dem Charakter sowie die

Navigation in der Software wurden ausgearbeitet.
Weiter wurde in dieser Arbeit ein Charakter erstellt. Das Aussehen des Charakters wurde anhand der Vorlage, die im Kapitel 3 entstanden ist, umgesetzt. Dabei wurde auf das Modellieren der Charaktergeometrie sowie die Erstellung der Textur und des Skeletts eingegangen. Danach wurde auf die Nachbearbeitung der Animationsdaten einer MoCap-Aufnahme eingegangen sowie dabei häufig auftretende Probleme erklärt. Weiter wurde das Zusammenfügen der MoCap-Daten mit dem Charakter näher beleuchtet. Animationen, die nicht als Daten vorlagen, wurden aus vorhandenen Animationsdaten erstellt.
Da für die Anwendung eine Entwicklungsumgebung benötigt wurde, die in der Lage ist Echtzeitdarstellung sowie die Integration von Multi-Touch-Gesten zu gewährleisten, wurde in dieser Arbeit die Auswahl einer geeigneten Entwicklungsumgebung getroffen. Bei der Auswahl mussten folgende Kriterien erfüllt werden:

- Multi-Touch Anbindung
- Charakter Implementierung
- Möglichkeiten zur Animationsüberblendung
- Video & Sound Einbindung

In den zwei geeignetsten Entwicklungsumgebungen, instantreality und Unity3D, wurde je eine Testanwendung erstellt. Mit Hilfe der Testanwendung sollte eine endgültige Entscheidung getroffen werden, welche Software zur Umsetzung des Spieles genutzt wird. Nach dem Vergleich der beiden Testanwendungen wurde die Spielentwicklungssoftware Unity3D ausgewählt, da in instantreality unter anderem gravierende Probleme bei der Animationsüberblendung aufgedeckt wurden.
Da Unity3D eine GUI in der Entwicklungsumgebung besitzt, konnte das Komponieren (positionieren der einzelnen Objekte in einem Level) der einzelnen 3D-Welten im "Scene"-Fenster passieren. Mit Hilfe von C#-Skripten wurde die Steuerung der einzelnen Abläufe und Aktionen übernommen. Dabei stellt das `StateManager`-Skript sowie das `GameManager`-Skript das Fundament der Anwendung dar. Mit Hilfe der TUIO-Bibliothek konnte eine Anbindung an Geräte, die das TUIO-Protokoll unterstützen, realisiert werden. Die Steuerung der Anwendung sowie die Multi-Touch-Ansteuerung wurde anhand von zwei Skriptbeispielen vorgestellt. Weiter wurde die Verwendung von Audioclips sowie Videos in der Anwendung behandelt.
Am Ende dieser Arbeit wurden die wichtigsten Ergebnisse aufgeführt, Probleme aufgezeigt und auf mögliche Lösungen hingewiesen.

9.1 Ausblick

In dieser Arbeit wurde bei der Entwicklung der Multi-Touch-Anwendung, ein großer Problembereich abgedeckt. Aus Zeitgründen wurde deshalb nur ein Teil der Anwendung umgesetzt. Die drei entstandenen Level stehen exemplarisch für den Verlauf der Anwendung. Anhand von planetenspezifischen Zusatzaktionen, wurde gezeigt wie die einzelnen Planeten interessanter

