Hochschule der Medien, Stuttgart
Fachbereich Druck und Medien
Studiengang Medieninformatik

Modellierung, Darstellung und interaktive Animierung virtueller Charaktere in 3D-Echtzeitanwendungen

Die Entstehung eines Charakters und die dynamische Animierung eines ped-Systems für eine interaktive 3D-Welt

Diplomarbeit vorgelegt von

Valentin Schwind
Norman Pohl

Erstprüfer: Prof. Dr. Jens-Uwe Hahn
Zweitprüfer: Prof. Walter Kriha

Stuttgart, den 06.01.2009

Abstrakt

Die vorliegende Arbeit befasst sich mit der Entstehung und Animierung virtueller Charaktere in interaktiven 3D-Welten. Anhand des Computerspiels „Die Stadt NOAH“, ein studiengangübergreifendes Projekt an der Hochschule der Medien Stuttgart (HdM), und der Überarbeitung der Hauptfigur „Sophie Faber“, möchten wir den Prozess dieser Entstehung vorstellen, auf die besonderen Herausforderungen dieser Entwicklung eingehen und die Erkenntnisse aus dieser Arbeit aufzeigen.

Ziel ist es, die Techniken und Vorgehensweisen zu veranschaulichen, die für die grafische und programmiertechnische Entstehung solcher glaubwürdiger, dreidimensionaler Charaktere erforderlich sind. Wir lernen in dieser Ausarbeitung die Prozesse kennen, wie aus einem gezeichneten Konzept ein vollständiger, dreidimensionaler Charakter entsteht, der schließlich animiert und gesteuert werden kann. Schwerpunkt dieses Projekts ist hierbei die dynamische Animierung eines Charakters.

Untersucht werden dabei die Aspekte der Konzeption, Modellierung, Texturierung, des Skulptierens und der Animierung von Computerspielfiguren. Wir schauen auf andere Spiele-Technologien und blicken hinter die Kulissen modernen Charakter-Designs. Diese Arbeit zeigt auf, wie man Computerspielfiguren „lebendig“ werden lässt und welche Herausforderungen und Probleme dieses Vorhaben mit sich bringt.

Die praktischen Teile unserer Ausarbeitung sind in zwei zentrale Einheiten aufgeteilt, die von uns parallel bearbeitet wurden:

a. Die Entstehung einer glaubwürdigen, virtuellen Spielfigur.
b. Die Darstellung und Animierung dieser Figur mithilfe einer eigenen Applikation, die dynamische Bewegungsabläufe unter Verwendung eines inversen kinematischen (IK)-Systems generiert.

Das Ergebnis enthält neben der ausführlichen Beschreibung unserer Entwicklung, ergänzende Hintergrundinformationen, die den Entstehungsprozess inhaltlich abrunden.

Erklärung

Abgrenzung

Kapitel	Seite	Verantwortlicher
1 Einleitung	12	Norman Pohl, Valentin Schwind
2 Die Entstehung eines virtuellen Charakters	15	Valentin Schwind
3 Darstellung und interaktive Animation eines virtuellen Charakters in Echtzeit	88	Norman Pohl

Erklärung

Hiermit erklären wir, die vorliegende Diplomarbeit selbstständig und gemäß der obigen Abgrenzung angefertigt zu haben. Es wurden nur die in der Arbeit angegebenen Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut haben wir als solches kenntlich gemacht.

Wir versichern, dass wir bislang keine Prüfungsarbeit mit gleichem oder ähnlichem Thema bei einer Prüfungsbehörde oder einer anderen Hochschule vorgelegt haben.

Ort, Datum

Unterschrift: Valentin Schwind

Ort, Datum

Unterschrift: Norman Pohl

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

3D	Dreidimensional
AO	Ambient Occlusion
CPU	Central Processing Unit - Hauptprozessor eines Computers
FG	Final Gathering
FPS	Frames per Second
GPU	Graphics Processing Unit - Grafikprozessor zur Berechnung der Bildschirmausgabe
HdM	Hochschule der Medien, Stuttgart
HDRI	Hight Dynamic Range Images (hochaufgelöste Bilder mit hohem Kontrastumfang)
IK	Inverse Kinematik
OGRE	Object-Oriented Graphics Rendering Engine, siehe http://www.ogre3d.org
PDA	Personal Digital Assistant, tragbarer Kompaktcomputer
PNG	Portable Network Graphics, Dateiformat mit verlustfreier Kompression
PSD	Photoshop Dateiformat
RAM	Random Access Memory (Arbeitsspeicher)
RGBA	Farbkanäle: Rot, Grün, Blau und Alpha (Transparenz)
TCP	Tool Center Point

1 Einleitung

1.1 Projektvorstellung

1.1.1 Die Stadt NOAH

„Die Stadt NOAH“ ist ein Computerspiel und eine Gemeinschaftsproduktion von Studenten mehrerer Studiengänge an der Hochschule der Medien Stuttgart. In dem 3D-Point & Click-Adventure schlüpft der Spieler in die Rolle der Kommissarin „Sophie Faber“, und führt diese durch eine Stadt, die durch eine gewaltige Staumauer vor einer zerstörerischen Flut bewahrt wurde. Als die Sicherheit der Stadt plötzlich durch einen mysteriösen Terroristen gefährdet ist, beginnt Sophie mit ihrem Ermittlungen. Dabei muss sie zahlreiche Rätsel lösen und schließlich das Geheimnis um „NOAH“ selbst lüften.

Das Spiel verbindet zahlreiche Fachgebiete der Studiengänge an der Hochschule der Medien zu einem großen, studentischen Projekt, das von Semester zu Semester weiterentwickelt wird. Hinter der Idee des Spiels, steht nicht seine offizielle Vermarktung, sondern seine kontinuierliche technische und erzählerische Verbesserung. Studenten erhalten dadurch einen Einblick in die Welt der Spiele-Entwicklung. Programmierer und Künstler bekommen die Möglichkeit, gemeinsam ihre Vorstellungen umzusetzen.

Beteiligte Studiengänge an „NOAH“ sind:

- ***Medieninformatik*** – Programmierung, Grafik, Projektleitung
- ***Audiovisuelle Medien*** – Grafik, Sound, Musik & Dialoge
- ***Medienautor*** – Plot & Projektleitung
- ***Verpackungstechnik*** – CD Hüllen und Poster

Vor dieser Arbeit waren wir (Diplomanden des Studiengangs Medieninformatik) an der Programmierung der Grafik-Engine und Betretung der Figuren und Levels von „NOAH“ beteiligt. Das Team bestand dabei aus teilweise (Sommersemester 2007) über 60 Studenten.

1.1.2 Beschreibung

Die Idee für diese Diplomarbeit und der Weiterentwicklung des Spiels „Die Stadt NOAH“, kam uns dann im Februar 2008, Ende des Wintersemesters 07/08, als uns vom Projektteam eine Verbesserung, bzw. vollständige Überarbeitung der Hauptfigur „Sophie Faber“ vorgeschlagen wurde. Uns interessierte dabei besonders der Gedanke, Sophie nicht nur glaubwürdiger darzustellen, sondern auch mithilfe eines interaktiven Systems zu animieren, welches dynamisch auf die Eingaben des Spielers reagiert. Dabei sollen in Echtzeit gleichzeitig die Eingabe, die Berechnung ihrer Animationen und die Darstellung von Sophie erfolgen.

Durch voranimierte Charakteranimationen, also Bewegungen, die mit einer Animationssoftware erstellt werden, wirken Computerspielfiguren (insbesondere Menschen) unnatürlich und unglaubwürdig, wenn diese im Spiel zum Einsatz kommen. Hierfür sind vor allem die verschiedenen und veränderlichen Umgebungen eines 3D-Spiels verantwortlich, insbesondere dann wenn sich der Spieler innerhalb der Welt mit seiner Figur frei bewegen darf. Die Lösung dieses Problems liegt nun entweder darin, für jede mögliche Szenerie eine passende Bewegung zu entwerfen (was zu eine gewaltigen Anzahl an Animationen erforderlich macht), oder eine Alternative zu entwickeln, die es einer Figur ermöglicht, dynamisch mit ihrer Umgebung zu interagieren.

Grundlage der Programmierung von „NOAH“ ist die quelloffene Game-Engine OGRE. Im Laufe dieses Projekts haben wir jedoch keinen „Eingriff“ in die Engine oder das Spiel vorgenommen. Unsere Arbeit konzentriert sich zunächst auf die Überarbeitung der Hauptfigur Sophie Faber, sowie deren Animierung in einer eigenen Anwendung, die ebenfalls auf OGRE basiert. Diese Arbeit wirft einen allgemeinen Blick auf die Entstehung virtueller Charaktere und wurde von uns in zwei Teile gegliedert, die parallel bearbeitet wurden.

a. Der erste Teil beschäftigt sich zunächst mit der Erschaffung eines virtuellen Charakters. Wir beleuchten den vollständigen Entstehungsprozess von der Konzeption, Modellierung, Texturierung, bis hin zur Keyframe-Animierung einer Spielfigur am Beispiel von Sophie Faber. Die praktische Arbeit bestand aus ihrem vollständigen und animierbaren 3D-Modell. Verantwortlich für diesen Teil ist Valentin Schwind.
b. Der zweite Teil dieser Ausarbeitung behandelt die Darstellung und Animierung von Charakteren. Wir entwickelten auf Basis der freien 3D-Grafikengine OGRE, welche auch in „NOAH“ zum Einsatz kam, eine 3D-Echtzeitanwendung: ein eigenes Tool für die dynamische Animierung. Verantwortlich für diesen Teil ist Norman Pohl.

Wir möchten in dieser Ausarbeitung Sophies Entwicklung beschreiben, und anhand ihres Beispiels den allgemeinen Entstehungsprozess und die angewandten Techniken veranschaulichen.

1.1.3 Motivation

Der Grund für unser Interesse an diesem Projekt lag nicht nur an unserer Leidenschaft für 3D-Computerspiele und Animationen. Da wir schon als Betreuer und Entwickler an „NOAH“ mitgewirkt hatten und selbstverständlich an einer Weiterentwicklung interessiert sind, wählten wir dies als Thema für diese Diplomarbeit. Unsere Arbeit an dem Projekt und unsere bisherige Erfahrung erleichterten dabei den Einstieg.

1.1.4 Das Team

Norman Pohl (verantwortlich für die Programmierung) und Valentin Schwind (verantwortlich für die Entstehung von Sophies Modell) sind Studenten im 10. Semester, studieren beide Medieninformatik (Dipl.) an der Hochschule der Medien in Stuttgart.

1.1.5 Ansprechpartner

Betreuer und Prüfer für diese Diplomarbeit sind Prof. Dr. Jens-Uwe Hahn und Prof. Walter Kriha. Ansprechpartner für „Die Stadt NOAH“ sind die Mitglieder des (ehemaligen) NOAH Teams unter der Leitung von Thomas Fuchsmann. Verantwortliche für die technische Unterstützung an der Hochschule der Medien ist Beate Schlitter.

1.1.6 Dank

Dank für die Unterstützung geht an: Thomas Braun und Solveigh Jäger.

2 Die Entstehung eines virtuellen Charakters

2.1 Einführung

2.1.1 Allgemeines

Dieses Kapitel beschäftigt sich mit der visuellen Umsetzung von Charakteren. In der Einführung überblicken wir zunächst die wichtigsten Zusammenhänge der Theorie, lernen einige bekannte Spielfiguren kennen und gehen schließlich genauer auf Computerspiele, deren Möglichkeiten und Einschränkungen ein. Die Entstehungsprozesse von der Idee, bis hin zum fertigen, animierbaren Charakter sind die Inhalte dieses Kapitels und Vorrausetzung für die dynamische Animierung des Charakters in einem Spiel bzw. einer 3D-Echtzeitanwendung.

Wir zeigen den Entstehungsprozess eines solchen Charakters anhand der überarbeiteten Spielfigur Sophie Faber aus „NOAH“. Das Spiel und dessen Engine setzen natürlich einige Richtlinien und Demarkationen voraus, die eine allgemeine Beschreibung der Entstehung erschweren. Unserer Erfahrung und Ansicht nach, sind jedoch genau diese Einschränkungen sehr wichtig, da sie maßgeblich Einfluss auf die Techniken und das Aussehen eines Charakters haben. OGRE, unsere eigene Anwendung und natürlich das Spiel selbst, legen die Richtlinien fest, die Sophies Erscheinungsbild und natürlich auch ihre Technik gemeinsam prägen.

2.1.2 Die vier Stufen der Immersion

Mithilfe von Computern und Programmen, ist es möglich neue, virtuelle Welten zu betreten. Die Immersion umschreibt dabei die Erfahrung eines Menschen in eine virtuelle Welt einzutauchen. Computerspiele ermöglichen eine starke, emotionale Beteiligung eines Spielers, wobei dieser Prozess natürlich vom Spiel, der Persönlichkeit des Spielers, aber auch von der Dauer des Spiels abhängig ist. Der britische Autor und Forscher Richard Bartle, bekannt durch seine Forschungen im Bereich der künstlichen Intelligenz und seinen Arbeiten an Multiplayer-Online-Spielen, unterschied in seiner Arbeit „Designing Virtual Worlds“ zwischen vier unterschiedlichen Stufen der Immersion [Bartle, 2003, S. 154]:

- Spieler/Player
- Avatar
- Charakter/Character
- Persona

Ein ***Spieler*** ist eine echte menschliche Person, die sich mit der interaktiven Welt bewusst auseinandersetzt. Der Spieler sieht die Figur dabei als Medium, um mit ihr virtuelle Umgebung zu verändern oder zu beeinflussen. Die meisten Spieler kontrollieren dabei ein Objekt oder eine einfach Form, um sich in der virtuellen Welt zu Recht zu finden.

Ein ***Avatar*** ist eine repräsentative, meist visuelle Darstellung des Spielers in der virtuellen Welt. Der Spieler betrachtet seinen Avatar als Stellvertreter und spricht von „seiner“ Figur in der dritten Person. Der Spieler bestimmt dabei meistens selbst das Aussehen des Avatars. Viele Avatare haben die Form eines Wesens oder einer anderen Grafik.

Die Stufe der Immersion, in der faszinierte Spieler am häufigsten zu finden sind ist, ist der ***Charakter***. Der Spieler hört auf über den Repräsentanten im Spiel nachzudenken und überträgt seine Gefühle und seine Persönlichkeit auf die Figur in der virtuellen Welt. Hier spricht der Spieler entweder von sich (in Form seiner Figur) in der ersten Person („Ich habe das Monster getötet“) oder wie von einem guten Freund, in der zweiten Person („Gordon hat den

Generator zerstört“). Den virtuellen Charakter bezeichnet man daher auch als Simulakrum („Abbild, Spiegelbild“) des Spielers, da die Spielfigur mit seiner Person verwandt oder ihm ähnlich ist.

Die extremste Stufe nennt Bartle ***Persona***. Hierbei verschwindet die Spielfigur und der Spieler wird Teil der virtuellen Realität. Der Spieler befindet sich darin, erfüllt kein Rollenspiel mehr, sondern **wird** selbst zur Figur. Es kommen dabei keine sensorischen Filter, Projektion oder dazwischenliegende Medien zum Einsatz.

Die Entstehung des Charakters, d.h. der dritten und häufigsten Stufe der Immersion, bildet den Schwerpunkt dieser Ausarbeitung. Dabei ist sie nur ein geringer und vor allem aber subjektiver Teil, der Handlungen und Persönlichkeit einer Spielfigur. Sie beschreibt die Empfindung, die der Spieler im Spiel haben wird, und stellt einen der wichtigsten Faktoren dar, die später auch zu einem intensiveren Erlebnis und damit maßgeblich zum Erfolg des Spiels beitragen kann. Ziel der Spielentwickle ist es daher die Immersion durch glaubwürdige, lebendige Charaktere zu verbessern.

