UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Objektbasierter Ansatz für die Animation von Charakteren mittels Inverser Kinematik

Diplomarbeit

Zur Erlangung des Grades einer Diplom-Informatikerin
im Studiengang Computervisualistik

vorgelegt von
Michelle Kristin Martin

Erstgutachter:	Prof. Dr. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Dipl.-Inform. Stefan Rilling (Institut für Computervisualistik, AG Computergraphik)

Koblenz, im Oktober 2007

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

UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Aufgabenstellung für die Diplomarbeit

von Michelle Kristin Martin

(Matr.-Nr. 203110096)

Thema: Objektbasierter Ansatz für die Animation von Charakteren mittels Inverser Kinematik

In interaktiven virtuellen Umgebungen, sei es ein Spiel oder eine Simulation, in denen 3D Charaktere und Objekte vorkommen, findet in der Regel auch durch den Nutzer ausgelöste Interaktion zwischen den Charakteren und den Objekten statt. Die Visualisierung dieser Interaktionen ist durch einfache Forward-Animation nicht zufriedenstellend lösbar, da eine Vielzahl an vorab erzeugten Animationen notwendig ist, um für jedes Objekt eine angepasste Animation bereit zu stellen. Darüber hinaus ist die korrekte Positionierung von Charakter und Objekt zueinander notwendig, damit diese Animationen präzise und realistisch wirken. In dynamischen Umgebungen, wo die Position der Objekte variabel ist, oder Objekte zur Szene hinzugefügt werden können, ist die Visualisierung nur durch Standard-Animationen lösbar, z.B. das ungerichtete Ausstrecken eines Armes, was unpräzise ist und wenig natürlich wirkt.

Die Inverse Kinematik bietet die Möglichkeit, skelettbasierte Charaktere unabhängig von vorab definierten Animationen zu animieren. Mit dieser Technik ist es möglich, Interaktionen zwischen Charakteren und Objekten realistisch zu visualisieren. Indem die zur Steuerung der Animation notwendigen Daten bei dem interaktiven Objekt gespeichert werden, kann ein solches System auch in dynamischen Umgebungen problemlos funktionieren.

Ziel dieser Diplomarbeit ist die Entwicklung und Implementierung eines objektbasierten Ansatzes für die Animation von Charakter-Objekt-Interaktionen in dynamischen Umgebungen und die Entwicklung eines geeigneten IK-Algorithmus. Es wird auf einer bestehenden Animationsbibliothek aufgebaut, die bereits skelettbasierte Keyframe Animation beherrscht. Bei der Umsetzung soll darauf Wert gelegt werden, dass zum einen die entstehenden Bewegungsabläufe möglichst natürlich wirken. Zum anderen soll der designtechnische Aufwand für die Anwendung des Systems minimiert werden, indem z.B. ein geeignetes Tool für die Festlegung der Objektparameter implementiert wird. Als Programmiersprache kommt C++ zum Einsatz und als Animationsbibliothek Cal3D.

Schwerpunkte dieser Arbeit sind:

1. Einarbeitung in die Materie
2. Entwurf des objektbasierten Animationssystems
3. Implementation des Systems
4. Bewertung und Dokumentation der Ergebnisse

Koblenz, den 23.April 2007

– Prof. Dr. Stefan Müller –

Erklärung

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

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

Ort, Datum Unterschrift

Objektbasierter Ansatz für die Animation von Charakteren mittels Inverser Kinematik

Inhaltsverzeichnis

1. Motivation zu dieser Arbeit....1
1.1. Der Trend zum Realismus....1
1.2. Die Problematik realistischer Animationen....4
1.3. Ziel der Arbeit....9

2. Inverse Kinematik....10
2.1. Was ist Inverse Kinematik....10
2.2. Inverse Kinematik in Spielen....13
2.3. Inverse Kinematik für den menschlichen Arm....14

3. Objektbasierter Ansatz....17
3.1. Grundidee....17
3.2. Definition der Ziele....19

4. Anforderungserhebung....20
4.1. Durchführung und Ergebnisse....21
4.2. Primitive....22
4.3. Griffposen....23
4.4. Interaktionen....25

5. Praktische Umsetzung....27
5.1. Aufbau des Frameworks....27
5.2. Skelettidentifizierung....28
5.3. Die fünf Primitive....32
5.3.1. Säulen-Primitiv....34
5.3.2. Kugel-Primitiv....36
5.3.3. Box-Primitiv....36
5.3.4. Kreditkarten-Primitiv....39
5.3.5. Knopf-Primitiv....39
5.4. Funktionsumfang und deren Umsetzung....40
5.4.1. Die Aktionen....42
5.5. Griffposen....45
5.6. IK Berechnungen....46
5.7. Animation....49
5.8. Sonderfälle....51
5.8.1. Bewegte Charaktere und Objekte....52
5.8.2. Entfernung des Objektes ist zu groß....52
5.8.3. mehrfache Unterarme....53

6. Ergebnisse und Bewertung....54

7. Fazit und Ausblick....57

Literaturverzeichnis....59

Verzeichnis der Code Listings

Listing 4.1: Die beliebtesten Computerspiele.....20
Listing 4.2: Die häufigsten Interaktionen mit Objekten.....21
Listing 4.3: Die Objekte mit den meisten Interaktionen.....21
Listing 4.4: Die 4 Teil-Interaktionen.....25
Listing 5.1: Die virtuellen Funktionen in den Basisklassen IKCharBase und IKObjBase.....28
Listing 5.2: Unterschiede zwischen verschiedenen Skelettmodellen.....29
Listing 5.3: Gemeinsamkeiten zwischen verschiedenen Skelettmodellen.....29
Listing 5.4: Funktionskopf der Funktion NewInteraction().....40
Listing 5.5: Kosinussatz und Berechnung des Ellenbogenwinkels γ 47
Listing 5.6: Funktionskopf der Funktion SetBone().....51

Abbildungsverzeichnis

Abbildung 1: Lara Croft (©Eidos Interactive)....2
Abbildung 3: Screenshot Crysis, Cry-Engine 2 (©2007 Crytek)....3
Abbildung 2: Tech Demo Cry-Engine 2 (©Crytek)....3
Abbildung 4: Simon the Sorcerer - Chaos ist das halbe Leben (©RTL Enterprises)....5
Abbildung 5: Tomb Raider Anniversary (©Eidos Interactive)....5
Abbildung 6: BioShock (©Take 2 Interactive)....6
Abbildung 7: Obscure 2 (©Playlogic Entertainment, Inc)....6
Abbildung 8: Simon the Sorcerer - Chaos ist das halbe Leben (©RTL Enterprises)....7
Abbildung 9: Sam & Max - Season 1 - Reality 2.0 (©JoWood)....7
Abbildung 10: Jack Keane (©10tancle Studios)....8
Abbildung 11: Zelda - Twilight Princess (©Nintendo)....8
Abbildung 12: Berechnung für eine Knochenkette mit zwei Gliedern....10
Abbildung 13: mehrere Lösungsmöglichkeiten....11
Abbildung 14: Schritt 1 im Cyclic Coordinate Descent....11
Abbildung 15: Schritt 2 im Cyclic Coordinate Descent....12
Abbildung 16: drei verschiedene Iks....12
Abbildung 17: Charakter an einem Hang, mit und ohne IK (Konoko Payne ©Pierre Terdiman)....13
Abbildung 18: Bewegungsumfang des Schultergelenks....14
Abbildung 19: Bewegungsumfang des Unterarms....14
Abbildung 20: Verschiedene Armstellungen....15
Abbildung 21: Bewegungsumfang des Handgelenks....16
Abbildung 22: Grundkonzept des Editors....18
Abbildung 23: die 5 Primitive....22
Abbildung 24: Beispiel-Handstellung am Säulen-Primitiv....23
Abbildung 25: Ausschnitt aus der Fotoserie zur Ermittlung der Griffposen....24
Abbildung 26: Die fünf Griffe der fluidischen Hand (Forschungszentrum Karlsruhe)....24
Abbildung 27: Kommunikation zwischen Framework und Hauptprogramm....27
Abbildung 28: Darstellung eines Skelettmodells....30
Abbildung 29: Körpermaße des Menschen nach DIN 33402....31
Abbildung 30: Handrotation ohne und mit Berücksichtigung der zusätzlichen Unterarme....31
Abbildung 31: UML-Schema der Primitiv-Klassen....32
Abbildung 32: Die Handachsen....33
Abbildung 33: gültige Armstellungen....33
Abbildung 34: Korrektur des Berührungspunktes....34
Abbildung 35: Berechnung für das Säulen-Primitiv....34
Abbildung 36: Ausrichtung der Daumen in Richtung der Klinge....35
Abbildung 37: Berücksichtigung der Normale beim Kugel-Primitiv....36
Abbildung 38: das Box-Primitiv und die Spannweite der Hand....37
Abbildung 39: das Box Primitiv auf einem Buch....37
Abbildung 40: zweihändiges Tragen einer Kiste....38
Abbildung 41: Ausrichtung des Kreditkarten-Primitives....39
Abbildung 42: das Knopf-Primitiv....39
Abbildung 43: Umgreifen am Lenkrad mit eingeblendeten Primitiven....41
Abbildung 44: Kombinationsmöglichkeiten mit Forward-Animationen....43
Abbildung 45: eine Teekanne gerade halten....44
Abbildung 46: das Arm-Dreieck....46
Abbildung 47: Rotation des Oberarms um sich selbst....47

Abbildung 48: steife Armhaltung....48
Abbildung 49: Heranziehen des Armes an den Körper....49
Abbildung 50: Animationsverlauf für Kopf, Schultern, Arm, Hand und Finger....50
Abbildung 51: Fehler durch Nichtberücksichtigung der zusätzlichen Unterarme....53
Abbildung 52: Übergabe eines Buches unter zwei Charakteren....54
Abbildung 53: Erstellung eines Box-Primitives....55
Abbildung 54: Genauigkeit der Ergebnisse....55
Abbildung 55: Aufnehmen einer Kiste aus einem Kofferraum....56
Abbildung 56: interaktive Szene in einem Auto....56

1. Motivation zu dieser Arbeit

Die Computerspiele-Industrie wächst seit Jahren ständig und unaufhaltsam. Beleg dafür sind eine wachsende Zahl Spieler und die stärkere Verbreitung von spezialisierter Spiele-Hardware, sei es für den PC oder in Spiele-Konsolen. Da mittlerweile auch Frauen und Erwachsene mittleren Alters als Zielgruppe erkannt und erschlossen wurden, ist der Markt gigantisch. Im Jahre 2006 wurden in Amerika 7,4 Milliarden US-Dollar für Konsolen- und PC-Spiele ausgegeben (Quelle: ESA[1]). Dabei ist der Anschaffungspreis für die Hardware oder etwaiges Zubehör nicht eingerechnet.

Als Folge des Wachstums gibt es auch eine Vielzahl an Spiele-Entwicklern, die daran mitverdienen möchten. Moderne Computerspiele können längst nicht mehr auf Hinterhöfen von einem kleinen Team aus hochmotivierten Idealisten produziert werden. Unter den Entwicklern, zumindest innerhalb eines Genres, herrscht entsprechend große Konkurrenz.

Um sich auf dem Markt gegen diese durchzusetzen, greifen die Studios zu der Entwicklung neuer Technologien, um sich von der Masse abzusetzen. Manche entwickeln neue Eingabegeräte, wie Nintendo mit dem „Nunchuk“ für die „Wii“ Konsole[1], andere setzen auf die Entwicklung neuer Spielkonzepte, wie z.B. Will Wrights Spore[2].

Der Bereich, in dem sich diesbezüglich in den letzten 15 Jahren wohl am meisten getan hat, ist die Computergrafik. Der Konkurrenzkampf der Entwicklerstudios ist der treibende Motor für die Entwicklung immer neuer Visualisierungstechnologien. Die Hardware Industrie hat auf diesen Trend reagiert und die großen Hersteller bringen mittlerweile beinahe halbjährlich neue Grafikkarten heraus, um mit den Hardware-Ansprüchen der neuesten Spiele fertig zu werden.

Um im Konkurrenzkampf gegen andere Entwicklerstudios nicht unterzugehen, gewinnen gerade in den letzten Jahren auch andere Technologien als die grafische Darstellung an Bedeutung. Ein immer wichtigerer Faktor in heutigen Videospielen ist beispielsweise eine realistisch wirkende Physiksimulation. Ein anderes, bisher sich weniger dynamisch entwickelndes Verkaufsargument sind glaubhafte Animationen. Einen Teilaspekt dieses Bereiches, nämlich die Interaktion zwischen den Händen einer Spielfigur und der virtuellen Welt, möchte ich in dieser Arbeit behandeln.

1.1. Der Trend zum Realismus

Neben dem Ziel sich von der Konkurrenz abzuheben hat das Streben nach überzeugender Grafik in Spielen auch noch einen weiteren Zweck. Immer realistischere Grafik soll die Immersion vertiefen. Immersion bedeutet das 'Eintauchen' in eine virtuelle Realität bis hin zu dem Grad der vollständigen Identifikation mit der eigenen Spielfigur. Je tiefer die Immersion, desto intensiver ist das Spielerlebnis. Ego-Shooter bauen sehr auf die Immersion, indem sie das Geschehen permanent aus

1 Wii – eine Ende 2006 auf dem Markt gebrachte Spielkonsole für den Fernseher. Ihr Controller, der Nunchuk, verfügt über Bewegungssensoren. So lassen sich die Spielfiguren über Arm- und Handbewegungen steuern.

2 Spore – ein Spiel, das mehrere Genres in sich vereint. Es geht darum, ein Lebewesen aus einem Einzeller mutieren zu lassen und durch verschiedene Stadien der Evolution zu begleiten, bis am Ende eine Zivilisation entsteht, die ins Weltall fliegen kann. Während der verschiedenen Entwicklungs-Phasen ändert sich die Spielweise immer wieder.

der Sicht des Spielers darstellen. Eingeblendete Informationen wie z.B. eine Lebensenergieanzeige, Munition oder eine Übersichtskarte werden in vielen neueren Spielen dieser Gattung auf ein Minimum reduziert, um das Eintauchen nicht zu behindern.

Ein starkes Mittel für die Immersion ist ein hoher Grad an visueller Glaubwürdigkeit. Daraus ist in einigen Spiele-Genres ein regelrechter Wahn nach realistischer Grafik entstanden. Dabei wird jeder Aspekt des Spieles berücksichtigt. Die Beleuchtung wird mit aufwändigen und komplexen Verfahren berechnet, weiche Schatten gehören längst zum Standard. Um dabei dennoch ressourcenschonend zu arbeiten, werden immer neue Tricks, wie vorberechnetem Licht in Form von Lightmaps, erfunden. Schnelle Bewegungen verursachen unter Umständen Bewegungsunschärfe, zusätzlich zur bereits vorhandenen Tiefenunschärfe. Auch die Texturen werden dank des größeren Speichers neuer Grafikkarten immer detailreicher. Dies trägt deutlich zur visuellen Qualität bei, erfordert aber auch eine Menge Aufwand bei der Erstellung, sowie bei der Verwaltung in Form von geschicktem Speichermanagement.

Abbildung 1: Lara Croft (©Eidos Interactive)

Entwicklung des Charakters über 8 Spiele hinweg von 1994 bis 2007
Die Polygonzahl des Charakters steigt, die Beleuchtung wird komplexer und die Texturen auf den Oberflächen im Hintergrund detailreicher.

Feuer und Rauch müssen mittlerweile volumetrisch sein und selbst Wolken sind heute echte Objekte und keine zweidimensionalen Himmels-Texturen mehr. Das Wetter wird aufwändig simuliert, vom Regen bis zu malerischen Sonnenuntergängen. Alle Objekte in einem Spiel werden von einer Physik-Engine kontrolliert, um ein realistisches Verhalten zu simulieren. Die Künstliche Intelligenz soll sich ebenso natürlich und menschlich verhalten können, wie der Spieler, im Team zusammenarbeiten und für Abwechslung im Spiel sorgen.

Abbildung 2: Screenshot Crysis, Cry-Engine 2 (©2007 Crytek)

Dreidimensionale Wellen, volumetrische Wolken und aufwändige Vegetation.

Die 3D Modelle werden unter Zuhilfenahme medizinischer Atlanten entworfen und mit Shadern nachbearbeitet und mittels Normal- und Displacement-Mappings wird ihnen eine zusätzliche Oberflächenstruktur verliehen. Die Modelle selbst bestehen aus immer mehr Polygonen. Selbst die unwichtigsten Massen-Gegner sollen sich noch sichtbar unterscheiden. Die Haut der Charaktere bekommt durch prozedurale Shader Poren und Unebenheiten verliehen. Und schon seit langem sind alle Charaktere skelettbasiert, um eine bessere Animation zu ermöglichen. Alle neu entwickelten Technologien müssen dabei selbstverständlich Echtzeit-fähig sein.

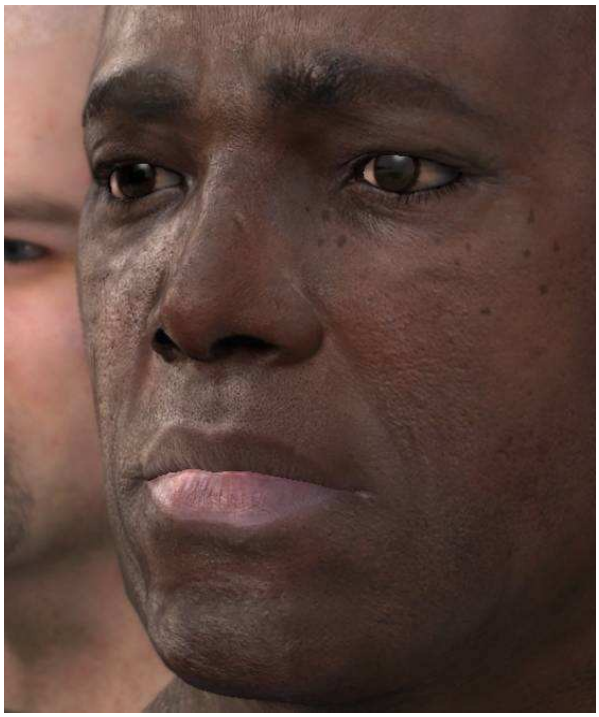

Abbildung 3: Tech Demo Cry-Engine 2 (©Crytek)

Die Haut bekommt mittels prozeduraler Shader Oberflächenstruktur, wie Poren und Unreinheiten.

1.2. Die Problematik realistischer Animationen

Auch bei den Animationen macht der Trend zum Realismus keinen Halt. An der Spitze der Technologie stehen beispielsweise Verfahren um Muskel zu simulieren, indem sie die Haut über den Muskeln aufbäumt, wenn diese sich zusammenziehen. Und die Gesichtsanimation ist und wird in den kommenden Jahren noch ein großes Thema sein. Der größte Schritt Richtung Realismus wurde durch skelettbasierte Charaktere und das Motion Capture Verfahren vollzogen.

Ein Charaktermodell auf einem Skelett aufzubauen eröffnet viele Möglichkeiten für die Animation. Die Grundidee ist dabei, dass alle Oberflächenpunkte des Modells nicht mehr einzeln animiert werden müssen, sondern nur noch das zugrunde liegende Skelett. Da dieses nur aus Linien und Gelenken besteht, ist dies wesentlich leichter und schneller zu bewältigen. Zudem lassen sich einmal erstellte Animationen so problemlos auf andere Charaktere übertragen. Die einzelnen Eckpunkte des 3D Modells berechnen ihre Position anhand der Orientierung der Knochen. Dabei kann ein Punkt von mehr als einem Knochen beeinflusst werden. Die Summe aller Einflüsse der Knochen ist 1.

Beim Motion Capture werden die Bewegungen von Schauspielern aufgenommen und auf ein solches Skelett übertragen. Die daraus entstehenden Animationen sind sehr realistisch und wirken absolut natürlich. Als Alternative zum Motion Capture können die Knochen mittels Keyframe Animation von Hand animiert werden. Dieses Verfahren erreicht jedoch wegen der Komplexität menschlicher Bewegungen niemals die gleiche Natürlichkeit wie das Motion Capture Verfahren. Bei beiden Verfahren werden Position und Rotation der Knochen zu festgelegten Zeitpunkten in den so genannten Keyframes gespeichert und zwischen diesen wird beim Abspielen der Rotation und Transformation interpoliert.

Bei solchen Animationen sind jedoch die Grenzen schnell erreicht, da nur zuvor erstellte Animationen wiedergegeben werden können. Zwar lassen sich mehrere Animationen auch miteinander mischen, so dass aus einer Gehen-Animation und einer Winken-Animation die Bewegung eines Charakters, der beim Gehen winkt, entsteht. Allerdings führt das Mischen von Motion Capture Animationen in der Praxis oftmals zu hässlichen Artefakten, wie z.B. wackelnde Gliedmaßen. Dies liegt an der Komplexität der einzelnen Bewegungen, die gerade beim Motion Capture Verfahren oft viele, atomar zu vernachlässigende Abweichungen enthalten.

Daraus ergibt sich, dass es alleine mit der Keyframe Animation keine Möglichkeit gibt, den Spieler dynamisch auf seine Umwelt reagieren zu lassen. Dies wird insbesondere in der Interaktion mit Objekten oder anderen Charakteren deutlich. Die Interaktion wird in der Regel vom Nutzer ausgelöst. Visualisiert wird der Vorgang, indem eine zuvor festgelegte Animation abgespielt wird, die z.B. den Charakter den Arm ausstrecken lässt, wenn ein Gegenstand aufgehoben werden soll. Diese Animation wird jedoch von der Position des Charakters und des aufzunehmenden Objektes nicht beeinflusst. Deshalb bietet sie nur eine unbefriedigende Lösung. Nur wenn sich der Charakter und das Objekt in exakt der gleichen Position zueinander befinden, wie es bei der Aufnahme der Animation vorgesehen wurde, ist es möglich, den Vorgang glaubhaft und präzise darzustellen. Selbst wenn beides kontrolliert werden kann, so ist immer noch eine Vielzahl an Animationen notwendig, um eine angepasste Animation für jeden Gegenstand und jede Situation im Spiel zu berechnen.

Gänzlich unlösbar ist das Problem in dynamischen Umgebungen, etwa dann, wenn der Charakter vom Spieler steuerbar ist und die meisten Objekte durch die Physik-Engine kontrolliert werden. Andere Szenarien in denen solche Animationen nicht ausreichen, sind von Spielern generierte Inhalte, wie sie in die Sims 2 oder Second Life, oder auch in Mods[3] vorkommen.

3 Mod – Modification; Viele Spiele-Entwickler legen ihren Spielen Editoren bei, damit Spieler ihre eigenen Inhalte schaffen können. Der Umfang kann sich von eigenen Level, die erstellt werden können, bis hin zu der Möglichkeit erstrecken, komplett eigene Spiele mit der Spiele-Engine zu entwickeln. Dabei können die originalen Spielinhalte

Dieses Problem taucht in alten und in neuen Spielen gleichermaßen auf und führt immer wieder zu einem Bruch der Immersion. Die Spiele-Entwickler finden dabei ganz unterschiedliche Lösungen und Tricks, um die teilweise sehr unzulänglichen Animationen zu verschleiern. Meist wird die Interaktion durch eine Standard-Animation dargestellt, wie z.B. das Ausstrecken eines Armes. Soll ein Charakter einem anderen etwas geben, so greifen beide, während sie ihre eigenen Animationen abspielen, in der Regel aneinander vorbei. Da die Finger meist überhaupt nicht animiert sind, werden die Gegenstände oft gar nicht dargestellt, oder sie schweben über der Hand. Diese grafische Inkonsistenz wird mangels einer besseren Alternative hingenommen.

Abbildung 4: Simon the Sorcerer - Chaos ist das halbe Leben (©RTL Enterprises)

Simon gibt einen Gegenstand an Rotkäppchen. Der Arm wird ungerichtet nach vorne ausgestreckt, das Objekt wird nicht dargestellt und Rotkäppchen streckt nicht einmal ihre Hand aus.

Darüber hinaus werden Gegenstände oft auch immer gleich platziert, z.B. entweder auf dem Boden, oder auf einer vorher festgelegten Tischhöhe, so dass mit zwei verschiedenen Animationen beide Bereiche optisch hinreichend abgedeckt werden können. Im Spiel Tomb Raider (alle Teile) liegen Gegenstände z.B. ausschließlich auf dem Boden. Manche Spiele machen gar keine Animation und lassen den Gegenstand einfach nur geschickt ins Inventar der Spielfigur wandern.

Abbildung 5: Tomb Raider Anniversary (©Eidos Interactive)

Alle Gegenstände befinden sich immer auf dem Boden. Zum Aufnehmen wird eine Animation abgespielt, bei der Lara sich runterbeugt. Danach wird das Objekt einfach ausgeblendet.

wie Figuren und Objekte in der Regel weiterverwendet und durch eigene ergänzt werden.