und die Anwendung abwechslungsreicher gestaltet werden kann. Die entstandene Anwendung kann so als Ausgangsbasis für zukünftige Erweiterungen oder als Anregung für weitere Projekte dienen. Die Ansätze für sinnvolle Multi-Touch-Gesten, in so einer 3D-Anwendung, könnten in anderen Arbeiten erweitert werden. In dieser Arbeit hat sich gezeigt, dass bei vielen Interaktionen mit dem Charakter ein Single-Touch die sinnvollste Lösung ist. Multi-Touch-Gesten dienen eher zur Navigation in der Anwendung. Die Kombination aus Single- und Multi-Touch-Gesten macht bei der Bedienung von Multi-Touch-Anwendungen aber den Reiz aus. Mit einem Ansatz, bei dem der Charakter frei in einer 3D-Szene gesteuert werden kann, können mehr Multi-Touch-Gesten eingesetzt werden. Dadurch, dass die Bedienung von Multi-Touch-Geräten einfach und intuitiv ist, steigt die Nachfrage nach solchen Geräten, was die Entwicklung dieser vorantreibt. Zudem wird die Verbreitung durch immer besser und günstiger werdende Hardware gefördert. Vor allem im Bereich der Mobiltelefontechnologien setzten immer mehr Hersteller auf Multi-Touch-Displays. Die Anzahl der Anwendungen für solche Geräte steigt immer mehr. Aber auch Betriebssysteme ermöglichen heutzutage den Einsatz von Multi-Touch. So könnten Multi-Touch-Displays etablierte Eingabegeräte, wie zum Beispiel die Maus, noch mehr ergänzen. Die allgemeine Fragestellung zur Verknüpfung von Multi-Touch-Geräten und interaktiven Charakteren bietet weiteres Potential für Forschungs- und Entwicklungsarbeiten. Besonders die Entwicklung bzw. Weiterentwicklung von Touch-Gesten in Zusammenhang mit Charakteren gibt viel Spielraum für Neues. Es ist abzusehen, dass die Verknüpfung von 3D-Multi-Touch-Anwendung mit interaktiven Charakteren mehr an Bedeutung gewinnt. Hierbei wird der Charakter sehr wahrscheinlich nicht im Vordergrund der Anwendung stehen, vielmehr könnte dieser als "Ansprechpartner" dienen. Eine Entwicklung, die in diese Richtung zeigt, ist die neue Generation von Navigationssystemen, in denen interaktive Charaktere integriert sind. Hierbei ist die Animation der Charaktere sehr wichtig, denn nur wenn die Animationen der Charaktere realistisch wirken, werden die Charaktere vom Benutzer auch anerkannt. Da sich die Verfahren der Motion-Capture-Systeme weiter verbessern, wird in Zukunft die Möglichkeit bestehen, noch natürlichere und realistischer wirkende Charaktere in 3D-Anwendungen zu integrieren.

Anhang A

Skriptübersicht

Skriptname	Funktion
`StateManager`	Regelt die Zustände in der Anwendung.
`GameManager`	Enthält alle wichtigen Variablen.
`CameraFacing-Billboard`	Sorgt dafür das die Planeten-Labels immer gleich groß und zur Kamera gewandt sind.
`CameraFade`	Dient zum ein- und ausblende des Kamerabildes.
`CamController`	Ist für die Navigation in der Szene und die Kamerabewegungen zuständig.
`FingermarkController`	Kontrolliert die Abläufe falls auf das Display des Charakters getoucht wird.
`FootController`	Sorg dafür das die Hüpfanimationen gestartet werden.
`InfoController`	Kontrolliert den Info-Button (Aktiviert und Deaktiviert ihn und spielt passende Videos ab).
`JetPackController`	Steuert Start und Landung (Videos, Sounds Animationen) und startet und beendet den Flugzustand.
`JetPackSpeedChanger`	Wandelt die Touches in Geschwindigkeitswerte um.
`JetPackSpeed-Controller`	Steuert die Flughöhe anhand der Geschwindigkeit.
`LevelManager`	Verwaltet die Liste der besuchten Level und kennt das aktuelle geladene Level sowie das zuletzt geladene. Das Skript ist persistent.

`LevelController`	Steuert anhand von Funktionen, das Starten sowie das wechseln von Leveln.
`LevelChange`	Bei Auswahl eines Menüobjektes übergibt das Skript das zu ladende Level an den `LevelController`. Weiter überprüft das Skript, ob ein Level geladen werden darf.
`PlanetController`	Dient der Steuerung des Planeten (Drehen des Planeten, starten und beenden der Laufanimation).
`PlanetMenuController`	Kontrolliert das Ein- und Ausblenden der Planeten-Buttons.
`HeadLookController`	Berechnet die Überblendung der gewählten Bereiche und stellt Einstellungsmöglichkeit für die Überblendungsstärke und Bereiche zur Verfügung.
`HeadLook`	Überblendet die Kopfanimation und setzt das Ziel, welches angeschaut werden soll.
`CometController`	Rotiert den Kometen und setzt das `HeadLookTarget` wenn er in Reichweite ist.
`CamPlaneAdjust`	Positionier ein Plane in Sichtfeld der Kamera, auf der dann die Objekte mit normalisierten Koordinaten positioniert werden können.
`GUIScreenPositioning`	Wandelt die Bildschirmauflösung in normalisierte Koordinaten um, so das Menüobjekte auf der Kameraplane positioniert werden können.
`VideoController`	Enthält alle Videos die im Monitor des Charakters laufen und Steuert das Abspielen.
`AudioController`	Steuert die Sounds eines Objektes.
`SatelliteController`	Steuert die Umlaufbahn des Satelliten, die Satellitenstrahlen und das Abspielen des Videos.
`LightPulser`	Lässt Lichter und "LensFlares" mit einstellbarer Frequenz und Intensität pulsieren.
`ObjectPulser`	Lässt Objekte mit gewünschter Frequenz und Größe pulsieren.
`MeshMorpher`	Morpht das Aussehen einer Objekteoberfläche zu einer anderen.