2.1.3 Figuren mit Charakter

Der Begriff des Charakters im Hinblick auf Computerspiele, kommt ursprünglich von den traditionellen Rollenspielen, in welcher der Spieler in die Rolle eines fiktiven Charakters schlüpft. In sogenannten Pen-&-Paper-Rollenspielen ist ein Charakter dabei eine fiktive Figur, die höchstens durch eine Zeichnung oder einfache Grafik dargestellt wird. Die einzelnen Mitspieler einer „Heldengruppe“ steuern in einem solchen Spiel jeweils ihre Figur im Rahmen einer vorgegebenen Geschichte (meistens durch einen Moderator erzählt und beschrieben). Sie treffen dabei Entscheidungen, die vom Regelwerk des Rollenspiels zugelassen sind und beeinflussen mit ihren Helden den Ablauf der Geschichte. Hierbei entwickelt sich mit der Zeit die Persönlichkeit eines Charakters, der Spieler verbessert dessen Attribute und Fähigkeiten.

Die eigentlich richtige Bezeichnung für den Spiele-Charakter, wäre Figur. Der Begriff Charakter beinhaltet in diesem Kontext jedoch nicht nur die äußere Darstellung, sondern auch die Persönlichkeit und Eigenschaften dieser Figur. Durch die Immersion entsprechen diese meist denen des Spielers, sodass der Charakter quasi auch die Dispositionen und Wesenszüge einer Figur beinhaltet. Die Figur wird dadurch zu einem Individuum, das zwar nur auf dem Bildschirm existiert, für den Betrachter aber lebendig wirkt.

Wir wollen die Lebendigkeit dieser Figuren hervorheben und sprechen daher im weiteren Verlauf dieser Ausarbeitung von ***Charakteren***. ***Virtuelle*** Charaktere sind dabei Figuren, die allein am Computer entstanden sind. Die schier unbegrenzten Möglichkeiten der Computergrafik lassen dabei jede Phantasie der Künstler Wirklichkeit werden. Und so entstehen die unterschiedlichsten Kreaturen, die in Bildern, Animationen und Spielen eine Funktion übernehmen können. Charaktere müssen dabei nicht immer mit dem Benutzer interagieren oder dessen Repräsentant in einer virtuellen Umgebung sein, sondern können auch eine komplett neue Funktion übernehmen, z.B. die eines Schauspielers. Solche Charaktere tauchen in der Werbung auf, erscheinen auf Plakaten und Postern, und bewegen sich in Film und Fernsehen. Natürlich müssen solche Charaktere nicht unbedingt wie Menschen aussehen, sondern können auch in der Form von Monstern, Fabelwesen oder sonstigen fiktiven Kreaturen in Erscheinung treten. In der Ich-Perspektive sogenannter Ego-Shooter werden Charaktere im Laufe eines Spiels nicht sichtbar. Obwohl keine Darstellung dieser Figuren erfolgt, sprechen wir von virtuellen Charakteren, da der Spieler sich in einer virtuellen Welt mit dessen Spielfigur identifiziert, und sein Verhalten auf sie überträgt.

2.1.4 Das Uncanny Valley

Eines der rätselhaftesten Phänomene, das bei der Darstellung von künstlich geschaffenen Figuren auftritt, ist das zunächst widersprüchlich erscheinende „Uncanny Valley“ [Mori, 1970, S. 32-35], ein Effekt in der menschlichen Psyche, der aus der Robotertechnik seit etwa Anfang der 70er Jahre bekannt ist. Es geht dabei um den gefühlten Verlauf des Anthropomorphismus (Menschenähnlichkeit) eines Betrachters, der auf eine künstliche Figur (z.B. ein Android oder Avatar) blickt.

Obwohl man annehmen könnte, dass sich der Anthropomorphismus proportional zum Grad des Realismus verhält, d.h. dass der Betrachter eine Figur immer weiter „akzeptiert“, je glaubwürdiger sie aussieht, so zeigt sich durch zahlreiche Versuche, dass die Akzeptanz der Menschen zu solchen Figuren, ab einem gewissen Punkt plötzlich sehr stark abnimmt. Sie steigt erst wieder, wenn ein menschlicher Charakter nicht mehr vom Original zu unterscheiden ist. Dieses Phänomen wird auch deutlich, wenn Menschen abstrakte, tierähnliche oder bewusst künstlich gehaltene Charaktere (z.B. verniedlichte Roboter) als sehenswerter empfinden, als solche, die menschenähnlich sind.

Die Erklärung dafür liegt in der Psychologie des Menschen: Das Gehirn bemisst künstlich geschaffene Figuren (ob virtuell oder real), zuerst instinktiv danach, ob sie menschlich wirken **wollen**. Wird von solchen Figuren der Anspruch erhoben, wie ein Mensch auszusehen, werden sie automatisch mit einem Menschen verglichen. Winzige unnatürliche Fehler im Aussehen oder in ihren Bewegungen führen zu Irritationen, die vom menschlichen Gehirn als negativ und befremdlich bewertet werden. Versucht eine Figur erst gar nicht menschlich zu sein, wird sie vom Betrachter anders betrachtet. Ist dies der Fall, sucht der Zuseher nach menschlichen Merkmalen (z.B. zwei Augen, zwei Arme, zwei Beine), menschenähnlichem Verhalten, Lauten oder Bewegungen. Hier findet eine Neubewertung der Glaubwürdigkeit statt, die bei abstrakten Figuren einfacher abläuft. Kindern fällt diese Neubewertung besonders leicht.

Folgende Grafik verdeutlicht den Verlauf des Uncanny Valleys [MacDorman, 2005, S. 1]:

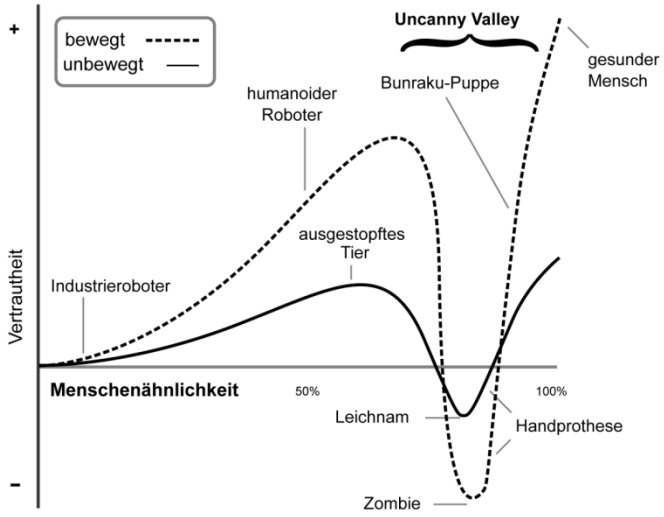


Abbildung 1: German Illustration of the Uncanny Valley: Picture under the GNU Free Documentation license. Version vom 19. Februar 2008, Quelle: Wikipedia, Autor: Tobias K.

Ein reelles Beispiel für diesen Effekt zeigt sich im „Madame Tussauds“-Museum in London. Die Wachsfiguren prominenter Persönlichkeiten wirken auf den Betrachter trotz ihres hohen Detailgrads und ihrer natürlichen Farben unheimlich und befremdlich. Dies lässt sich nicht nur allein mit der starren Pose der Figuren erklären, sondern auch dadurch, dass trotz des hohen Glaubwürdigkeitsgrades eben keine **vollständige** Übereinstimmung mit dem Original (einem echten Menschen) herrscht (wie bei einem Photo).

Auch Wissenschaftler, die sich mit dem Bau von menschenähnlichen Robotern (Androiden) beschäftigen (z.B. die japanischen Prototypen Repliee Q1 und Geminoid HI-1), haben Probleme das Uncanny Valley zu überwinden.

Computeranimationen, die virtuelle Charaktere als Darsteller verwenden, haben es schwierig, da kleine Fehler im Aussehen oder in der Animierung sofort kritisiert und negativ bewertet werden. Das deutlichste Beispiel hierfür ist der komplett computeranimierte US-amerikanische Kinofilm „Die Legende von Beowulf". Darin wurden existierende Hollywood-Schauspieler durch virtuelle Doubles ersetzt. Das Problem wird also dadurch verstärkt, dass der Zuschauer die echten Personen schon kennt, die sich zwar nur noch in kleinen Details von ihren animierten Partnern unterscheiden. Jedoch deuten genau diese Unterschiede darauf hin, dass mit den virtuellen Charakteren wohl etwas nicht stimmt.

Das Uncanny Valley zu überwinden, zählt derzeit wohl zu einer der größten Herausforderung der heutigen Computergrafik. Während die 3D-Computerspiele im Realismus- und Glaubwürdigkeitsgrad sich noch weiter steigern, befinden sich die weitaus aufwendigeren Charaktere aus Computeranimationen mitten drin – im Uncanny Valley.

2.1.5 Technische Grenzen

Computersysteme sind aufgrund unterschiedlicher Probleme und Begrenzungen nur bedingt zur Darstellung von großen, virtuellen Welten und äußerst detaillierten Charakteren fähig. Anwendungen mit Echtzeitdarstellung leiden besonders unter der begrenzten und veränderlichen Leistungsfähigkeit von Computern. Aufwendige Szenen, physikalische Simulationen und komplexe Figuren mindern die Qualität, die Genauigkeit oder Quantität von Geometrie. Wir haben die wichtigsten Punkte dieser Begrenzungen zusammengefasst:

2.1.5.1 Die Leistungsfähigkeit von Computern

Zwar nimmt dank Mehrkernsystemen und schnelleren Prozessoren die Rechenleistung von CPUs und GPUs immer weiter zu, doch liegt sie besonders bei 3D-Echtzeitanwendungen, physikalische Simulationen und komplexen Lichtberechnungen (Radiosity, Global Illumination, Sub-Surface-Scattering, Caustics, Refraktionen, Reflexionen, etc.) noch weit hinter der erforderlichen Schnelligkeit zurück.

Für realistischere Szenen ist dabei besonders die **Kombination** unterschiedlicher Rendertechniken erforderlich (z.B. Raytracing, Caustics und Global Illumination), die aber selbst von sehr schnellen Computern nur mit sehr viel Zeitaufwand berechnet werden können. Während sich Computerspiele zahlreicher Tricks bedienen, um dieses Problem zu umgehen, ist eine exakte Berechnung realistischer Szenen immer noch sehr zeitintensiv und liegt weit über einer Echtzeitdarstellung. Bei Computeranimationen (d.h. beim Rendern von Bilddateien, statt der Grafikwiedergabe in Echtzeit) ist es möglich, mithilfe zusätzlicher CPUs oder ganzer Computersysteme (sogenannter Renderfarmen), die Berechnung zu beschleunigen.

2.1.5.2 Speicher

Grafik- und Arbeitsspeicher werden besonders bei großen und vielen Maps sehr schnell ausgelastet. Zwar werden heute Grafikkarten mit deutlich mehr Speicher gebaut, als zur Wiedergabe eines einfachen Monitorsignals notwendig ist, doch benötigen heutige 3D-Beschleunigungskarte den Zwischenspeicher zum Darstellung von Grafiken in Echtzeit. Hochaufgelöste Texturen und HDRIs (High Dynamic Range Images) haben praktisch zu einer Explosion der benötigten Speichergrößen (derzeit bei 256 bis 512 MB RAM, bei neuen Grafikkarten sogar bis zu 1 GB) geführt. Eine Obergrenze ist noch nicht abzusehen.

2.1.5.3 Maschinengenauigkeit

Algorithmen physikalischer Simulationen oder exakter Beleuchtungsberechnungen sind wegen begrenzter Adressräume oft sehr ungenau. Divisionen, Berechnungen von Integralen oder Ableitungen, führen oft zu Rundungsfehler von Gleitkommazahlen (reelle Zahlen), und teilweise zu drastischen Darstellungsfehlern in Simulationen, Animationen oder gar beim Rendern. Durch die Einführung von 64-Bit-Architekturen (bei CPUs, Programmen und Betriebssystemen), die auch eine Vergrößerung des Arbeitsspeichers zulassen, ist es wahrscheinlich, dass dieses Problem mit der Zeit stark verringern wird.

2.1.6 Charaktere auf Bildern

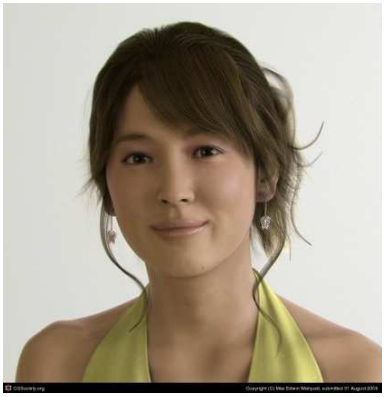


Abbildung 2: Indonesian girl, Copyright: Max Wahyudi, Quelle: maxed-win.cgsociety.org

Bilder sind die wohl älteste und verbreitetste Form Charaktere darzustellen. Charaktere transportieren Botschaften, erregen Aufsehen und vermitteln Emotionen. Aber egal ob im Werbe-, Web- oder im Kunstbereich: die Bedeutung digitaler Darsteller nimmt dabei immer mehr zu. Sie lassen sich in kurzer Zeit an nahezu jedes Bedürfnis anpassen, sind veränderbar, zeitlos und wirken glaubwürdig und echt.

Um Verwechslungen mit Photos zu vermeiden, sind viele virtuelle Computerfiguren sogar häufig auch in bewusst stilisierter oder übertriebener Form anzutreffen. Mithilfe von Bildbearbeitungsprogrammen lassen sich Charaktere auf Bildern leicht überarbeiten und retuschieren.

2.1.7 Charaktere in Computeranimationen

Computeranimationen sind Filme, die mithilfe von Computergrafikprogrammen entstanden sind. Generell werden dabei Bilder in Einzelbildern gerendert (berechnet) und in Dateien gespeichert. Werden diese Einzelbildfolgen mit mehr als 24 Bildern pro Sekunde abgespielt, ergibt sich eine flüssige Bewegung und man spricht von einer Animation.

Die bekanntesten Beispiele für Computeranimationen, sind die Filme der Pixar Animation Studios, die mit Filmen wie Toy Story (der erste komplett computeranimierte Kinofilm), Die Monster AG, Findet Nemo und Ratatouille berühmt geworden sind. Diese abendfüllenden Filme, die komplett ohne real gedrehte Szenen auskommen, zeichnen sich besonders durch zahlreiche Innovationen bei der Entwicklung von Simulations- und Animationstechniken aus, die von Produktion zu Produktion verbessert werden. Pixar setzt jedoch weitestgehend auf comichafte Charaktere oder Tiere mit menschlichen ähnlicher Physiologie, um das „Uncanny Valley“ zu umgehen. Der Erfolg computeranimierter Filme lässt sich anhand der Schließlung der Zeichentrickabteilung von Walt Disney beobachten, die im Januar 2006 auch die Pixar Animation Studios für 7,4 Milliarden US-Dollar aufgekauft haben.

Der erste computeranimierte Spielfilm mit weitestgehend glaubwürdigen, menschlichen Charakteren (speziell der Figur Aki Ross) war die US-amerikanische, japanische Co-Produktion „Final Fantasy: Die Mächte in dir“[1]. Obwohl

[1] Final Fantasy: Die Mächte in dir – Der Titel des Films lässt auf eine Verfilmung des gleichnamigen Computerspiels schließen. Tatsächlich hat er nur wenig mit der Spielreihe gemeinsam.

der Film mehr als 130 Million Dollar gekostet hatte und sehr stark auf die Verwendung von Motion Capturing zurückgegriff, blieb ihm ein Erfolg an den Kinokassen verwehrt. Grund ist unter anderem wieder das „Uncanny Valley".