Dies zieht teilweise unerwünschte Einschränkungen im Leveldesign nach sich. So muss der Untergrund in der direkten Umgebung von Gegenständen möglichst eben sein, damit bei der Aufnahmeanimation Arme und Beine des Charakters nicht durch die Geometrie greifen. Außerdem geschieht die Positionierung von Gegenständen sehr gleichförmig und unrealistisch, wenn sie nur auf dem Boden oder auf Oberflächen wie z.B. Tischen mit einheitlicher Höhe platziert werden dürfen. Ein Charakter kann z.B. nicht ein Gewehr aus einer Halterung an der Wand nehmen, da die Standardanimation für einen solchen Fall ungeeignet ist. Soll für eine besondere Szene dennoch das Objekt gesondert platziert werden, muss eine Animation extra dafür erstellt werden. Dann jedoch darf das Objekt nachträglich nicht mehr in seiner Position verändert werden.

Abbildung 6: BioShock (©Take 2 Interactive)

Hier wird ein Schraubenschlüssel aufgehoben. Während der Schlüssel auf dem Boden langsam durchsichtig wird, fährt die Hand mit einem weiteren Schlüssel in der Hand von unten ins Bild. Für eine kurze Zeit sind zwei Schraubenschlüssel im Bild zu sehen.

Abbildung 7: Obscure 2 (©Playlogic Entertainment, Inc)

In der Standard-Animation greift der Charakter mit der linken Hand nach unten (obere Bildreihe). Nach der Hälfte der Animation springt der aufgenommene Baseball-Schläger aber in die rechte Hand (untere Bildreihe).

In klassischen Abenteuerspielen mit starrer Kameraeinstellung wäre es theoretisch möglich die Position von Charakter zum Objekt zu kontrollieren. Dennoch ist der Aufwand des Erstellens einer eigenen Animation für jeden Gegenstand unverhältnismäßig. Daher wird auch hier häufig eine Standardanimation eingesetzt. Dies führt teilweise zu skurrilen Bildern, wenn etwa ein Charakter nach unten greift, aber einen Gegenstand vom obersten Regal nimmt. Viele Spiele versuchen solche Situationen so oft wie möglich zu verstecken, indem sie den Charakter z.B. vor den zu nehmenden Gegenstand stellen und ihn so verdecken, oder die Kamera geschickt während der Aktion in eine andere Richtung schwenken lassen.

Abbildung 8: Simon the Sorcerer - Chaos ist das halbe Leben (©RTL Enterprises)

Wenn Simon die Karnickel-Falle aufbauen soll, stellt er sich immer so zur Kamera, daß er die Falle verdeckt. Diese taucht aus dem Nichts auf dem Boden auf, nachdem Simon sich runter gebeugt hat.

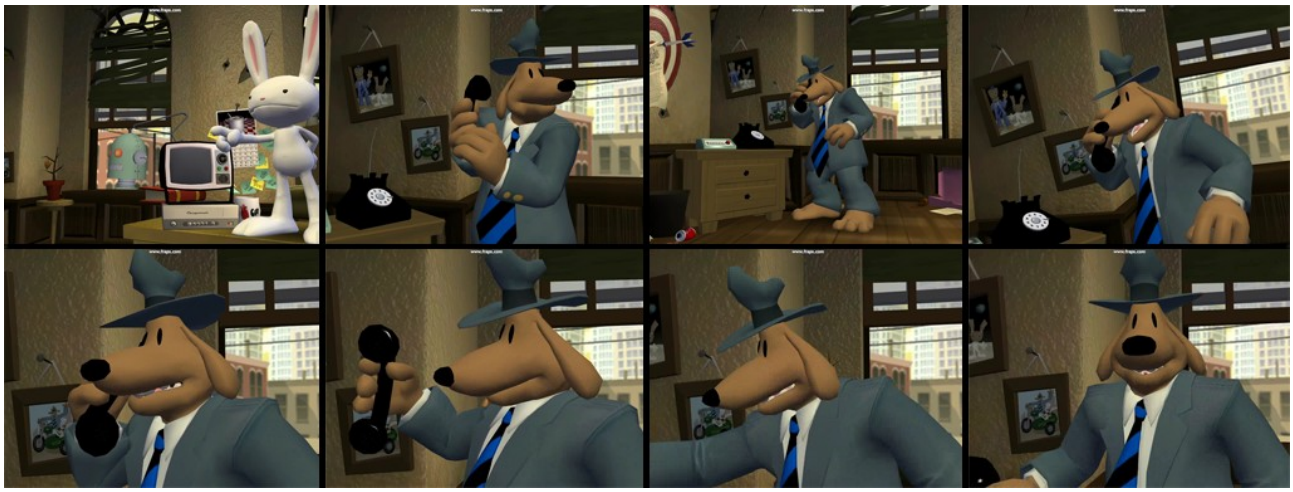

Abbildung 9: Sam & Max - Season 1 - Reality 2.0 (©JoWood)

Elegant gelöst durch Kameratricks: Wenn die Kamera von Max zu Sam wechselt, hält dieser den Telefonhörer bereits in der Hand. Die Kamera zoomt dann während des Telefonats immer weiter heran. Wenn Sam den Hörer auflegt, greift seine Hand wie zufällig aus dem Bild heraus.

Um realistischer zu wirken, bliebe daher nur der Weg, für jede Kombinationsmöglichkeit eine eigene Animation zu erstellen. Dieser Arbeitsaufwand und die damit verbundenen Kosten wären allerdings extrem hoch, zumal gerade in Adventures überproportional viel Interaktion mit Objekten stattfindet. Für Interaktionen mit Gegenständen über das Aufnehmen hinaus wird daher in der Regel ebenfalls eine Standardanimation abgespielt. Dies ist nicht selten die gleiche Animation, wie für das Aufheben oder Benutzen eines Objektes.

Abbildung 10: Jack Keane (©10tancle Studios)

In der oberen Bilderreihe hebt Jack eine Flasche vom Boden auf. Sie wird zwar zu seiner rechten Hand positioniert, jedoch völlig falsch. In der unteren Bildreihe 'benutzt' Jack eine Stange mit einem Metallgerüst. Es wird die gleiche Animation abgespielt, die jedoch in diesem Fall noch weniger zur gezeigten Aktion passt. Die Stange erscheint darüber hinaus im Gerüst, bevor Jack anfängt, sich zu bewegen.

In Spielen mit variabler Kamera wird das Positionierungsproblem häufig recht einfach gelöst: Ist die Animation passend für eine bestimmte Interaktion mit einem Objekt erstellt worden, wie etwa das Ziehen eines Hebels oder das Öffnen einer Kiste, so wird der Charakter kurzerhand vom Spiel an die richtige Position gesetzt, damit die Animation korrekt aussieht. Um diese kleine Schummelei gut zu verstecken, wird die Kamera dabei für gewöhnlich starr gehalten und die Aktion lässt sich erst auslösen, wenn der Charakter der eigentlichen Zielposition ohnehin schon sehr nahe ist, so dass der unvermeidliche Sprung zur Zielposition nicht zu groß ausfällt.

Abbildung 11: Zelda - Twilight Princess (©Nintendo)

Will man die Kiste öffnen, positioniert das Spiel den Charakter innerhalb eines Frames direkt vor der Kiste. Daraufhin wird eine Standardanimation abgespielt. Das gefundene Objekt schwebt danach vor dem Charakter.

Selbst bei Kisten, Truhen und Hebeln, für die es eigens angefertigte Animationen gibt, darf an den Modellen nach dem Erstellen nichts mehr geändert werden, damit die Animation weiterhin dazu passt. Das führt dazu, dass dieselben Modelle durch das ganze Spiel hinweg immer wieder verwendet werden. Dies bedeutet grafische Unstimmigkeiten, insbesondere dann, wenn das Spiel an mehreren sehr verschiedenen Orten spielt; z.B. wenn in Indianerhütten die gleichen Kisten stehen, wie im Hochsicherheitslabor. Außerdem wird das häufige Wiederverwenden der gleichen assets[4] von den Spielern auch zunehmend kritisch aufgenommen, da die Wiederholungen den Eindruck einer 'billigen Abfertigung' und 'aufgewärmter Kost' vermitteln.

1.3. Ziel der Arbeit

Der Zwang zur Entwicklung immer neuer Technologien hat den Entwicklungsaufwand vieler Spiele enorm in die Höhe getriebenen. Aufwändigere Grafiken und Spiele-Engines erfordern mehr Künstler, Grafiker, Designer und Programmierer, weshalb die Teams immer größer werden. Bereits jetzt liegt die Entwicklungszeit für einen Ego-Shooter bei über 3 Jahren, und es entstehen Kosten bis in den zweistelligen Millionenbereich. Neue Techniken, die entwickelt werden sollen, müssen daher nach Aufwand und Nutzen gegeneinander abgewogen werden.

In dieser Arbeit soll daher eine echtzeitfähige Lösung entwickelt werden, die genaue und natürlich aussehende Animationen zur Visualisierung von Charakter-Objekt-Interaktionen dynamisch mithilfe von Inverser Kinematik erstellt. Gleichzeitig soll der Aufwand, der für die Nutzung anfällt, minimiert werden, um möglichst geringe zusätzliche Entwicklungskosten zu generieren. Dabei sollen möglichst viele der zuvor genannten Probleme der Spieler-Objekt Interaktion vermieden werden, um einen höheren Grad an Realismus bei diesen Animationen zu erreichen.

4 assets - so werden in der Spielebranche die nicht zum Programmcode gehörenden Spielbestandteile genannt, insbesondere Grafiken und 3D Modelle, ferner aber auch Sounds, Musik und sogar Skripte.

2. Inverse Kinematik

2.1. Was ist Inverse Kinematik

Die Inverse Kinematik stammt ursprünglich aus der Robotik. In der Industrie werden seit Jahren Roboter in der Produktion eingesetzt, z.B. Roboter-Arme in einer Produktionsstraße für Autos. Ein solcher Roboter-Arm besteht aus mehreren steifen Gliedern, die über Gelenke verbunden sind. Am Ende der Kette ist ein Werkzeug angebracht und die Basis des Armes ist fest verankert, kann aber unter Umständen einen mobilen Unterbau besitzen. Die Stellung eines jeden Gelenkes beeinflusst die Position des Werkzeugs am Ende des Armes.

Diese Arme müssen in der Lage sein, ihre Aufgaben an einem Werkstück auszuführen, dessen Position nicht millimetergenau bestimmt werden kann. Daher können hier keine vorher festgelegten Bewegungsabläufe abgespult werden. Diese, praktisch oft nicht einsetzbaren, Bewegungsabläufe werden auch „Forward-Kinematik“ genannt. Bei dieser Methode würde der Endpunkt des Armes durch die vorher gespeicherten Gelenkstellungen bestimmt. Benötigt wird aber eine inverse Berechnung – also, die Gelenkstellungen müssen anhand der Zielposition und der Armposition berechnet werden.

Unter Zuhilfenahme von Kameras, Markern und Lasern wird für das Werkzeug eine Ziel-Position im Raum auf dem Werkstück ermittelt. Die Winkel für die einzelnen Gelenke müssen nun so berechnet werden, dass der Endpunkt des Armes auf den Zielpunkt zeigt. Die Inverse Kinematik liefert Methoden, diese einzelnen Winkel zu berechnen.

Das Grundprinzip besteht also darin, mathematisch die Winkel der einzelnen Gelenke zu bestimmen, die zur gewünschten Ausrichtung führen. Hier ist eine Rechnung für einen Arm mit zwei Gliedern im zweidimensionalen Raum[2].

$$\theta_2 = \operatorname{acos}\left(\frac{Target.x^2 + Target.y^2 - Length1^2 - Length2^2}{2 \cdot Length1 \cdot Length2}\right)$$

$$\theta_1 = \theta_3 - \theta_4$$

$$\theta_3 = \operatorname{atan}\left(\frac{ydistance}{xdistance}\right)$$

$$\theta_4 = \operatorname{atan}\left(\frac{Length2 \cdot \sin(\theta_2)}{Length1 + Length2 \cdot \cos(\theta_2)}\right)$$

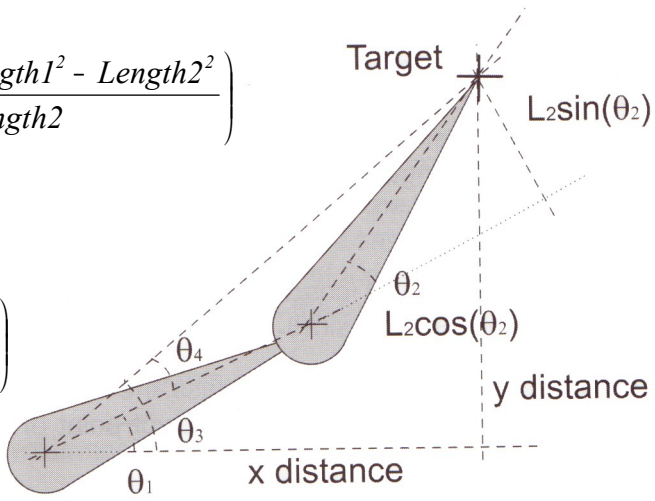

Abbildung 12: Berechnung für eine Knochenkette mit zwei Gliedern

Die Problematik besteht darin, dass es keine eindeutige Lösung für einen mehrgliedrigen Arm mit fester Basis und einen fixen Zielpunkt gibt. Je mehr Glieder ein solcher Arm hat, desto mehr Lösungen gibt es auch. Im zweidimensionalen Raum gibt es für einen Arm mit zwei Gliedern bereits zwei Lösungen. Für drei Glieder sind es bereits vier Lösungen und für vier Glieder sind es acht. Jedes zusätzliche Glied verdoppelt die Zahl der möglichen Lösungen, es gibt also 2^{n-1} Lösungen für n Glieder. Durch das Hinzufügen einer weiteren Dimension werden sie mathematisch unendlich.

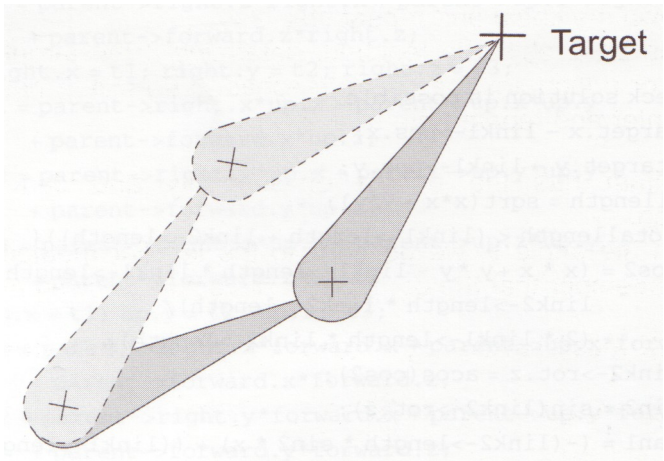

Abbildung 13: mehrere Lösungsmöglichkeiten

Bereits für eine Kette mit zwei Gliedern gibt es keine eindeutige Lösung.

In der IK gibt es verschiedene Methoden und Algorithmen, um diese Problematik zu lösen. Einer der bekanntesten Algorithmen ist der Cyclic Coordinate Descent (CCD). Diese Technik wurde erstmals von Li-Chun Tommy Wang und Chih Cheng Chen in einem Artikel über Robotik[3] vorgestellt. Der Algorithmus durchläuft in einer Schleife alle Glieder, bzw. Gelenke, vom letzten bis zum ersten. Er dreht dabei jeden Abschnitt so, dass der Endpunkt des Armes so weit wie möglich auf den Zielpunkt zeigt.

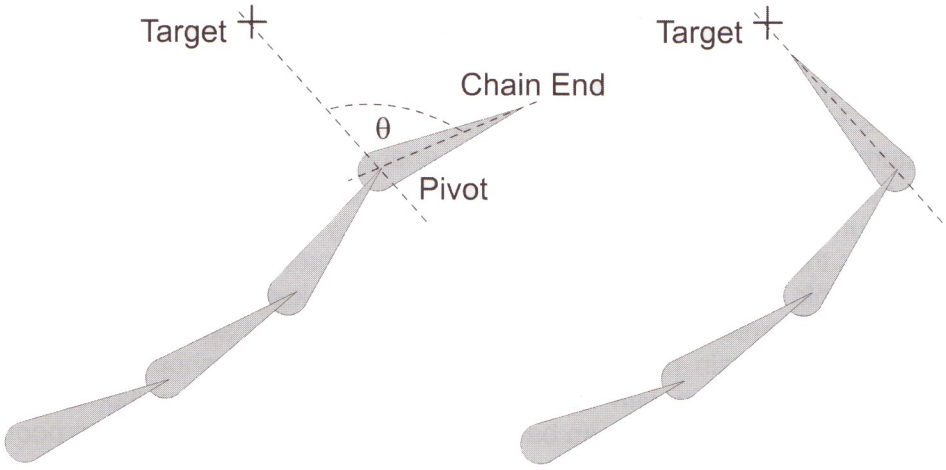

Abbildung 14: Schritt 1 im Cyclic Coordinate Descent

Das hinterste Kettenglied wird in Richtung des Zielpunktes gedreht. Danach springt der Algorithmus zum nächsthöheren Kettenglied.

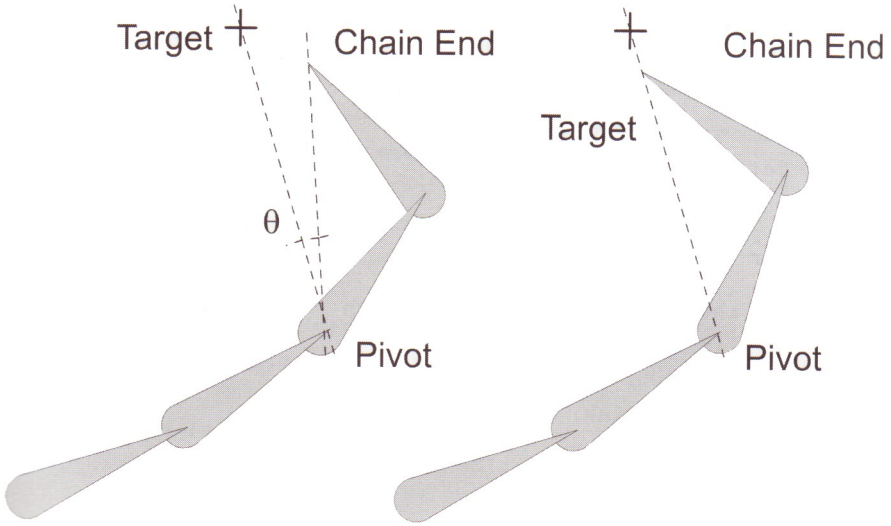

Abbildung 15: Schritt 2 im Cyclic Coordinate Descent

Jedes weitere Kettenglied wird so gedreht, daß der Vektor vom aktuellen Gelenk bis zum Endpunkt der Kette auf das Ziel zeigt.

Dies geschieht, indem einerseits der Vektor vom aktuellen Gelenk zum Endpunkt des Armes ermittelt wird, und der Vektor vom Gelenk zum Zielpunkt andererseits. Der Winkel zwischen diesen beiden Vektoren ist dann der Winkel, um den das aktuelle Gelenk gedreht wird. Nach jedem Durchlauf wird überprüft, ob der Zielpunkt bereits hinreichend genau getroffen wurde. Um Endlosschleifen zu vermeiden, wird nach einer festgelegten Maximalzahl an Durchläufen abgebrochen.

Der CCD ist ein sehr robuster Algorithmus, der in der Regel bereits nach wenigen Schleifendurchläufen das Ziel erreicht. Er kann im dreidimensionalen Raum eingesetzt werden, indem die Basis des Armes in Richtung des Zielpunktes gedreht wird. So können alle restlichen Glieder mit dem einfachen zweidimensionalen Verfahren berechnet werden.

Trotzdem kann die ermittelte Lösung nicht immer die gewünschte sein. Die Ergebnisse des CCD resultieren durch die Art der Berechnung darin, dass die weiter am Ende des Armes liegenden Gelenke stark abgeknickt werden, während die Gelenke näher an der Basis kaum bewegt sind. Dies kann Nachteile z.B. bei der Lastenverteilung mit sich bringen. Daher gibt es eine Reihe Modifikationen des CCD, z.B. mit einem Maximalwinkel, der pro Schleifendurchlauf an jedem Gelenk gedreht werden darf, um die Drehungen weiter nach hinten zu verlagern.

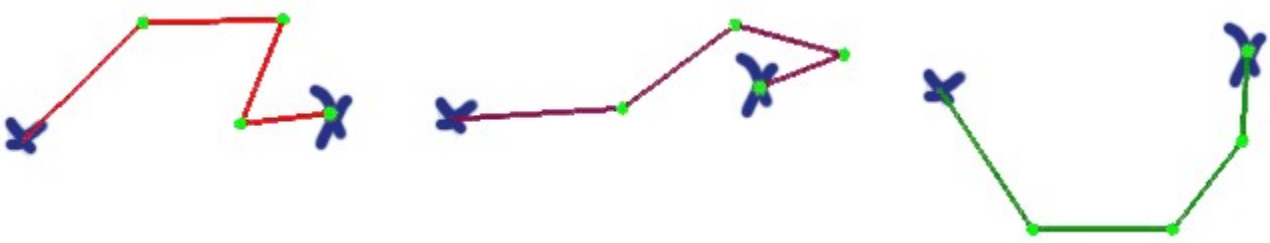

Abbildung 16: drei verschiedene Iks

Mit dem gleichen Start- und Zielpunkt liefert jede IK ein anderes Ergebnis. Die ersten beiden Ketten wurden mit CCD-Varianten gelöst, die dritte durch eine analytischen Algorithmus auf Basis einer Kreisannäherung.

Die Idee der Inversen Kinematik kann auf die Animation von skelettbasierten Charakteren übertragen werden. So können mittels einer IK theoretisch die Winkel an Knie und Hüfte, bzw. Schulter und Ellenbogen berechnet werden. Hier müssen jedoch weitere Beschränkungen gelten. Menschliche Gelenke verfügen in der Regel über einen geringeren Winkelbereich als ein Roboterarm und jedes einzelne menschliche Gelenk unterscheidet sich vom nächsten. So kann sich z.B. ein Knie nur in eine Richtung beugen, während das Schultergelenk über drei Freiheitsgrade verfügt[4].

2.2. Inverse Kinematik in Spielen

Inverse Kinematik wird bereits in der Spiele-Entwicklung eingesetzt. Vor allem bei der Erstellung von Forward Kinematik, also den regulären Keyframe Animationen. Viele 3D Modell-Editoren, wie z.B. 3D Studio Max, bieten die Möglichkeit, mit einer simplen IK die Stellungen von Hüfte- und Kniegelenk, sowie Schulter- und Ellenbogengelenk zu berechnen. Dabei sind teilweise die Winkelbeschränkungen, wie sie die menschliche Anatomie diktiert, bereits berücksichtigt. Die Animateure brauchen lediglich noch die Hand oder den Fuß des Charakters bewegen. Dies erleichtert den Aufwand der Animationserstellung teilweise ungemein.

Wegen der notwendigen Winkelbeschränkungen und dem angestrebten natürlichen Aussehen der Bewegung wird IK in Spielen selbst eher spärlich eingesetzt. Die häufigste Variante ist eine Steuerung des Kopfes eines Charakters, der sich unabhängig vom Rest des Körpers bewegen soll, um einen Gegenstand oder eine Person in der Nähe anzusehen, mit der eine Interaktion stattfinden kann.

Auch an anderen Stellen wird das Skelett des Charakters direkt während des Spieles bewegt, z.B. um eine vorhandene Animation in die gewünschte Richtung zu drehen. So kann eine Animation, die den Spielcharakter die Waffen nach vorne zielen lässt, durch Drehung der Wirbelsäule so verändert werden, dass der Charakter zur Seite zielt. Diese Effekte können oftmals aber auch durch geschicktes Mischen von Keyframe-Animationen mit unterschiedlichen Gewichtungen erreicht werden. Lässt Animation A den Arm nach oben zeigen und Animation B nach rechts, so kann man durch Mischen von A und B den Arm nach schräg rechts oben bewegen. In modernen Spielen wird IK auch eingesetzt, um die Beine und Füße des Charakters anzupassen, wenn dieser an einem Hang steht. Die Füße sollen trotz des schrägen Untergrundes flach auf dem Boden aufliegen, und ein Bein muss angewinkelt werden, um einen natürlichen Stand zu erreichen [5].

Abbildung 17: Charakter an einem Hang, mit und ohne IK (Konoko Payne ©Pierre Terdiman)

Die Füße und Beine werden passend zum Hang platziert.

2.3. Inverse Kinematik für den menschlichen Arm

Die Inverse Kinematik ist ein Teil der Lösung zur Visualisierung komplexer Interaktionen, da sie die Rotationen für den Arm berechnen kann, die die Hand zu einer vorgegebenen Stelle führen. Um ein natürlich aussehendes Ergebnis zu erzielen, muss die Inverse Kinematik, die zur Berechnung der Gelenkstellungen eingesetzt werden soll, speziell auf den menschlichen Arm zugeschnitten sein.

Bei der Berechnung der Gelenke eines Armes sind zwei Knochen zu berücksichtigen; der Oberarm und der Unterarm. Die Schulter stellt den Fixpunkt des Armes dar. Durch einen vorgegebenen Punkt, an dem sich das Handgelenk befinden soll, ist auch der Zielpunkt definiert. Benötigt werden die Rotationen für das Schultergelenk, an dem der Oberarm sitzt, und für das Ellenbogengelenk. Diese Gelenke sind in der Medizin gut erforscht und ihre Freiheitsgrade sind bekannt.