`ObjectActivator`	Schaltet den Renderer und Collider eines Objekts aus und kann den Layer ändert, damit Objekte unsichtbar werden und nicht beeinflusst werden können.
`ObjectFadeOnTouch`	Blendet zugewiesene Objekte, über das `FadeObjectByAlpha`-Skript, nach einem Touch ein oder aus.
`ObjectRotator`	Rotiert Objekte um verschiedene Achsen.
`RecursiveDontDestroyOnLoad`	Setzt Objekte persistent, so dass sie in neuen Levels nicht unverändert vorhanden sind.
`PersistenceChecker`	Sind persistente Objekte doppelt vorhanden sind, so wird das neu erstellte Objekt zerstört.
`NewtrixController`	Kontrolliert die Idle-Animationen und spielt die Warteanimationen ab.
`TexOffset`	Animiert die Texturkoordinaten.
`FadeObjectByAlpha`	Blendet den Alphawert (Transparenzwert) des Objektes ein und aus.
`TouchEventManager`	Baut eine bidirektional Verbindung zum "TUIO-Stream" auf und ermittelt anhand von Raycasts, ob bei einem Touchevent ein Touchobjekt getroffen wurde. Es schickt betroffene Objekte die Touchevents.
`SceneManager`	Überprüft, ob ein `TouchEventManager` in der Szene vorhanden ist. Und sorgt dafür, dass immer nur einer existiert.
`TouchableObject`	Basisklasse alles Objekte, die auf Touches reagieren. Das Skript eines Touchobjektes muss von diesem Skript abgeleitet sein.
`CrosshairController`	Setzt ein Fadenkreuzobjekt an die Bildschirmposition eines Touches.

Anhang B

Bilder der Anwendung

Abbildung B.1: Newtrix schaut auf den Kometen der an der Erde vorbeifliegt.