Zwar werden derzeit nur selten abendfüllende Computeranimationen mit virtuellen Menschen produziert, dafür tauchen diese aber als Spezialeffekte in Realfilmen immer häufiger auf. Um echte Darsteller keiner unnötigen Gefahr auszusetzen oder für Stunts, die von Menschen nicht ausgeführt werden können, werden in zahlreichen Spielfilmen virtuelle Stuntmen oder Doubles eingesetzt, die in den Realfilm eingefügt werden. Ziel der Künstler ist es natürlich den Unterschied zwischen Real- und Animationsfigur zu verbergen.

Bekanntes Beispiel hierfür ist die Verfilmung der „Der Herr der Ringe"-Trilogie, wo Computeranimationen und echte Aufnahmen geschickt miteinander kombiniert wurden. Der Einsatz virtueller Stuntmen, Massensimulationen zur Animierung ganzer Armeen und die Erschaffung zahlreicher, phantasievoller Kreaturen zählen zu den Meilensteinen der Charakteranimation. Besonders die Animationen des Wesens „Gollum" fügen sich perfekt in den Realfilm ein und gelten bislang noch als unerreicht.

Die Reihe computeranimierter Filme hat sich in den letzten Jahren immer weiter fortgesetzt, jedoch gibt es nur vereinzelt Anzeichen dafür, dass echte Darsteller vollständig von virtuellen Charakteren ersetzt werden könnten. Bislang werden menschliche Charaktere in Computeranimationen meist für visuelle Effekte verwendet, die mit Real- oder Modellaufnahmen allein nicht möglich gewesen wären.

2.1.7.1 Das Emily-Projekt

Die kalifornische Firma Image Metrics stellte 2008 das sogenannte „Emily Project" vor, in welchem die US-amerikanische Schauspielerin Emily O'Brien ein digitales, aber äußerst glaubwürdiges CG-Double bekam. Über „Performance-Driven Facial Animations", sprich mithilfe von Realaufnahmen, über die der digitalisierte Klon von Emilys Gesicht gelegt und angepasst wurde, schufen die Künstler einen nahezu vollkommen glaubwürdigen Charakter für Computeranimationen und verkündeten die Überwindung des Uncanny Valleys.

Die Technik arbeitet anstelle des klassischen Motion Capturings mit einem speziellen Rotoscopingverfahren, das sehr viel mehr Details und Feinheiten bei der Animation des Gesichtes beachtet. Da man auf Zähne, Lippen, Zunge und Augen keine Reflektorpunkte aufkleben kann, und die klassischen Motion Capturing-Verfahren weder Faltenbildung noch die 50 Muskeln eines Gesichts aufzeichnen können, werden Aufnahmen einer gewöhnlichen Kamera per Rotoscoping nachgeahmt.

Image Metrics arbeitet hier mit einer Bibliothek aus „über 75 hochaufgelösten Blendingtextures- und 30 unterschiedlichen Displacement Maps"[Image Metrics, 2008], die für jede angewendete Mimik in Emilys Gesicht überblendet werden können. Die Maps wurden mit einer speziellen Vorrichtung aufgenommen, die von allen Seiten gleichmäßiges, diffuses Licht auf ihr Gesicht strahlt und von jeder angewandten Miene Photoaufnahmen macht.

Wir haben das Emily-Video angesehen und es genauer unter die Lupe genommen. Einige Menschen wurden während der Animation von uns dazu befragt, ob „ihnen etwas auffällt". Dies geschah, bevor im Video gezeigt wird, dass Emily nicht echt ist. Wenn man anfangs als Zuschauer nicht wusste, dass es sich um einen virtuellen Charakter handelte, befanden sich die Zuschauer (egal ob Fachmann oder Laie) tatsächlich in dem Glauben, dass das digitale Double von Emily echt ist. Erst als sie über den „Betrug" aufgeklärt wurden, waren die Probanden imstande, gewisse Makel dieser äußerst beeindruckenden Animationen zu identifizieren. Zu erwähnen ist nochmal, dass nur Emilys Gesicht, nicht der restliche Körper oder ihre Umgebung computeranimiert sind.

2.1.8 Charaktere in interaktiven Anwendungen

Virtuelle Charaktere finden aber auch in Software- oder Online-Anwendungen immer mehr Einsatz. Um z.B. eine persönliche Bindung mit dem Benutzer oder mit dem Kunden herzustellen, wird ihm im Internet oder in einer Anwendung ein virtueller Berater zur Seite gestellt. Dieser hilft dem Benutzer bei der Suche nach Informationen oder benachrichtigt Kunden über den Stand ihrer Einkäufe. Firmenmaskottchen werden als virtuelle Charaktere plötzlich zum Leben erweckt. Ihnen wird Persönlichkeit verliehen, auf deren Sympathie ein Kunde ansprechen soll.

Abbildung 3: Eve - Kundenberaterin, Copyright: Yellow Strom, Quelle: www.yellowstrom.de

Ein bekannter (animierter) Charakter aus dem Internet ist Eve, die virtuelle Kundenberaterin der Firma Yellow Strom [Yello Strom GmbH, 2008]. Diese reagiert mithilfe einer Sprechblase auf unterschiedliche Eingaben des Benutzers und liefert Informationen zum Service des Unternehmens. Eve reagiert dabei nicht nur auf Fragen, sondern auch auf Begrüßungen, Beleidigungen und Komplimente. Wenn Eve z.B. dreimal beleidigt wurde, verschwindet sie eine Zeit lang.

Ein weiteres bekanntes Beispiel einer Anwendung, ist der Tamagotchi (zu Deutsch: Ei-Uhr), ein kleines tragbares Elektronikspielzeug aus Japan, das sich als virtuelles Haustier (Küken) besonders gegen Ende der 90er weit unter Kindern und Teenagern verbreitet hat. Tamagotchis haben nach ihrer Aktivierung unterschiedliche Bedürfnisse (Schlafen, Essen, Trinken), die bis zu ihrem Lebensende oder nach einem manuellen Neustart erfüllt werden müssen. Die Geräte entwickeln mit der Zeit eine eigene „Persönlichkeit“ und passen sich dem Verhalten ihres Besitzers an. Konkretes Ziel der Anwendung gibt es nicht, jedoch ist es üblich dem Tamagotchi ein möglichst langes und zufriedenes Dasein zu bereiten.

Sogar in Autos halten virtuelle Charaktere inzwischen Einzug. Im neuen Radio-Navigationssystem RNS510 von Volkswagen ist ein sogenanntes „Avatar-basierte Benutzerinterface“ integriert. Der virtuelle Charakter „Carla“, erklärt dem Fahrer als „audiovisuelle Hilfsoption“ die Funktionsweise des Gerätes [Volkswagen AG, 2008]. Carla bildet dabei eine Weiterentwicklung klassischer Navigationssysteme, die bislang nur mit einer Stimme auskommen. Sie erscheint auf dem Display des Gerätes und erklärt dessen Funktionen.

Anhand dieser Beispiele sieht man, dass Figuren in Anwendungen und Geräten eingesetzt werden können, um mit dem Benutzer zu kommunizieren und ihm einen leichteren Zugang zu einer Technik, zu einem Produkt oder einer Dienstleistung zu ermöglichen. Wie beim Tamagotchi können solche Charaktere dabei auch der Unterhaltung dienen. Aufgrund begrenzter Ressourcen oder technischer Möglichkeiten wirken Charaktere in solchen Anwendungen aber in ihrer Darstellung noch sehr künstlich und unangenehm. Oft wird kritisiert, dass solche Charaktere den Benutzer mit ablenkenden Animationen, ständigen Einblendungen oder ungewolltem Erscheinen nerven. Ein solches Beispiel ist die Animation des Hundes in der Suche des Betriebssystems Windows XP, die von sehr vielen Benutzern gleich nach seinem Erscheinen deaktiviert wird.

Charaktere in solchen Anwendungen werden als „Gegenüber“ zum Nutzer betrachtet. Der Nutzer springt nicht in die Rolle einer anderen Figur, sondern bleibt er selbst (erste Stufe der Immersion) und interagiert mit der virtuellen Figur, anstatt sie zu steuern.

2.1.9 Charaktere in Computerspielen

Computerspielfiguren gibt es praktisch seit Computerspiele existieren. Die Immersion eines Spiels wird dann verbessert, wenn Menschen in die Rolle eines Charakters schlüpfen können, dessen Steuerung übernehmen und die Welt aus seiner Sicht erleben. Meist ist dies nur in Computerspielen der Fall. Zwar kommen verhältnismäßig viele Titel ohne Charaktere aus (z.B. Puzzles, Strategie- und Geschicklichkeitsspiele), trotzdem beinhaltet nahezu jedes Spiel, mit einem gemeinhin ausgearbeiteten Handlungsrahmen oder einer spannenden Geschichte, mindestens einen Charakter, der dann entweder abstrakt oder visuell dargestellt wird[2].

Der Schwerpunkt dieser Ausarbeitung liegt, wie schon erwähnt, auf der Erschaffung und Animierung der Figur Sophie aus dem Computerspiel „NOAH". Bevor wir uns allerdings dem Entstehungsprozess dieses Charakters widmen, möchten wir zunächst noch auf einige andere Computerspielfiguren eingehen, die im Lauf der Zeit große Berühmtheit erlangt haben. Einige Merkmale dieser Figuren haben uns natürlich auch beeinflusst und waren Ausgangspunkt für neue Ideen.

2.1.9.1 Mario (Super Mario Bros., 1983)

Die Videospielfigur und das Werbe-Maskottchen des Unternehmens Nintendo ist wohl der beliebteste und bekannteste Held der Computer-, Konsolen- und Videospiele-Welt. Die „Super-Mario"-Titel wurden nahezu 300 Millionen Mal verkauft und gilt damit als erfolgreichste Spiele-Reihen überhaupt. Die Figur und ihre Abenteuer wurden als auf die Kinoleinwand gebracht und als TV-Serie umgesetzt – mit wenig Erfolg. Die Umsetzungen für PCs war auch wenig erfolgreich. Nintendo blieb allein mit dem Konsolenmarkt, besonders mit der Wii-Konsole erfolgreich und treu.

Super Mario wurde zunächst durch die Arcade-Automaten in den USA, Konsolen für TV-Geräte sowie durch die Verbreitung tragbarer Videospielsystemen (Game Boy) bekannt. Besonders die Titelreihen Super Mario Bros., Super Mario Land, sowie Super Mario World waren zu Beginn äußerst erfolgreich und wurden auf verschiedene Plattformen portiert. Später kamen Renn-, Sport-, Party- und Spaßtitel hinzu, die besonders im Split-Screen-Modus, d.h. mit mehreren Spielern an einer Konsole, beliebt waren.

Das Design des kleinen, dicken italienischen Klemptners mit Schnauzer, roter Latzhose, blauem Hemd und Mütze, entstand durch die begrenzten Ressourcen der damaligen Hardware, die nur eine maximale Auflösung der Spielfigur von 16x16 Pixel erlaubte. Durch die Mütze war es nicht nötig Haare darzustellen. Die „Knubbelnase" und bunte Kleidung diente dazu die einzelnen Körperteile deutlicher voneinander zu unterscheiden. In späteren Spielen änderte sich die Farbkombination der Kleidung von Super Mario. Sein comichaftes Aussehen blieb aber bis zu den heutigen 3D-Spielen von Nintendo erhalten.

Abbildung 4: Super Mario All Stars für SuperNES

2.1.9.2 Lara Croft (Tomb Raider, 1996)

Zu den bekanntesten Computerspielfiguren zählt natürlich Lara Croft, die Heldin der Action-Adventure-Reihe „Tomb Raider". Die von der Firma Eidos Interactive publizierte und von Core Design (später Crystal Dynamics) entwickelte Serie, gehört mit neun PC-Spieltiteln, mehreren Neuauflagen und 32 Millionen verkauften Exemplaren zu einer der meistverkauften Computerspielreihen der Welt. Der Charakter-Designer Toby Gard entwickelte 1996

[2] Beispiel ist das Echtzeit-Strategiespiel „Homeworld", das trotz seiner epischen Geschichte und Heldensage, nahezu vollständig ohne Charaktervisualisierungen auskommt. Die Geschichte und Dialoge werden nur aus dem Off heraus erzählt.

die Figur einer Archäologin, die glaubwürdig, menschlich, geschickt und sehr stark wirken sollte. Durch „grazile, akrobatische Bewegungen“ sollte sie sich durch virtuelle 3D-Welten bewegen. „Mit ihren sehr weiblichen Attributen eigentlich in erster Line für Männer programmiert, habe sie überraschend auch Frauen angesprochen und damit eine Wende eingeleitet. Bis dahin war Computerspielen in erster Linie etwas für Jungs.“[Haake, 2007]

Als erste Computerspielfigur, hat Lara Croft es auch auf die Kinoleinwand geschafft. Ihre Figur wurde durch die US-amerikanische Schauspielerin Angelina Jolie verkörpert. Inzwischen existieren zahlreiche Titel für Konsolen, Handys und sogar PDAs. Der Vertrieb wird von einer großen Anzahl an Merchandise-Artikeln begleitet, die überwiegend mit der Figur Lara Croft zusammenhängen.

Abbildung 5: Lara Croft, Screenshot aus „Tomb Raider – Legend“

Der Computerspieler übernimmt in der 3D-Welt die Steuerung in der Dritten-Person-Ansicht (Sichtkamera hinter der Spielfigur). Der Spieler muss dabei unterschiedliche Herausforderungen überwinden. Er stellt in unterschiedlichen Levels seine Geschicklichkeit unter Beweis, löst Rätsel, entdeckt Geheimnisse und schaltet verschiedene Gegner aus. Meist befindet man sich in unterschiedlichen Teilen der Welt, und ist auf der Suche nach verschollenen Artefakten.

Der große Erfolg von Tomb Raider lässt sich auch anhand der erotischen Darstellung der Figur erklären. Brüste, Hüfte und Lippen wurden übertrieben gezeichnet und stilisiert[3]. Ihre Haare sind zu einem Zopf geflochten, einige Strähnen hängen ins Gesicht. Ihre Bekleidung wechselt in den Spielen, besteht aber meist aus braunen Hotpants, einem kurzem grünen Oberteil, dunklen Stiefeln und Handschuhen. Sie trägt an beiden Oberschenkeln Holster für Waffen, in einen kleinen Rucksack befindet sich das Inventar (z.B. Fernglas und Kompass).

Laras Charakter und Fähigkeiten werden im Laufe der Serie immer weiter ausgebaut und erweitert. Die Modelle und Textur der Figur, die Levels und Gegner werden mit jedem Teil detaillierter und glaubwürdiger. Ihre Persönlichkeit ändert sich ebenfalls. So wird sie in dem Titel Lara Croft: Tomb Raider – Angel of Darkness ernsthafterer und finsterer dargestellt. In Lara Croft: Tomb Raider – Legend erfährt man Näheres aus ihrer Kindheit.

2.1.9.3 Gordon Freeman (Half-Life, 1998)

Ein Charakter der bekanntesten und berühmtesten Ego-Shooter-Reihe (mit der Perspektive aus der Sicht der Spielfigur) ist Gordon Freeman aus der Serie „Half-Life“. Der Computerspieler schlüpft in die Rolle eines brillanten Wissenschaftlers, dem bei einem geheimen Experiment in dem US-Forschungszentrum Black Mesa in New Mexico ein fataler Fehler unterläuft. Bei dem Versuch einen außerirdischen Kristall als Energiequelle zu nutzen, öffnet sich das Tor zu einer fremden Dimension. Fortan muss sich der Spieler nicht nur mit außerirdischen Kreaturen herumschlagen, sondern auch mit Marines und dem Black-Ops-Spezialkommando, das alle Zeugen des Vorfalls versucht auszuschalten.

Da der Spieler das Spiel aus der Sicht der Figur erlebt, ist Gordon Freeman in seinem HEV-Anzug nicht sichtbar. Die Besonderheit des Charakters liegt in seiner Rolle und in seinem Einfluss auf das Spiel. Der Charakter spricht nicht, es gibt keine Zwischensequenzen oder Briefings, die den Spieler instruieren. Alle Aktionen geschehen durch die Augen der Spielfigur und werden von einer gewissen Hilflosigkeit begleitet, da der Spieler von den Ereignissen praktisch überrannt wird, isoliert und machtlos gegenüber seinen Feinden scheint. Der Spieler lernt daher einen besonderen Umgang mit der Figur Gordon Freeman, der eine tragische Mitschuld an der Katastrophe trägt.