Das Schultergelenk ist ein Kugelgelenk mit drei Bewegungsachsen. Die sagittale Achse erlaubt die Abduktion und Adduktion, die transversale Achse die Anteversion und Retroversion, und die Rotationsachse die Drehung um sich selbst (Quelle: Lehrbuch Anatomie[4]).

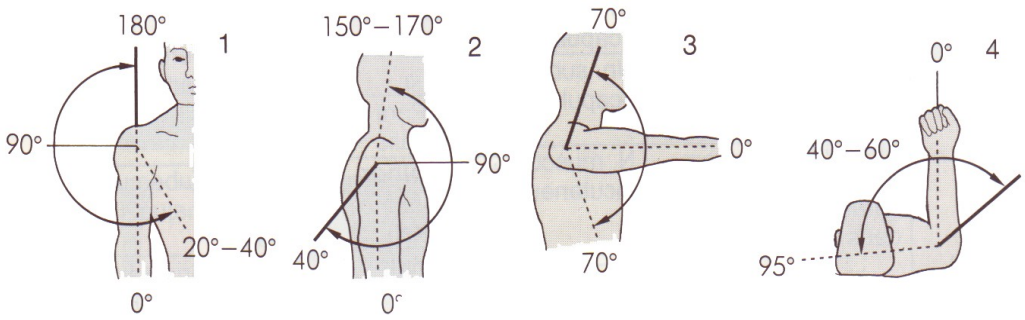

Abb. 823a-d. Bewegungsumfänge des Schultergelenks einschließlich Schlüsselbeingelenke (mittlere Meßwerte von gesunden jungen Erwachsenen nach der Neutralnullmethode). *[bh1]*

1 Abduktion und Adduktion
2 Retroversion und Anteversion
3 Außen- und Innenrotation bei abduziertem Arm
4 Außen- und Innenrotation bei adduziertem Arm

Abbildung 18: Bewegungsumfang des Schultergelenks

Das Ellenbogengelenk ist beim Menschen weitaus komplizierter aufgebaut, als bei einem virtuellen Charakter, da es sich aus drei Knochen zusammensetzt, dem Humerus (Oberarm), dem Radius (Speiche) und der Ulna (Elle). Dadurch verfügt das Ellenbogengelenk des Menschen über eine zusätzliche Bewegungsachse, nämlich der Rotation um sich selbst. Dabei wird die Speiche, die im Ellenbogen verankert ist, an ihrem anderen Ende (nahe des Handgelenkes) um die Elle rotiert.

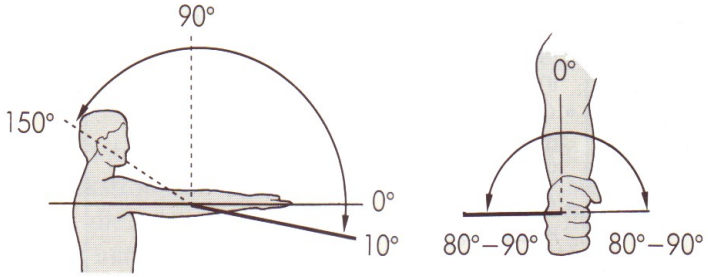

Abbildung 19: Bewegungsumfang des Unterarms

Bei virtuellen Charakteren wird der Unterarm jedoch fast immer vereinfacht und nur mit einem einzelnen Knochen repräsentiert, der sich auch nur im Ellenbogen einknicken kann. Eine Drehung um die Längsachse des Unterarms wird daher ausschließlich von der Hand ausgeführt.

Bereits im zweidimensionalen Raum gibt es in der Inversen Kinematik zwei mögliche Lösungen für einen Arm mit zwei Knochen. Durch das Hinzukommen einer dritten Dimension gibt es unendlich viele. Indem eine Achse durch den Start- und den Zielpunkt gelegt wird, um die der Arm rotiert wird, können theoretisch beliebig viele weitere gültige Lösungen ermittelt werden. Durch die Freiheitsgrade wird dieser Lösungsbereich ein wenig eingegrenzt, bleibt aber unendlich.

Abbildung 20: Verschiedene Armstellungen

Die Stellung des Ellenbogens ist variabel.
Es kann sogar zur Selbstkollision kommen.

Der vorgestellte CCD Algorithmus ist für die Berechnungen eher ungeeignet. Sein Einsatzgebiet liegt aufgrund der rekursiven Berechnung bei mehrgliedrigen Knochenketten, die im Gegenzug keine besonderen Winkelbeschränkungen aufweisen. Darüber hinaus ist der CCD nicht dafür ausgelegt, eine natürliche Armhaltung nachzuempfinden.

Yoshibito Koga hat in seinem Artikel „Planning Motions with Intentions“[6] eine IK vorgestellt, bei der die Art der auszuführenden Aktion bei der Berechnung mit einfließt, z.B. Aufnehmen und Wenden eines Schachbrettes. Je nach Art der Aktion ist eine andere Armhaltung und ggf. sogar eine andere Handhaltung erforderlich. Dabei setzt er voraus, dass die Handposition und Griffpose extern vorgegeben sind. Auch zusätzliche Informationen über die Beschaffenheit des Objektes, wie z.B. Größe oder Gewicht, können Einfluss auf die Armhaltung nehmen.

Da Computerspiele und andere virtuelle Umgebungen aber in der Regel interaktiv sind, kann über die Absicht einer Aktion keine Aussage getroffen werden, bzw. diese kann sich im Verlaufe der Aktion ändern. Da die Architektur der Programme zudem meist sehr abstrakt aufgebaut ist, kann auch das Hauptprogramm nur bedingt Informationen über die Absicht der Aktion weitergeben. Das gleiche gilt für zusätzliche Informationen über die zu greifenden Objekte. Sind diese nicht bereits im Spiel vorhanden, müssten sie hinzugefügt werden, was zusätzlichen Aufwand in der Entwicklung verursachen würde. Zudem soll das hier zu entwickelnde Framework auf beliebig kombinierbare Teil-Interaktionen aufbauen, um die vom Spiel benötigte Flexibilität zu liefern. Diese müssen frei von einer Handlungsintention bleiben.

Da über die Intention der auszuführenden Handlung keine Aussage gemacht werden kann und soll, muss die errechnete Stellung des Armes möglichst neutral und allgemein gültig sein. Dazu kann

sich die IK an der Ziel-Handhaltung am Objekt orientieren. Indem die Winkel am Handgelenk in die Berechnung des Armes einfließen, kann eine entspannte Handhaltung erreicht werden und eine neutrale Armstellung berechnet werden.

Grundlage dafür kann die in medizinischen Untersuchungen der Hand ermittelte so genannte 'funktionale' Handstellung sein, die von der Nullstellung der Hand abweicht. Aus dieser Stellung heraus finden nach empirischen Messungen die meisten Aktionen der menschlichen Hand statt. Dabei befindet sich das Handgelenk in 20° Extension und 10° Adduktion. In dieser Arbeit wird daher eine eigene IK implementiert, die die Winkel am Handgelenk berücksichtigt, um eine neutrale Armstellung zu berechnen.

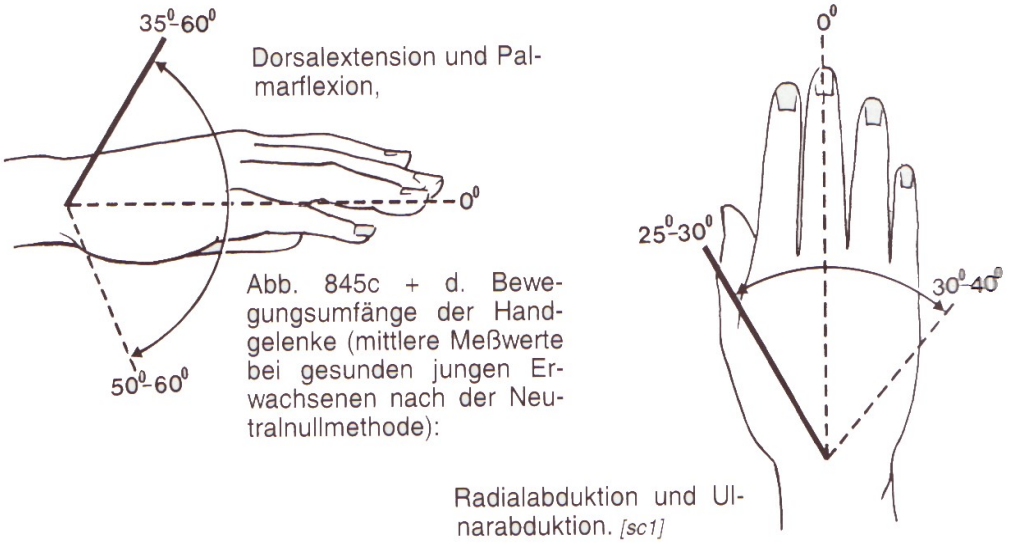

Abbildung 21: Bewegungsumfang des Handgelenks

3. Objektbasierter Ansatz

Obwohl es die Inverse Kinematik bereits lange gibt, wird sie dennoch bislang nicht oder nur äußerst selten für die Charakter-Objekt-Interaktion eingesetzt. Der Grund dafür ist, dass die Berechnungsmethode alleine nicht ausreicht.

Eine Inverse Kinematik kann nur arbeiten, wenn es bereits einen Zielpunkt für die Knochen-Kette gibt, die animiert werden soll. Die Bestimmung eines solchen Zielpunktes auf einem Objekt gestaltet sich aber äußerst schwierig, vor allem, wenn Position und Orientierung des Charakters und des Objektes sich ständig ändern können.

Darüber hinaus muss die IK für die Bewegungsgrade eines menschlichen Armes konkret angepasst werden, um ein natürliches Ergebnis zu erzielen.

Und abschließend müssen auch noch Hand und Finger animiert werden. Die Handhaltung in Relation zum Objekt ist je nach Art des Gegenstandes verschieden. Auch die Fingerstellungen und der Grad der Schließung der Finger sind abhängig vom Objekt. Und darüber hinaus beeinflusst die Handstellung zu einem gewissen Grad auch die Haltung des Armes.

3.1. Grundidee

In dieser Arbeit soll ein Framework entwickelt werden, das selbstständig unter Einsatz von Inverser Kinematik für beliebige Charaktere und Objekte Animationen zur Visualisierung von Interaktionen erstellt und diese auch abspielen kann. Eine eigens entwickelte Inverse Kinematik übernimmt dabei die Berechnung der Gelenkstellungen des Charakters.

Bevor dies geschehen kann, muss jedoch ein Zielpunkt für den Arm, bzw. die Hand gefunden werden, mit dem die Berechnungen durchgeführt werden können. Dieser muss sich an einer sinnvollen Stelle auf der Oberfläche des zu greifenden Objektes befinden, damit der Arm und die Hand in der Endstellung möglichst natürlich wirken. Darüber hinaus müssen auch die Finger und die Hand so animiert werden, dass der realistische Eindruck entsteht, die Hand würde das Objekt tatsächlich greifen. Dies sind die eigentlichen Probleme bei der Erstellung eines solchen Animationssystems.

In dieser Arbeit soll diese Problematik rückwärts vom Ziel zum Fixpunkt angegangen werden. Das bedeutet, dass zuerst eine geeignete Position auf der Oberfläche des Objektes ermittelt werden soll, an dem ein Charakter dieses Objekt aus seiner momentanen Stellung heraus gut greifen kann. Dann wird abhängig vom Objekt eine passende Finger- Handstellung gewählt. Durch Rückrechnung ergibt sich aus diesen Daten ein Zielpunkt für das Handgelenk. Mit diesem Zielpunkt kann über die IK eine natürliche Armstellung berechnet werden. Abschließend wird die Armstellung noch anhand der Handhaltung korrigiert.

Ein beliebig aufgebautes 3D Modell daraufhin zu untersuchen, wo es angefasst und gegriffen werden könnte, erfordert eine aufwändige Untersuchung der einzelnen Polygone. Der Vorteil dieser automatischen Untersuchung liegt darin, dass kein zusätzlicher Aufwand für Designer und 3D Artists entsteht. Da die Dreiecke und Polygone jedoch kaum etwas über die tatsächliche Art und

Beschaffenheit des Objektes aussagen, ist es nur schwer möglich, auf diesem Wege geeignete Informationen darüber zu gewinnen, wie dieser Gegenstand gegriffen werden soll. Darüber hinaus ist eine Analyse der Dreiecke überaus zeitaufwändig, da selbst kleine Objekte aufgrund der steigenden Leistungsfähigkeit von Grafikkarten aus immer mehr Polygonen bestehen. Eine Berechnung in Echtzeit ist daher für solch einen Ansatz unwahrscheinlich. Eine Vorberechnung aller Objekte würde hingegen viel Speicherplatz einnehmen, oder die Ergebnisse wären vielleicht nicht geeignet für einen Einsatz in dynamischen Umgebungen, wo sich Position von Charakter und Objekt ständig ändern können. Darüber hinaus bietet diese Methode den Designern keine Möglichkeiten, Einfluss darauf zu nehmen, wo ein Objekt gegriffen werden soll.

Um diese Probleme anzugehen, wird in dieser Arbeit daher ein objektbasierter Ansatz vorgestellt, der nicht gänzlich vollautomatisch ist, aber den zusätzlichen Aufwand beim Objektdesign gleichzeitig minimiert. Dies bedeutet, das die für die Berechnungen notwendigen Informationen beim Objekt gespeichert werden, wobei deren Umfang gleichzeitig so gering wie möglich gehalten wird. Es handelt sich bei diesen Informationen um einfache, geometrische Primitive, die als Hilfsobjekte auf dem Objekt verankert sind und Teile des 3D Modells vereinfacht repräsentieren. Ein ähnliches Modell benutzen bereits Physik-Engines für ihre Berechnungen. Durch diese Methode wird die komplexe Oberfläche eines Objektes stark vereinfacht. So können weitere Berechnungen stellvertretend für das Objekt stattdessen auf diesen Primitiven ausgeführt werden.

Die Primitive werden dabei relativ zum Objekt positioniert und rotiert. So kann ihre Position durch die Position und Orientierung des Objektes stets korrekt in Weltkoordinaten berechnet werden. Durch den einfachen Aufbau der Primitive entstehen nur wenige Daten, die gespeichert werden müssen. Neben der Position und Rotation genügen meist ein bis drei weitere Werte, die die Größe und den Umfang des geometrischen Körpers angeben. Weitere Informationen werden nicht benötigt. Die Hilfsobjekte werden mittels eines Editors am Objekt verankert. So haben Designer und 3D Artists die Möglichkeit, Einfluss auf die Art zu nehmen, wie und wo das Objekt gegriffen wird.

Abbildung 22: Grundkonzept des Editors

Das Primitiv wird am Griff der Teekanne ausgerichtet, skaliert und rotiert.

Diese Primitive lösen gleich beide Probleme der Griff-Findung auf einmal. Sie dienen einerseits als Markierung dafür, wo ein Gegenstand gegriffen werden kann - zum Beispiel kann mit einer Säule der Griff an einer Teekanne repräsentiert werden - und zum anderen sagen sie etwas darüber aus wie das Objekt gegriffen werden soll; zum Beispiel weist der Umfang des Primitives darauf hin, wie weit sich die Finger schließen sollen. Das Framework nutzt die Primitive, um in Echtzeit abhängig von der Stellung des Charakters und der Lage des Objektes einen geeigneten Zielpunkt auf der Oberfläche des Primitives für die Inverse Kinematik zu errechnen. Wie das genau geschieht, ist in Kapitel 5.3. Die fünf Primitive beschrieben.

3.2. Definition der Ziele

Ziel dieser Arbeit ist die Erstellung eines objektbasierten Frameworks, das mittels Inverser Kinematik und geometrischen Primitiven, natürlich aussehende Animationen zur Visualisierung von Charakter-Objekt-Interaktionen berechnet. Die Animationen beschränken sich dabei ausschließlich auf die Arme bzw. Hände eines virtuellen Charakters, damit diese mit anderen Animationen gemischt werden können und nicht zu stark in den Ablauf des Spieles eingegriffen wird. Es handelt sich um eine reine Visualisierung. Nicht berücksichtigt werden sollen daher Kollision mit der Umgebung und dem Charakter selbst, sowie wie Physikalisierung des Charakters oder der Objekte. Weitere Animationen, wie etwa das Hinknien, Vornüberbeugen des Oberkörpers oder die Wegfindung und das Bewegen des Charakters zu dem Objekt, müssen von außerhalb über das Hauptprogramm gesteuert werden. Da dem Framework nichts über den Szenenaufbau oder den momentanen Status des Charakters bekannt ist, kann es diese Funktionen nicht übernehmen. Würde das System beispielsweise einen Charakter, der bereits auf dem Boden liegt und robbt, mit einer Hinknien-Animation belegen, so wäre das Ergebnis optisch sehr unattraktiv.

Dabei soll das System einen möglichst geringen zusätzlichen Aufwand für die Entwicklung verursachen. Die Bedienung des Frameworks muss robust und leicht erlernbar sein. Die Erstellung der Primitive muss einfach, intuitiv und schnell erfolgen können. Daher müssen die meisten Berechnungen in Echtzeit während des Spielablaufs durchgeführt werden können.

Um dem Realismus-Trend zu folgen, sollen die erstellten Animationen und ermittelten Griffposen möglichst natürlich wirken und dem tatsächlichen menschlichen Bewegungsablauf nachempfunden sein. Die Inverse Kinematik und die Animationen müssen daher speziell für dieses Einsatzgebiet entwickelt werden. Auch die Finger- und Handhaltung soll bestimmt werden. Die zum Einsatz kommende Inverse Kinematik muss den anatomischen Vorgaben für die menschlichen Gelenke des Armes aus der Humanmedizin genügen.

Darüber hinaus muss ein angemessener Funktionsumfang geboten werden, der die häufigsten Interaktionen abdecken kann. Neben dem Greifen eines Objektes müssen daher weitere, sinnvolle Funktionen implementiert werden. Dies kann auch die Modifikation des Objektes bzw. seiner Position und Orientierung mit einschließen.

Grundlage aller Berechnungen ist ein Set an Primitiven, das umfangreich genug sein muss, um den Großteil aller Objekte abzudecken.

Die erstellten Animationen sollen auch interpoliert und über das Framework abgespielt werden. Die Kommunikation mit dem zugrunde liegenden Animationssystem soll über ein geeignetes, abstraktes Interface stattfinden.

Um den genauen Umfang und die Details der einzelnen Bestandteile des Frameworks zu bestimmen, wird eine Anforderungserhebung durchgeführt und das Framework nach den Ergebnissen entworfen.

4. Anforderungserhebung

In der Anforderungserhebung wurde eine Reihe Spiele daraufhin untersucht, welche Charakter-Objekt Interaktionen in ihnen vorkommen. Darüber hinaus wurden die Objekte, die Teil der Interaktionen sind, gesammelt und daraufhin analysiert, welche Primitive für eine sinnvolle Repräsentation der Stellen, an denen diese gegriffen werden, geeignet sind.

Da das Angebot an Spielen und Genres sehr groß ist, dient eine Liste der beliebtesten Spiele von einem der größten deutschen Spiele-Magazine als Grundlage für die Auswahl und Gewichtung der Spiele-Genres.

```
Charts:
1. Gothic 3
2. World of Warcraft
3. Anno 1701
4. Elder Scrolls Oblivion
5. Battlefield 2
6. Diablo 2
7. Half Life 2
8. Warcraft 3
9. Guild Wars
10. Medieval 2
11. Far Cry
12 Titan Quest
13. F.E.A.R.
14. Counterstrike
15. Dark Messiah of Might and Magic
16. Tackmania United
17. Supreme Commander
18. Neverwinter Nights 2
19. Battlefield 2142
20. Call of Duty 2
```

Listing 4.1: Die beliebtesten Computerspiele
Lesercharts aus der GameStar 05/2007

Nach Genres sortiert ergeben sich aus den 20 Spielen auf der Liste 7 Shooter, genauer sogar Militär-Shooter, 5 Action Rollenspiele, 2 MMORPGs[5], sowie 4 Strategiespiele, ein Autorennspiel und der Spiele-Klassiker Diablo II. Die Strategiespiele enthalten keine oder nur indirekt durch den Spieler steuerbare Charaktere, die darüber hinaus auch keine nennenswerte oder gar keine Interaktion mit ihrer Umgebung haben können. Daher werden diese in der Untersuchung nicht berücksichtigt. Da es sich bei Diablo II um ein 2D Spiel handelt, fällt auch dieses aus der Wertung, da sich diese Arbeit um die Animation von Charakteren in einer 3D Umgebung dreht.

5 MMORPG – Massively Multiplayer Online Roleplay Game; Diese Spiele sind ausschliesslich online spielbar, in der Regel gegen eine monatliche Grundgebühr. Man spielt in Echtzeit und gleichzeitig mit bis zu mehreren Tausend anderen Spielern zusammen in einer Welt. Dabei übernimmt man die Rolle eines bestimmten Helden, den man im Laufe des Spieles aus- und fortbildet

Die meisten Spiele der resultierenden Liste bieten die Möglichkeit zur Erstellung eigener Maps und Mods, so dass sich die Umgebung tatsächlich durch User generierte Inhalte dynamisch erweitern lässt. Daher ist eine Erweiterung des Hauptspieles um ein entsprechendes Animationssystem sinnvoll.

4.1. Durchführung und Ergebnisse

Die einzelnen Spiele werden untersucht und eine Liste mit den häufigsten Interaktionen und den beteiligten Objekten erstellt, um daraus eine Liste mit Funktionen und Griffposen für das Framework abzuleiten. Dabei werden nicht nur die Spiele auf der Liste, sondern auch ähnliche Spiele des gleichen Genres untersucht, um ein breiteres Spektrum und einen besseren Durchschnitt zu erhalten.

```
Kämpfen/Schlagen
Aufnehmen (z.B. Waffen, Munition)
Sammeln/Harvesten
Werfen
Annehmen (Tausch/Handel mit anderen Charakteren)
Öffnen (Kisten, Truhen etc.)
Öffnen (Türen. auch Auto-Türen)
Benutzen eines Gegenstandes in Kombination mit
      einem anderem (z.B. Schlüssel/Schloss,
      Keycard/Computer)
Crafting/Handwerk
Taschendiebstahl
Durchsuchen („Looten“)
Bedienelemente (Displays, Numpads, Knöpfe)
Gegenstand bewegen (Hebel, Lenkrad)
```

Listing 4.2: Die häufigsten Interaktionen mit Objekten

```
Maschinengewehr (und andere Großkaliber)
Pistole (und andere Kleinkaliber)
Munition
Schwert
Bogen
Messer
Axt
Stab
Granate
Erste-Hilfe-Kit
Flaschen (verschiedene Formen)
Ring
Türgriff
Wagentürgriff
Lenkrad
Sprengsatz
Schlüssel
Buch
Hebel
Knopf (Bedienelemente und Schaltflächen)
Deckel (Kisten, Truhen etc.)
KeyCard
Papier/Zettel
```

Listing 4.3: Die Objekte mit den meisten Interaktionen

Da die Häufigkeit der Interaktionen und der eingesetzten Gegenstände stark davon abhängig ist, wie und von wem das Spiel gespielt wird, können diese Listen nicht vollkommen repräsentativ sein. Die Reihenfolge der Listeneinträge gibt daher nicht zwingend konkreten Aufschluss über die tatsächliche Häufigkeit. Dennoch bildet diese Aufstellung eine aussagekräftige Basis für den Entwurf des Frameworks.

Zu berücksichtigen ist, dass durch das Framework nicht nur die bereits in Spielen eingesetzten Interaktionen besser visualisiert werden sollen, sondern darüber hinaus auch Interaktionen möglich gemacht werden sollen, die bislang gerade deshalb gemieden wurden, weil es kaum möglich war, sie zufriedenstellend darzustellen. Daher soll das Framework so allgemein wie möglich funktionieren und sich nicht nur auf die bereits bestehenden Interaktionen beschränken.

4.2. Primitive

Die Gegenstände auf der Liste der am häufigsten in Interaktionen vorkommenden Objekte wurden dahingehend analysiert, mit welchen geometrischen Primitiven sich die Objektteile, an denen der Gegenstand gegriffen wird, am besten abbilden lassen.

Dabei ist zu beachten, dass die Gegenstände auf verschiedenen Richtungen gegriffen werden können und alle Griffoptionen berücksichtigt werden müssen. Darüber hinaus muss bei einigen Gegenständen, insbesondere bei kleineren und einfachen, eventuell das gesamte Objekt repräsentiert werden können, z.B. bei einer Granate oder einem Buch. Als Resultat ergeben sich 5 Primitive.