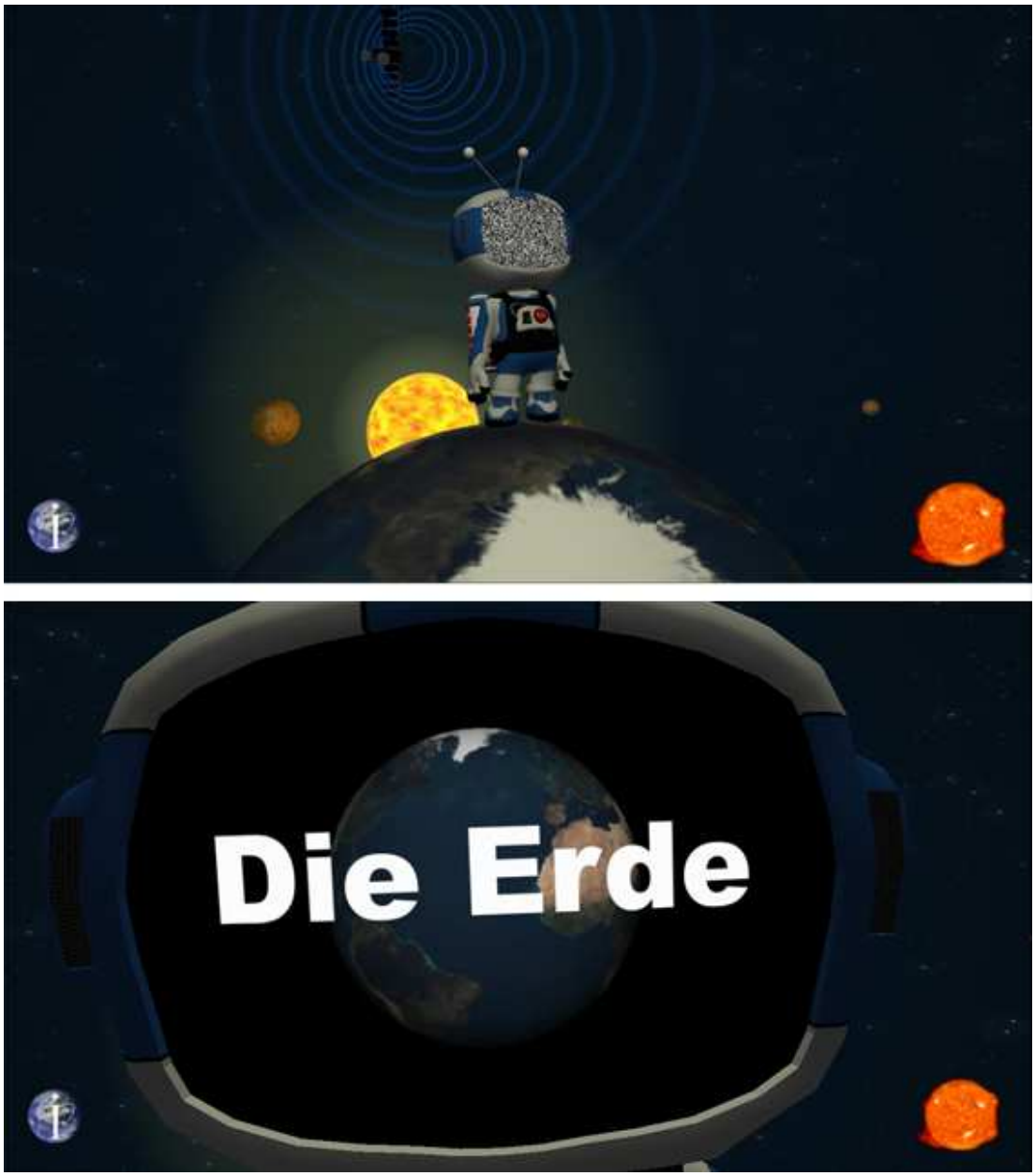

Abbildung B.2: Oben: Newtrix wird durch die Satellitenstarahlen gestört. Unten: Das Informationsvideo der Erde wird auf Newtrix Kopf abgespielt.

Abbildung B.3: Oben: Scannanimation von Newtrix nach dem Besuch eines Planeten. Unten: Labels mit den Planeteninformationen.

Abbildung B.4: Oben: Laufanimation während der Planet gedreht wird. Unten: Abwischen des Fingerabdrucks nach Drücken des Displays.

Abbildung B.5: Oben: Hüpfanimation nach dem Double-Touch am Fuß. Unten: Planetenauswahlmenü mit dem drei auswählbaren Planeten.

Abbildung B.6: Oben: Newtrix winkt dem Benutzer zu wenn keine Eingabe erfolgt. Unten: Fluganimation mit sichtbaren Steuerungs-Buttons.

Glossar

API	Eine API ("application programming interface") ist eine Programmierschnittstelle, die eine Programmanbindung auf Quelltextebene definiert.
Bikubische Interpolation	Die bikubische Interpolation bestimmt Zwischenwerte innerhalb eines zweidimensionalen, regulären Rasters.
Blobs	Ein Blob ist bei der Touch-Erkennung in der Software "Community Core Vision" (CCV) ein heller leuchtender Punkt. Anhand der erkannten Blobs wertet die Software aus wie viele Touches vorhanden sind.
Bump-Map	Ein "Bump-Map" ist eine Graustufen-Bilddatei, über die Tiefeninformationen in die Oberfläche einer Geometrie eingearbeitet werden.
Coroutine	In der Informatik sind Coroutine Programm-Komponenten, die ihren Ablauf unterbrechen und später wieder aufnehmen können. Weiter können Coroutinen parallel ausgeführt werden. In Unity3D ist mit Hilfe von Coroutinen eine zeitliche Steuerung der Funktionen Möglich.
Framework	Ein Framework ist ein Programmiergerüst, das in der Softwaretechnik, insbesondere im Rahmen der objektorientierten Softwareentwicklung, verwendet wird. Hierbei ist es selbst noch kein fertiges Programm, sondern stellt den Rahmen zur Verfügung, indem der Programmierer eine Anwendung erstellt kann.
GUI	Eine grafische Benutzeroberfläche ("Graphical User Interface") ist eine Software-Komponente, die dem Benutzer eines Computers bzw. einer Software die Interaktion mit der Maschine bzw. einer Anwendung über grafische Symbole erlaubt.
Gyroskop	Gyroskope oder auch Kreiselinstrument, misst einachsige Drehgeschwindigkeiten. Das Messprinzip ist entweder mechanisch oder optisch. Ein typisches mechanisches Gyroskop besteht aus rotierenden Kreiseln, die in Kardanringen montiert sind. Bei optischen Gyroskopen erfassen Bildsensoren Änderungen im Rotationsverhalten eines Kreisels, die durch Bewegungen entstehen.