[3] Es geht das Gerücht, dass die große Oberweite der Lara Croft auf eine anfangs fehlerhafte Programmierung zurückzuführen ist.

Ein besonderes Merkmal der Half-Life Engine ist die Unterteilung der Levels in viele kleinere Sektionen, die je nach Transferpunkt kurz geladen werden. Der Spieler erhält dadurch den Eindruck sich in einem großen, zusammenhängenden Komplex zu befinden. Half-Life bietet Interaktionen mit Objekten (mithilfe einer Phyisk-Engine), mit der Geschichte und Einflüsse der Umwelt auf die Figur. Soundeffekte mit Raumklang und ein eigener Soundtrack auf CD intensivieren das Spielerlebnis.

Abbildung 6: GordonFreeman im Mulitplayer von Halflife

Die Figur Gordon Freeman wird außerhalb des Spiels mit Bart, schwarzer Brille und orangefarbenen MARK IV/V HEV (Hazardous Environment) – Anzug dargestellt. In seiner Hand trägt er meistens eine Brechstange, mit der er sich den Weg freischlägt. Eine weitere wichtige Figur in Half-Life ist der mysteriöse G-Man, ein Regierungsbeamter mit Anzug, Krawatte und Aktenkoffer, der im Laufe des Spiels öfters in Erscheinung tritt, dessen Rolle aber lange unklar bleibt.

2.1.9.4 Nr. 47 (Hitman, 2000)

Ein namenloser Mann wird in einem Genlabor durch das Erbgut fünf verschiedener Super-Krimineller erzeugt. Er ist der Klon aus einem rumänischen Forschungslabors und trägt die Nr. 47, sowie einen Barcode auf seinem Hinterkopf, um ihn mit Scannern leichter identifizieren zu können. Er wird Auftragskiller und kleidet sich unauffällig, mit einem schwarzem Anzug, Hemd, Krawatte sowie einem Aktenkoffer, in dem er ein zerlegtes Scharfschützengewehr transportiert. Die „Hitman"-Reihe erzählt dabei die Geschichte des Klons, der im ersten Teil seinen „Vater" und fast alle seine Brüder tötet. Den einzigen verbliebenen Bruder tötet er am Ende des zweiten Teils.

Abbildung 7: Screenshot aus dem Sequel Hitman 2

Der erste Teil der Hitman-Reihe nutzt als erstes Spiel überhaupt eine Ragdoll-Physik-Engine, um Bewegungen bewusstloser Körper zu simulieren. Eine Besonderheit des Spiels ist dabei auch die Möglichkeit das Missionsziel zu erreichen, ohne Schüsse aus einer Waffe abzufeuern. Dabei wird eine Vielzahl von Lösungswegen angeboten, um lautlos das Ziel zu neutralisieren, ohne dass anderen Personen im Spiel zu Schaden kommt. Zudem können auch herumliegende Gegenstände zur Lösung des Ziels zu verwendet werden.

Wie die Tomb-Raider-Serie hat es Hitman, allerdings mit nur mäßigem Erfolg auf die Kinoleinwände geschafft. Die Spiele-Reihe wurde aber durch die detaillierte und konsequente Charakterzeichnung von Nr. 47 weltbekannt und mit vier Teilen weitergeführt und bildet einen Meilenstein in der Entwicklung der Spiele-Charakter.

2.1.9.5 Max Payne (Max Payne, 2001)

Wohl eine der tragischsten Computerspielfiguren ist der gleichnamige Protagonist der Reihe „Max Payne", die für PC von Remedy Entertainment produziert und bislang von den Labels Gathering, Rockstar und Take Two veröffentlicht wurde. Die Ansicht im Spiel ist aus der dritten Person und bietet als besonderes Feature des Gameplays die sogenannte „Bullet Time", die der Spieler selbst auslösen kann, um die Zeit langsamer laufen zu lassen und gezielter Gegner auszuschalten oder mit verschiedenen Sprüngen vorbeischießenden Projektilen auszuweichen. Das Spiel enthält viele Hommagen an andere Filme und bezieht sich auf Figuren und Orte der nordischen Mythologie. Durch innere Monologe in den Zwischensequenzen und im Spiel teilt Max Payne dem Spieler seine Emotionen mit. Das Spiel an sich ist eine Rückblende und wird quasi im Nachhinein erzählt. Es wirkt durch solche Elemente film-

ähnlicher und atmosphärischer, als andere Spieltitel und wird häufig als Beispiel angeführt, dass Computerspiele auch eine Kunstform darstellen können.

Der Charakter Max Payne ist Polizist und vergleichbar mit klassischen Personen aus dem Film Noir, da er von Selbstzweifeln und Vorwürfen leben muss und sich für die Ereignisse schuldig fühlt, die zum Tode seiner Familie geführt haben. Da diese von Einbrechern getötet wurde, die unter der neuartigen Droge „Valkyr" standen, beschließt Max Payne als verdeckter Ermittler für die Drogenfahndung zu arbeiten. Payne wird im Laufe des Spiels weiter von Albträumen verfolgt und sucht die Drahtzieher hinter dem Kartell.

Abbildung 8: Screenshot aus Max Payne

Das Charakter-Design von Max Payne beruht auf dem Schreiber der Geschichte des Spiels, Sam Lake. Auf sein charakteristisches Gesicht wird auch häufig im Spiel angedeutet. Max Payne trägt meistens eine schwarze Lederjacke (oder Mantel) über einem weißen Hemd, dunkle Hose und Schuhe.

2.1.10 Charaktere in Machinimas

Eine relativ junge Form der Videokunst sind Machinimas (Kunstwort aus: machine, cinema, animation). Ein Machinima ist ein computeranimierter Film, der nicht mithilfe von Videodaten, sondern durch eine 3D-Engine gerendert wird (ähnlich wie animierte Zwischensequenzen aus Spielen). Die Filme werden praktisch nur mithilfe von Spiele-Engines produziert. Viele Machinimas entstehen jedoch auch aus Spielen heraus, werden zu Videos zusammengeschnitten und online verbreitet.

Ein sehr bekanntes Beispiel ist die deutsche Machinima-Serie „Der Abgrund in uns". Die Folgen sind regelmäßig auf der PC-Zeitschrift PC ACTION enthalten und auch Online angeschaut werden können. Inzwischen gibt es auch zahlreiche Internetseiten, Festivals & Wettbewerbe, die für eine immer größere Verbreitung solcher Animationen sorgen. Ein beliebtes Spiel bzw. die Plattform zum Erstellen sogenannter InGame-Videos ist das MMORPG „World of Warcraft" von Blizzard.

Eine besondere Art dieser Filme, erzählt Handlungen ausschließlich mit animierten Objekten (z.B. Fahrzeugen). Diese ersetzen die Charaktere, die Dialoge dann aus dem Off (z.B. die Piloten eines Flugzeugs) heraus erzählen. Die Animationen werden mithilfe von Simulatoren erstellt oder durch Spiele aufgezeichnet und mithilfe von Videoschnittprogrammen zu einer Handlung zusammengestellt.

2.1.11 Medien im Vergleich

Wir wissen inwiefern sich die Möglichkeiten der Darstellung und die Erwartungshaltung des Betrachters bei Charakteren unterscheiden. In Spielen bzw. in 3D-Echtzeitanwendungen liegt dabei die große Schwierigkeit, dass viele negative Faktoren und sehr hohe Erwartungen aufeinander treffen. Das Uncanny Valley vertieft sich durch die Voraussetzung, dass der Spieler mit seinem Objekt interagieren kann. Frames müssen sehr viel schneller berechnet werden müssen, als bei Animationen, die in Einzelbilder gerendert und in Dateien abgespeichert werden können. Bei Machinimas häuft sich das Problem, dass Anwender oft nicht wissen, ob die kleinen Programme auf ihren PCs überhaupt abgespielt werden können. Es macht daher Sinn die einzelnen Medienplattformen, in denen virtuelle Charaktere dargestellt werden können, genauer anzusehen. Im weiteren Verlauf dieser Ausarbeitung ist der Fokus auf die Darstellung von Charakteren in Echtzeitanwendungen gerichtet.

Wir möchten hervorheben, dass die Entwicklung bei Echtzeit 3D-Anwendungen sehr weit vorangeschritten ist, und werden im Laufe dieser Arbeit noch einige Methoden kennenlernen, wie Spiele dieses Handicap verringern.

	Anwendungen	Filme	Bilder	Spiele	Machinimas
Interaktionen (z.B. Steuerung)	ja	nein	nein	ja	nein
Animationen	ja	ja	nein	ja	ja
Simulationen (z.B. physikalisch)	je nach Art (selten)	ja	ja (bei 1 Frame)	ja	ja
Anzahl der Dreiecke	je nach Art (Echtzeit o. in Dateien gerendert)	unbegrenzt (derzeit bei ca. 1 Mio.)	unbegrenzt (derzeit bei ca. 1 Mio.)	begrenzt (derzeit bis ca. 20.000)	begrenzt (derzeit bei ca. 10.000)
Texturspeicher	abhängig vom Medium	unabhängig	unabhängig	abhängig vom Grafikspeicher (derzeit bis 1GB RAM)	abhängig vom Grafikspeicher (derzeit bis 1GB RAM)
Beleuchtung	je nach Art	sehr genau	sehr genau	einfach	einfach
Nachträgliche Bildkorrektur	nicht möglich	möglich (aufwendig)	möglich (meistens der Fall)	nicht möglich	nicht möglich
Echtzeit	ja	nein	nein	ja	ja
Ausgabe	Computer, diverse Geräte	Computer, Fernsehen, Kino	Computer, Druck	Computer, Videokonsolen	Computer
Realismus (Stand: 2008)	niedrig	hoch (nicht perfekt)	sehr hoch (fast perfekt)	mittel (stark am Wachsen)	mittel

Tabelle 1: Vergleich von Medien in Bezug auf Darstellung von Charakteren

2.1.12 Einschränkungen bei 3D-Echtzeitanwendungen

Wie in der Tabelle oben zu sehen ist, haben 3D-Modellierer für einen virtuellen Charakter speziell bei Spielen einige Einschränkungen durch Hard- und Software zu beachten. Für unsere Arbeit bedeutet dies konkret:

- Die Anzahl der Dreiecke („Polycount“) muss für den Charakter begrenzt werden. Aufgrund unserer Erfahrungen mit Spielen (speziell der OGRE-Engine aus „NOAH“) entscheiden wir uns den Charakter mit etwa 10.000 Dreiecke zu modellieren. Für ältere Grafikkarten wird die Geometrie später zu einem niedrigeren LoD (Level of Detail, Detailgrad) reduziert.
- Die maximale Texturauflösung („Mapsize“) ist bei den meisten Grafikarten noch auf 2048 x 2048 Pixel begrenzt. Wir wählen daher die maximale Auflösung für die Texturen und entscheiden uns womöglich während der Weiterentwicklung für eine Reduzierung der Auflösung, um evtl. die Geschwindigkeit des Spiels zu verbessern. Dieses Format bedeutet bei einer Bittiefe von 4 Byte pro Pixel (RGBA[4]): 2048 x 2048 x 4 ≈ 16 MB Speicher.
- Die Anzahl der Maps pro Objekt sind von der Hardware (pro Renderpass) auf max. 8 limitiert. Nur durch Addition oder Multiplikation wäre es möglich noch zusätzliche Maps hinzuzufügen.

[4] Farbkanäle: Rot, Grün, Blau, Alpha mit jeweils 8 Bit

- Die Kanäle für Texturkoordinaten sind auf maximal 16 beschränkt. Da es bei Charakteren allerdings unnötiger Aufwand bedeuten würde mehrmals zu „unwrappen“ [5], wird hier nur ein Kanal verwendet. Zudem werden Texturkanäle für zusätzliche Effekten bei der internen Informationsübertragung von Vertex- zu Pixelshadern verwendet. Bei Objekten und Levels werden meist 2 oder 3 zusätzliche Kanäle für z.B. Pixellighting oder Normalmapping gebraucht.
- Es gibt eine begrenzte Anzahl an virtuellen Knochen („Bones“) für ein IK-System bzw. pro Objekt, welches ein solches System verwendet. Da auf Grafikkarten mit Vertex Shader 2.0 und 3.0 genau 256 Constant Float Register zur Verfügung stehen und für jede Bone-Transformation 3 Register benötigt werden, hat ein IK-System etwa ca. 210 Register zur Verfügung. Die übrigen Register werden u.a. für Kamera-, Licht und Objektpositionen verwendet. Dies bedeutet, dass uns etwa 70 Bones für den Charakter zur Verfügung stehen.

2.1.13 Zusammenfassung der Prozesse

Bei der Analyse (1) der Möglichkeiten und Einschränkungen wird unter anderem deutlich, wie Spiele-Programmierer und Designer zusammenarbeiten müssen. Die Entwickler legen Richtlinien fest, welche die 3D-Artists im Rahmen ihrer Kreativität einhalten. Obwohl wir natürlich parallel an der Entwicklung gearbeitet und regelmäßig Daten miteinander abgeglichen haben, ist dieses Kapitel im Folgenden nach dem gleichen Prozess strukturiert, der auch in der Spiele-Entwicklung beim Design von Charakteren allgemein üblich oder zumindest vergleichbar ist. Konzeptzeichnungen (2) legen früh das Design fest, auf dem die Modellierung (3) des Charakters basiert. Das Texture Mapping (4) bildet die Grundlage für das Sculpting (5) und die Texturierung (6), da beide Techniken Texturkoordinaten benötigen. In vielen vergleichbaren Projekten erfolgt das Sculpting (Displacement Painting oder High-Resolution Modeling) erst nach der Texturierung. Unsere Absicht war es aber nicht Details der Textur zu übernehmen und diese im Relief der Normal Map hervorzuheben, sondern Details auf das Mesh zu zeichnen und diese erst dann zu texturieren. Die Anwendung beider Techniken ist von Fall zu Fall verschieden. Beim Rigging (7) wird ein virtuelles Skelett erstellt und animiert, wodurch der Charakter tatsächlich „lebendig“ wirkt. Das Rigging gehört zu den heikelsten Aufgaben der Entstehung, da hier die Scheitelpunktanzahl des Objekts nicht mehr verändert werden soll. Entstehen neue Punkte im Mesh, verändert sich die Nummerierung der Vertices, was bedeutet, dass der Rigging-Prozess wiederholt werden muss. Mithilfe eines Plugins erfolgt der Export (8) nach OGRE und schließlich die dynamische Animation und Interaktion innerhalb der Engine.