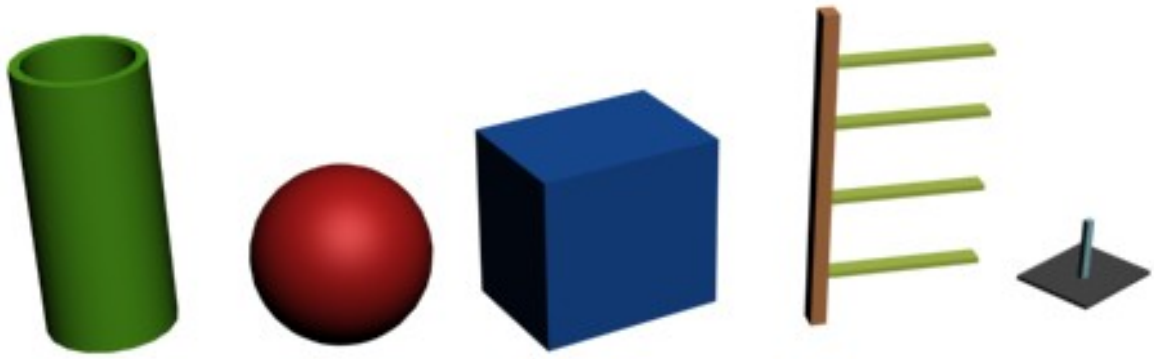

Abbildung 23: die 5 Primitive

von links nach rechts: Säule, Kugel, Box, Papier, Knopf

Das Säulen Primitiv ist das Allround-Talent unter den Primitiven, da es bereits alleine einen Großteil der Objekte abdecken kann. Es eignet sich besonders für Griffe aller Art. Dazu zählen z.B. Körbe, Schwerter, Äxte, Messer, Türklinken, Lenkräder und Flaschen, sowie alle Arten von Tragegriffen.

Die Kugel ähnelt der Säule und soll vor allem spezieller geformte Gegenstände sauber repräsentieren, wie zum Beispiel ein Hebelknauf, Kugeln oder Handgranaten.

Die Box ist ein weiteres sehr vielseitig einsetzbares Primitiv. Sie eignet sich vor allem für solche Gegenstände, die vollständig durch ein Primitiv repräsentiert werden müssen, weil man sie entweder überall greifen könnte, oder vollständig umschließen soll. Dazu zählen neben großen und kleinen Kisten z.B. Bücher, Med-Kits, Sprengsätze und auch Türgriffe.

Bei den letzten beiden Primitiven handelt es sich um 2D Primitive. Das Papier (oder auch Kreditkarten) -Primitiv repräsentiert eine Kante an einem sehr dünnen Gegenstand. Die heraus zeigenden Querstreben (im Bild gelblich eingefärbt) zeigen an, in welcher Richtung sich das Innere des Objektes befindet, so dass die Kante immer von außen gegriffen werden kann. Dieses Primitiv eignet sich für dünne Gegenstände alle Art, auch sehr kleine. Neben Papier, Kreditkarten und Klemmbrettern z.B. auch für Ringe und Schlüssel.

Aufgabe der Primitive ist es, abhängig von ihrer eigenen Position und der des Charakters, einen passenden Berührungspunkt zu ermitteln und einen Zielpunkt für die Inverse Kinematik zu bestimmen. Darüber hinaus sollen sie die Haltung der Hand bestimmen. Der Verlauf der Handachse ist bei jedem Primitiv verschieden, so wird eine Säule in der Regel nicht senkrecht von der Seite, sondern in einem Winkeln zwischen 15°-25° gegriffen.

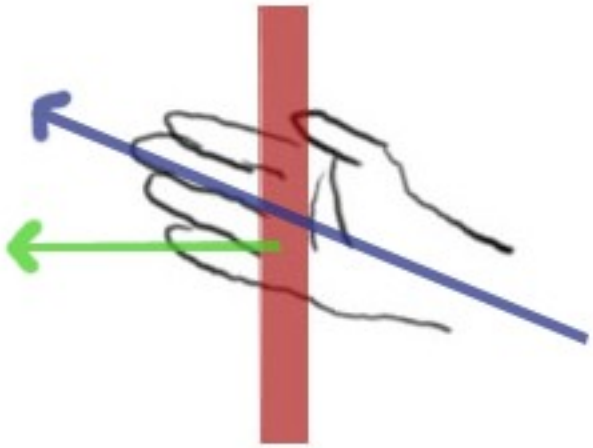

Abbildung 24: Beispiel-Handstellung am Säulen-Primitiv

Wie die Handachse bei den einzelnen Primitiven verläuft, wird in Kapitel 4.3. Griffposen näher untersucht. Damit die Finger korrekt animiert werden können ist das Primitiv darüber hinaus auch für die Wahl einer geeigneten Fingerstellung verantwortlich.

4.3. Griffposen

Im nächsten Schritt sollen Stellungen für die Finger ermittelt werden, die jeweils zu den Primitiven passen und zu den Objektteilen, die sie repräsentieren. Die Grundidee ist, dass jedes Primitiv eine Standard-Griffpose erhält, die dann entsprechend der Ausmaße des Primitives angepasst wird. Diese Verknüpfung von Primitiv und Fingerstellung ist notwendig, da das Primitiv auch die Handhaltung bestimmt. Die Griffposen müssen mit Rücksicht auf diese Haltung modelliert werden und lassen sich daher auch nicht einfach auf andere Primitive übertragen. Theoretisch ist dies jedoch möglich, da eine gewisse Trennung zwischen den beiden erhalten bleiben muss, um dem Primitiv die Möglichkeit zu lassen, eine andere Griffpose als die Standardstellung zu wählen. Dies ist zum Beispiel bei dem Box-Primitiv notwendig, da dieses abhängig von der Größe anders gegriffen werden muss. Ein weiterer Grund für das Wählen einer anderen Haltung könnte sein, wenn ein Objekt mit zwei Händen gegriffen wird, statt nur mit einer.

Die Griffposen müssen neutral sein, da keine Information über die Absicht des Charakters bekannt ist, bzw. sich diese Absicht aufgrund der Interaktivität des Spieles ändern könnte. Um einerseits eine Basis für die Modellierung zu haben, aber auch um gleichzeitig die Griffposen für die verschiedenen Primitive zu untersuchen, wurde eine Fotoserie erstellt. Anhand der verschiedenen Haltungen wird auch die Stellung der Handachse deutlich. Auch die Modellierung der Griffposen basiert auf diesen Haltungen. Für die praktische Umsetzung siehe Kapitel 5.3. Die fünf Primitive.

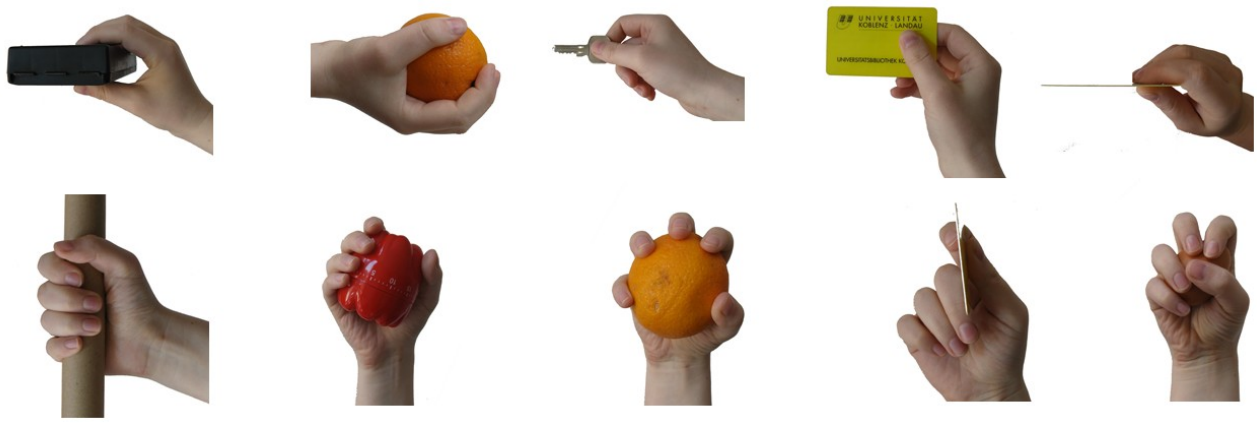

Abbildung 25: Ausschnitt aus der Fotoserie zur Ermittlung der Griffposen

Da in Computerspielen und anderen virtuellen Umgebungen die Hände der Charaktere meist nur vereinfacht dargestellt werden und in der Regel nur sehr klein im Bild zu sehen sind, und da die Möglichkeiten in der Animation weitaus weniger vielfältig sind, als die der menschlichen Hand, können einige Stellungen zusammengefasst werden. So ähneln sich die Fingerhaltungen für Papier, Ring, Schlüssel und Kreditkarte weit genug, um sie mit der gleichen Fingerstellung abdecken zu können.

Untersuchungen über häufige Fingerstellungen, bzw. Griffposen werden in der Medizintechnik vor allem im Zusammenhang mit Prothesen gemacht. Am Institut für Angewandte Informatik wurde die so genannte fluidische Handprothese[7] entwickelt, die neben einem Feedbacksystem über den gegriffenen Gegenstand einige gebräuchliche Griffe beherrscht. Da es die Technik mittlerweile erlaubt, Gliedmaßen wie Hände und ganze Arme durch motorisierte Prothesen oder einen künstlichen Roboterarm zu ersetzen, musste Forschung betrieben werden, um diesen eine sinnvolle Funktionalität zu geben. Dabei wurden für die fluidische Handprothese fünf verschiedene Schlüssel-Griffposen[8] ermittelt.

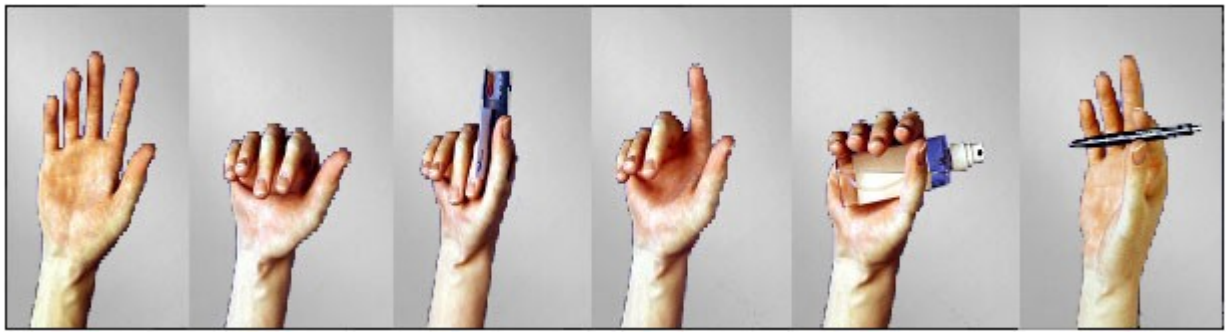

Abbildung 26: Die fünf Griffe der fluidischen Hand (Forschungszentrum Karlsruhe)

Unter Berücksichtigung beider Ergebnisse wurden fünf Grundstellungen für die Finger erarbeitet, die zu den jeweiligen Primitiven passen. Diese können bereits in ausreichender Qualität alle in

Spielen besonders häufig vorkommenden Objekte greifen. Eine weitere, sechste Stellung stellt eine flach auf einer Oberfläche aufliegende Hand dar.

4.4. Interaktionen

Anhand der Liste der am häufigsten vorkommenden Interaktionen soll ein Set aus Teil-Interaktionen entstehen, die das Framework beherrschen muss. Dabei gibt es jedoch Einschränkungen, da nicht alle Interaktionen voll berücksichtigt werden können.

Zum einen wird Handwerksarbeit in allen vorhandenen Spielen entweder durch eine Standard-Arbeitsanimation visualisiert, die unabhängig vom tatsächlich bearbeiteten Objekt und den Zutaten abläuft, oder ohne Animation ausschließlich über ein separates Fenster geregelt. In allen Spielen wird der Handel mit anderen Charakteren ebenfalls in einem separaten Fenster ohne Animation geregelt. Da hier theoretisch eine beliebige große Anzahl von Objekten den Besitzer wechseln kann, ist eine Visualisierung auch nicht sinnvoll. Daher werden diese beiden Aktionen nicht weiter berücksichtigt.

Das Sammeln (Harvesten) ist ein Sonderfall, da, sofern es sich nicht um das einfache Aufnehmen eines Gegenstandes handelt, hierbei einige Gameplay Elemente Einfluss haben. So wird z.B. die Dauer des Sammelns durch die Fähigkeit bestimmt, die der Spieler darin besitzt. Während der Dauer des Sammelns wird wiederholt die gleiche Animation abgespielt. Daher wird auch diese Aktion nicht gesondert behandelt.

Die restlichen Interaktionen lassen sich bei näherer Betrachtung aufteilen in die folgenden Teil-Aktionen. Diese beeinflussen nicht nur den Arm des Charakters, sondern auch die Position und Orientierung des Gegenstandes wird ggf. vom Framework übernommen.

1. Hand zum Objekt führen und zugreifen

2. Objekt in der Hand festhalten, Objekt positionieren
   (verschiedene Trage-Methoden)

3. Hand am Objekt festhalten
   (Hand und Arm folgen dem Objekt)

4. Objekt loslassen / Finger öffnen

Listing 4.4: Die 4 Teil-Interaktionen

Diese vier Basis-Interaktionen soll das Framework beherrschen. Durch Kombination lassen sich alle gängigen Interaktionen darstellen. Einen Gegenstand aufzunehmen lässt sich aus den Aktionen 1 (Hinführen und Greifen) und 2 (Objekt tragen) erstellen. Ein Rad zu drehen funktioniert durch die Kombination von 1 und 3 (am Objekt festhalten), da die Hände auf dem Objekt verankert bleiben. Wird das Rad nun gedreht, entsteht der Eindruck, der Charakter würde das Rad drehen.

Die Aufsplittung in Teil-Aktionen bietet darüber hinaus eine gewisse Flexibilität: Es kann auf Änderungen dynamisch reagiert werden, z.B. durch Abbruch der laufenden und Start einer neuen Aktion. Darüber hinaus können neue Aktionen durch Kombination geschaffen werden.

Bei genauer Betrachtung wird deutlich, dass Aktion 2, das Tragen eines Objektes, in mehreren

Varianten implementiert werden muss, um den verschiedenen Arten von Gegenständen und den benötigten Interaktionen gerecht zu werden. Dabei wird es zwei Grundvarianten geben. Bei der einen wird der Arm während des Tragens vollständig vom Framework animiert, bei der anderen werden nur die Finger animiert und das Objekt in der Hand platziert. Dies ist nötig, um vorhandene Keyframe-Animationen mit verschiedenen Gegenständen zu kombinieren. So kann z.B. eine Kampf-Animation, die den Charakter nach vorne schlagen lässt, mit verschiedenen Gegenständen oder Waffen in der Hand ausgeführt werden.

5. Praktische Umsetzung

Aufgabe des in dieser Arbeit entwickelten IKFrameWork ist die Animation eines oder beider Arme eines skelettbasierten Charakters und, sofern nötig, auch die passende Positionierung und Ausrichtung eines Objektes. Dabei ist es abstrakt genug aufgebaut, um mit verschiedenen Animationssystemen zusammenzuarbeiten.

5.1. Aufbau des Frameworks

Das IKFrameWork ist als DLL angelegt, um leicht in andere Projekte eingebunden werden zu können. Nach einer vollständigen Integration kann das Framework vom Hauptprogramm eingesetzt werden, um für ein Charakter-Objekt-Paar eine Interaktion darzustellen. Alle Kommunikation läuft dabei über das IKFrameWork ab, das die Interaktion verwaltet, bis sie vom Hauptprogramm aus beendet wird. Zum Start einer Interaktion muss das Hauptprogramm dem Framework lediglich ein Charakter-Objekt-Paar und eine gewünschte Aktion mitteilen. (siehe dazu Kapitel 5.4. Funktionsumfang und deren Umsetzung)

Für jedes Charakter-Objekt-Paar erzeugt das IKFrameWork einen eigenen IKSolver, der für alle Berechnungen dieser Interaktion zuständig ist. Das Framework verwaltet eine Liste von IKSolvern, die alle aktuell aktiven Interaktionen repräsentieren. Der IKSolver berechnet abhängig von der Art der Interaktion Animationen für Oberarm, Unterarm, Hand und Finger. In jedem Update wird überprüft, ob die Animationen noch gültig sind und die interpolierten Daten für jeden einzelnen Knochen werden direkt an das Animationssystem übermittelt. Als Schnittstellen werden dabei der Charakter und das Objekt genutzt, die an der Interaktion beteiligt sind.

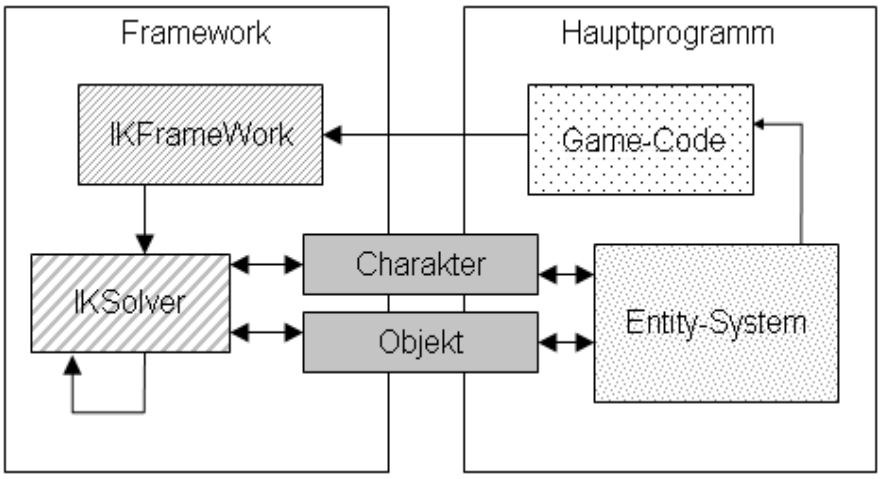

Abbildung 27: Kommunikation zwischen Framework und Hauptprogramm

Die Basisklassen IKCharBase und IKObjBase stellen die Schnittstelle zum Entity- und Animationssystem des Hauptprogramms dar. Alle Objekte, die Gegenstand einer Interaktion sein

können sollen, müssen von der Klasse IKObjBase abgeleitet sein. Alle Charaktere, die skelettbasiert sind und eine Interaktion ausführen können sollen, müssen von IKCharBase abgeleitet sein. Beide Klassen definieren einige abstrakte Funktionen, die bei der Ableitung implementiert werden müssen. Diese Funktionen dienen zur Kommunikation mit dem Entity- und Animationssystem. Sie werden zum Beispiel genutzt, um die Position und Rotation des Objektes abzufragen oder zu setzen.

IKCharBase		IKObjBase	
vector	GetBonePosition(int)	vector	GetPosition()
quaternion	GetBoneRotation(int)	quaternion	GetRotation()
vector	GetPosition()	void	SetPosition(vector)
quaternion	GetRotation()	void	SetRotateion(quaternion)
int	GetRootBone()		
int	GetChildrenCount(int)		
int	GetBoneChild(int, int)		
void	SetBone(...)		

Listing 5.1: Die virtuellen Funktionen in den Basisklassen IKCharBase und IKObjBase
Es sind nur die wichtigsten Funktionen gelistet.

Die in IKCharBase definierte Funktion *SetBone()* ist das Kernstück der Schnittstelle. Diese Funktion wird vom IKSolver jedes Frame mit den aktuellen Transformationsdaten für jeden Knochen, der vom Framework animiert wird, aufgerufen. Das Animationssystem kann das Charaktermodell dann mit diesen Daten passend animieren. Für eine vollständige Auflistung und Erklärung aller Parameter siehe Kapitel 5.7. Animation.

Neben den abstrakten Methoden definiert IKCharBase auch eine virtuelle Funktion *IKStartFrame()*. Die Implementierung dieser Funktion ist optional und dient nur der besseren Zusammenarbeit mit dem Animationssystem. Sie wird vom IKFrameWork für jeden Charakter, der Teil einer Interaktion ist, aufgerufen, bevor die aktualisierten Knochen-Animationen übermittelt werden. Dies dient dazu, um ggf. Vorbereitungen zu treffen, um diese Daten zu empfangen.

Das Framework selbst dient dem Hauptprogramm als Schnittstelle zur Steuerung der Interaktion. Über eine Identifikationsnummer (id) kann auf eine beliebige Interaktion zugegriffen werden, um diese zu beenden, oder zu modifizieren. So kann die Art einer laufenden Aktion gewechselt werden, um beispielsweise einen gegriffenen Gegenstand nun aufzuheben.

Jedes Frame muss die Update Funktion des Frameworks mit einem Zeitintervall aufgerufen werden. Dieser Aufruf muss nach dem Update des Animationssystems erfolgen. Andernfalls kann das Framework nicht auf eine erfolgte Bewegung des Charakters oder des Objektes reagieren. Dadurch würde eine Abweichung von einem Frame entstehen, der zu einer zittrigen Animation oder ungewollten Kollision mit dem Objekt oder sich selbst führen kann. Erfolgt der Aufruf nach der Bewegung des Charakters und des Objektes, tritt dies nicht auf, da das Framework mit den aktuellen Daten arbeiten kann.

5.2. Skelettidentifizierung

Obwohl wegen der einfachen Animierbarkeit mittlerweile nahezu alle Charaktere in Spielen auf einem Skelett basieren, gibt es unter diesen Skeletten große Unterschiede. Es gibt kein einheitliches Skelett, auf dem die meisten der Spielfiguren aufgebaut sind, und das daher als Grundlage für das Framework dienen könnte.

Dennoch benötigt das Framework Informationen über die einzelnen Knochen des Charakters, insbesondere über die Knochen des Armes und der Hand und der Finger. Aber auch die Wirbelsäule ist für Berechnungen relevant, da sie zur Ermittlung der Körperachsen dient, also z.B. zur Bestimmung, wo für die Figur oben ist.

Eine mögliche Lösung für dieses Problem, wäre, das Hauptprogramm diese Informationen an das Framework weitergeben zu lassen, also z.B. die benötigten Knochen zu ermitteln und in ein passendes Format zu konvertieren. Dies bedeutet jedoch zusätzlichen Aufwand für die Entwickler. Darüber hinaus werden auch innerhalb eines Spieles verschiedene Skelette eingesetzt, um den vorkommenden Charakteren gerecht zu werden. Daher wäre der Ablauf für solch eine Knochenbestimmung uneinheitlich und sehr aufwändig.

Um die Arbeit daher weg von den Entwicklern und hin zum Framework zu verlagern, werden das Skelett eines Charakters automatisch analysiert und die benötigten Knochen identifiziert. Als Grundlage dienen eine Reihe unterschiedlicher Skelette von Charakteren aus Computerspielen, sowie das sehr weit verbreitete „Biped“.

Das Biped ist ein vorgefertigtes humanoides Skelett, das bei der Modellierungsumgebung 3D Studio Max© mitgeliefert wird. Es ist vorgefertigt für die Animation humanoider Modelle und bereits mit Winkel-Einschränkungen für die einzelnen Gelenke versehen und wird daher sehr häufig eingesetzt. Durch eine Reihe an Parametern kann dieses Skelett modifiziert werden, so dass selbst Bipeds nicht einheitlich sind. So kann z.B. die Anzahl der Finger und der Fingerknöchel variiert werden, oder die Figur kann einen Schwanz bekommen.

Als Gegenstück zum Biped stehen die manuell erstellten Skelette, die mit einer Vielzahl von Editoren erzeugt werden können. Die dabei auftretenden Unterschiede sind noch größer, als unter den Bipeds. Die Untersuchung aller Testskelette, sowie des Bipeds, ergab eine Liste mit den häufigsten Abweichungen. Auf diese Unterschiede muss bei der Identifikation Rücksicht genommen werden und der Algorithmus darf sich nicht auf diese Knochen stützen. Stattdessen soll die Liste der Gemeinsamkeiten als Anhaltspunkt für die Knochenidentifikation dienen.

- unterschiedliche Anzahl Unterarme
- unterschiedliche Anzahl Wirbelknochen
- teilweise keine Hand, bzw. keine Finger vorhanden
- unterschiedliche Anzahl Schulterknochen
- Zahl der Finger variabel
- Zahl der Fingerknochen variabel
- von der Wirbelsäule kann ein Schwanz abgehen

Listing 5.2: Unterschiede zwischen verschiedenen Skelettmodellen

- Wurzelknochen ist die Hüfte
- min. ein Schulterknochen vorhanden
- die Arme gehen vom gleichen Wirbelknochen ab
- nur ein Oberarmknochen
- die Hand ist immer das Kind des ersten Unterarms

Listing 5.3: Gemeinsamkeiten zwischen verschiedenen Skelettmodellen

Ein Skelett ist ein hierarchisches Konstrukt ähnlich einem Szenengraphen. Ein Knochen hat einen oder mehrere Kinderknochen, und einen Vaterknochen. Dabei geben alle Knochen wie bei einem Szenengraphen ihre Transformationen an die Kinder weiter. Wie bei einem Szenengraphen gibt es einen Wurzelknoten, der keinen Vaterknoten besitzt. Bei allen untersuchten Skeletten sitzt dieser in der Hüfte und an ihm hängt die Wirbelsäule. Von dieser gehen von unten nach oben die Extremitäten (Schwanz, Beine, Arme) ab und sie endet immer mit dem Kopf.