Haptische Wahrnehmung	Als haptische Wahrnehmung bezeichnet man das aktive Erfühlen von Konturen, Oberflächentextur, Gewicht, Größe usw. eines Objekts durch Integration aller Hautsinne und der Tiefensensibilität.
Immersion	Immersion ist ein Bewusstseinszustand, der im Kontext der virtuellen Realität das Eintauchen in eine künstliche Welt beschreibt. Hierbei erlebt der Benutzer auf Grund einer fesselnden und anspruchsvollen (künstlichen) Umgebung eine Verminderung der Wahrnehmung seiner eigenen Person.
Joints	Englisch für Verbindung, verbinden. Bezeichnen bei virtuellen Charakteren, die Gelenke in einem Skelett.
Kapazitive Kopplung	Kapazitive Kopplung bezeichnet die Übertragung von Energie von einem Schaltkreis zu einem anderen.
Latenz	Die Latenz ist ein Maß für die zeitliche Verzögerung in einem System. Synonyme für Latenz sind Reaktionszeit bzw. Verzögerungszeit.
Mesh	Ein "Mesh" ist in der 3D-Computergrafik das Polygonnetz eines 3D-Objekts.
Morphing	Beim Morphing werden zwischen zwei einzelnen Objekten Übergänge berechnet. Dabei wird das Äußere eines Objektes zu dem eines Anderen interpoliert.
Motion-Capturing	Unter Motion-Capture versteht man eine Technik, die es ermöglicht, Bewegungen so aufzuzeichnen und in ein von Computern lesbares Format umzuwandeln, dass diese Bewegungen zum einen analysieren und zum anderen auf im Computer generierte 3D-Modelle übertragen werden können.
Multi-Touch	Mit Multi-Touch wird die Fähigkeit eines berührungsempfindlichen Eingabegeräts bezeichnet, gleichzeitig mehrere Berührungen, in den meisten Fällen Finger, zu erkennen.
OSC	Das "Open Sound Control" ist ein nachrichtenbasiertes Kommunikationsprotokoll, welches hauptsächlich für die Echtzeitverarbeitung von Sound über das Netzwerk und Multimedia-Installationen verwendet wird. Steuersignale können von Hardware oder Software erzeugt und dann via OSC in Form von Nachrichten (OSC-Messages) an eine Schnittstelle weitergegeben werden und so eine Ausgabe steuern.
Open Source	"Open Source" (Offene Quelle) ist eine Palette von Lizenzen für Software, deren Quelltext öffentlich zugänglich ist. Durch die Lizenzen kann eine Weiterentwicklungen einer Software gefördert werden.