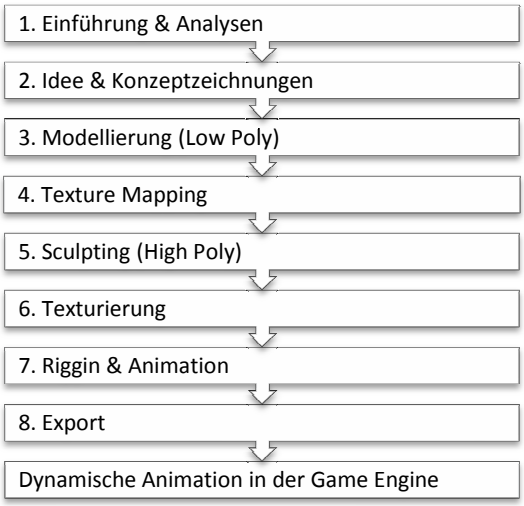


Abbildung 9: Entstehungsprozesse

[5] beschreibt umgangssprachlich den Vorgang der manuellen oder automatischen Zuweisung von Texturkoordinaten

2.1.14 Auswahl der Software

2.1.14.1 Modellierungs- und Animationssoftware

Da das Computerspiel „NOAH auf Basis der OGRE-Engine läuft, benötigten wir ein 3D-Computergrafik- und Animationsprogramm, für das es ein funktionierendes OGRE-Export-Plugin gibt und bei dem eine umfangreiche Funktionalität vorhanden ist. Als 3D-Softwarepaket zur Modellierung und Animation unseres Charakters standen zunächst folgende Pakete für die vorhandenen PCs mit Windows-Betriebssystemen zur Auswahl:

- ***Autodesk 3ds Max***: umfangreiche Funktionalität, gute Bedienbarkeit, höchste Verbreitung
- ***Autodesk Maya***: hohe Funktionalität, spezialisiert auf Computeranimationen
- ***Softimage XSI***: umfangreiche Funktionalität, gute Bedienbarkeit, weite Verbreitung
- ***Microsoft Truespace***: geringe Funktionalität, wenig Verbreitung
- ***Blender 3D***: hohe Funktionalität, hohe Verbreitung, OpenSpource
- ***Maxon Cinema 4D***: mittelmäßige Funktionalität, einfache Bedienbarkeit
- ***Chumbalumsoft Milkshape***: geringe Funktionalität, einfache Bedienbarkeit
- ***NewTek LightWave***: geringe Funktionalität, geringe Verbreitung
- ***SIdeFX Houdini:*** geringe Funktionalität, geringe Verbreitung

Aufgrund unserer Erfahrung, der bisherigen Arbeit an „NOAH“ und anderen Projekten, entschieden uns für Autodesks 3ds Max 2008 auf der 32- bzw. 64-Bit Version von Windows XP. Der umfangreiche Unwrapping-Editor, die einfache Bedienung, der Support, sowie die notwendige Funktionalität insbesondere beim Textur-Baking, bei Skeleton Animationen mit Biped und Bones, sowie die Auswahl an verfügbaren Exportern für OGRE waren entscheidene Vorteile des Programms zur Erstellung eines Charakters.

Über weite Teile des Projekts verwendeten wir die 64-Bit Version des Programmpakets, welche besonders bei der Animation und bei Nutzung des integrierten Mental Rays-Renderers für bessere Ergebnisse sorgte. Beim Export war leider nur die 32-Bit Version zu gebrauchen, da es für 64-Bit keinen geeigneten Exporter gab.

2.1.14.2 Sculpting-Tools

Mit Ausnahme des von Blender 3D unterstützte kein Programm besondere Sculpting-Funktionen, um mit der Hilfe von Normal Maps zusätzliche Details auf die Oberfläche eines 3D Objekts zu zeichnen. Dazu haben wir ein zusätzliches Programm benötigt, wovon uns 5 verschiedene zur Auswahl standen:

- ***Pixologic ZBrush***, mit ZMapper Plugin (beste Funktionalität und gute Bedienbarkeit, sehr ausgereift)
- ***Autodesk Mudbox*** (beste Bedienbarkeit, unausgereift)
- ***Nevercenter Silo 3D*** (rudimentäres Sculpting, wenig Support)
- ***Luxology Modo*** (rudimentäres Sculpting)
- ***Blender 3D*** (noch in der Entwicklung)

ZBrush 3.1war für uns insbesondere in Verbindung mit dem ZMapper Plugin, das umfangreichste und leistungsfähigste Sculpting-Tool. Es kombiniert eine relativ intuitive Oberfläche mit zahlreichen Einstellungen für sehr präzise Normal Maps. Aufgrund einiger Programmfehler, Instabilität oder noch zu geringer Funktionalitäten kamen andere Programme nicht in Frage.

2.1.14.3 Texturierung-Tools

Für unsere umfangreichen Arbeiten zur Texturierung benötigten wir ein 2D-Bildbearbeitungsprogramm. Hier standen uns folgende Pakete zur Auswahl:

- ***Adobe Photoshop*** (gute Bedienbarkeit, größte Verbreitung)
- ***Corel Photopaint*** (gute Bedienbarkeit, hohe Funktionalität)
- ***Corel Paint Shop Pro*** (gute Bedienbarkeit, geringe Funktionalität)

Aufgrund unserer Erfahrungen mit dem Programm und seiner hohen Funktionalität und weiten Verbreitung wählten wir Adobe Photoshop CS3.

2.1.14.4 OGRE-Export Plugins

Für 3ds Max gab es eine Reihe an Plugins für den Export nach OGRE. Hier standen folgende Tools zur Verfügung:

- ***LEXIExporter***
- ***Mesh & Animation-Exporter***
- ***ACE Studios oFusion***
- ***OgreMax Scene Exporter***
- ***Scene Exporter***

Über weite Teile des Projekts verwendeten wir den LEXIExporter und LEXIView für die Export nach OGRE, da diese besonders in Verbindung mit 3ds Max 2008 (64-Bit Variante) am wenigsten Probleme bereitet hatten. Als mehr Funktionen für den Export der Animationen erforderlich waren, steigen wir auf den leistungsfähigeren oFusion Exporter um.

2.1.14.5 Zusätzliche Tools und Plugins

Sonstige Software, die für diese Diplomarbeit verwendet wurde:

- ***LEXIView:*** 3D-Viewer auf Basis der OGRE-Engine und zum Überprüfen der Datenbestände
- ***OgreCommandLineTools:*** zur Überprüfung der Mesh Daten
- ***Blender 3D:*** OpenSource 3D-Programm mit einigen Features
- ***Subversion SVN***: Versionsverwaltung der Diplomarbeit und der Projektdaten
- ***Microsoft Groove:*** zum direkten Austausch von Daten
- ***Microsoft Word***

2.2 Idee & Konzepte

2.2.1 Allgemeines

Die Entstehung eines Charakters beginnt natürlich mit der Idee, einen solchen zu erschaffen. Umstände, Hintergründe und die Methoden dieser Entwicklung beeinflussen das Design, wie auch die Persönlichkeit eine Figur. Dabei spielen die Gedanken seiner Erfinder die wichtigste Rolle. Egal ob in Animationen oder Anwendungen, Charaktere besitzen individuelle Entstehungsgeschichten, die sich in einigen Aspekten sehr ähneln, aufgrund der unzähligen Möglichkeiten in dieser Entstehung aber niemals gleichen. Charakteren wird beim Entwicklungsprozess daher nicht nur ein gewisses Maß an Persönlichkeit verliehen, sondern auch Individualität. Unser Projekt ist dabei besonderes interessant, da eine bestehende Figur neu überarbeitet wird. Charaktere entwickeln sich, und ihre Grundlage ist ihr Design, wovon es zwei mögliche Entstehungsformen gibt: Das Art- und das Story-Driven-Design

2.2.2 Design

2.2.2.1 Art-Driven

Künstler kommen beim Design eines Charakters manchmal auf sehr ungewöhnliche Ideen. Der Erfinder von „Pac-Man“ Toru Iwatani kam z.B. durch eine angefangene Pizza auf den Entwurf seiner Spielfigur. Besitzt die Geschichte einen vernachlässigbaren Wert bei der Gestaltung, wird einfach auf ein gutes oder interessantes Aussehen der Figur geachtet. Man spricht dabei von „Art-Driven“-Design.

2.2.2.2 Story-Driven

Bestimmt eine Geschichte oder die virtuelle Welt das Aussehen eines Charakters, spricht man von „Story-Driven“-Design. Hier werden Verhalten und Persönlichkeit vor dem eigentlichen Erscheinungsbild festgelegt. Oft bevorzugen Künstler eine detaillierte Beschreibung ihres Charakters, um ihn wirklich verstehen, zeichnen und modellieren zu können **Es ist eine ungültige Quelle angegeben.**. Beispiel für dieses Vorgehen ist unsere Figur Sophie Faber.

2.2.3 Sophie Faber

„Sophie Faber ist Mitte zwanzig, Single und Dammschutzkommissarin bei der Dammsicherheit der Stadt „NOAH“. Sie ist sportlich und jung, ein keltischer Typ mit heller Haut und leicht rötlichem Haar. Als Uniform trägt sie eine typische Polizei-Multifunktionsweste. Sophie gilt als introvertiert und ernst, ist mutig und intelligent und frägt sich, ob ihre Eltern noch leben. Sie ist bei ihrem Aussehen und ihrem Arbeitsplatz sehr reinlich. Ihre Stimme ist sanft und ruhig, klar und nicht sehr hoch. Sie ist in der Stadt NOAH ohne Eltern aufgewachsen. Aufgrund besonderer Begabungen übernahm der Staat ihre Erziehung. Sie ist neu in ihrem Job und noch etwas unsicher. Sie wird daher von ihren Kollegen nicht ernst genommen. Da ihre Kollegen sehr faul sind, muss sie selbst die Verfolgung der Terroristen übernehmen. Sophie ist in jedem Teilmasterplot involviert, da sie der Hauptcharakter ist.“ [Noah Team, 2007]

Die Vorstellung der Autoren von „NOAH“ lässt trotz dieser ausführlichen Beschreibung noch allerhand Spielraum beim Charakter-Design zu. Wichtig für die Designer ist, genau diese Vorgaben zu berücksichtigen und aufgrund der festgelegten, visuellen Merkmale die Figur im Konzept weiter auszubauen.

2.2.4 Erstes Konzept

Abbildung 10: Sophie Faber Painting, Quelle: Team NOAH

Das erste Konzept von Sophie, ist ein digitales Painting, eine mit einem Grafiktablett erstellte Zeichnung von Studenten der HdM. Das Bild zeigt allgemein die Stimmung der Umgebung von „NOAH“, den Typ und die Eigenschaften der jungen Polizistin, und legt die wichtigsten Rahmenbedingungen für ihre äußeres Erscheinungsbild fest. Dazu gehören die Farbnuancen, der grobe Schnitt der Kleidung und die Konturen ihres Charakters.

Sophies Gesicht ist unkenntlich, verschwommen gezeichnet und lässt somit Spielraum für die Vorstellungen der 3D-Künstler. Sophies Frisur ist eine Hochsteckfrisur mit dunklen Haaren, die allerdings zuvor jedoch von den Autoren zu Beginn als „leicht rötlich“, [Noah Team, 2007] beschrieben worden waren. Das Bild war Vorlage bei Sophies erstem Modell, welches etwa zwei Jahre vor dieser Ausarbeitung entstanden ist.

2.2.5 Das Vorgängermodell

Abbildung 11: Sophies erstes Modell

Sophies erstes Modell beruhte auf dem Konzept aus Abbildung 10 und entstand gegen Ende 2006. Ihre Haare sind auffällig gelborange und besitzen keine Transparenz. Aufgrund von Problemen beim Export besaß ihr Mesh an vielen Stellen eine falsche Ausrichtung der Normalen. Das Halfter, eine Handytasche und ihr rechter Handschuh sind untexturiert geblieben.

2.2.6 Zweites Konzept

Abbildung 12: Sophie Faber Konzeptzeichnung, Quelle: Solveigh Jäger

Sophies zweiter Zeichnungsentwurf, der nach der Modellierung des ersten Modells entstanden ist, besteht aus einer Bleistiftzeichnung, welche eingescannt und nachträglich digital koloriert wurde. Das Bild entstand im Sommer 2008 und besitzt andere Proportionen und Farben, als das Vorgängermotiv. Es zeigt Sophie in einer anderen, etwas stilisierten Form – eine dynamische, junge Frau mit einer Pistole in der linken Hand, deren Äußeres aber weniger auf die introvertierte Sophie aus der Beschreibung passt. Wichtigster Unterschied gegenüber dem ersten Entwurf ist die äußerst detaillierte Zeichnung ihres Gesichts, und dass ihre Haare zu einem Pferdeschwanz gebunden sind.

Da im ersten Konzept kein Gesicht zu erkennen war und die Haare darin nicht der Beschreibung entsprachen, entschieden wir uns Kopf und Haare das zweiten Konzepts weiter zu verwenden und es mit dem restlichen Körper der ersten Zeichnung zu einer dritten zu kombinieren.

2.2.7 Drittes Konzept

Abbildung 13: Mode, Farben und Ausrüstung Quelle: Solveigh Jäger

Im dritten Konzept, einer ebenfalls nachträglich kolorierten Bleistiftzeichnung, werden zusätzliche Elemente von Sophies Aussehen bestimmt. Zu sehen sind einige Farbwerte als Richtlinie für die spätere Farbgebung bei der Texturierung. Elemente ihrer Kleidung und Ausrüstung, sowie die detaillierte Zeichnung und Bemalung ihrer Kleidung. Abgebildet ist noch ein PDA, welches in NOAH häufig zum Einsatz kommt.

Dieser Entwurf wurde quasi zum Leitfaden für Sophies Entstehung. Viele Merkmale ihres Aussehens wurden detailliert und konsequent umgesetzt, was bei der Modellierung zwar weniger Freiheiten, aber auch weniger Arbeit bedeutet.

An dieser Stelle entsteht daher oft ein Graben zwischen 2D- und 3D-Künstlern, wenn sie jeweils ihre eigenen Vorstellungen umgesetzt haben wollen.

2.3 Modellierung

2.3.1 Allgemeines

Unter der Modellierung verstehen wir die Konstruktion eines dreidimensionalen Objekts mithilfe einer entsprechenden Software. Prinzipiell können alle Oberflächen sichtbarer, fester und flüssiger Objekte mithilfe von Polygonen dargestellt werden. Polygone bestehen aus Dreiecken, Dreiecke aus drei Kanten und diese bestehen immer aus zwei Scheitelpunkten.

2.3.2 Vorbereitungen

Wie erwähnt erfolgte die Modellierung des Charakters mit 3ds Max und gewöhnlicher Polygon-Modellierung. Es gab keine Verwendung von NURBS oder Spline-Cages[6]. Dabei sollte der Charakater mit maximal 10.000 Dreiecken (Dreiecke, Triangles bzw. Tris) modelliert sein und natürlich entsprechend der Vorlagen auszusehen haben. Die Vorbereitungen für die Modellierung beinhalteten:

1. Die Sammlung von Photos von Menschen, die Ähnlichkeit mit Sophie aufwiesen,
2. Kontrolle der Maße und Skalierung ihres Körpers,
3. Analyse des ersten Modells von Sophie,
4. Neumodellierung.

Die Voraussetzungen zur Modellierung eines glaubwürdigen Charakters setzen unter anderem zunächst einige Anatomische Grundkenntnisse über den menschlichen Körper aus. Die Recherche konzentrierte sich zunächst auf den Verlauf von Muskeln unter der Haut, auf Gesichtscharakteristiken und auf Proportionen der Protagonistin.[7]

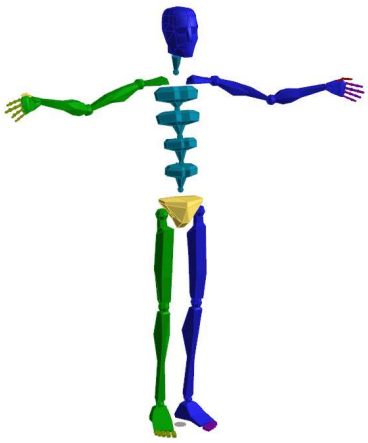


Abbildung 14: Grundpose des Bipeds für die Modellierung

Es gilt bei der Modellierung nicht nur zu beachten, dass Gesichtskonturen und Muskelverläufe dem der Kanten des Drahtgittermodells angepasst werden, sondern dass auch, Falten und Hautverzerrungen für die spätere Animation berücksichtigt werden. Der Charakter sollte in einer symmetrischen Pose modelliert sein, in der er einfacher geriggt[8] und für zukünftige Bewegungen als Ausgangsstellung verwendet werden kann.

Zu den Vorbereitungen innerhalb von 3ds Max zählen vor allem die Anpassungen der Einheiten und der Skalierungen. Da die Größe der Sophie nicht weiter angegeben ist, haben wir diese einfach auf 1,74m festgelegt. Das Standard-Biped in 3ds Max (siehe Abbildung 14) wurde zur Vorlage für die Pose und groben Proportionen der Sophie.