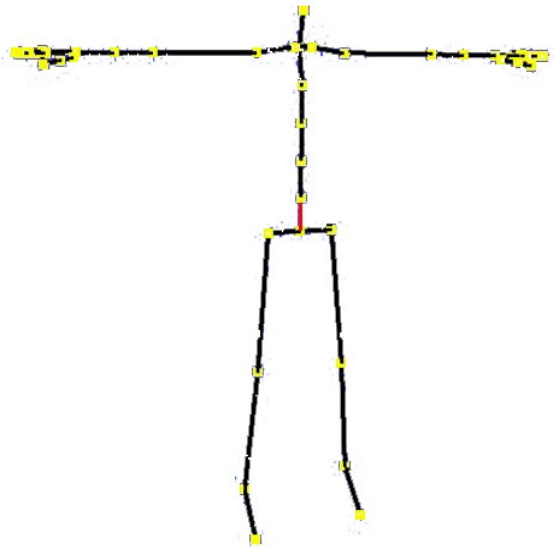

Abbildung 28: Darstellung eines Skelettmodells

Diese Gemeinsamkeit macht sich die Skelettidentifizierung zunutze: Ausgehend vom Wurzelknoten wird entlang der Wirbelsäule nach den Armen gesucht. Da sowohl die beiden Beine, als auch die Arme jeweils vom gleichen Wirbel abgehen, wird der 2. Wirbel gesucht, der mehr als 2 Kinder hat. Ein Wirbel hat in der Regel mindestens ein Kind, nämlich den nächsten Wirbel. Geht von ihm ein Schwanz ab, so hat er zwei. Die Beine sind das erste Paar, das abgeht, daher muss der zugehörige Wirbel mindestens 3 Kinder haben. Da es theoretisch möglich ist, dass sowohl der Schwanz als auch die Beine vom gleichen Wirbel abgehen, wird der erste Wirbel mit mehr als 2 Kindern als der Wirbel definiert, an dem die Beine hängen. Der nächste Wirbel in der Kette mit mehr als 2 Kindern muss dann der gesuchte Wirbel sein, an dem die Arme hängen.

Da die Skelette je nach Beweglichkeit ggf. mehr als einen Schulterknochen aufweisen, muss nun versucht werden, den Oberarm zu identifizieren. Es wird nach zwei Kriterien gesucht: Zum einen nach der Anthropometrie. Anthropometrie ist die Lehre von der Ermittlung und Anwendung der Maße des menschlichen Körpers. In der Norm DIN 33402 [9] werden die menschlichen Proportionen zusammengefasst[6]. Da auch die meisten Modelle in Computerspielen und demzufolge auch deren zugrunde liegenden Skelette nach anatomischen Vorgaben modelliert sind, kann über die Knochenlänge nach Ober- und Unterarm gesucht werden. Der Oberarm ist gefunden, wenn der aktuelle Knochen länger als 85% der Länge des nachfolgenden Knochens ist.

6 Anthropometrie – die DIN Norm spielt eine große Rolle für die Ergonomie, insbesondere bei der Gestaltung von Arbeitsplätzen, Möbeln und von Menschen zu bedienende Maschinen.

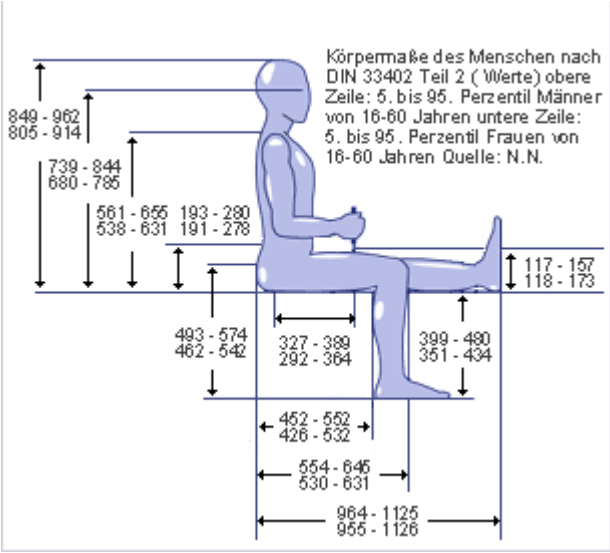

Abbildung 29: Körpermaße des Menschen nach DIN 33402

Das zweite Kriterium stammt aus der Charakter-Modellierung. Um eine bessere Animation des Modells im Unterarmbereich und an der Hand zu erzielen, werden in den Skeletten virtueller Charaktere zusätzliche Unterarme eingebaut, die sich ein wenig mit der Hand mitdrehen und so das Modell des Unterarms zusätzlich verzerren, um optisch schönere Ergebnisse zu erzielen. Diese zusätzlichen Unterarme hängen als Kinder am Oberarm. Daher kann als zweites Kriterium der Oberarm als der Knochen bestimmt werden, der mehr als ein Kind hat.

Abbildung 30: Handrotation ohne und mit Berücksichtigung der zusätzlichen Unterarme

Nach Identifikation des Ober- und Unterarms untersucht der Algorithmus noch die Hand und ihre Kinder, um Informationen über die Anzahl und Länge der Finger zu erhalten. Da viele Figuren in Spielen nicht vollständig animierte Rüstungsteile wie Handschuhe tragen, sind oft weniger als 5

Finger vorhanden, oder die Finger bestehen nur aus zwei Knöcheln. Das Framework kann mit Fingerzahlen von 0 bis 5 und einer Knöcheltiefe von 3 umgehen. Zusätzliche Knöchel oder Finger werden nicht animiert.

Neben der Animation der Arme und Hände dient die Identifizierung auch dazu, später an einem animierten Charakter die relativen Achsen zu ermitteln. Relativ zu diesen Achsen werden die Berechnungen für die Inverse Kinematik ausgeführt. Dabei wird die x-Achse als die Linie von der linken Schulter bis zur rechten Schulter angesehen. Die y-Achse definiert sich durch die Ausrichtung des Wirbelknochens, an dem die Arme angehängt sind. Aus diesen beiden wird mittels Kreuzprodukt die nach vorne zeigende Achse gebildet.

Die Charakteridentifizierung muss zu Beginn für jeden Charakter, der an einer Interaktion beteiligt sein könnte, nur einmalig ausgeführt werden.

5.3. Die fünf Primitive

Die Primitive bilden das Herzstück des Frameworks und sie haben die wichtigste Funktion. Ihre Aufgabe ist es, eine natürlich wirkende Haltung zu erreichen. Sie bestimmen die Position und die Art und Weise, wie ein Objekt gegriffen wird. Sie geben den Zielpunkt vor, mit dem die IK die Armwinkel berechnen kann.

Nachdem dem Framework vom Hauptprogramm ein Charakter-Objekt Paar übergeben wurde, ruft es in einem ersten Schritt das gewählte Primitiv auf, um einen Zielpunkt zu ermitteln. Da jedes Objekt beliebig viele Primitive besitzen kann, kann das zu benutzende Primitiv vom Hauptprogramm ebenfalls vorgegeben werden.

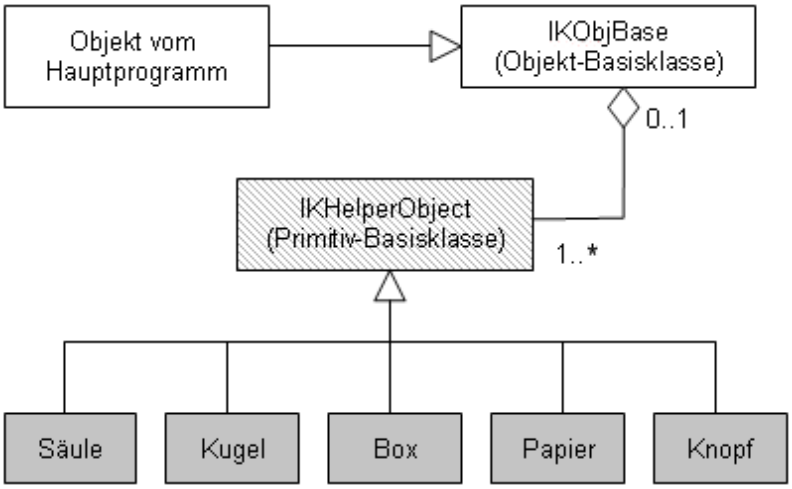

Abbildung 31: UML-Schema der Primitiv-Klassen

Das Primitiv berechnet dann abhängig von seinem Typ einen Berührungspunkt. Ein Berührungspunkt ist der Punkt auf der Oberfläche des Primitives, an dem die Handfläche auf das Objekt treffen soll. Darüber hinaus gehören neben der Position auch noch zwei Vektoren zu dem Berührungspunkt, die den Verlauf der Handachse bestimmen und in Richtung des Handrückens zeigen. Diese beiden Vektoren sind zueinander rechtwinklig.

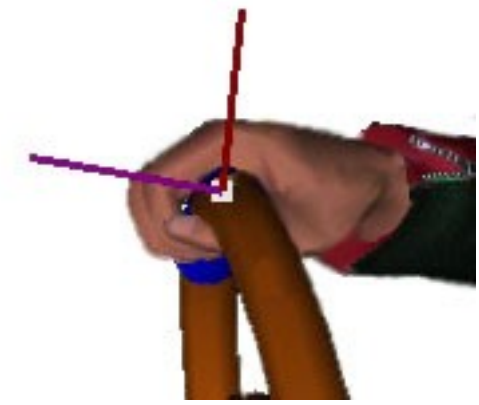

Abbildung 32: Die Handachsen

Die Längsachse (violett) und die Handrückenachse (rot)

Für die Bestimmung der Position des Berührungspunktes gibt es keine eindeutige Lösung. In der Regel gibt es eine ganze Reihe gültiger Lösungen. Jedes Primitiv hat daher eine eigene Methode zur Berechnung der Position.

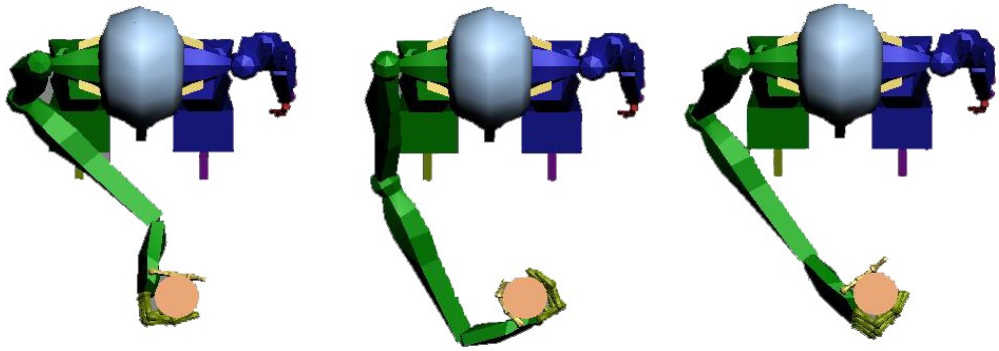

Abbildung 33: gültige Armstellungen

Gezeigt sind nur gültige, für den Menschen einnehmbare Haltungen. Dennoch wirken nicht alle erlaubten Armstellungen auch natürlich.

Der Punkt auf der Oberfläche des Primitives allein reicht noch nicht aus, um die Gelenkstellungen des Armes daraus zu berechnen. Zuerst muss diese Position noch um zwei Größen angepasst werden. Zum einen wird für die Inverse Kinematik nicht ein Zielpunkt für Hand oder Finger benötigt, sondern die Zielposition des Handgelenkes. Daher muss der Berührungspunkt entlang der ermittelten Handachse nach hinten geschoben werden und zwar um etwa 75% der Handlänge, da an dieser Stelle die meisten Objekte berührt werden.

Zum anderen ist die Dicke der Hand zu berücksichtigen. Würde der Arm anhand des ermittelten Punktes einfach direkt berechnet werden, würde der Handknochen, bzw. die Finger, auf dem Punkt an der Oberfläche des Primitives zum Liegen kommen. Die Vertices, die die Polygone des 3D Modells bilden, liegen jedoch um diesen Knochen herum. Als Ergebnis würde die Hand zur Hälfte unter der Oberfläche des Primitives und damit des Objektes verschwinden. Daher muss der Berührungspunkt um die Hälfte der Handdicke nach außen in Richtung des Handrückens verschoben werden.

Abbildung 34: Korrektur des Berührungspunktes

Ohne Korrektur befindet sich die Hälfte der Hand im Inneren des Objektes.
Die Finger sind hier zur besseren Veranschaulichung nicht animiert.

Die Berechnung der Position des Berührungspunktes, sowie der Handachsen, wird passend zum jeweiligen Arm ausgeführt, so dass für den rechten und linken Arm jeweils natürlich aussehende Haltungen eingenommen werden können. Neben diesen Informationen bestimmt das Primitiv auch die passende Griffpose.

Neben einer genaueren Vorstellung der einzelnen Primitive, soll im Folgenden ausführlich erläutert werden, mit welcher Methode jeweils der Berührungspunkt und die Handachsen berechnet werden.

5.3.1. Säulen-Primitiv

Die Säule ist, wie bereits erwähnt, eines der am vielseitigsten einsetzbaren Primitive. Körbe, Schwerter, Äxte, Messer, Türklinken, Lenkräder und Flaschen, sowie alle Arten von Tragegriffen können mit ihr umgesetzt werden. Neben ihrer Position und Rotation muss für die Säule lediglich die Länge und der Radius gespeichert werden. Der menschliche Arm bewegt sich innerhalb eines Kugelraumes um den Schulterpunkt. Diesen Umstand macht sich das Säulen-Primitiv zunutze, um einen gültigen Berührungspunkt zu bestimmen.

Bei der Berechnung wird im Wesentlichen eine Ebene quer durch die Säule gelegt und diese mit der Bewegungskugel des Armes geschnitten. Der entstehende Schnittkreis bestimmt den Berührungspunkt auf der Oberfläche des Primitives. Die Tangente am Schnittpunkt bestimmt den Vektor zum Handrücken und das Kreuzprodukt aus der Flächennormalen und diesem Vektor ergibt die Handachse.

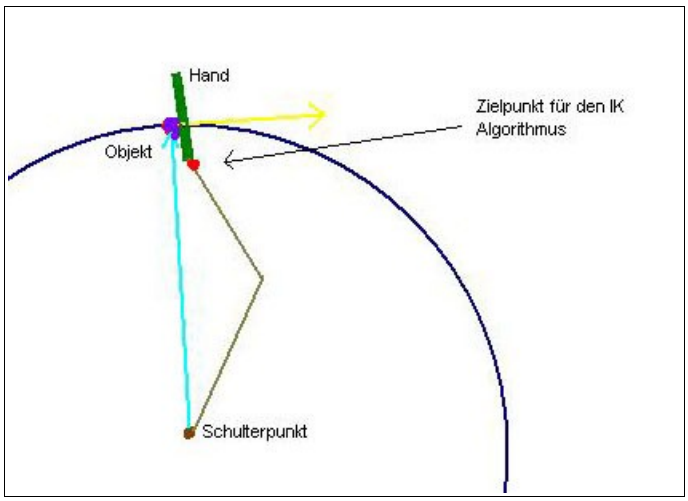

Abbildung 35: Berechnung für das Säulen-Primitiv

Die Flächennormale entspricht dabei der Normalen der Säule, genauer gesagt, deren Mittelachse, die sich wiederum aus der Rotation des Objektes und der des Primitives ergibt. Anhand der y-Achse des Charakters wird diese Normale ggf. noch gewendet, so dass sie relativ gesehen für den Charakter nach oben zeigt. Bei dieser Methode bildet der Schulterpunkt den Mittelpunkt des Kreises. Da ein Arm aber durch das Ellenbogengelenk seinen Handlungsraum nach außen verlagern kann, führt dies in den äußeren Bereichen zu einer zu stark nach hinten gedrehten Handachse. Dieser Umstand wird berücksichtigt, indem bei der Berechnung der Schulterpunkt entlang des seitwärts ausgestreckten Armes auf die halbe Entfernung zum Ellenbogen und nach unten verschoben wird.

Um darüber hinaus die Berechnung zu vereinfachen und keinen Schnittkreis mit einer Kugel bilden zu müssen, wird der verschobene Schulterpunkt in die besagte Ebene projiziert. Aus dem Vektor zwischen dem projizierten Punkt und dem Säulenmittelpunkt ergibt sich direkt die Handachse. Durch Kreuzprodukt mit der Normalen bestimmt sich der Handrückenvektor. Abschließend wird der Berührungspunkt auf der Oberfläche ermittelt, indem der Handrückenvektor mit dem Radius der Säule multipliziert wird und zum Mittelpunkt der Säule addiert wird. So ergibt sich ein Punkt auf dem Rand der Säule. Eine Säule wird allerdings nicht im 90° Winkel von der Seite gegriffen (siehe Abbildung 24: Beispiel-Handstellung am Säulen-Primitiv), daher muss der Vektor, der die Handachse bestimmt, noch um 15° um den Handrückenvektor gedreht werden, um eine natürlich aussehende Handhaltung zu erreichen. Die Griffpose für die Säule wird dazu passend an einer um 15° gedrehten Säule modelliert.

Das Säulen-Primitiv verfügt als eines von zwei Primitiven über die Möglichkeit, die Normale auf Anweisung des Hauptprogramms mit in die Berechnung einzubeziehen. In diesem Fall bedeutet das, dass das Hauptprogramm erzwingen kann, dass der Daumen der Hand beim resultierenden Berührungspunkt immer in die Richtung der ursprünglichen Normale zeigt. Dazu wird die Normale nicht gewendet, selbst wenn sie nach unten zeigt. Dadurch kann bei der Ausstattung des Objektes mit Primitiven optional noch mehr Kontrolle darüber ausgeübt werden, wie das Objekt gegriffen werden soll. Nützlich kann diese Funktionalität z.B. bei Hiebwaffen wie Schwertern oder Äxten sein.

Abbildung 36: Ausrichtung der Daumen in Richtung der Klinge

Ein Schwert wird in der Regel immer so gegriffen, dass der Daumen in Richtung der Klinge zeigt. Wenn die Normale so modelliert wurde, dass sie in Richtung der Klinge zeigt, so kann die Waffe korrekt z.B. aus einem Halfter gezogen werden. Vom Hauptprogramm kann bestimmt werden, ob der Daumen entlang der Normalen oder in die entgegengesetzte Richtung zeigen soll. Die Angabe dieses Parameters ist aber optional.

5.3.2. Kugel-Primitiv

Die Kugel ist in ihrem Einsatzgebiet nicht ganz so vielseitig, wie die Säule. Dennoch wird sie als eigenständiges Primitiv benötigt, da sich einige Griffe nur durch eine Kugel repräsentieren lassen, wie z.B. ein Hebelknauf. Außer einem Radius muss neben Position und Rotation für das Kugel-Primitiv nichts gespeichert werden. Darüber hinaus kann eine Kugel aufgrund ihrer Symmetrie von jeder Richtung aus gegriffen werden. Daher eignet sie sich besonders, wenn der ganze Gegenstand mit dem Primitiv repräsentiert werden soll, wie z.B. bei einer Granate oder einem Apfel.

Die Berechnung des Berührungspunktes verläuft aufgrund der Ähnlichkeit zur Säule vergleichbar ab. Da die Kugel jedoch anders als die Säule kein oben oder unten definiert, wird keine Ebene durch ihren Mittelpunkt gelegt. Der Schulterpunkt wird auf die Hälfte des Oberarmes nach außen geschoben, aber nicht nach unten, wie es bei der Säule der Fall ist. Denn die Handachse wird direkt durch den Vektor von diesem Punkt zum Kugelmittelpunkt bestimmt. Der Daumen zeigt grundsätzlich Richtung der Körpermitte des Charakters. Falls sich das Objekt unterhalb der Schulterhöhe befindet, wird die Handachse dann noch um diese Daumenachse 15° nach hinten, als Richtung des Körpers, gedreht, um eine für die Hand entspanntere Stellung einzunehmen. Aus dem Kreuzprodukt ergibt sich dann der Handrückenvektor.

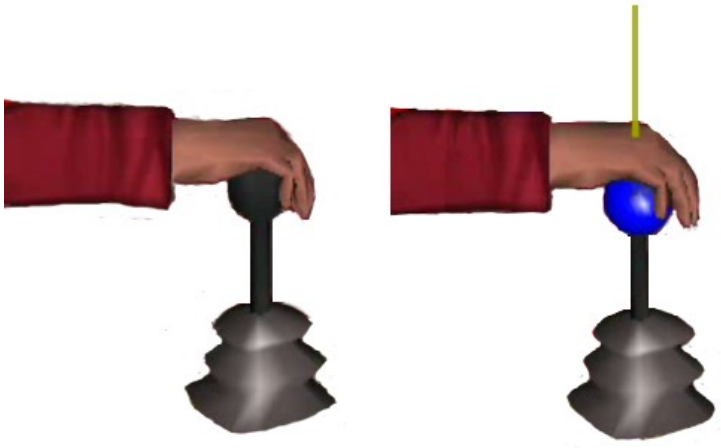

Abbildung 37: Berücksichtigung der Normale beim Kugel-Primitiv

Die Normale erzwingt in diesem Fall eine gerade Handhaltung auf dem Schalthebel. Rechts sind Kugel und Normale eingeblendet.

Als zweites Primitiv kann auch die Kugel optional die Normale berücksichtigen, die sich aus der Rotation des Standardvektors (0, 1, 0) ergibt. Die Normale wird im Editor beim Erstellen angezeigt und kann somit ausgerichtet werden. Beim Kugel-Primitiv bedeutet die Berücksichtigung der Normalen, dass der Handrückenvektor dieser entspricht. Das bedeutet, die Hand legt sich immer so auf die Kugel, dass die Normale durch den Handrücken hindurch 'sticht'. Dies kann vor allem bei Hebeln sehr sinnvoll sein, die eindeutig ein Hauptanwendungsgebiet des Kugel-Primitives darstellen.

5.3.3. Box-Primitiv

Die Box gehört sicherlich zu den am häufigsten eingesetzten Primitiven und eignet sich zur Repräsentation aller Arten von Kisten und sonstigen eckigen Gegenständen, wie Bücher, CDs, Tempo-Packs, Med-Kits, etc. Im Vergleich zu den anderen Primitiven benötigt die Box mehr Daten, um definiert zu werden. Neben Position und Rotation müssen insgesamt drei Kantenlängen gespeichert werden.

Auch die Berechnung eines Berührungspunktes ist für die Box etwas aufwändiger als für die restlichen Primitive. Der Unterschied ist, dass sich bei einer Kiste ab einer gewissen Größe die Art ändert, wie sie getragen wird. Auch die Griffpose ändert sich. Daher wird abhängig von den Kantenlängen die Berechnung unterschiedlich ausgeführt. Entscheidend ist die Spannweite der Hand.

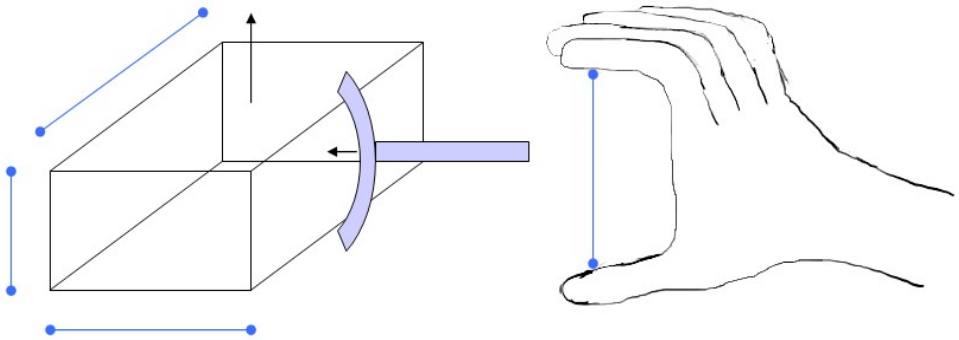

Abbildung 38: das Box-Primitiv und die Spannweite der Hand

Abhängig von der Hand, bzw. der Fingerlänge kann ein Charakter einen Gegenstand mit nur einer Hand greifen oder muss beide Hände gebrauchen. Dies kann bedeuten, dass der eine Charakter einen bestimmten Gegenstand nicht mehr einhändig greifen kann, ein anderer, mit längeren Fingern jedoch dazu in der Lage ist. Als Richtwert gilt das 1,5fache der Handlänge. Ist die kürzeste Seite der Box länger als dieser Wert, so wird die alternative Berechnung durchgeführt.

Ist der Gegenstand mit einer Hand greifbar, wird zunächst die Kante bestimmt, an der die Box gegriffen wird. Dazu werden die Kanten ihrer Länge nach sortiert. Um bequem greifen zu können, wird an einem rechteckigen Gegenstand in der meisten Zahl der Fälle entlang der längsten Kante gegriffen und die Hand schließt sich um die kürzeste Kante. Ein Buch würde man z.B. auf diese Weise greifen.

Ist die längste und die kürzeste Kante ermittelt, so stehen zwei Seiten zur Auswahl, an denen gegriffen werden kann. Es wird dann der Abstand der Seitenmittelpunkte zur Schulter ermittelt und die zur Schulter nähere Seite ausgewählt. Der Seitenmittelpunkt stellt dann den Berührungspunkt dar. Als nächstes wird bestimmt, ob die Hand von oben oder von unten an die Box greifen soll. Für gewöhnlich zeigt der Daumen beim Griff einer Box nach oben. Als Referenzvektor wird der Vektor vom Schulterpunkt bis zum Seitenmittelpunkt genutzt.