Parser	Ein Parser (to parse = "analysieren") ist ein Computerprogramm, dass in der Computertechnik für die Zerlegung und Umwandlung einer beliebigen Eingabe in ein für die Weiterverarbeitung brauchbares Format zuständig ist.
Pitch	"Pitch" heißt übersetzt Tonhöhe. In der Audiotechnik versteht man unter "pitchen" das verändern der Tonhöhe.
Potentiometer	Ein Potentiometer ist ein elektronisches Widerstandsbauelement, dessen Widerstandswerte mechanisch (durch Drehen oder Verschieben) veränderbar sind. Es hat mindestens drei Anschlüsse und wird vorwiegend als stetig einstellbarer Spannungsteiler eingesetzt.
Polygon	Polygon oder auch Vieleck ist ein Begriff aus der Geometrie. Ein Polygon erhält man, indem man mindestens drei voneinander verschiedene Punkte miteinander verbindet, sodass durch den entstandenen Linienzug eine zusammenhängende Fläche (Figur) entsteht. Auch diese so entstandene Fläche wird oft Polygon genannt. In der 3D-Computergrafik werden beliebige (auch gekrümmte) Oberflächen als Polygonnetz modelliert. Insbesondere Dreiecksnetze eignen sich besonders gut zur schnellen Darstellung von Oberflächen.
Rendering	Bildsynthese oder rendern (dt.: berechnen) bezeichnet in der Computergrafik die Erzeugung eines Bildes aus Rohdaten (auch Szene genannt). Eine Szene ist ein virtuelles räumliches Modell, das Objekte und deren Materialeigenschaften, Lichtquellen sowie die Position und Blickrichtung eines Betrachters definiert. Computerprogramme zum Rendern von Bildern werden Renderer genannt.
Rigid-Body	Ein "Rigid-Body" (Harter Körper) reagiert auf physikalische Kräfte wie Schwerkraft oder Wind in Dynamischen Animationen.
Rotoskopie	Rotoskopie ist ein bei der Herstellung von Animationsfilmen genutztes Verfahren zum Zeichnen der Bilderfolgen. Dabei werden (meist eigens aufgenommene) Filmszenen Einzelbild für Einzelbild von hinten so auf eine Mattglasscheibe projiziert, dass der Animator sie abzeichnen kann (wie beim Durchpausen). Das Verfahren wurde vor allem für realistisch gestaltete menschliche Charaktere eingesetzt, besonders dann, wenn sehr komplexe Bewegungen wie z. B. Tanzszenen gefordert waren.
Samplingrate	Samplingrate oder Abtastrate, ist in der Signalverarbeitung die Häufigkeit, mit der ein kontinuierliches Signal abgetastet und in ein zeitdiskretes Signal umgewandelt wird.

Szenengraph	Ein Szenengraph ist eine Datenstruktur, die häufig bei der Entwicklung computergrafischer Anwendungen eingesetzt wird. Es handelt sich um eine objektorientierte Datenstruktur, mit der die logische, in vielen Fällen auch die räumliche Anordnung der darzustellenden zwei- oder dreidimensionalen Szene beschrieben wird.
Telepräsenz	beschreibt den Zustand sich in einer entfernten Umgebung anwesend zu fühlen. Je höher der Grad der Immersion, desto mehr fühlt sich der Benutzer in der entfernten Umgebung.
Tracking	In dieser Arbeit steht "tracking" für die Erkennung und Verfolgung von Punkten. Beim Motion-Capturing sind es die Marker oder Sensoren. Bei der Multi-Touch-Erkennung die Touches.
TUIO	"Tangible User Interface Objects" kurz (TUIO) ist ein offenes Framework, welches ein Protokoll und eine API (Programmschnittstelle) für Multi-Touch-Oberflächen definiert.
UV-Layout	Ein "UV-Layout" ist die zweidimensionale Ausbreitung der Oberfläche einer Geometrie.
UV-Koordinaten	"UV-Koordinaten" sind Texturkoordinaten eines Objekts.
UV-Mapping	"UV-Mapping" ist die Generierung eines UV-Layouts auf Basis verschiedener Techniken.
Vertex	"Vertex" sind Eckpunkt einer Polygon-Geometrie.
Weighting	"Weighting" bezeichnet die manuelle Korrektur der Abhängigkeit zwischen Skelett und Geometrie.
X3D	Extensible 3D, kurz X3D, ist eine auf XML basierende Beschreibungssprache für 3D-Modelle, die in einem Webbrowser angezeigt werden können.
XML	Die "Extensible Markup Language" (erweiterbare Auszeichnungssprache), abgekürzt XML, ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdaten. Ein Grundgedanke hinter XML ist es, Daten und ihre Repräsentation zu trennen, um Daten beispielsweise einmal als Tabelle und einmal als Grafik auszugeben, aber für beide Arten der Auswertung die gleiche Datenbasis im XML-Format zu nutzen.
Zugriffs-modifikatoren	Zugriffsmodifikatoren sind Schlüsselwörter in Programmiersprachen, die den Grad definieren, in dem Teile eines Programms auf andere Teile eines Programms, wie beispielsweise Variablen, Funktionen oder Klassen, zugreifen können. Sie regeln die Sichtbarkeit dieser Teile innerhalb eines Programms.

Literaturverzeichnis

[AN07] Peter Haberäcker Alfred Nischwitz, Max Fischer. *Computergrafik und Bildverarbeitung*. Friedr. Vieweg & Sohn Verlag, Wiesbaden, 2007.