---

[6] Die Modellierung mit Splines bzw. Spline-Cages findet in dieser Ausarbeitung keine Erwähnung.

[7] Wichtige Referenz und Nachschlagewerk bei der Modellierung des menschlichen Körpers war das Zeichenbuch: „Hogarth's Zeichenschule". [Hogarth, 2001]

[8] „geriggt": Gewichtung der Scheitelpunkte an das IK-System des Charakters (siehe Kapitel 2.7.3)

2.3.3 Durchführung

Ausgehend vom Gesicht wurde zunächst ein unbekleideter Körper von Sophie modelliert, um korrekte Proportionen ihrer Anatomie zu erhalten. Ziel war es dabei einen Körper zu schaffen, der mit einer geringen Anzahl von Polygonen („Low Poly"), sowie auch als Mesh mit hoher Subdivision realistisch wirkt. Der Grund hierfür ist, dass das Mesh im späteren Verlauf als geglättete Variante im Sculpting-Tool benötigt wird, um zusätzliche, feine Strukturen innerhalb der Normal Map zu erzeugen („High Poly"). Die Erhöhung der Subdivisions, d.h. die Glättung und Unterteilung wird in 3ds Max mithilfe des Turbosmooth-Modifikators erzielt, der gegenüber traditioneller Mesh-Glättung auch Glättungsgruppen berücksichtigt und voneinander trennt. Beispielsweise bilden die Lippen im Gesicht der Sophie eine eigene Glättungsgruppe. Der Turbosmooth-Modifikator unterscheidet diese voneinander und lässt eine scharfe Kante zwischen Gesicht und Lippen.

Quadmodellierung

Zwar entstehen bei der späteren Glättung des Meshes (unabhängig von der Form des Polygons) stets viereckige Flächen, es empfiehlt sich aber bei der Modellierung darauf zu achten, diese auch schon im Low Poly-Mesh zu verwenden (2 Dreiecke pro Polygon). Vierecke werden bei der Unterteilung durch Subdivisions gleichmäßig und am Besten unterteilt (siehe Abbildung 15, oben). Findet eine Unterteilung oder Tesselierung mit mehr oder weniger als vier Scheitelpunkten pro Polygon statt, entstehen (für uns) unbrauchbare Strukturen und Unregelmäßigkeiten bei der Glättung an den Ecken (siehe Abbildung 15, unten rechts). Ein guter Vorsatz bei der Erstellung eines Low- und eines High Poly-Charakter ist daher die weitestgehend einheitliche Modellierung mit Vierecken[9] (sogenannte Quads).

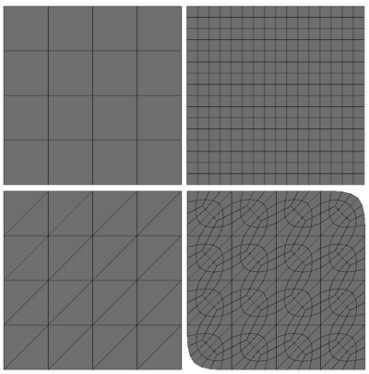


Abbildung 15: Subdivisions mit Vierecke, und Dreiecken

Spiegelung

Bei der Spiegelung der Hälften hilft der Symmetrie-Modifikator in 3ds Max, so dass nur eine Körper bzw. Gesichtshälfte modelliert werden muss. Im Gegensatz zu getrennten, zwei gespiegelten Instanzen der beiden Mesh-Hälften, verbindet der Symmetrie-Modifikator diese zusätzlich zu einem zusammenhängenden Mesh, welches den weiteren Modellierungsprozess sehr vereinfacht. Bei der natürlichen Modellierung muss jedoch berücksichtigt werden, dass zwei Gesicht- und Körperhälften nicht vollkommen symmetrisch sind. Um Unregelmäßigkeiten zwischen der linken und rechten Seite zu erzeugen, werden nach der Symmetrierung nachträglich Geometrie und Texturen verändert.

Modellierungsschritte

Wir entwickelten das Mesh der Sophie in folgenden Modellierungsschritten:

1. Modellierung von Kopf & Körper
2. Modellierung und Anpassung der Kleidung (Polizeiuniform)
3. Löschen der Geometrie unter der Kleidung
4. Zusätzliche Anpassungen (Haare, Falten, Kleidung)
5. Reduzierung der Polygone & Überprüfung der Geometrie

Die Trennung von Kopf und Körper ermöglichte es uns Sophies Aussehen zu verändern, ohne die aufwendigen Regionen am Kopf erneut modellieren zu müssen. Die Kleidung bzw. das Äußere von Sophie können so nachträglich

[9] Anmerkung: Die Polygone werden beim Export in das OGRE-Dateiformat wieder vollständig in Dreiecke unterteilt, da OGRE keine Polygone unterstützt.

verändert werden. Weiterer Vorteil dieser Methode war die Verbesserung bei der Ausnutzung des Texturspeichers. Zeigt die Ansicht im Spiel Sophies Gesicht aus der Nähe (im sogenannten CloseUp), so kann die Textur der Büste eine höhere Auflösung besitzen, als die des restlichen Körpers, wodurch Sophies Gesicht sehr viel detaillierter wirkt.

2.3.4 Kopf & Gesicht

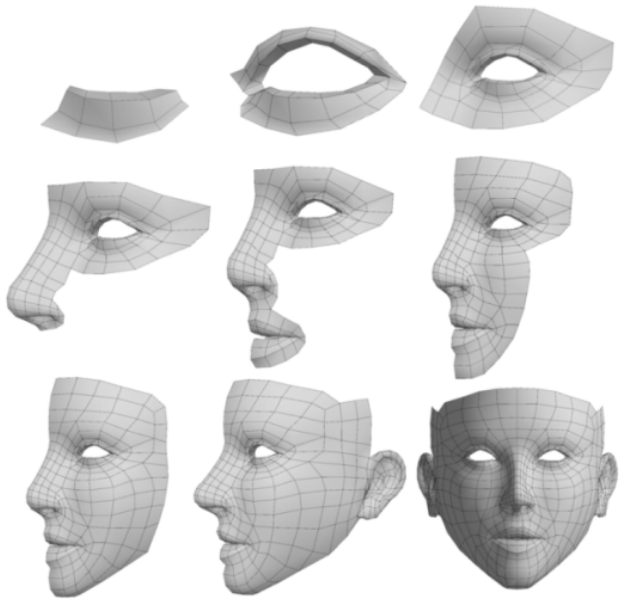


Abbildung 16: Neun Modellierungsschritte von Sophies Gesicht

Die Modellierung von Kopf und Gesicht, gehört vermutlich zu den wichtigsten Aufgaben bei der Entwicklung des Charakters. Sophies erste Geometrie entstand bei den Augen, auf die der Betrachter erfahrungsgemäß zuerst blickt. Eine ringförmige Muskelstruktur umgibt die Lider, die im Kantenverlauf um die Augen herum zu erkennen sind. Wie in Abbildung 16 zu sehen, entwickelte sich daraus die Nase und der entspannte Mund, ebenfalls mit seiner ringförmigen Muskelstruktur. Ohren wurden separat gemodelt und nachträglich an das Gesicht angefügt.

Während in Computeranimationen unabhängige oder über eine Hierarchie miteinander verbundene Objekte und auch Gruppen animiert werden können, muss bei 3D-Spielen darauf geachtet werden, dass alle Teile des Meshes einem Bone des IK-Systems zugeordnet. D.h. Augen können nicht einfach mit dem Schädel verbunden werden, damit sie diesem einfach folgen (was programmiertechnisch machbar, aber sehr aufwendig wäre). Sie sollten mit dem IK-System des Charakters zusammenhängen. Ein Auge ist dann praktisch ein „Knochen", der mit dem Schädel verbunden ist und nur über seine Rotation (in der Mitte des Auges) animiert wird. Augäpfel, aber auch Zähne und Haare wurden also nachträglich modelliert, eingefügt und dann mit dem Gesichtsobjekt vereinigt.

Wie schon erwähnt, war es unser Ziel eine „Low Poly"-Geometrie zu machen, die auch nach ihrer Glättung mithilfe von Subdivisions „gut" als „High Poly"-Variante aussieht. Zwar gibt es zwischen den zwei Modellierungsarten große Unterschiede (besonders in Bereichen mit sehr viel Kurven und Krümmungen), dennoch fanden wir durch die geschickte Aufteilung von Kanten schnell zu einem akzeptablen Mittelmaß. Beide Varianten ähneln sich daher sehr (siehe Abbildung 17).

Schwierigkeiten bereitete eine gleichmäßige Verteilung der Polygone im Gesicht, sowie im Mundinnenraum. Da die Zähne, Zunge und Rachen nur beim Sprechen sichtbar sind, aber dennoch einiges an Geometrie benötigen, mussten hier entsprechend Abstriche bei der Ausmodellierung gemacht und entsprechende Lösungen bei der späteren Texturierung gesucht werden.

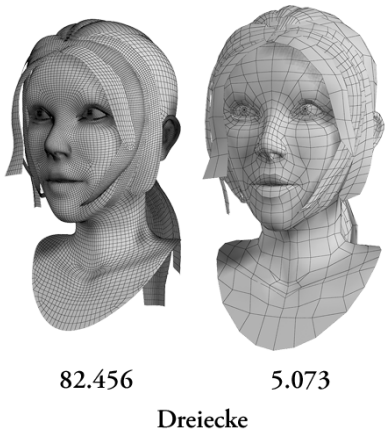


Abbildung 17: Sophies Kopf als High (links) und Low-Poly Variante (ohne Polygon Reduzierung)

2.3.4.1 Haare & Wimpern

Sophies Haar und ihre Wimpern sind Geometrie, die mit einer transparenten Textur versehen wird[10]. Aufgrund des Vorgängermodells ihres Charakters war es zunächst geplant, Sophie eine unveränderliche Hochsteckfrisur zu geben, die sich beim Gehen nicht bewegt. Um ihr allerdings mehr Glaubwürdigkeit zu verleihen, welche die Möglichkeiten bestehender Animationstechniken weitestgehend ausnutzt, beschlossen wir uns an das zweite und dritte Konzept von Sophie zu halten (siehe Seite 33, Abbildung 12 und Abbildung 13), die lange Haare zu einem Pferdeschwanz zu binden und diese mithilfe des IK-Systems zu animieren.

2.3.5 Körper

Sophies restlicher Körper richtet sich größtenteils an die Vorlage der Konzeptzeichnungen, sowie an Fotos, die wir zur Überprüfung anatomischer Details hinzugezogen haben. Wie beim Gesicht, wurde bei der Modellierung des Körpers auf die Polygonstrukturen bei Muskeln und Gelenken geachtet, sodass diese bei der späteren Biegung der Haut gleichmäßig gedehnt und zusammengestaucht werden. Zusätzlichen Einfluss auf die Modellierung hatte natürlich auch das menschliche Skelett[11], welches die Proportionen und Gelenkstellen des Körpers vorgibt, sowie an vielen Stellen auch die Hautstruktur beeinflusst (z.B. an Rippen, Schulter und Gelenken).

Abbildung 18: Vergleich mit anatomischer Vorlage

2.3.6 Kleidung

Durch die vollständige Modellierung des Körpers war es uns nun möglich eine passende Kleidung für Sophie zu suchen. Die Polizeiuniform besteht aus einem Shirt, einem Lederoberteil und einer entsprechenden langen Hose. An einem Gürtel befinden sich drei Taschen für das Equipment (PDA), an ihrem rechten Oberschenkel befindet sich ein Halfter für eine Waffe. An ihren Händen sind Fingerhandschuhe mit offenen Fingerenden, an ihren Füßen trägt sie Polizeistiefel und auf den Trägern ihres Oberteils befinden sich ebenfalls zwei kleine Täschchen.

Zur Modellierung der Kleidung haben wir einige Tricks angewendet. An vielen Bereichen (z.B. bei den Handschuhen), haben wir die Geometrie nicht neu modelliert, sondern die bestehende Geometrie geklont und um wenige Millimeter mithilfe des Push-Modifikators „aufgeblasen“. Zur Modellierung der Knickfalten, besonders im Hüftbereich und um die Füße. nutzten wir die Cloth-Simulation von 3ds Max (Cloth-Modifikator), um schnell und einfach Falzen zu erzeugen. An Stellen der Kleidung, die dabei zu sehr „durcheinander“ geraten waren, nutzten wir das „Relax“-Tool, um betroffene Bereiche wieder zu glätten.

Aufhebung der Symmetrie

Ein nur schwer umkehrbarer und daher wichtiger Schritt bei der Modellierung des Charakters ist die Aufhebung der Symmetrie von Körper und Kleidung. In 3ds Max spricht man vom „Collapsen“ des Symmetrie-Modifikator, d.h. die Symmetrie wird fester Bestandteil der Geometrie. Dies ist notwendig, um auf beiden Körperhälften Unregelmäßigkeiten und Falten im Anzug zu erzeugen, Sophies Halfter mit dem Mesh der Kleidung zu verbinden und verschiedene Strähnen ihres Haars auf beiden Seiten unterschiedlich heraus zu modellieren.

---

[10] Siehe Kapitel 2.6: Texturierung auf Seite 38.

[11] Während unserer Recherche fanden wir das 3D-Modell aus einer Kernspintomographie von einem erwachsenen Mann. Dies half uns insbesondere bei der späteren Positionierung von Schulter- und Kniegelenken, war jedoch für die Anpassung der Haut und der sichtbaren Oberfläche unbrauchbar.

2.3.7 Polygon Reduzierung

In Spielen muss beachtet werden, dass nur begrenzt Dreiecke zur Verfügung stehen, weil Grafikkarten diese nur bis zu einer bestimmen Anzahl flüssig darstellen können. Bei unserer Spielfigur, mussten wir daher darauf schauen, wie viele Dreiecke die Sophie hat und wie sie verteilt sind. Wir sind jedoch zunächst der Philosophie gefolgt, der Anzahl der Dreiecke bei der anfänglichen Modellierung keine Beachtung zu schenken. Das heißt, es wurde zunächst nicht auf die Anzahl der Polygone geschaut, sondern nachträglich die Anzahl „überflüssiger" Geometrie auf maximal 10.000 Dreiecke reduziert. Dies hatte den Vorteil unseren Ideen bezüglich der Modellierung freien Lauf zu lassen, und mit dem bloßen Auge entscheiden zu können, an welcher Stelle das Mesh seine Polygone behalten darf und wo nicht. Meistens behielt es seine Geometrie in der Umgebung von kurvigen Stellen und starken Krümmungen, und verlor sie an eben Stellen, wie Bauch und Rücken.

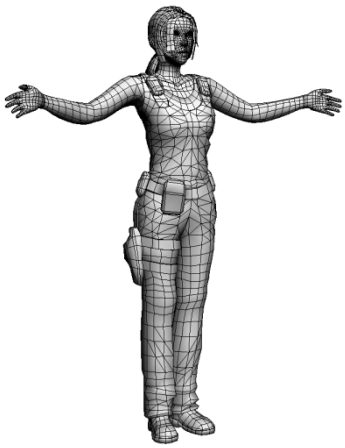


Abbildung 19: Modell von Sophie nach der Reduktion von Geometrie

Da Polygone des Körpers unter der Kleidung nicht sichtbar sind, wurden diese natürlich entfernt. Sophies Anzug mit Ausnahme der Arme, Finger und Brust verdeckt sämtliche Haut, somit konnte ein sehr großer Teil der Polygone des Körpers gelöscht werden. Nach der Entfernung der Polygone unter der Kleidung, vereinigten wir Körper und Kleidung zu einem Objekt. Sophies Kopf blieb weiterhin separat erhalten, offene Polygone, mit Ausnahme der Schnittlinie zwischen Kopf und Brust, wurden alle offenen Flächen mithilfe des „Weld"-Tools verschmolzen.