Abbildung 39: das Box Primitiv auf einem Buch

Die Seite an der gegriffen wird,
wird automatisch gewählt

Danach werden die Handachsen bestimmt. Als initiale Achsen stellt der Vektor vom Mittelpunkt der Box zum Berührungspunkt den Handrückenvektor dar, und der Kantenvektor entlang der längsten Seite, der vom Körper weg zeigt, stellt die Handlängsachse dar. Um eine natürliche Handhaltung einzunehmen, werden die Vektoren 40° nach innen gedreht (Hand wird nach unten geklappt, Flexion) und 15 Grad nach unten (Hand wird nach außen gedreht, Ulnarabduktion). Dazu wird eine passend modellierte Griffpose gewählt.

Kann die Box nicht mit einer Hand umschlossen werden, wird, wie bereits erwähnt, eine alternative Berechnungsform gewählt. Diese geht davon aus, dass die Box oder Kiste so groß ist, dass sie eigentlich mit zwei Händen getragen werden muss. Daher wird der Berührungspunkt so berechnet, als ob eine zweite Hand vorhanden wäre. Dies muss jedoch nicht zwingend der Fall sein, siehe dazu auch Kapitel 5.4. Funktionsumfang und deren Umsetzung.

Es gibt mehrere, sich unterscheidende Arten, eine Box mit zwei Händen zu tragen. Die hier gewählte Methode wird jedoch in Einzelfällen nur mit einer Hand ausgeführt werden. Darüber hinaus kann über die Intention der Interaktion keine Aussage getroffen werden. Es ist also unklar, ob die Box berührt wird, um sie zu tragen, oder ob sie z.B. zur Seite geschoben werden soll. Daher müssen Berührungspunkt und Griffpose so gewählt werden, dass sie für all diese Fälle eine gewisse optische Plausibilität liefern. Erreicht wird dies mit einem flachen Griff auf einer oder beiden Seiten der Kiste.

Die Hände befinden sich dabei auf gegenüberliegenden Seiten auf der Kiste. Bestimmt werden die Seiten anhand eines Vergleiches der Kanten mit der x-Achse des Charakters, also mit der Linie von der linken Schulter zur rechten. So wird das Seitenpaar gewählt, bei dem der Vektor von einem Mittelpunkt zum anderen den geringsten Unterschied zu dieser x-Achse aufweist. Die Handachsen ergeben sich aus den übrigen Kantenvektoren der Box. Dabei zeigen die Daumen nach oben und die übrigen Finger weg vom Körper.

Als Berührungspunkt gilt der Mittelpunkt der Seite, die der jeweiligen Schulter am nächsten gelegen ist. Bei sehr großen Kisten muss die Handposition jedoch noch korrigiert werden. Ist die Kiste an der Seite, an der die Hand aufliegt, in der Tiefe (also entlang der Handachse) mehr als 3mal so lang, wie die Hand, so wird der Berührungspunkt so weit entlang der Handlängsachse Richtung Körper verschoben, dass die Hand nicht tiefer als bis zu ihrer 1,5fachen Länge in die Kiste greift.

Abbildung 40: zweihändiges Tragen einer Kiste

Die Hände liegen nicht mittig auf den Seiten auf, sondern sind ein Stück zum Körper geschoben.

5.3.4. Kreditkarten-Primitiv

Das Kreditkarten-Primitiv ist das erste der beiden 2D Primitive. Die Griffposen dieser Primitive sind starr und werden nicht abhängig von der Größe des Primitives angepasst. Die Kreditkarte steht stellvertretend für alle dünnen Karten, Papier oder Pappe-ähnlichen Gegenstände, wie Kreditkarten, Ausweise, Tabletts oder Zettel. Aber auch andere, eher selten vorkommende, dünne Gegenstände lassen sich mit ihm repräsentieren, wie z.B. Ringe und Schlüssel.

Das Primitiv besteht nur aus einer Kante und einem Anzeiger, in welcher Richtung sich der Gegenstand befindet. Über die Kante wird der Gegenstand dann gegriffen.

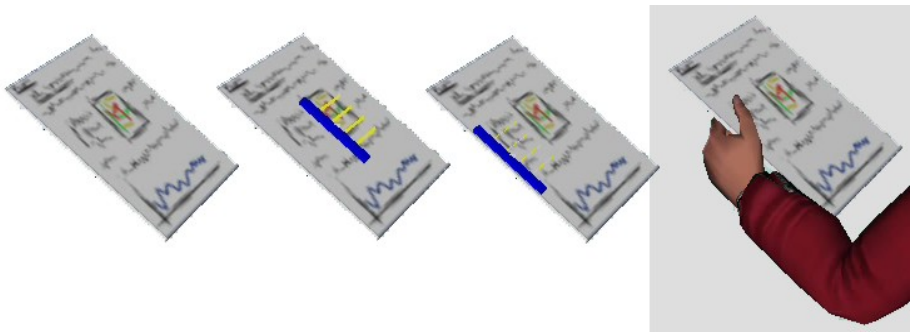

Abbildung 41: Ausrichtung des Kreditkarten-Primitives

Der Berührungspunkt befindet sich in der Mitte der Kante, eine Berücksichtigung der Kantenlänge findet nicht statt. Dies dient dazu, beim Platzieren des Primitives mehr Kontrolle darüber zu haben, wo genau das Objekt gegriffen wird.

Die Handlängsachse entspricht dem Kantenvektor, der vom Körper weg zeigt. Der Handrückenvektor zeigt in die entgegengesetzte Richtung des Objektinnern und wird aus der Rotation des Standardvektors (1, 0, 0) gewonnen. Um eine natürliche Handhaltung zu erreichen, wird die Hand um den Handrückenvektor noch einige Grade nach unten gedreht.

5.3.5. Knopf-Primitiv

Das Knopf Primitiv ist ebenfalls ein 2D Primitiv. Es dient dazu, eine Stelle auf einer Oberfläche zu markieren, auf die der Zeigefinger zeigen soll. Haupteinsatzgebiet sind daher Knöpfe und Schaltflächen. Da die Normale auf dem Quadrat auch gleich die Seite definiert, aus der auf die Fläche gezeigt werden soll, benötigt dieses Primitiv keine weiteren Daten neben seiner Position und Rotation.

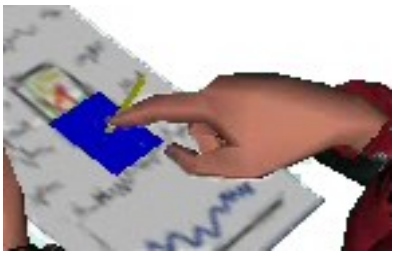

Abbildung 42: das Knopf-Primitiv

Der Berührungspunkt ist etwas aufwändiger zu berechnen, da er als ein Punkt auf der Handfläche definiert ist, das Knopf-Primitiv jedoch so konzipiert ist, mit der Fingerspitze berührt zu werden. Daher wird über die Länge der Fingerknochen und der Krümmung des Zeigefingers in der modellierten Griffpose die Handposition rückgerechnet. Dabei wird davon ausgegangen, dass das

letzte Glied des Zeigefingers parallel mit der Normalen läuft. Um daraus den Handlängsachsenvektor zu bestimmen, wird die Normale um 45° in Richtung der Schulter gedreht. Genau wie beim Kreditkarten-Primitiv ist die Griffpose für den Knopf starr.

5.4. Funktionsumfang und deren Umsetzung

Um das Framework einzusetzen und eine Interaktion zu visualisieren, muss das Hauptprogramm nur mit dem IKFrameWork kommunizieren. Um eine neue Interaktion zu starten, muss es mindestens einen Charakter, der von der Klasse IKCharBase abgeleitet ist, sowie ein Objekt, das von der Klasse IKObjBase abgeleitet ist, an das Framework übergeben. Das Objekt muss darüber hinaus mindestens ein Primitiv besitzen. Darüber hinaus können aber optional noch weitere Angaben an das Framework gemacht werden.

```
int NewInteraction (IKCharBase *chr, IKObjBase *obj,
        float duration = 0.6f, IK_ARM arm = RIGHT_ARM,
        IK_ACTION_TYPE action = GRAB, unsigned int primitiveNr = 0,
        int direction = 0, int leftArm = -1);
```

Listing 5.4: Funktionskopf der Funktion *NewInteraction()*

Bei den ersten beiden Parametern handelt es sich um den Charakter, der die Aktion ausführen soll, und das Objekt, das Gegenstand der Aktion sein soll. Diese Angaben sind die minimalen Daten, die das Framework benötigt. Ist das Skelett des Charakters nicht identifiziert, oder verfügt das Objekt über keine Primitive, so wird die Aktion abgelehnt. In diesem Fall liefert die Funktion einen negativen Wert zurück. In allen anderen Fällen wird eine Identifikationsnummer zurückgeliefert, über die das Hauptprogramm auf die Interaktion zugreifen kann.

Optional können aber noch eine Reihe zusätzlicher Parameter genutzt werden. Der Wert *duration* gibt in Sekunden an, wie lange die Interaktion dauern soll. Dieser Wert ist nur relevant bei der Aktion, bei der der Charakter nach dem Objekt greift. Er gibt an, wann die Hand um das Objekt geschlossen sein soll, also die Gesamtdauer der Animation. Der Standard-Wert von 0.6 Sekunden ist optisch ermittelt worden. Das IKFrameWork stellt zusätzlich eine Funktion *bool HasGrabbed(int* id*)* zur Verfügung, die immer dann 'true' zurück liefert, wenn der Charakter das Objekt mit der Hand umschlossen hält.

Der Parameter *arm* definiert, mit welchem Arm die Aktion ausgeführt werden soll. Hier kann zwischen rechtem Arm, linken Arm und beiden gewählt werden. Um eine Aktion mit zwei Armen auszuführen, gibt es zwei Lösungen. Die eine besteht darin, einfach für jeden Arm eine eigene Interaktion zu starten. Dies gibt einem die Möglichkeit, die Arme unabhängig voneinander zu manipulieren, d.h. die Aktion nur eines Armes zu beenden oder zu verändern. Die andere Möglichkeit besteht darin, an diesen Parameter *BOTH* zu übergeben. Das Framework erzeugt intern ebenfalls zwei IKSolver, stimmt diese aber aufeinander ab. Darüber hinaus werden sie über die gleiche id verwaltet. Beide Arme greifen in der Standardeinstellung auf das gleiche Primitiv zu.

Die Angabe, welche Arme für die Visualisierung verwendet werden sollen, muss vom Hauptprogramm erfolgen. Selbst, wenn ein zweiter Arm benötigt werden würde, z.B. um ein sehr großes Objekt zu tragen, so darf das Framework den anderen Arm nicht selbstständig mit animieren. Dies liegt darin begründet, dass die Kontrolle über die Animation beim Hauptprogramm verbleiben muss. Das Framework kann keine Aussage darüber treffen, ob der zweite Arm gerade

eingesetzt werden darf. In Ego-Shootern hält die rechte Hand z.B. in der Regel eine Waffe und alle Aktionen werden mit der linken Hand ausgeführt. Daher beschränkt sich das Framework auf den Arm, der beim Aufruf von *NewInteraction()* angegeben wurde.

Der fünfte Parameter gibt die Aktion an, die ausgeführt werden soll. Wie in der Anforderungserhebung ermittelt, beherrscht das Framework verschiedene Teilinteraktionen. Die möglichen Aktionen sind Greifen, Festhalten und verschiedene Versionen von Tragen. Die einzelnen Teilaktionen werden in Kapitel 5.4.1. noch genauer erläutert. Da der überwiegende Teil aller Interaktionen damit beginnt, dass der Charakter seine Hand zum Gegenstand führt, ist das Zugreifen der Standardwert für diesen Parameter. Durch eine weitere Funktion kann der Typ der Aktion jederzeit geändert werden.

Verfügt das Objekt über mehr als ein Primitiv, so kann mit dem Parameter *primitiveNr* der Index des gewünschten Primitives angegeben werden. Dies kann für eine Reihe von Anwendungen notwendig sein, z.B. um jeden Arm an eine andere Stelle des Objektes greifen zu lassen, z.B. verschiedene Knöpfe auf einer Schalttafel, oder, durch eine geschickte Reihenfolge der Primitive, einen bestimmten Effekt zu erzielen, wie z.B. das Umgreifen an einem Lenkrad. Ist der angegebene Index ungültig, so wird stattdessen automatisch das erste Primitiv ausgewählt, daher ist auch der Standardwert 0.

Abbildung 43: Umgreifen am Lenkrad mit eingeblendeten Primitiven

Durch geschickte Anordnung der Primitive am Lenkrad entsteht der Effekt, als greife der Charakter beim Drehen des Lenkrades mit den Händen um. Tatsächlich wird, wenn das Handgelenk zu tief steht, die HOLD-Aktion vom Hauptprogramm beendet und eine neue GRAB-Aktion mit einem Primitiv mit einem höheren bzw. tieferen Index gestartet.

Die letzten beiden Parameter spielen nur unter besonderen Umständen eine Rolle. Bei dem Wert *direction* kann angegeben werden, ob und wie die Normale vom Primitiv bei der Berechnung des Berührungspunktes berücksichtigt werden soll. Dies ist nur relevant, wenn es sich bei dem Primitiv um eine Säule oder eine Kugel handelt. Der Standardwert von 0 bedeutet, dass die Normale keine Rolle spielt. Ist der Wert positiv oder negativ, so richtig das Säulen-Primitiv den Daumen entsprechend des Vorzeichen entlang oder entgegengesetzt zur Normalen aus. Die Kugel legt die Normale durch den Handrücken. Bei ihr ist es irrelevant, ob der Wert positiv oder negativ ist, da dies auf das Ergebnis keinen Einfluss hat. Wurde bei der Angabe, mit welchem Arm die Aktion ausgeführt werden soll, festgelegt, dass beide Arme zum Einsatz kommen, so wird der hier angegebene Wert für den rechten Arm genutzt und für den linken Arm invertiert.

Der letzte Parameter, *leftArm*, ist nur relevant, wenn bei *arm* angegeben wurde, dass beide Arme für die Interaktion eingesetzt werden sollen. Hier kann optional für die linke Hand ein anderes Primitiv angegeben werden, falls nicht beide Hände auf das gleiche Primitiv greifen sollen. Wurde dieser Parameter angegeben, so wird der Parameter *direction* nicht für die linke Hand invertiert, sondern ebenfalls direkt weiterverwendet. Daher kann *leftArm* auch dazu eingesetzt werden, beide Hände auf das gleiche Primitiv und in die gleiche Richtung zugreifen zu lassen, indem hier das gleiche Primitiv wie bei *primitiveNr* angegeben wird.

5.4.1. Die Aktionen

Wie in der Anforderungserhebung ermittelt, muss das Framework vier verschiedene Teil-Interaktionen beherrschen. Im Folgenden soll deren Umsetzung näher erläutert werden.

Die Aktion Greifen trägt die Bezeichnung *GRAB*. Sie führt die Hand zum Objekt und schließt die Finger um das Primitiv. Direkt nach dem Erzeugen eines IKSolvers lässt dieser vom gewählten Primitiv einen Berührungspunkt berechnen, zu dem der Arm, bzw. die Hand dann hingeführt wird. Dabei durchläuft der Arm mehrere Animationsphasen, Näheres dazu in Kapitel 4.3. Griffposen.

Hat sich die Hand um das Objekt geschlossen, wird die Aktion *GRAB* beendet und geht automatisch über die Festhalten-Aktion *HOLD*. Ab diesem Zeitpunkt liefert ein Aufruf der Funktion *HasGrabbed()* immer 'true' zurück. Im Festhalte Modus bleibt die Hand fest in der gewählten Griffpose auf dem Objekt, als wenn sie dort angeklebt wäre. Wird das Objekt bewegt, so wird der Arm neu berechnet, damit die Hand dem Objekt folgen kann. (siehe Abbildung 43)

Dies kann für eine Reihe von Aktionen, die sehr genau sein müssen, ausgenutzt werden. Um zum Beispiel eine KeyCard exakt durch ein Lesegerät zu führen, muss lediglich die Karte durch das Gerät gezogen werden. Da der Arm und die Hand sich automatisch mit ihr bewegen, entsteht der Eindruck, als zöge die Hand die Karte durch das Gerät. Weitere Einsatzmöglichkeiten sind Lenkräder, die gedreht werden oder etwa Hebel, die umgelegt werden. Um dem Hauptprogramm zusätzliche Informationen zu bieten, lässt sich die aktuelle Position des Handgelenkes, bzw. der Handfläche (sofern Finger vorhanden sind) über eine zusätzliche Funktion abfragen. Selbstverständlich wäre dem Hauptprogramm auch eine direkte Abfrage der Position des Handknochens möglich, doch dazu muss die id dieses Knochens bekannt sein. Da das Framework eine Skelettidentifizierung durchführt, kann es diese nützliche Information zuverlässig für alle Charaktere bieten.

Zusätzlich zum Festhalten am Objekt beherrscht das Framework mehrere Methoden, einen Gegenstand, der gegriffen wurde, zu tragen. In allen Trage-Modi bestimmt das Framework die Position und Rotation des Objektes während der Aktion. Über abstrakte Funktionen in der Basisklasse (siehe Kapitel 5.1. Abschnitt Basisklassen) wird diese gesetzt. Die Funktionalität hängt daher von einer korrekten Implementierung dieser Funktionen ab.

Da gerade in modernen Spielen nahezu alle Gegenstände physikalisiert sind, kann unter Umständen die Position des Objektes nicht direkt gesetzt werden, solange die Physikberechnung durchgeführt wird. Das Ergebnis können grafische Fehler in der Form hin und her springender Objekte sein. Um dies zu vermeiden wurde in die Basisklasse IKObjBase eine virtuelle Funktion *IKStartCarrying()* eingebaut. Es ist nicht zwingend erforderlich, diese Funktion in den abgeleiteten Objekten zu implementieren. Sie dient lediglich der besseren Abstimmung mit einer eventuell vorhandenen Physik-Engine. Sie wird vom IKFrameWork aufgerufen, sobald dieses den Auftrag bekommt, das Objekt zu tragen und soll dem Hauptprogramm die Gelegenheit geben, die Physikalisierung des Objektes temporär zu deaktivieren. Wird die Interaktion beendet oder zu einer Aktion gewechselt, die nicht die Position des Objektes setzt, so wird die Funktion *IKStopCarrying()* aufgerufen, in der die Physikalisierung wieder eingesetzt werden kann. Dies macht es möglich, einen Gegenstand z.B. erst aufzunehmen und anschließend zu werfen.

Um verschiedene Arten von Interaktionen darstellen zu können, beherrscht das Framework diverse Tragemodi. Die Standard-Variante *CARRY* ist dazu gedacht, das Objekt zu tragen und gleichzeitig eine Forward/Keyframe Animation abzuspielen, die auch den Arm bewegt. Das Framework überschreibt lediglich die Finger mit der korrekt angepassten Griffpose. Außerdem platziert es das Objekt korrekt in der Hand. Dazu wird zum einen ein Berührungspunkt auf dem Objekt errechnet. Zum anderen werden anhand des animierten Skelettes der Punkt auf der Handfläche, an dem das Objekt aufliegen soll, bestimmt, sowie die Handachsen errechnet. Dann wird das Objekt so in der

Hand platziert und rotiert, dass der Berührungspunkt mit seinen Achsen mit dem Punkt auf der Hand und den ermittelten Handachsen aufeinander zum Liegen kommt.

Dieser Trage-Modus kann bei einer Reihe von Aufgaben eingesetzt werden. Allen voran hat jeder Charakter in der Regel eine eigene, generische Trage-Animation in Form eine Forward-Animation. Dabei wird die Hand meist vor dem Körper gehalten und der Gegenstand irgendwie darin platziert. Bei Ego-Shootern muss die Hand dabei so hoch gehalten werden, dass sie am Rande des Blickfeldes erscheint. Ein anderes Einsatzgebiet ist z.B. das Ausführen einer Angriffsanimation, z.B. eines Schlages. Mittels des Frameworks kann ein beliebiger Gegenstand wie eine Waffe in der Hand platziert werden. Unter Verwendung einer Werfen-Animation kann jeder Gegenstand geworfen werden.

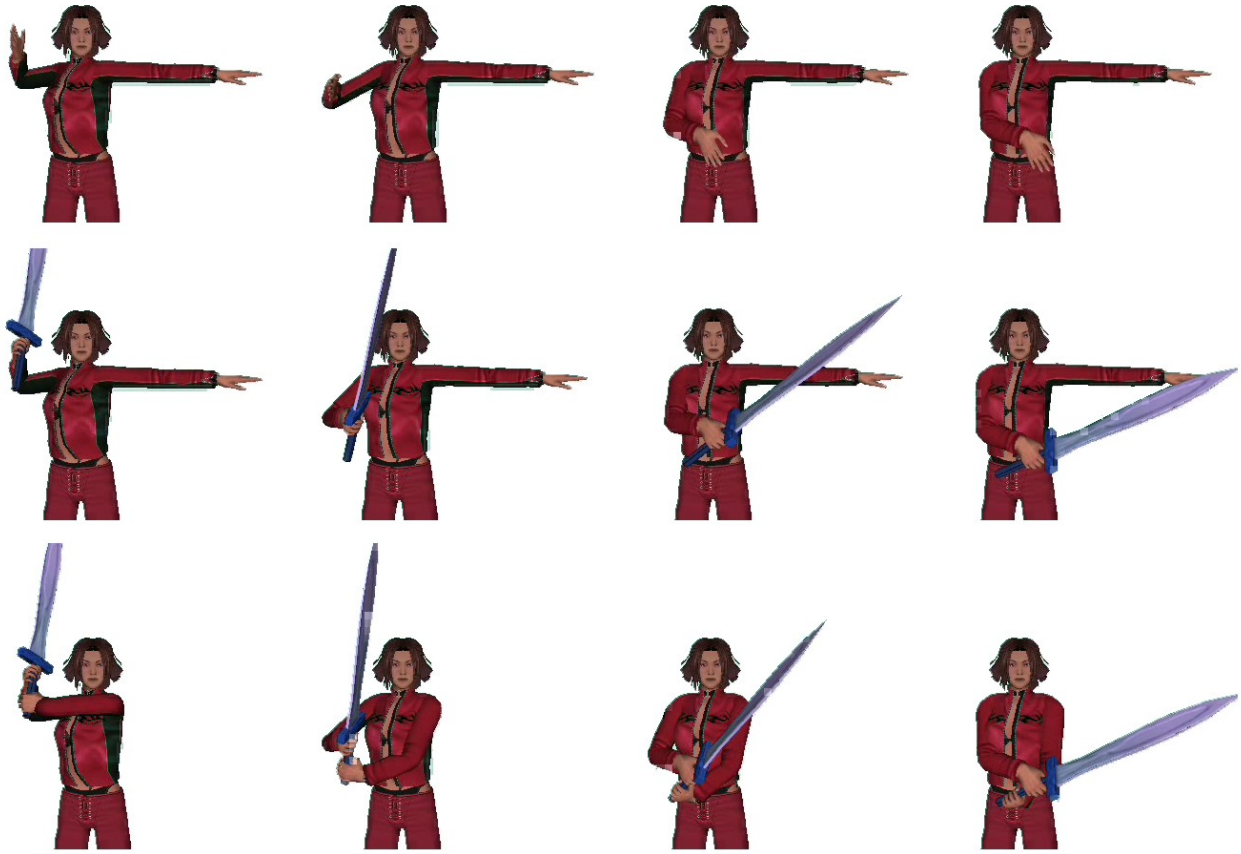

Abbildung 44: Kombinationsmöglichkeiten mit Forward-Animationen

In der obigen Bilderreihe wurde die Trage-Interaktion mit der rechten Hand gestartet. Anschließend wurde eine zweite Interaktion mit der linken Hand gestartet, die den Auftrag *HOLD* erhielt. Der rechte Arm wird dann mit der Schlag-Animation animiert. Das Framework übernimmt die Finger-Animation, platziert das Schwert in der Hand und steuert den kompletten linken Arm, so dass der Eindruck entsteht, es würde ein zweiarmiger Schlag mit dem Schwert ausgeführt werden.

Dabei bestimmt die Position der rechten Hand, wie und wo das Objekt platziert wird. Ist gewünscht, dass der linke Arm die Position vorgibt, so muss nur die entsprechende Interaktion mit der Aktion *CARRY* für den linken Arm, statt dem rechten gestartet werden und die Interaktion für den rechten Arm den Auftrag *HOLD* erhalten. Werden zwei Interaktionen gestartet, die beide den Auftrag *CARRY* erhalten, so wird automatisch die zuletzt erzeugte Interaktion die Positionierung des Objektes übernehmen.

Wird der *CARRY* Auftrag an eine Interaktion gegeben, die mit beiden Armen ausgeführt werden soll, so wird ein spezieller Zweiarm-Modus eingesetzt. Dabei wird das Objekt vor den Körper in einem passenden Abstand vor dem Brustkorb platziert. Dann werden die Hände jeweils auf die errechneten Berührungspunkte gelegt und ausgerichtet. Tatsächlich wird also eigentlich das Objekt platziert und die Hände halten sich lediglich daran fest. Das Objekt wird dabei so gedreht, dass die Arme in etwa den gleichen Abstand vom Körper haben, da dies am ehesten einer generell natürlichen Haltung entspricht. Ausgangspunkt dieser Drehung ist die Achse von einer Schulter zur anderen, an der die Linie, die von einem Berührungspunkt zum anderen läuft, ausgerichtet wird. Wenn der Charakter in einer natürlichen Geh-Animation die Schultern mitschwingt, wird auch das Objekt mitgedreht, was einen sehr realistischen Eindruck erweckt.