[Aut08] C't Author. *Windows 7 mit "Multitouch"-Oberfläche*. Heise Verlag, C't Ausgabe 12/2008 S.37, Hannover, 2008.

[Aut09] NUI Group Authors. *Multi-Touch Technologies*. NUI Group, 2009.

[Bad08] Patrick Bader. *Entwicklung eines virtuellen Eingabegeräts für Multitouch Screens*. Bachelor Thesis - Fraunhofer IAO / HdM Stuttgart, Stuttgart, 2008.

[Bic01] Daniela Bickler. *Zielgruppe Kinder - Handlungsspielräume eröffnen, Abhängigkeiten vermeiden.* Verantwortung in audiovisuellen Medien. Heft 17 - Nomos Verlagsgesellschaft, Baden-Baden, 2001.

[Bir09] Jeremy Birn. *Lighting & Rendering: 3D-grafiken meisterhaft beleuchten- realistische Texturen entwickeln*. Pearson Education, München, 2009.

[CH09] Franz Koller Christine Höflacher, Michael Burmester. *Touch it, move it, scale it - Multitouch*. Studie - Hochschule der Medien Stuttgart und User Interface Design GmbH (UID), Stuttgart, 2009.

[Dac04] Raimund Dachselt. *Eine deklarative Komponentenarchitektur und Interaktionsbausteine für dreidimensionale multimediale Anwendungen*. Der Andere Verlag, Tönning, 2004.

[DJ06] Stephan Neunreither und Friedrich Wagner Dietmar Jackèl. *Methoden der Computeranimation*. Springer Berlin Heidelberg, Heidelberg, 2006.

[Ebn09] Dominik Ebner. *Avatare- Voraussetzungen für den erfolgreichen Einsatz virtueller Berater*. GRIN Verlag, München, 2009.

[Fle99] Bill Fleming. *3D Modeling & Surfacing*. Morgan Kaufmann, San Francisco, California, 1999.

[Fri10] Stefan Friedrich. *Neue Technologien in der Logistik - Identifikation und Analyse potentieller Einsatzfelder der Multi-touch-technologie*. GRIN Verlag, München, 2010.

[Fug04] Torben Fugger. *3D-Game-Engine*. Universität Koblenz Landau, Koblenz/Landau, 2004.

[Gru08] Stephan Grunwald. *Sound als Feedbackmöglichkeit in Computerspielen: Eine Analyse am Beispiel von Echtzeit-Strategie und First-Person-Shooter*. GRIN Verlag, München, 2008.

[Gui05] Marc-André Guindon. *The Modeling & Animation Handbook*. John Wilwy & Sons, Inc., Hoboken, New Jersey, 2005.

[Hei04] Andreas M. Heinecke. *Mensch-Computer-Interaktion*. Carl Hanser Verlag, München, Wien, 2004.

[Hel09] Sabine Heller. *Charakter-animation in Film und Fernsehen: Analyse und Entwicklung von zwei- und dreidimensionalen Charakteren*. GRIN Verlag, München, 2009.

[Hor02] Eduard Horber. *Motion Capturing*. Forschungsbericht - Universität Ulm, Ulm, 2002.

[Ing08] Michael Ingrassia. *Maya for Games: Modeling and Texturing Techniques with Maya and Mudbox*. Focal Press, Burlington, USA, 2008.

[JFP06] Andrzej Skowron James F. Peters. *Transactions on rough sets V*. Springer, Heidelberg, 2006.

[JG02] Elisabeth André Justine Cassell Eric Petajan Norman Badler Jonathan Gratch, Jeff Rickel. *Creating interactive virtual humans: Some assembly required*. IEEE Computer Society, Intelligent Systems Juli/August 2002 S. 54-63, Washington, 2002.

[Kaj07] Kai Kajus. *MotionCapturing Leitfaden*. Hochschule Mannheim, Mannheim, 2007.

[Ker04] Isaac V. Kerlow. *The Art Of 3D Computer Animation And Effects*. John Wilwy & Sons, Inc., Hoboken, New Jersey, 2004.

[Kna06] Björn Knafla. *Wie kommt der Drache ins Computerspiel?* Universität Kassel, Kassel, 2006.

[Kra01] Bernd Kracke. *Crossmedia-Strategien: Dialog über alle Medien*. Gabler Verlag, Wiesbaden, 2001.

[Kra07] Robert Krause. *Seminar Game Development Engines and Middleware*. Universität der Bundeswehr München, München, 2007.