2.3.8 Konsistenz der Meshes

Für ein sauberes Mapping, ein problemloses Rigging und für den späteren Export nach OGRE, sollte das Mesh konsistent und fehlerfrei sein. Dies bedeutet, dass alle Flächen vollständig, bei zusammenhängenden Flächen weitestgehend geschlossen und keine isolierten Scheitelpunkte mehr vorhanden sein sollten (sprich: Vertices, die zu keinem Polygon gehören, werden gelöscht). Darüberhinaus sollte das Objekt keine veränderte Skalierung besitzen, da diese beim Rigging in 3ds Max zahlreiche Probleme verursacht. Die Skalierung beider von Kopf- und Körper bleibt also immer bei 100%.

2.3.9 Mesh-Daten

Geometrie von Kopf und Körper im Vergleich zum Vorgängermodell (vor 2 Jahren) haben zum Vergleich:

	Objekt	Polygone	Dreiecke	Edges	Vertices
1	Sophie_Head	1.987	3.834	4.263	2.299
2	Sophie_Body	3.280	6.114	6.356	3.071
	Neues Modell:	5.267	9.948	10.619	5.370
	Altes Modell:	3.846	6.791	7.267	3.425

Tabelle 2: Anzahl der Polygone von Sophies neuem Modell

2.4 Texture Mapping

2.4.1 Allgemeines

Um die Flächen von 3D-Objekten mit Texturen auszustatten, benötigt man Informationen darüber, an welcher Stelle die Geometrie auf der Textur zu liegen hat. Dies geschieht über die Texturkoordinaten. Diese legen fest, wie ein (oder mehrere) Texturen auf den Polygonen eines Objekts abgebildet werden. Diese Koordinaten werden auch als UVW-Koordinaten bezeichnet und besitzen (wie die Scheitelpunkte im Raum) ein eigenes Koordinatensystem, worin die Positionen aller Punkte eines Objekts gespeichert sind. Da in Computerspielen nur die Höhe und die Breite einer Fläche benötigt werden, spricht man hier nur von UV-Koordinaten. Der sichtbare Bereich dieser Koordinaten liegt in einem quadratischen Bereich zwischen den UV-Koordinaten [0,0] und [1,1]. Jede Textur wird unabhängig von ihrer Auflösung (und dessen Format) innerhalb dieser Koordinaten abgebildet. Ist eine Position größer oder kleiner als dieser Bereich, wiederholt sich eine Textur.

Zu der vermutlich unbeliebtesten Aufgabe bei der Erschaffung texturierter Objekte gehört schließlich das aufwendige Vergeben solcher Texturkoordinaten. Insbesondere bei komplexen Charakteren, bei denen dieses Verfahren ordentlich und damit langwierig vonstatten gehen sollte, bedeutet dies viel Zeit und Disziplin. Dies ist notwendig, um Verzerrungen, Überlappungen und Wiederholungen einer Textur zu vermeiden. Und obwohl es zahlreiche automatische Projektions-Techniken gibt, sollte man (besonders bei organischer Modellierung) weiterhin per Hand Texturkoordinaten vergeben, um das beste Ergebnis zu erzielen. Das ganze Verfahren nennt man „Texture Mapping“, für den Prozess zur manuellen Vergabe von Texturkoordinaten verwenden wir gerne den englischen Begriff „Unwrapping“ (zu Deutsch: „auspacken“).

2.4.2 Techniken

Zu den automatischen Projektionstechniken des Texture Mappings gehören Box Mapping, das hier nicht zum Einsatz kam, sphärisches, zylindrisches und planares Mapping. Darüberhinaus gibt es das Flatten Mapping, worin jede Fläche eines Objekts einzeln (nacheinander nach Größe sortiert) auf die Texturkoordinaten projiziert wird. Eine Neuerung in 3ds Max, die beim Unwrapping der Sophie Verwendung fand, ist das Pelt Mapping. Wir stellen hier die verwendeten Techniken vor und an welchen Stellen des Körpers diese zum Einsatz kamen.

2.4.2.1 Zylindrisches Mapping

Für die Arme, Beine und Finger von Sophie gebrauchten wir das klassische, zylindrische Mapping, mithilfe man einen langen, rundlichen Körper über eine Röhre aufrollen und auf die gesamte Texturfläche projizieren kann. Die Achse des Zylinders sollte entlang des entsprechenden Körperteils ausgerichtet sein. Mithilfe des „Best Align“-Werkzeugs erkennt 3ds Max (meistens) automatisch die optimale Ausrichtung des Zylinders an die entsprechenden Flächen. Etwaige Verzerrungen durch Knöchel und Muskeln wurden mithilfe des „Relax“-Tools nachträglich im Unwrap-Editor von 3ds Max geglättet. Das „Relax“-Tool sorgt für eine Normalisierung der Abstände zwischen den Scheitelpunkten. Durch das Mapping entstehen einzelne, zusammenhängende Netzflächen, die dann im Editor skaliert und positioniert werden können.

2.4.2.2 Planares Mapping

Für einfache Flächen (z.B. an Bauch und Rücken) benutzten wir teileweise die Textur-Mapping-Ebene des einfachen, planaren Mappings, welches die Positionen der Scheitelpunkte aus einer Projektionsebene gewinnt, die sich nach Wunsch verschieben, rotieren und skalieren lässt. Bei dieser Projektion sollte allerdings auf die Rückseite eines Objekts geachtet werden, die sich bei einer planaren Projektion evtl. mit der Vorderseite überlappt. Sollte eine planare Projektion womöglich spiegelverkehrt erfolgt sein, lässt sich dies mit dem „Mirror"-Tool des Unwrap-Editors schnell korrigieren.

2.4.2.3 Sphärisches Mapping

Die einzige Verwendung sphärischen Mappings fand bei den Augen statt. Ziel war es dabei die Projektionsachse genau in die gewünschte Blickrichtung zu drehen, sodass die beiden Pupillen (die nach der Texturierung aufgemalt wurden) genau in die richtige Richtung blicken. Da Augen fast nie genau parallel blicken, sondern stets etwas in die Mitte schielen (selbst wenn sie in die Entfernung blicken), musste die Projektionsachse beim sphärischen Mapping etwas gedreht werden.

2.4.2.4 Pelt Mapping

Das neue Pelt Mapping in 3ds Max ist das hilfreichste Unwrapping-Tool bei organischen Körpern. Mit dieser Funktion lassen sich unebene Bereiche (wie Gesicht, Kleidung und Schuhe) separat selektieren, aufrollen und glätten. Vergleichbar mit dem eher unansehnlichen Häuten und Aufspannen eines Tierpelzes, lassen sich mit dem Pelt Mapping verhältnismäßig schnell, große Bereiche des Körpers komfortabel in die richtige Position bringen und manuell aufspannen.

Das Pelt Mapping benötigt zunächst einen definierten Bereich, in dem es angewendet werden soll. Die gewünschten Flächen werden über zusammenhängende Kanten mit den sogenannten „Seams" voneinander abgetrennt. Vergleichbar sind die Seams mit den Nahtstellen von Kleidungen, die (im weitesten Sinne) ähnlich zusammengesetzt werden und dessen Abgrenzungen beim Unwrapping gerne gewollt sind. Zur Auswahl des Seams gibt es in 3ds Max mehrere Techniken. Zum komfortablen Selektieren verwendeten wir gerne die „Point To Point Seam"-Variante, bei der nur Anfang- und Zielpunkt eines Seams ausgewählt werden mussten.

Das Pelt Mapping funktioniert nun über den sogenannten „Stretcher" (standardmäßig in der Form eines Kreises), dessen äußere Punkte durch die Seams bestimmt wurden. Alle ausgewählten Flächen, die beim Pelt Mapping berücksichtigt werden sollen, liegen beim Texture Mapping innerhalb des Seams. Die einzelnen „Streching Points" können von Hand oder mithilfe des „Sketching"-Tools positioniert werden. Das Sketching-Tool bietet hierbei die Möglichkeit, die markierten Seams-Punkte in einem Rechteck, einem Kreis, einer Linie und eine Freihandform zu positionieren. So lässt sich praktisch der „Rahmen" frei gestalten, in dem die Flächen aufgespannt werden sollen.

Nachdem sich der Benutzer für eine Reihe von Einstellungen entschieden hat, spannt er die Flächen innerhalb des Strechers auf. Dieser Prozess wird mit einer komplexen Berechnung und mit der ausgewählten Anzahl an Iterationen solange simuliert, bis sich die Texturkoordinaten innerhalb des Seams gleichmäßig aufgerollt haben. Eine nachträgliche Glättung mithilfe des „Relax"-Tools ist manchmal hilfreich, um kleine Verzerrungen wieder „auszubügeln".

Der Nachteil des Pelt Mappings ist das (an manchen Stellen) unkontrollierbare Verhalten bei der Simulation. Trotz mehrfacher Versuche gelang es z.B. nicht mit Pelt-Mapping die Hosenbeine optimal aufzurollen.

2.4.2.5 Spline Mapping

Eine sehr junge Mapping Technik, die erst mit 3ds Max 2009 Einzug fand, (und uns sogar spontan veranlasste, kurzzeitig die Trial Version dieser Software zu testen) ist das Spline Mapping. Diese Technik gibt einem Objekt Texturkoordinaten anhand eines (manuell) gezeichneten Splines, also entlang der Hülle einer Kurve. Spline Mapping ist dabei praktisch für die Zuordnung von Texturkoordinaten gekrümmter Objekte mit einem zylindrischen Querschnitt, wie eine Schlange oder ein Tentakel, oder geschwungene flache Oberflächen, wie eine gewundene Straße. Je näher das Spline dabei an die tatsächliche Form des Objektes angepasst wird (teilweise werden Objekte auch aus solchen Splines modelliert), desto genauer sind die Texturkoordinaten. Für die Sophie wäre das Spline Mapping womöglich an Armen und Beinen hilfreich gewesen, wenn die Ergebnisse des zylindrischen Mappings uns nicht zufriedengestellt hätten.

2.4.2.6 Manuelles Unwrapping

Nicht immer bringt eine Mapping Technik auf Anhieb das gewünschte Resultat. Oftmals müssen Verbindungen nachträglich gelöst oder neu geschaffen werden, um die jeweiligen Bereiche besser auf die quadratische Fläche des Editors anzupassen. In diesem Fall muss man zur unbeliebten manuellen Vergabe von Texturkoordinaten übergehen und alle notwendigen Flächen, Kanten und Scheitelpunkte per Hand verschieben. Hierbei ist es notwendig sich zunächst einen Überblick über den „Geometriesalat“ zu verschaffen und Schritt für Schritt die jeweiligen Arbeitsbereiche in logische Einheiten zu isolieren (z.B. die Selektion nach Elementen oder Körperregionen).

Regeln

Für ein sauberes Unwrapping und zur Vermeidung von Überlappungen haben wir eine Liste von Regeln erstellt, die uns bei der Arbeit sehr geholfen haben:

- Die standardmäßig grün markierten Kanten müssen immer am Rand der texturierten Bereiche liegen.
- Kanten dürfen sich nicht überkreuzen. Flächen nicht überlagern.
- An Scheitelpunkten darf zwischen zwei Kanten kein Winkel mit mehr als 180° anliegen. (Dies ist darin begründet, dass Polygone immer in Dreiecke unterteilt werden. Sollte der Winkel eines Scheitelpunktes größer als 180° sein, gibt es eine Überlappung.)
- Ressourcen sind begrenzt. Den gesamten quadratischen Bereich (so gut es geht) ausnutzen.
- Skalierungen und Verhältnisse gemäß der Originalgeometrie berücksichtigen.
- Niemals aufgeben.

Mapping Hilfen

Ein selbsterstelltes, gekacheltes Material mit einer quadratischen Schachbretttextur, war eine sinnvolle, optische Hilfestellung beim Unwrapping. Durch die Aktivierung der Anzeige der Map im Viewport von 3ds Max vereinfacht sich die Arbeit beim Mapping enorm. Die quadratische Anordnung des Musters hilft bei der richtigen Skalierung aller Geometriemuster und (besonders) ihrer Größenverhältnisse untereinander. Die Pfeile auf unserer Textur zeigen immer auf den rechten Bildrand: ein einfacher Indikator bei der Rotation zusammenhängender Texturelemente. Die farbigen Kacheln oben links und unten rechts markieren die Stellen, an welchen sich die Map wiederholen.

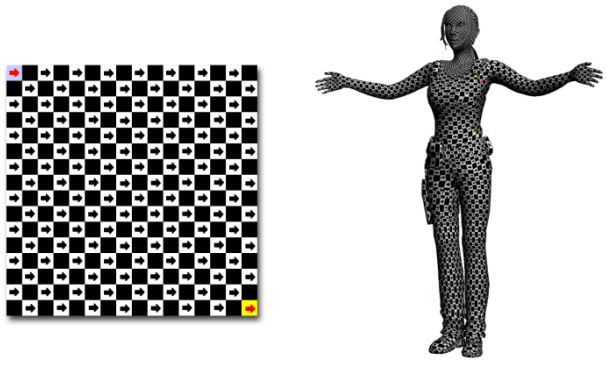


Abbildung 20: Schachbrettmuster für das Unwrapping

Weitere Hilfen waren die Soft Selection-Tools, mit denen sich über einen Einflussbereich zusätzliche Geometrien bearbeiten ließen, die Snap Funktionen zum Einrasten auf dem Hintergrundgitter, die Stitch Funktion zur Sortierung quadratischer, runder oder freihändiger Formen.

Über automatisches Unwrapping

Nur die Kombination der vorgestellten Techniken liefert bei einem Charakter (wie Sophie) und dem derzeitigen Stand der Software, akzeptable Resultate. Dazu gehören:

- Möglichst wenig Seams, sprich Trennlinien zwischen den Texturflächen
- Gleichmäßige Verteilung der Scheitelpunkte mit wenig Verzerrungen
- Keine Überlappungen
- Sortierung

Der Traum von einem einfachen, komfortablen und automatisierten Programm zur Vergabe von Texturkoordinaten, ist bis heute noch nicht realisierbar. Dies hängt offensichtlich mit der Natur des Unwrappings zusammen. Die einwandfreie Projektion dreidimensionaler Objekte auf ein zweidimensionales Feld lässt sich allzu nicht einfach über Algorithmen lösen, sondern verlangt zur Bearbeitung und Entscheidung das menschliche Auge.