Neben dem Standard Trage-Modus, in dem lediglich die Finger animiert werden, gibt es eine Unterform, die auch den Handknochen mitanimiert. Dieser Modus nennt sich *CARRY_STEADY* und soll dazu dienen, Gegenstände gerade zu halten, wie z.B. ein Wasserglas oder eine Teekanne.

Dazu werden beim Aufnehmen des Objektes dessen aktuelle Rotation und die „up-Axis“ des Charakters festgestellt. Anschließend wird das Objekt nur noch um diese Achse herum rotiert, so dass verhindert wird, dass es zur Seite kippen kann. Bei der „up-Axis" des Charakters handelt es sich nicht um die y-Achse, die zur Berechnung in der Inversen Kinematik eingesetzt wird. Diese ist hier nicht verwertbar, da der Charakter z.B. nach vorne gebeugt sein könnte, um das Objekt aufzunehmen. In diesem Fall wäre das Ergebnis nicht korrekt. Die „up-Axis“ wird bei der Skelettidentifizierung in der Nullstellung des Skelettes ermittelt und verändert sich nur mit der Gesamtrotation des Charakters, nicht aber mit seiner Animation. Das bedeutet, wenn die „up-Axis“ beim Charakter in der Nullstellung nach oben zeigt, so zeigt sie auch dann noch nach oben, wenn der Charakter gerade der Länge nach auf dem Boden liegt. Befindet sich der Charakter jedoch in Schwerelosigkeit, oder geht z.B. mit Magnetstiefeln an der Decke entlang, so ist auch die „up-Axis“ korrekt gedreht.

Abbildung 45: eine Teekanne gerade halten

Die dritte Tragevariante ist ein Spezialfall, der eher selten gebraucht wird, der aber der Vollständigkeit halber ebenfalls unterstützt wird. Der Modus CARRY_AS_IS lässt den Charakter den Gegenstand so tragen, wie er aufgenommen wurde, mit seiner Rotation und Position relativ zum Körper. Dazu wird der Arm in der Stellung eingefroren, in der er sich beim Aufnehmen des Objektes befindet. Diese Variante kann zum Beispiel eingesetzt werden, wenn spezielle Gegenstände in einem bestimmten Abstand zum Körper getragen werden sollen.

Als vierte Teilaktion muss der Charakter ein gegriffenes Objekt auch wieder loslassen können. Dazu muss im Animationssystem die aktuelle Haltung des Armes und der Hand in eine andere Animation oder Haltung übergeblendet werden. Dies geschieht mithilfe von Gewichtungen, die bei der Berechnung der finalen Skelettposition einfließen. Da das IKFrameWork selbst keine Gewichtungen für seine Knochendaten liefert, sondern die Stellung der Knochen einfach

überschreibt, bzw. mit Gewichtung 1 in die Berechnung einfließt, und darüber hinaus das Framework auch die Animation, in die überblendet werden soll, nicht kennt, muss dieses wie üblich im Animationssystem stattfinden. Damit dies geschehen kann, darf das Framework jedoch keine weiteren Knochendaten schicken. Die Funktion *TerminateInteraction(int* id*)* beendet daher die Interaktion und erlaubt es dem Animationssystem, den Arm und die Hand in eine andere Stellung zu bewegen.

Um möglichst robust zu sein, überprüft das Framework alle Daten ständig auf ihre Gültigkeit. Gibt es einen Fehler beim Erstellen einer Interaktion, so wird eine negative id zurückgegeben. Tritt während des Ablaufes ein anderer Fehler auf, z.B. bei der Skelettidentifizierung, so wird eine Fehlermeldung in eine Log-Datei geschrieben und je nach Schwere des Fehlers ein Abbruch erzwungen oder das Programm regulär fortgesetzt.

5.5. Griffposen

Um auch die Finger korrekt animieren zu können, enthält das Framework mehrere Griffposen, die alle von der Basisklasse IKGripPose abgeleitet sind. Ausgewählt wird die entsprechende Griffpose vom Primitiv. Die Aufgabe jeder Griffpose ist es, sich an die Größe des zu greifenden Primitives anzupassen und die korrekte Fingerstellung zu finden. Dabei muss neben der Stellung der Finger, die das Objekt, bzw. das Primitiv, umschließen, auch eine Stellung für die geöffneten Finger bereit gestellt werden, da sich eine Hand, bevor sie nach etwas greift, zunächst erst einmal öffnet. Auch diese Öffnung geschieht abhängig von der Primitivgröße unterschiedlich weit.

Dazu wurden für jede Griffpose zwei Fingerhaltungen in einem 3D Editor modelliert. Die eine Animation stellt die maximale Schließung der Hand für die gewählte Griffpose dar, die andere die maximale Öffnung. Abhängig von der Fingerstellung berechnet jede Griffpose daher auf eine andere Art und Weise, welchen Umfang sie maximal und minimal umgreifen kann. Den Umfang des Griffes erhält sie vom Primitiv. Bei der Säule ist dieser Wert z.B. der Durchmesser, bei der Box hingegen die Länge der kürzesten Seite. Anhand dieses Wertes und der minimalen und maximalen Spannweite des Griffes, kann die Griffpose mittels linearer Interpolation zwischen den beiden Animationen eine finale Animation für die Finger errechnen, die das Primitiv umschließt. Durch eine Skalierung des Wertes wird anschließend noch eine Fingerhaltung für die geöffnete Hand ermittelt, da das Zugreifen der Hand nicht aus der Neutral-Nullstellung heraus geschieht.

Die Griffpose für den Button (ausgestreckter Zeigefinger) und für die große Box (flache Hand) bilden dabei Sonderfälle: Für sie ist nur eine Fingerstellung modelliert, da sie nicht umgreifen müssen und daher auch keine vorherige Öffnung der Finger stattfindet. Daher findet in diesen Klassen auch keine weitere Berechnung statt.

Wegen des Aufwands beim Erstellen und Skinning[7] des Charakters werden häufig nicht alle Finger mit allen Knöcheln modelliert. Trägt ein Charakter z.B. Handschuhe, ist dies auch gar nicht nötig. Darüber hinaus bewegen sich die drei hinteren Finger, Mittel-, Ringfinger und kleiner Finger, in der Regel gleichförmig, so dass sie von einem einzelnen Knochen animiert werden können. Daher haben Skelette oft nur drei Finger und je nach Komplexität der vorhandenen Animationen unterschiedlich viele Knöchel.

Alle Animationen im Framework sind für die Maximalzahl von fünf Fingern mit je drei Fingerknöcheln erstellt worden. Angewandt werden können sie jedoch auf jede beliebige Hand. Sind keine Finger vorhanden, bleibt die Griffpose untätig. Ist die Finger- oder Knöchelzahl

7 Skinning – Zuweisung der Eckpunkte eines 3D Modells zu einem oder mehreren Knochen eines Skelettes, die sie animieren sollen. Der Fingerknochen des Mittelfingers darf z.B. nur Eckpunkte vom Mittelfinger beeinflussen. In aller Regel müssen diese Zuweisungen bei so kleinen Strukturen wie Fingern mühsam von Hand gemacht werden.

niedriger, so werden die vorhandenen Animationen angewandt, ausgehend vom Daumen. Selbst bei einer Hand mit nur drei oder vier Fingern, ist davon auszugehen, dass der am weitesten innen liegende Finger am ehesten dem Daumen entspricht, und der zweite Finger am ehesten dem Zeigefinger. So wird ein korrektes Verhalten selbst bei unterschiedlicher Fingerzahl gewährleistet. Sollte die Finger- oder Knöchelzahl höher sein, so werden die zusätzlichen Gliedmaßen nicht berücksichtigt.

5.6. IK Berechnungen

Es wurde eine eigene, auf die speziellen Gegebenheiten und Ziele des Frameworks zugeschnittene, IK implementiert, die die Winkelstellungen der Arme anhand der von den Primitiven vorgegebenen Zielpunkte berechnet. Als erster Schritt musste zunächst das Skelett des Charakters in eine klar definierte Ausgangshaltung gebracht werden. Diese wird durch die Achsen des Charakters selbst definiert und ist daher abhängig von seiner aktuellen Körperhaltung. Siehe dazu auch Kapitel 5.2. Skelettidentifizierung. Die Nutzung dieser relativen Achsen ist notwendig, um den tatsächlichen Bewegungsspielraum des Armes zu nutzen. Dies ist insbesondere bei der Anpassung des Armes an die Handwinkel notwendig, um eine natürliche Haltung einzunehmen.

Da der Arm immer aus seiner aktuellen, animierten Haltung heraus bewegt werden soll, muss zunächst eine Rotation für das Schultergelenk ermittelt werden, die den Arm, für den die Berechnung ausgeführt wird, zunächst seitlich gerade ausstreckt. Der Armknochen wird dazu so gedreht, dass er parallel zur x-Achse des Körpers, die durch die beiden Schultern definiert wird, verläuft. Die Hand wird so rotiert, dass die Handinnenfläche dabei nach unten zeigt. Aus dieser Grundhaltung heraus werden die weiterführenden Berechnungen angestellt. Alle ermittelten Rotationen werden auf die bereits vorhandenen Rotationen aufgerechnet. Daher wird wegen der effizienten Multiplikation mit Quaternionen gearbeitet. Abhängig davon, ob es sich um den linken oder rechten Arm handelt, werden die Rotationen ggf. umgekehrt.

Dabei wird der Arm mit seinem Oberarmknochen und dem Unterarmknochen als Dreieck betrachtet. Die dritte Seite bildet die Verbindungslinie von Schulterpunkt zum Zielpunkt für das Handgelenk.

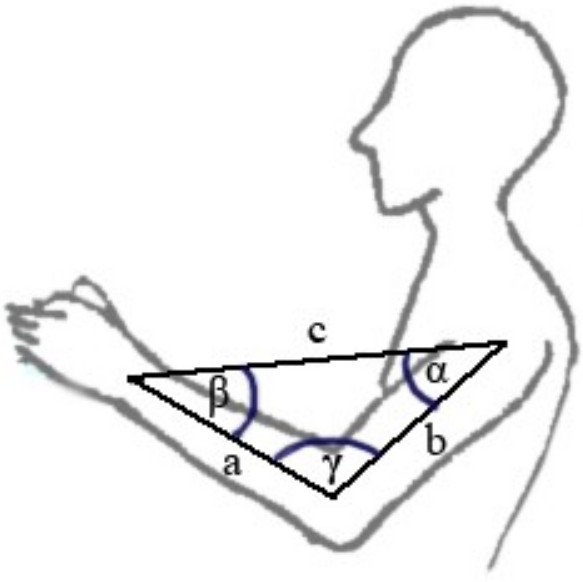

Abbildung 46: das Arm-Dreieck

Da das Ellenbogengelenk ein Scharniergelenk ist, das sich nur in eine Richtung einklappen kann, ist die Berechnung des Winkels für dieses Gelenk am einfachsten, da dieser Winkel eindeutig ist. Mithilfe des Kosinussatzes kann der Winkel γ anhand der Längen der Kantenlängen *a*, *b* und *c* errechnet werden.

$$\text{Kosinussatz für } \gamma: \; c^2 = a^2 + b^2 - 2\,a\,b\cos\gamma$$

$$\gamma = acos\frac{a^2 + b^2 - c^2}{2\,a\,b}$$

Listing 5.5: *Kosinussatz und Berechnung des Ellenbogenwinkels γ*

Da sich der Arm zu diesem Zeitpunkt noch in der Ausgangsstellung befindet, entspricht die Drehachse der y-Achse des Charakters. Damit ist die Berechnung für den Unterarm bereits abgeschlossen.

Der Oberarm wird in drei Schritten berechnet: Zuerst wird er am Körper seitlich herunter geklappt. Dies entspricht einer 90° Rotation um die z-Achse des Charakters. Danach zeigt der eingeklappte Unterarm je nach Stärke der Rotation gerade nach vorne vom Körper weg. Anschließend wird der Oberarm um sich selbst gedreht, so dass der Unterarm auf den Zielpunkt zeigt.

Da der Oberarm gerade nach unten geklappt wurde, entspricht die Rotationsachse der y-Achse des Charakters. Um den Winkel zu ermitteln, um den gedreht werden muss, wird der Vektor vom Schulterpunkt zum Zielpunkt, der ja bereits in der Berechnung des Ellenbogens ermittelt wurde, in die Ebene projiziert, die durch die y-Achse aufgespannt wird. Der gesuchte Rotationswinkel ist der Winkel zwischen dem projizierten Vektor und der z-Achse.

Abbildung 47: Rotation des Oberarms um sich selbst

Der Vektor von Schulter zu Handgelenk wird so projiziert, dass er in einem rechten Winkel zu y-Achse steht.

In einem dritten Schritt wird der Arm schließlich angehoben, um das Handgelenk auf den vorgegebenen Zielpunkt zu bringen. Die Rotationsachse stellt die um den Oberarm gedrehte x-Achse dar. Die Schulter, der Ellenbogen, das Handgelenk und der Zielpunkt liegen alle in einer Ebene. Um den Winkel zu bestimmen, um den der Oberarm nach oben gedreht werden muss, wird zunächst der Winkel zwischen dem Vektor vom Schulterpunkt zum Zielpunkt und dem Oberarmvektor gemessen. Der Oberarm zeigt zu diesem Zeitpunkt gerade nach unten und wird somit durch die negative y-Achse repräsentiert. Der gesuchte Winkel lässt sich durch den errechneten Winkel und den Winkel α aus dem Armdreieck, das bereits zu Beginn vorgestellt wurde, ermitteln.

Nun ist der Arm vollständig berechnet und das Handgelenk befindet sich am vorgegebenen Zielpunkt. Der Ellenbogen zeigt direkt nach unten. Der Arm wird in einem letzten Schritt noch so angepasst, dass die Haltung natürlich wirkt und die Winkel an der Hand entspannt werden. Dabei werden sowohl die Position des Zielpunktes als auch die Winkel am Handgelenk berücksichtigt. Gesucht wird der kleinstmögliche gültige Winkel, da beobachtet werden kann, dass in der natürlichen, entspannten Armhaltung der Ellenbogen weitestgehend nach unten zeigt.

Die Anpassung des Ellenbogens ist aus mehreren Gründen notwendig: Zum einen kann eine Armhaltung, bei der der Arm in Richtung der gegenüberliegenden Körperseite greift, schnell zu Selbstkollision mit dem 3D Modell des Oberkörpers führen. Zum anderen wirkt ein direkt nach unten zeigender Ellenbogen steif und unnatürlich, insbesondere, wenn die Winkel am Handgelenk stark abgeknickt sind.

Abbildung 48: steife Armhaltung

Als Rotationsachse dient für diese letzte Rotation der bereits mehrfach eingesetzte Vektor vom Schulterpunkt zum Zielpunkt. Eine Drehung um diese Achse verändert nichts an der Position des Handgelenkes. Lediglich der Ellenbogen wird nach außen und oben rotiert.

Um die Hand einbeziehen zu können, müssen zunächst die Winkel an der Hand berechnet werden. Dabei sind nur zwei der drei Winkel von Relevanz – die Flexion/Extension (hoch- und runterklappen), sowie die Adduktion zur Seite. Die Rotation um sich selbst kann vernachlässigt werden. Durch die Berechnung der Berührungspunkte durch die Primitive wurde hier bereits dafür gesorgt, dass der Winkel im zulässigen Bereich liegt. Die Handachsen werden mit den Achsen des Berührungspunktes zur Deckung gebracht.

Um die Berechnung des Winkels für die Anhebung des Ellenbogens möglichst effizient zu gestalten, wird lineare Interpolation eingesetzt. Die Handwinkel werden für den Arm mit maximal nach unten zeigendem Ellenbogen berechnet. Dann wird der Arm um 100° gedreht, was in etwa der maximal möglichen Drehung des menschlichen Arms in dieser Richtung entspricht. Anschließend werden die Handwinkel erneut berechnet und es wird der kleinste Drehwinkel zwischen 25° und 100° interpoliert, an dem beide Handwinkel innerhalb einer erlaubten Standardabweichung von der funktionalen Handstellung liegen (siehe auch Kapitel 2.3. Inverse Kinematik für den menschlichen Arm).

Doch nicht nur die Winkel der Hand werden berücksichtigt, sondern es soll durch das Anheben auch eine Selbstkollision weitestgehend verhindert werden. Daher wird die Rotation des Oberarmes um sich selbst als Indikator dafür genommen, wie weit der Arm zur anderen Körperseite zeigt. Ist eine Korrektur notwendig, wird abhängig von diesem Grad ein Winkel zwischen 25° und 100° für die Anhebung des Ellenbogens ermittelt.

Abschließend wird der größere der beiden zuvor berechneten Winkel für die tatsächliche Rotation eingesetzt, wobei der Winkel minimal 25° betragen muss. Winkel unter diesem Wert führen zu regelmäßiger Selbstkollision und einem optisch unrealistischen Eindruck.

Zum Abschluss werden noch einmal die Winkel der Hand berechnet. Diese können nicht zusammen mit der Anhebung des Armes interpoliert werden, da das Ergebnis nicht genau genug ist. Daher werden die Handwinkel nach Drehung des Armes in seine finale Stellung erneut ermittelt.

5.7. Animation

Die Berechnung der finalen Armstellung reicht jedoch für eine realistische Animation nicht aus. Auch die Bewegung des Armes soll überzeugend wirken. Dazu wurden die natürlichen Bewegungsabläufe beim Menschen näher untersucht. Da es kaum veröffentlichte Studien zur Bewegung des Armes beim Greifen eines Gegenstandes gibt, basiert die Animation des Frameworks auf eigenen Beobachtungen und der optischen Bewertung der Ergebnis-Animationen.

Bei der Betrachtung der Bewegung des Armes und der Hand beim Greifen eines Gegenstandes, fällt zunächst auf, dass sich die Finger erst kurz vor dem Zugreifen öffnen. Erst auf dem letzten Abschnitt der Annäherung an das Objekt spreizen sich die Finger gerade weit genug auf, um das Objekt gut umschließen zu können. Auch die Hand richtet sich nicht bereits zu Beginn der Bewegung vollständig aus. Stattdessen rotiert sie grob in eine passende Stellung und erst kurz vor dem Öffnen der Finger knickt sich das Handgelenk entsprechend der finalen Handhaltung ein. Dies gibt motorisch durchaus Sinn, da die Hand nicht länger als nötig in einer möglicherweise unentspannten Stellung gehalten werden muss, was Muskelkraft erfordert.

Beim Bewegungsfluss des Arms ist zu erkennen, dass die Hand nicht in einer direkten Linie von ihrem Ausgangspunkt zu ihrem Ziel bewegt wird, sondern in einer Kurve. Dabei wird die Hand zunächst ein Stück zum Körper angezogen, bevor sie wieder ausgestreckt wird. Dies erfüllt zwei Funktionen: Zum einen spart es aufgrund des Hebelgesetzes Kraft, da der Schwerpunkt näher an den Drehpunkt verlagert wird. Zum anderen hat dieser Mechanismus den praktischen Nutzen, dass die Hand zunächst von ihrer momentanen Position Abstand nimmt und sich dann aus der Entfernung für das neue Objekt ausrichtet. Wenn die Hand bereits auf oder an einem Objekt liegt, wird so z.B. verhindert, dass sie durch das Objekt hindurch greift. Für das Zielobjekt ergibt sich der Vorteil, dass dieses nicht zwingend von der Seite gegriffen werden soll, von der sich die Hand nähert. Auch hier wird eine Kollision verhindert und Zeit für die Ausrichtung der Hand gewonnen. Wie weit die Hand eingezogen wird, hängt von der Entfernung von der Ausgangsposition der Hand und dem Zielpunkt ab.

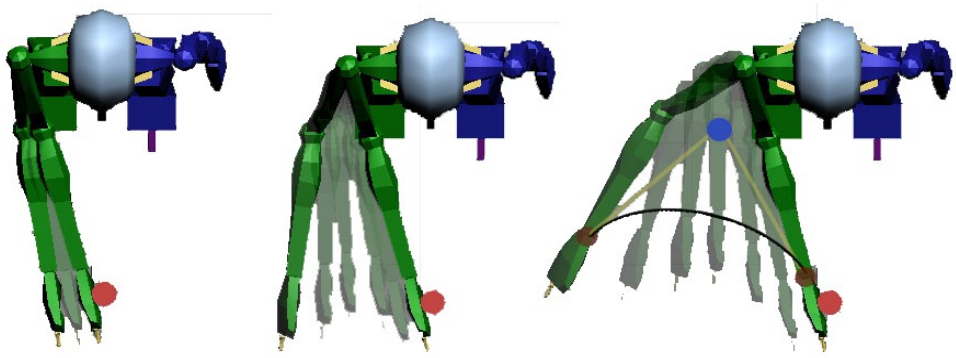

Abbildung 49: Heranziehen des Armes an den Körper

Abhängig vom Abstand wird die Hand unterschiedlich weit eingezogen.

Diese Beobachtungen wurden bei der Konzeption der Animation im Framework berücksichtigt. Für den Arm wurden drei Bewegungsphasen angelegt. Die erste zieht den Arm abhängig vom Abstand von Ziel- und Startpunkt an den Körper heran, die zweite Phase streckt den Arm wieder aus, und die dritte Phase stellt die finale Annäherung an das Objekt dar.

Die Bewegung der Hand wurde in zwei Teilanimationen umgesetzt. In der ersten Phase rotiert sich die Hand lediglich um sich selbst, die anderen beiden Achsen werden nicht berücksichtigt. So kann eine grobe Ausrichtung am Objekt stattfinden. Erst im letzten Abschnitt der Bewegung werden auch die restlichen Winkel auf das Handgelenk angewandt. Um ein optisch möglichst ideales Ergebnis zu erzielen, wird die Rotation der Hand um sich selbst teilweise an eventuell vorhandene zusätzliche Unterarme weitergegeben. Dabei entfällt auf jeden Unterarm ein gleich großer Anteil. Siehe hierzu auch Kapitel 5.2. zusätzliche Unterarme.

Die Bewegung der Finger ist auf den letzten Abschnitt der Animation begrenzt. Etwa zeitgleich zur Ausrichtung der Hand werden die Finger entsprechend der vom Primitiv ausgewählten, angepassten Griffpose geöffnet und dann um das Primitiv geschlossen. Mit der vollständigen Schließung der Finger endet die Bewegung.

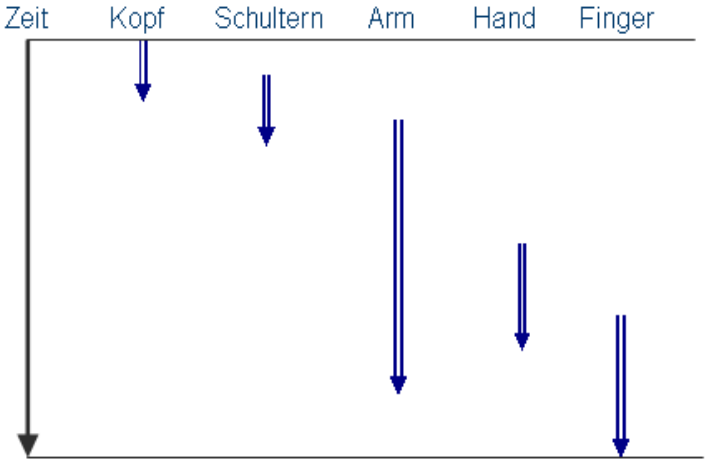

Abbildung 50: Animationsverlauf für Kopf, Schultern, Arm, Hand und Finger

Die zeitlichen Verhältnisse wurden durch Beobachtung ermittelt.

Auf Basis dieses zeitlichen Ablaufes werden die einzelnen Animationsphasen für die Knochen des Armes und der Hand kontrolliert. Für jeden Knochen wird eine eigene Animation erstellt. Die Klasse IKAnimation speichert die aktuelle Rotation des Knochens, da die Bewegung aus dieser Stellung heraus beginnen soll. Weiterhin speichert sie die gewünschte Zielrotation und Start- und Endzeit der Animation. Zwischen den Quaternionen, die die Rotationen repräsentieren, wird mittels Slerp (spherical linear interpolation) abhängig von der verstrichenen Zeit interpoliert. Für jeden animierten Knochen wird eine IKAnimation erzeugt.

Da nur die Start- und Zielrotationen gespeichert werden, was zwei Keyframes entspricht, repräsentiert eine IKAnimation immer nur eine einzelne Bewegungsphase. Der Grund dafür ist, dass sich die Zielrotationen aufgrund von Charakter- oder Objektbewegungen ändern können und die Quaternione erneut berechnet werden müssen. Siehe hierzu auch Kapitel 5.8.1. Bewegte Charaktere und Objekte. Weil virtuelle Charaktere in der Regel ständig in Bewegung sind und selbst im Stillstand eine Atmungsanimation den Brustkorb und die Schultern ständig hebt und senkt, wird Rechenzeit gespart, indem nur die gerade aktive Animation aktualisiert wird. Ist eine Animation für einen Knochen abgeschlossen, beginnt für diesen die nächste Bewegungsphase.