[Lov07] Jörn Loviscach. *Siggraph 2007: Computergrafik und Interaktion*. Heise Verlag, C't Ausgabe 18/2007 S. 40, Hannover, 2007.

[Mah08] Keywan Mahintorabi. *Maya 2008 3D-grafik und 3D-animation*. Hüthig Jehle Rehm, Heidelberg, 2008.

[MB05] Manfred Brill Michael Bender. *Computergrafik: Ein anwendungsorientiertes Lehrbuch*. Hanser Verlag, München, 2005.

[Men00] Alberto Menache. *Understanding motion capture for computer animation and video games*. Morgan Kaufmann, 2000.

[Mul08] L.Y.L. Muller. *Multi-touch displays: design, applications and performance evaluation*. Master's Thesis - Universiteit van Amsterdam, Amsterdam, 2008.

[Oli07] Gary Oliverio. *Maya 8 Character Modeling*. Wordware Publishing, Inc., Plano, Texas, 2007.

[Ric08] Sebastian Richter. *Digitaler Realismus: Zwischen Computeranimation und live-action. Die neue Bildästhetik in Spielfilmen*. transcript Verlag, Bielefeld, 2008.

[Sch04] David Scherfgen. *3D-Spiele-programmierung mit DirectX 9 und C++*. Carl Hanser Verlag, München, Wien, 2004.

[Sch06] Maximilian Schönherr. *Maya 7*. Pearson Education, München, 2006.

[Ste04] Daniel Steger. *Motion Capture mit optisch-magnetischem Trackingsystemen in VR-Applikationen*. Technische Universität Chemnitz, Chemnitz, 2004.

[Ste08] Lennart Steinke. *Spieleprogrammierung - Das bhv Taschenbuch*. Hüthig Jehle Rehm, Heidelberg, 2008.

[TAM08] Naty Hoffman Tomas Akenine-Möller, Eric Haines. *Real-Time Rendering*. A.K. Peters Ltd., Natick, MA, USA, 2008.

[TE09] Norbert Neuß Wilfried Teusch Arne Busse Tilman Ernst, Stefan Aufenanger. *Über Medien reden*. Bundeszentrale für politische Bildung, Bon, 2009.

[TF07] Simon Beckmann Thomas Fiedler. *Einblick in Motion-Capture*. Technisch Fachhochschule Berlin, Berlin, 2007.

[Tin09] Martin Tintel. *Character Design- Die Methoden und Mechanismen hinter dem Charakter Design*. GRIN Verlag, München, 2009.

[Trü07] Jonas Trümper. *Multi-Touch-Systeme und interaktive Oberflächen*. Forschungsbericht - TU Berlin, Berlin, 2007.

[UK08] Peter König Ulrike Kuhlmann. *Finger-fertig? Multitouch: Wunsch und Wirklichkeit*. Heise Verlag, C't Ausgabe 14/2008 S.150-155, Hannover, 2008.

[UL09] Jasmin Link Antonino Ardilio Andreas Schuller Janina Bierkandt Uwe Laufs, Micha Block. *Studie Multi-Touch Technologie, Hard-/Software und deren Anwendungsszenarien*. Fraunhofer IAO, Stuttgart, 2009.

[Vit08] Arnd Vitzthum. *Entwicklungsunterstützung für interaktive 3D-Anwendungen - Ein modellgetriebener Ansatz*. Dissertation - LMU München, München, 2008.

[Wal05] Doug Walker. *Learning Maya 7: The Modeling & Animation Handbook: The Modeling and Animation Handbook*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2005.

[Wei04] Christian Weisel. *Entwicklung eines markerbasierten Motion Capturing-Systems für Echtzeitanwendungen*. Fachhochschule Gießen-Friedberg, Friedberg, 2004.

[Wir04] Thomas Wirth. *Missing Links*. Hanser Verlag, München, 2004.

[YJ07] Patrick Dähne Johannes Behr Yvonne Jung, Tobias Franke. *Enhancing X3D for Advanced MR Appliances*. International Conference on 3D Web Technology (WEB3D) 12/2007, Perugia, Italy, 2007.

[YJ09] Johannes Behr Yvonne Jung. *Simplifying the Integration of Virtual Humans into Dialog-like VR Systems*. Software Engineering and Architectures for Realtime Interactive Systems (SEARIS) 2/2009, Lafayette, LA, USA, 2009.