Die Basisklasse IKCharBase verlangt die Implementierung der Funktion *SetBone()*, über die das Framework die interpolierten Rotationen für die Arm- und Handknochen an das Animationssystem weitergibt. Die Transformationen können nicht mit anderen Animationen geblendet werden, da diese das Ergebnis verfälschen würden. Die Gewichtung der vom Framework gesandten Daten muss daher immer 1 sein.

```
void SetBone (int boneId, IKVector& translation, IKQuaternion& rotation,
              IK_BONES flag, IK_ARM arm);
```

Listing 5.6: Funktionskopf der Funktion *SetBone()*

Die Funktion erhält vom Framework die id des Knochens, dessen Transformation gesetzt werden soll. Die weiteren Parameter sind die Translation und die Rotation. Die Translation wird vom Framework nicht verändert, sondern nach der Knochenidentifikation zu Berechnungszwecken gespeichert. Hier wird sie aus Effizienzgründen zusätzlich übergeben. Wenn eine Transformation für einen Knochen gesetzt wird, müssen im Animationssystem in der Regel beide Werte angegeben werden und so wird dem Hauptprogramm ein weiterer Aufruf zur Abfrage erspart.

Die beiden Parameter *flag* und *arm* sind wie die Translation nur Zusatzinformationen, die näheren Aufschluss über den Knochen geben, der gesetzt werden soll. Der Parameter *flag* kann dabei die Werte *IKARM*, *IKHAND* oder *IKFINGERS* haben und gibt an, zu welcher Kategorie der aktuelle Knochen gehört. Der Parameter *arm* bestimmt passend dazu die Körperseite und kann die Werte *RIGHT_ARM* oder *LEFT_ARM* annehmen. Dies kann hilfreich sein, falls das Animationssystem zur besseren Abwicklung von IK-Animationen eigene, interne Animationen speziell für das Framework erzeugt. Wird eine Interaktion beendet, kann so beispielsweise besser in eine eigene Animation überblendet werden. Dies ist zudem die einfachste Möglichkeit, das Framework in ein bestehendes Animationssystem zu integrieren, insbesondere dann, wenn man das System selbst nicht modifizieren kann. In der im Rahmen dieser Diplomarbeit entwickelten Demo-Applikation wurde im Einsatz mit Cal3D als Animationsbibliothek ebenfalls diese Variante gewählt, da die Bibliothek nicht verändert werden sollte. Siehe hierzu auch Kapitel 6. Ergebnisse und Bewertung.

5.8. Sonderfälle

5.8.1. Bewegte Charaktere und Objekte

Die Szenen, in der die Interaktionen, die vom Framework visualisiert werden, stattfinden sollen, sind nur selten statisch. Die Position des Objektes kann sich verändern, z.B. durch Physik oder Animation, oder der Charakter kann sich bewegen. Selbst ein scheinbar stillstehender Charakter durchläuft in der Regel noch eine Atmungsanimation, die den Brustkorb hebt und die Schultern bewegt. Alle Änderungen an der Position und Ausrichtung des Charakters und des Gegenstandes zueinander erfordern eine Neuberechnung der Armstellungen, da andernfalls das Ergebnis bereits bei kleinen Bewegungen ungenau wird. Die Hand könnte das Objekt nicht mehr erreichen oder sie würde auf der Oberfläche hin und her wackeln, oder sogar durch den Gegenstand hindurch greifen.

Eine ständige Neuberechnung der Armstellung erfordert jedoch Rechenzeit, die möglichst gespart werden soll. Daher müssen Regeln für die Neuberechnung aufgestellt werden. Als Grundlage für diese Regeln dienen die aktuelle Animationsphase beim Greifen und die Größe der Änderung.

In der ersten Armphase, in der die Hand in Richtung des Körpers gezogen wird, ist der Einfluss des Objektes auf die Bewegung noch sehr gering. Daher wird in dieser Phase grundsätzlich nicht neu berechnet. Fand jedoch eine Positionsänderung des Charakters oder des Objektes statt, so wird direkt nach Abschluss dieser Phase vollständig neu berechnet.

In der Phase, in der der Arm ausgestreckt wird, muss nach anderen Kriterien neu berechnet werden. Hier wird der Grad der Abweichung mit berücksichtigt. Aus den Änderungen an Position und Rotation des Objektes und der Schulter des Charakters wird eine Kennzahl errechnet. Liegt diese über einem Schwellwert, wird direkt neu berechnet. Andernfalls wird ein Zähler gestartet, der nach einer festen Anzahl an Frames ebenfalls neu berechnet. So wird gewährleistet, dass bei konstanten kleinen Änderungen, wie z.B. der Atmung des Charakters, nicht jedes Frame eine Neuberechnung erzwungen wird, große Abweichungen aber trotzdem rechtzeitig behandelt werden.

In der letzten Phase tritt die Hand mit dem Objekt in direkten Kontakt. Hier muss die Genauigkeit der Handpositionierung am größten sein. Daher wird hier selbst bei kleinen Änderungen sofort eine neue, angepasste Armstellung berechnet. Da der Charakter in den Aktionen 'Tragen' und 'Festhalten' ständigen Objektkontakt hat, wird diese Regel auch für diese Fälle angewandt.

Wurde eine neue Armstellung berechnet, so müssen alle Animationen für die einzelnen Armknochen ebenfalls aktualisiert werden. Zu diesem Zweck genügt es jedoch nicht, nur die jeweiligen Ziel-Keyframes mit den neu berechneten Rotationen auszutauschen. Dies würde zu teils starken visuellen Artefakten, wie einem springenden Arm, führen. Zusätzlich muss daher die aktuelle Position des Armes als Ausgangslage genommen werden, von der die Bewegung zum neuen Ziel startet.

5.8.2. Entfernung des Objektes ist zu groß

Um eine Interaktion zu starten, muss das Hauptprogramm lediglich einen Charakter und ein Objekt an das Framework übergeben. Für das Objekt muss mindestens ein Primitiv definiert sein und der Charakter muss ein identifiziertes Skelett besitzen. Weitere Bedingungen stellt das Framework nicht. Es wird auch nicht überprüft, ob der Gegenstand nahe genug am Charakter platziert ist, so dass er von ihm erreicht werden kann.

Das Hauptprogramm selbst hat möglicherweise nur begrenzte Möglichkeiten, die Entfernung abzuschätzen. Daher bietet das Framework eine Funktion *IsInReach()* an, die für ein Charakter-Objekt-Paar zurückmeldet, ob sich das Objekt in weniger als einer Armeslänge Entfernung zur jeweiligen Schulter befindet. Dies kann das Hauptprogramm nutzen, um z.B. den Charakter auf Basis des eigenen Wegfindungssystems näher an den Gegenstand heran zu bewegen, oder um zu entscheiden, ob dem Spieler für ein im Sichtfeld des Charakters befindliches Objekt eine Interaktionsmöglichkeit angegeben werden soll.

In einigen Fällen kann das Hauptprogramm den Charakter jedoch nicht näher an den Gegenstand heranbringen, z.B. weil es dem Spieler nicht die Kontrolle abnehmen will oder darf. In diesem Fall muss das Framework mit der Unerreichbarkeit des Gegenstandes umgehen. Um visuell eindeutig zu bleiben und nicht den Eindruck eines Fehlverhaltens zu erwecken, wird in diesem Fall die Hand so nah wie möglich an den Gegenstand heran bewegt, die Finger bleiben jedoch geöffnet. Würden sich die Finger trotz der Unerreichbarkeit schließen, würde entweder der Eindruck entstehen, als würden sie einen leeren Raum greifen, oder sie würde, bei einem nur sehr knapp außerhalb der Reichweite befindlichen Objekt, mit dem Gegenstand kollidieren. Beides würde optisch als Fehler gewertet werden.

Bekommt der Charakter in so einer Situation den Befehl, den Gegenstand aufzunehmen und zu tragen, so wird das Objekt wie gewohnt in der Hand platziert. Ein ähnliches Verhalten ist in Shootern zu beobachten, wo die Reichweite ganz bewusst besonders hoch eingestellt wird, um das

Aufsammeln von Munition und Waffen zu erleichtern. Wird im Gegenzug ein Objekt, das der Charakter festhält, außerhalb seiner Reichweite gezogen, so müssen sich die Finger ebenfalls erneut öffnen.

5.8.3. mehrfache Unterarme

Wie bereits in Kapitel 5.2. Skelettidentifizierung erklärt, verfügen manche Skelettmodelle für Charaktere über zusätzliche Unterarme, die für eine gleichmäßige Verzerrung des Modells sorgen sollen. Diese werden bei der Handrotation auch vom Framework eingesetzt, um visuell ansprechende Ergebnisse zu erzielen. Doch aus der Hierarchie des Armes ergibt sich ein weiteres Problem, dass das Framework zwingt, diese zusätzlichen Knochen besonders zu berücksichtigen.

Die Unterarme sind in der Skeletthierarchie nicht als Kinder an den eigentlich Haupt-Unterarm gehängt, sondern an den Oberarm. Dies verursacht eine Besonderheit bei der Rotation des Haupt-Unterarms. In der Hierarchie werden die Transformationen des Vaterknotens an die Kinder weitergegeben. Da die zusätzlichen Unterarme aber am Oberarm hängen, muss die Rotation des Haupt-Unterarms explizit auf sie übertragen werden. Andernfalls sind deutliche grafische Artefakte die Folge. Die zusätzlichen Unterarme können also in keinem Fall einfach ignoriert werden.

Abbildung 51: Fehler durch Nichtberücksichtigung der zusätzlichen Unterarme

6. Ergebnisse und Bewertung

Im Rahmen dieser Arbeit wurde eine Demonstrations-Applikation entwickelt, die bei der Bewertung der Ergebnisse behilflich sein soll. Sie beinhaltet zwei verschiedene Charaktere mit unterschiedlichen Skeletten, eine Reihe an unterschiedlichen Gegenständen und setzt den vollen Funktionsumfang des Frameworks ein.

Das in dieser Arbeit entwickelte Framework beherrscht die vier in der Anforderungserhebung ermittelten Teil-Interaktionen. Im Rahmen der Aktion 'Tragen' wurden sogar verschiedene Lösungen implementiert. Durch die Unterteilung der Aufgaben in einzelne Aktionen kann daher der Großteil der Interaktionen, die in Spielen vorkommen, umgesetzt werden. Durch Kombination sind darüber hinaus weitere, bislang nicht mögliche Aktionen, visualisierbar geworden. Neben dem Aufnehmen eines Gegenstandes kann dieser auch an andere Charaktere weitergegeben werden.

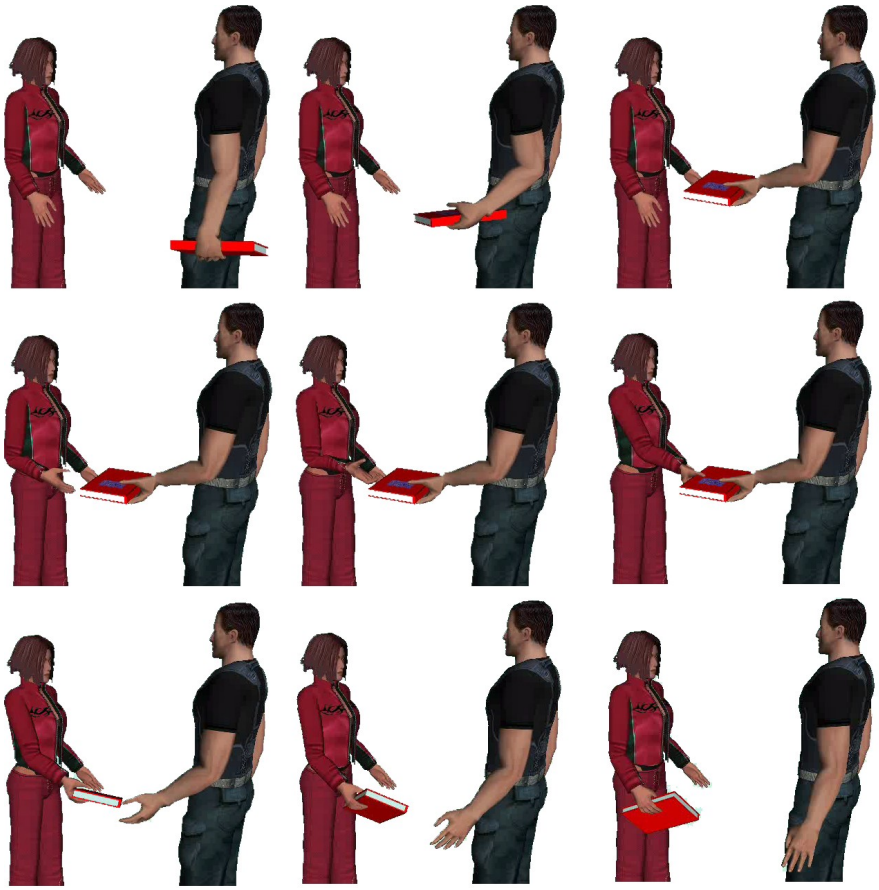

Abbildung 52: Übergabe eines Buches unter zwei Charakteren

Durch den objektbasierten Ansatz entsteht ein geringer zusätzlicher Aufwand bei der Erstellung der Objekte, da alle Gegenstände, die Teil einer Interaktion sein können sollen, mit Primitiven ausgestattet werden müssen. Durch die geringe Menge an Informationen, die pro Primitiv benötigt

wird und einen sinnvollen Editor zur Positionierung kann die benötigte Zeit jedoch sehr kurz gehalten werden. Im Test mit neun verschiedenen Objekten mit unterschiedlicher Anzahl an Primitiven wurde eine Durchschnittszeit von gut 30 Sekunden pro Primitiv ermittelt, unter der Voraussetzung, dass die Bedienung des Programms bereits bekannt ist. Da die Einarbeitung bei der großen Anzahl an Objekten in einem kommerziellen Spiel nicht ins Gewicht fällt, kann dies problemlos vorausgesetzt werden.

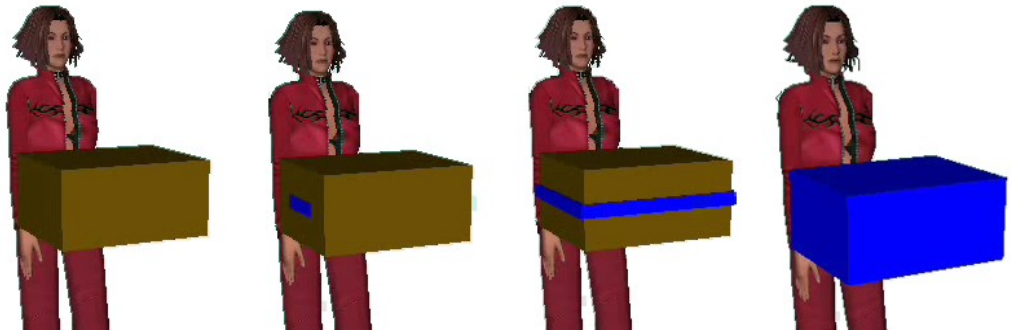

Abbildung 53: Erstellung eines Box-Primitives

Die fünf in der Anforderungserhebung ermittelten Primitive wurde mit einem jeweils eigenen Verfahren zur Berechnung eines passenden Berührungspunktes ausgestattet. Passend zu den Primitiven wird eine Griffpose ausgewählt. Obwohl die Objektoberfläche durch die Primitive nur angenähert wird und die Fingerstellung auf der Basis von zwei Stellungen abhängig von der Primitivgröße interpoliert wird, ist das Ergebnis optisch absolut hinreichend genau. Die Finger verdecken einen Teil der tatsächlichen Oberfläche, so dass kleine Abweichungen nicht auffallen und eine Kollision mit der Objektoberfläche wird durch eine gute Wahl bei der Skalierung des Primitives verhindert.

Abbildung 54: Genauigkeit der Ergebnisse

Durch die Umsetzung aller fünf Primitive können alle Gegenstände, die am häufigsten in Interaktionen vorkommen, repräsentiert werden. Eine eigens entwickelte IK berechnet passend dazu eine gültige, möglichst natürlich aussehende Armstellung auf Basis von Vorgaben aus der Medizin. Der Bewegungsfluss des Armes wird dabei an den menschlichen Bewegungsablauf angelehnt. Die Bewertung der Ergebnisse in diesem Bereich kann jedoch nur optisch erfolgen.

Da das Framework auch mit bewegten Charakteren und Objekten funktioniert, müssen andere Systeme, die ebenfalls zur Erhöhung des Realitätsgrades dienen, nicht umgangen werden, wie z.B. die Physikalisierung oder die Stillstand-Animationen des Charakters. Zudem können dadurch die Animationen des Frameworks mit den Animationen des Hauptprogramms kombiniert werden, um noch realistischere Bewegungen zu erhalten.

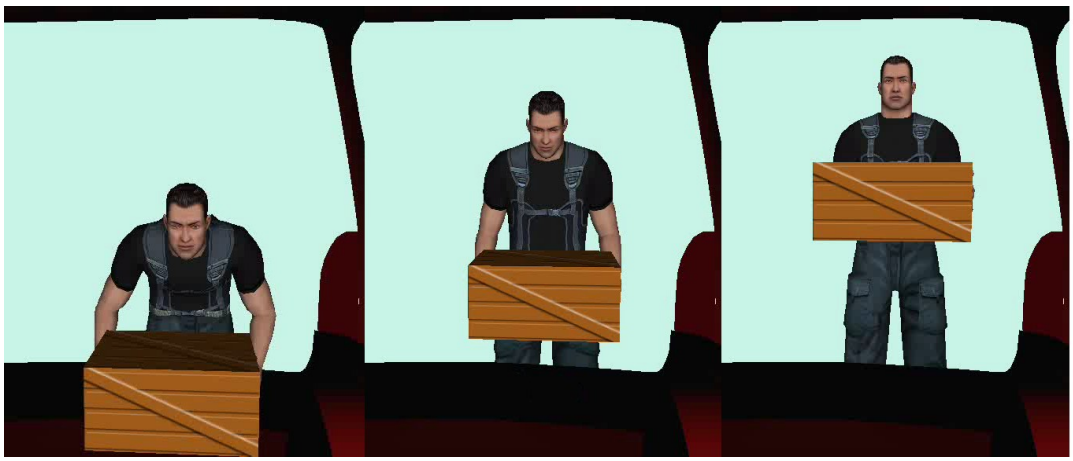

Abbildung 55: Aufnehmen einer Kiste aus einem Kofferraum

Bei der Beuge-Animation handelt es sich um eine Forward-Animation.

Abbildung 56: interaktive Szene in einem Auto

7. Fazit und Ausblick

Die Visualisierung von Interaktionen zwischen Charakteren und Objekten in Spielen konnte bislang nur unzureichend umgesetzt werden. Eine Lösung mittels Inverser Kinematik ist ohne eine Definition, wie und wo das Objekt gegriffen werden soll, nicht möglich.

Der in dieser Arbeit vorgestellte objektbasierte Ansatz, bei dem das Objekt durch manuell definierte Primitive direkt die Position und die Weise bestimmt, auf die es berührt werden darf, bietet eine sehr zufrieden stellende Möglichkeit, realistische Ergebnisse zu erzielen. Dabei wird der zusätzliche Aufwand bei der Ausstattung der Objekte mit Primitiven vertretbar gering gehalten. Auch die optische Genauigkeit der Ergebnisse ist trotz der Annäherung der Objektoberfläche durch ein geometrisches Primitiv absolut überzeugend.

Das entwickelte Framework bietet einen großen Funktionsumfang, so dass es sowohl in Spielen als auch in anderen virtuellen Umgebungen eingesetzt werden kann, um Interaktionen zu visualisieren. Die Unterteilung in Teil-Aktionen stellt allerdings einerseits eine Einschränkung dar. Da nichts über die Intention der Handlung bekannt ist, muss stets eine neutrale Darstellung gewählt werden. Durch die Aufsplittung können andererseits jedoch durch Kombination sogar mehr Vorgänge dargestellt werden, als in der Anforderungserhebung ermittelt.

Bei der Implementierung der IK zur Berechnung der Armhaltung hat sich gezeigt, dass unbedingt anatomische Gegebenheiten berücksichtigt werden müssen, um ein natürliches Aussehen zu erreichen. Aus diesem Grund sind Standardverfahren wie das CCD für die Berechnung ausgeschieden und ein eigenes Verfahren musste entwickelt werden. Dabei hat sich überraschenderweise gezeigt, dass die Berechnungen performant genug sind, um theoretisch jedes Update neu ausgeführt zu werden.

Die Natürlichkeit der Ergebnisse wird zweifelsfrei beeinflusst von der Qualität der Integration der Frameworks in das Hauptprogramm. In der Praxis zeigt sich, dass das Framework auch mit suboptimalen Bedingungen umgehen können muss, da es eher wie ein Dienstleister angesehen werden kann. Es kann keine Entscheidungen über die Animation des Charakters treffen, die über den gegebenen Arm hinaus gehen. Die Kontrolle über den gegenüberliegenden Arm und über die übrigen Knochen des Skelettes muss beim Hauptprogramm verbleiben. So kann das Framework weder erzwingen, dass ein zweiter Arm herangezogen wird, wenn der Gegenstand zur groß ist, um mit einer Hand gegriffen zu werden, noch kann es veranlassen, den Charakter vornüber zu beugen oder näher an das Objekt heran zu bewegen, wenn dieses sich außerhalb der Reichweite befindet.

Die genannten Einschränkungen können durch einen Ausbau des Frameworks noch weiter vermindert werden. So ist etwa die Berücksichtigung von Modifikatoren bei der Berechnung der Armstellung denkbar, z.B. von Gewicht, Emotionen oder Intention. Entsprechende Informationen könnten optional vom Hauptprogramm an das Framework weitergegeben werden.

Während das Framework, wie bereits erklärt, nur den zugewiesenen Arm bewegen darf, so kann dennoch eine Berücksichtigung des anderen Armes sinnvoll sein. So könnte beispielsweise verhindert werden, dass die beiden Arme miteinander kollidieren. In der vorliegenden Implementierung wurde kein Test der Arm-Knochen auf Selbstkollision durchgeführt, da dies den

Umfang dieser Arbeit deutlich überstiegen hätte. Um ein Durcheinander-Gleiten der Arme zu verhindern, müssen glaubwürdige Ausweich-Strategien entwickelt werden, wie etwa das Herumführen des einen Armes um den anderen, sowie eine vorausschauende Kollisionserkennung implementiert werden.

Darüber hinaus wäre es denkbar, das Framework so zu erweitern, dass es mit einer Physik-Engine zusammen arbeiten kann. Durch eine Voraussage der Objektposition und seiner Rotation könnte es möglich gemacht werden, z.B. geworfene Gegenstände glaubwürdig aufzufangen.

Leider konnten aufgrund der zeitlichen Beschränkung dieser Arbeit eine Reihe von sinnvollen Erweiterungen nicht mehr in das Framework aufgenommen werden. Dennoch konnte gezeigt werden, dass eine glaubwürdige und realistische Darstellung von Charakter-Objekt-Interaktionen mittels eines objektbasierten Ansatzes möglich ist. Daher ist zu hoffen, dass auch der Bereich der Animation in naher Zukunft mit den Fortschritten der anderen Gebiete, wie z.B. Grafik und Physik, im Sinne des Trends zum Realismus in der Computerspiele-Industrie mithalten kann.

Literaturverzeichnis

[1] Entertainment Software Association (2006) : Sales & Genre Data.
http://theesa.com/facts/gamer_data.php; Abruf: September 2007

[2] Nik Lever (2002) : Real-time 3D Character Animation. Focal Press, Oxford.
ISBN 0-240-51664-8

[3] Li-Chun Tommy Wang, Chih Cheng Chen (1991) : "A Combined Optimization
Method for Solving the Inverse Kinematics Problem of Mechanical Manipulators".
IEEE Transactions on Robotics and Automation, Ausgabe Vol. 7, No. 4, Seiten 489-499

[4] Prof. Dr. med. Dr. phil. Herbert Lippert (Januar 2000) : Lehrbuch Anatomie.
Urban & Fischer Verlag, München. ISBN 3-437-42360-6

[5] Coder Corner (2007) Pierre Terdiman : IK experiments.
http://www.codercorner.com/IK.htm; Abruf: Oktober 2007

[6] Y. Koga, K. Kondo, J. Kuffner, and J. Latombe (1994) : "Planning Motions with Intentions".
In Proceedings of SIGGRAPH 94, Ausgabe Juli 94, Seiten pp. 395 - 408

[7] Forschungszentrum Karlsruhe (2007) Institut für Angewandte Informatik : Prothetik.
http://www.iai.fzk.de/www-extern/index.php?id=1255&L=0; Abruf: Oktober 2007

[8] Dr. Ing. Stefan Schulz (2005) Fluidgruppe : Handprothese.
http://fifserver.iai.fzk.de/fluidgruppe/; Abruf: Oktober 2007

[9] Hees (2007) : Anatomie - Die Körpermaße des Menschen nach DIN 33402.
http://www.hees.de/raumkonzepte/aktuelles/index.php; Abruf: Oktober 2007

[10] Cal3D (2007) Gna! : Character Animation Library.
http://home.gna.org/cal3d/; Abruf: August 2007

[11] Dead Justice (2003) Cat Mother Ltd. : 3D Models; Free Content, GPL license.
http://catmother.sourceforge.net/; Abruf: April 2007