Folgende Entscheidungen können (derzeit) von keiner Software automatisch bestimmt werden:

- Auswahl von Flächen…
- …und deren Mapping mit einer geeigneten Technik (siehe oben).
- Optimale Entzerrung der Abstände zwischen den Scheitelpunkten. Damit ist **nicht** die gewöhnliche Normalisierung der Koordinaten beim „Relaxen“, sondern die Abstände und Winkel der **echten** Geometrie gemeint. Sprich, dass das Mapping so gut wie möglich an das reale Mesh approximiert wird.
- Die Bestimmung von Seams
- Auswahl einer geeigneten Skalierung (z.B. die Vergrößerung von Texturflächen, die im Spiel oder in Animationen höhere Auflösungsbereiche benötigen)
- Auswahl einer vorhandenen Textur (die womöglich ebenfalls Einfluss beim Unwrapping ausübt)

2.4.3 Anwendung

2.4.3.1 Texturkoordinaten von Kopf und Körper

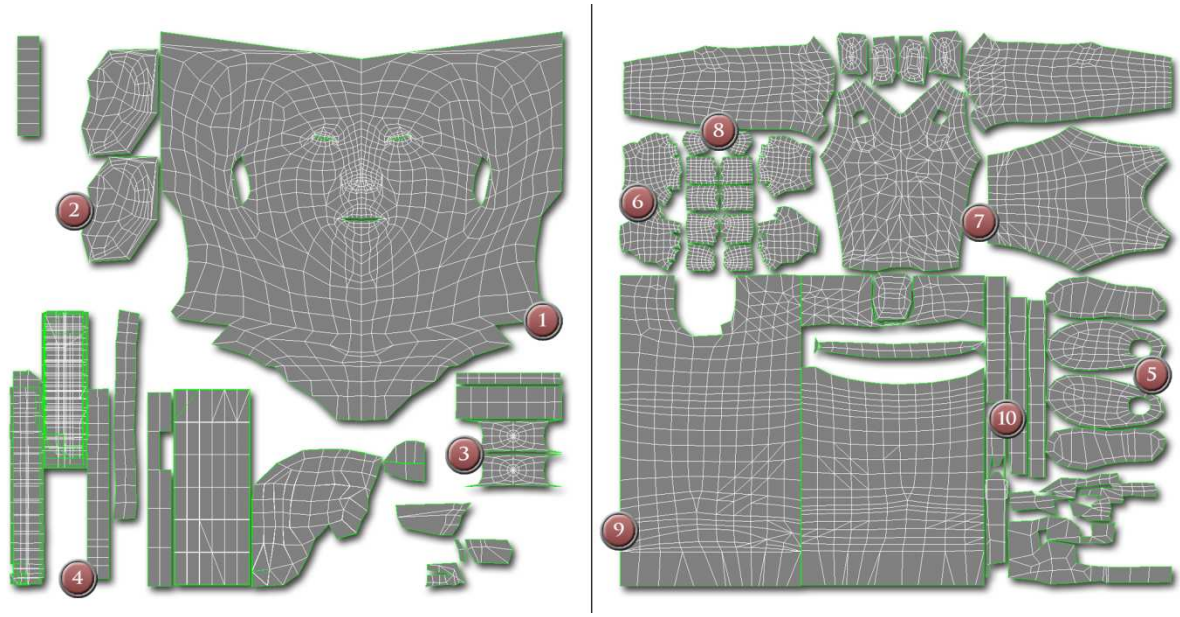


Abbildung 21: Texturkoordinaten von Kopf (links) und Körper

Anwendung der unterschiedlichen Mapping Techniken auf die jeweiligen Körperteile:

	Teil	Mapping Technik	Relax	Manuell	Gespiegelt
1	Gesicht	Pelt & Planares Mapping	ja	ja	ja
2	Ohren	Pelt Mapping	ja	ja	nein
3	Augen	Sphärisches Mapping	nein	nein	nein
4	Haare	Planares Mapping	nein	ja	nein
5	Schuhe	Pelt- (Schuh) & Planares (Sohle) Mapping	ja	nein	ja
6	Handschuhe	Pelt Mapping	ja	nein	ja
7	Oberteil	Pelt Mapping	ja	ja	ja
8	Arme/Finger	Zylindrisches Mapping	ja	ja	ja
9	Hosenbein	Zylindrisches Mapping	ja	ja	nein
10	Gürtel	Zylindrisches Mapping	nein	ja	nein

Tabelle 3: Zusammenfassung der angewandten Mapping Techniken

Die gespeicherten Drahtgitter der Texturkoordinaten von Kopf und Körper waren auch die Grundlage für die spätere Texturierung. Nachträglich Veränderungen der Texturkoordinaten waren nur bei der Positionierung der Haare nötig. Die Auflösung des gerenderten Templates der Texturkoordinaten betrug 2048x2048 Pixel und wurde als PNG verlustfrei gespeichert. Obwohl sie später natürlich nicht sichtbar sind, waren diese Renderings für uns Referenz und Grundlage aller Normal Maps und sämtlicher Texturen von Sophie.

2.5 Sculpting

2.5.1 Allgemeines

Große 3D-Softwarepakete sind die Grundlage zum Erstellen von Modellen und Computergrafiken und für viele Zwecke vollkommen ausreichend, doch die organische (Charakter-)Modellierung ist mit solchen Programmen sehr aufwendig und nur unzureichend lösbar. Viele Details und feine Unebenheiten (Poren auf der Haut, ausgeprägte Muskeln und Reliefs) sind für den Nutzer mit der Maus nur sehr schwer auf die Oberfläche organischer Körper aufzutragen. Solche Details sind jedoch notwendig, um organischen Figuren mehr Glaubwürdigkeit und Leben einzuhauchen.

Aufgrund ihrer Echtzeitfähigkeit leiden Computerspiele zusätzlich darunter, dass sie derartig viele Details nicht über die Geometrie darstellen könnten. Sie sind auf die Hilfe von Texturen und Normal Maps angewiesen, um Details per Textur auf Objekte zu bringen. Bei organischen Objekten sind solche Maps jedoch nicht ohne weiteres machbar, da Unebenheiten exakt platziert und gewichtet werden müssen. 2D-Bildbearbeitungsprogramme besitzen keine Funktion für eine 3D-Reliefzuordnung oder die Möglichkeit das Resultat darzustellen. Zwar gibt es die Möglichkeit durch einige Tools[12] Höheninformationen mithilfe von Schwarzweiß Bump Maps auf ein Objekt aufzutragen und in eine Normal Map zu konvertieren, jedoch nicht um diese schnell zu bearbeiten und gleich dessen dreidimensionales Ergebnis in einem Viewport anzusehen.

Sculpting Tools

1999 tauchte das erste Sculptingtool auf: ZBrush, ein Programm, dass die intuitive Bedienung, nämlich das Bemalen und Modellierung mithilfe von Pinseln, auf die Erstellung von 3D-Objekten überträgt. Sculpting bedeutet, dass der Benutzer nicht mit der Maus und dem „Zupfen“ von Scheitelpunkten beschäftigt ist, sondern mit der Relief- und Farbbemalung mit dem Cursor. Wie vom Erfinder gedacht, verwendeten wir dazu ein vorhandenes Grafiktablett (Wacom Graphire4).

Die Bemalung mithilfe von ZBrush, sowie die Exportierung eines hochaufgelösten Objekts werden in diesem Kapitel nicht angesprochen. Für die Textuirerung von Sophie verwendeten wir Photoshop, ihr Ausgangsmodell blieb auch nach der Editierung mit ZBrush gleich. In diesem Kapitel geht es allein um die Erzeugung einer detaillierten Normal Map und um die Verwendung des ZMappers Plugins (siehe Seite 49, Kapitel 2.5.5).

2.5.2 Software

Unsere Sculpting-Software war ZBrush in der Version 3.1 von Pixologic. Es besitzt die wohl umfangreichste Funktionalität und verwendet eine besondere Hybridtechnik, die 2D mit 3D-Techniken miteinander verbindet. Es ist als 3D-Malprogramm, wie auch als Modellierungs-Tool zu gebrauchen.

[12] Nvidia‘s Normal Map Konverter Filter für Photoshop oder das Freeware Tool „Crazy Bump“.

2.5.3 Funktionsweise

Zusätzlich zu den normalen RGB- und Alpha-Werten werden in ZBrush für jeden Pixel in einer Variablen (Z) die Höhen- bzw. Tiefeninformationen gespeichert (sogenannte „Pixols“). ZBrush ist von Grund auf so konzipiert, dass bestehende (Low Poly-)Modelle schon beim Start in ZBrush importiert werden können. Natürlich können über grobe Grundkörperstrukturen (Box, Kugel, Zylinder, usw.) auch vollständig neue Objekte erstellt werden. Das Programm verwendet die bestehenden Texturkoordinaten des Objekts zur Speicherung aller Z-Werte und stellt diese auf dem Bildschirm dar. Hier können nun mit der Maus oder per Grafiktablett zusätzliche Details hinzugefügt werden. Dieser Vorgang erinnert an Bildhauen oder die plastische Modellierung.

2.5.3.1 Import

Die beiden Ausgangsobjekte für das Skulptieren sind der Kopf und Körper von Sophie. Beide Objekte wurden getrennt und nacheinander bearbeitet. Da der Import in ZBrush nur über das OBJ-Format funktioniert, exportierten wir das Mesh aus 3ds Max in einer separaten Datei. Der Exporter in 3ds Max bietet dabei gleich einige nützliche Funktionen: Da in Maya und ZBrush die Z- und Y-Achsen des Koordinatensystems vertauscht sind, muss das Model rotiert werden. Exportiert werden ganze Polygonflächen (nicht nur Dreiecke), mit ihren Texturkoordinaten, Normalen und Glättungsgruppen.

Beim Start von ZBrush wird gefragt, welches Modell man importieren möchte. Dieses Objekt wird in ZBrush dann als „Tool“ bezeichnet, da es auch die Rolle eines Pinselwerkzeugs übernehmen kann. Die Modellierung findet dann auf dem Werkzeug statt. Das bedeutet, dass man nicht das ganze Dokument, sondern nur das Werkzeug separat speichern muss, um eine Skulptur abzulegen.

2.5.3.2 Glättung mit Subdivisions

Ist das Modell konsistent und fehlerfrei, wird es in ein „Dokument“ in ZBrush geladen. Der erste Schritt ist zunächst das „Low Poly“-Objekt soweit zu glätten, bis auch bei hohem Zoomfaktor auf das Objekt keine Kanten mehr entlang der Silhouette des Objekts sichtbar sind. Die Anzahl der Subdivisions, sprich die Unterteilungen in immer feinere Geometrie, ist vom Auge des Betrachters abhängig. Jedoch lässt sich allgemein sagen, dass etwa 4-6 Iterationen ausreichend sind. Je höher die Wiederholungen der Subdivisions sind, desto genauer, aber auch langsamer wird die Bearbeitung im Viewport von ZBrush. Das Programm bietet allerdings die Möglichkeit nachträglich zwischen einzelnen Stufen der Subdivisions umzuschalten ohne die modellierten Formen auf der detaillierten Geometrie zu verlieren. Da sich bei der Modellierung in ZBrush auch das importierte „Low Poly“-Objekt verändert, besteht die Möglichkeit dieses in ZBrush erneut zu exportieren und in 3ds Max zu laden. Wir haben diese Option nicht verwendet, da wir der Ansicht waren, dass die Original-Geometrie nur in 3ds Max verändert werden sollte.

2.5.3.3 Die Modellierung mit Pinseln

Der Pinsel wird in ZBrush durch eine Verschiebungs- und Verteilungsvariante, durch eine quadratische Bitmap und deren Alphakanal bestimmt. So können neben zahlreichen vorgefertigten Mustern auch eigene Bilddateien verwenden werden. Für unsere Modellierung waren die vorgefertigten Pinselmuster zunächst ausreichend. Sämtliche angewandten Pinselvariationen und Gestaltungsmuster in ZBrush können als Tool zusammengefasst werden, sodass modellierte Formen wiederum als Pinsel verwendet werden können.

Die Form des Pinsels kann dabei in das Objekt eingedrückt (Zsub) oder herausgehoben werden (Zadd). Als zusätzliche Möglichkeit in ZBrush ist es an dieser Stelle auch möglich Farbwerte (RGB) aufzutragen. Diese werden allerdings bei der späteren Erzeugung der Normal Map nicht berücksichtigt.

Häufige Verschiebungsvarianten der angewandten Pinsel (Brushes) in ZBrush waren:

- ***Blob:*** zufällige Höhen- und Tiefenverschiebung
- ***Flatten:*** planares Einebnen (z.B. am Gürtel)
- ***Magnify:*** zum Aufblähen einzelner Regionen
- ***Move:*** direkte Verschiebung von Geometrie
- ***Nudge:*** indirekte Verschiebung: vergleichbar damit in einem Farbtopf herumzurühren
- ***Slash:*** zum Einkratzen auf Oberflächen (z.B. verkratzte Stellen am Leder)
- ***Smooth:*** für die Glättung, um ungewollte Unebenheiten wieder auszubügeln
- ***Standard:*** wichtigster Pinsel für die Tiefen- und Höhenberechnung
- ***Stitch & Tracks:*** Pinsel zur Platzierung regelmäßiger Konturen (z.B. für Säume und Nähte)

Die Strichvarianten (die sogenannten Strokes) der Pinsel bestimmen mit welcher Methode die Brushes auf das Objekt gezeichnet werden. Mit dieser Option wird also nicht gesagt, wie der Pinsel verteilt werden soll, sondern an welche Stelle.

- ***Dots***: Standardpinsel zum Skulptieren mit dem Cursor
- ***DragRect:*** Das rechteckige Aufziehen eines einzelnen Musters mit gedrückter Taste
- ***FreeHand:*** Zusammenhängende Striche
- ***Spray:*** Vergleichbar mit einer Sprühdose, die zufällig Muster auf das Objekt überträgt
- ***DragDot:*** Festes, verschiebbares Muster, das mit gedrückte Maustaste positioniert wird

Die Bitmap des Alpha-Pinsels legt nun das genaue Muster eines Pinsels fest. Wie schon erwähnt, gibt es von ZBrush eine große Auswahl an Standardmustern, die für das Skulptieren von Sophie verwendet wurden. Zur Modellierung unregelmäßiger Hautirritationen und Ledermuster auf der Kleidung, verwendeten wir überwiegend die Alphas 01-04, 08, 09, 22-25, 56, 57, und 59, aufgelistet in folgender Abbildung:

Abbildung 22: Auswahl von Alpha Bitmaps als Pinselmuster in ZBrush 3.1

2.5.3.4 Symmetrie

Zu den wichtigsten Verbesserungen in ZBrush 3 zählt die Einführung der symmetrischen Bemalung, die unabhängig von der Geometrie eines Objekts stattfindet. D.h. auch wenn eine Seite des Objekts nicht mit der gegenüberliegenden Seite identisch ist, werden trotzdem beide Seiten (bei aktivierter Option und entlang der ausgewählten Achse) symmetrisch skulptiert oder bemalt. Deaktiviert man die Symmetrie, werden Pinsel nur auf einer Seite aufgetragen. Dies erweckt den Eindruckt, dass das Gesicht nicht völlig symmetrisch, sondern (wie in Wirklichkeit) auf beiden Gesichtshälften leicht unterschiedlich beschaffen ist. Wie schon in 3ds Max lag die Symmetrieebene der Sophie entlang der X-Achse.

2.5.3.5 Dokument, Material, Licht und Qualität

Jedes Dokument erhält in ZBrush nach seiner Erstellung eine Auflösung, die meist etwa dem sichtbaren Arbeitsbereich des Bildschirms angepasst wird. Ein Objekt bekommt ein Material (z.B. Wachs, Haut oder Metall) und jede Szene ein Beleuchtungssetup (z.B. eine 3-Punkt-Beleuchtung). Um Subdivisions mit der besten Geschwindigkeit darzustellen wird die Qualität der Renderoptionen im Viewport auf „Preview" zurückgesetzt. Nach dem Import und der Anpassung aller Einstellungen wird das gesamte Dokument gespeichert. Somit erscheint dieses nun beim nächsten Laden von ZBrush gleich zur Auswahl auf dem Startschirm.

2.5.4 Durchführung

2.5.4.1 Kopf

Um Probleme bei der Modellierung in ZBrush frühzeitig auszuschließen, musste Sophies Kopf (Abbildung 23, 1) ohne die Geometrie der Haare als OBJ gespeichert werden. Die Modellierung in ZBrush mit mehreren Polygonschichten ist unvorteilhaft, zumal eine Normal Map für die Haare ohne existierende Texturvorlage nicht glaubwürdig scheint. Wir beschlossen dies gegebenenfalls nachzuholen.

In 3ds Max wurden die Haare also von einer Kopie des Kopfes entfernt (Abbildung 23, 2). Da ZBrush nicht die Möglichkeit bietet nach Glättungsgruppen zu unterteilen, sprich an der Grenze zwischen zwei solcher Gruppen keine scharfen Kanten zu belassen, wurde Sophies Kopf mithilfe des Turbosmooth Modifikators schon in 3ds Max mit 2 Glättungsiteration unterteilt, sodass das Modell an den Augen und Lippen harte Kanten aufwies (Abbildung 23, 3). Schließlich wurde das Objekt mit etwa 62.000 Polygonen als OBJ exportiert.

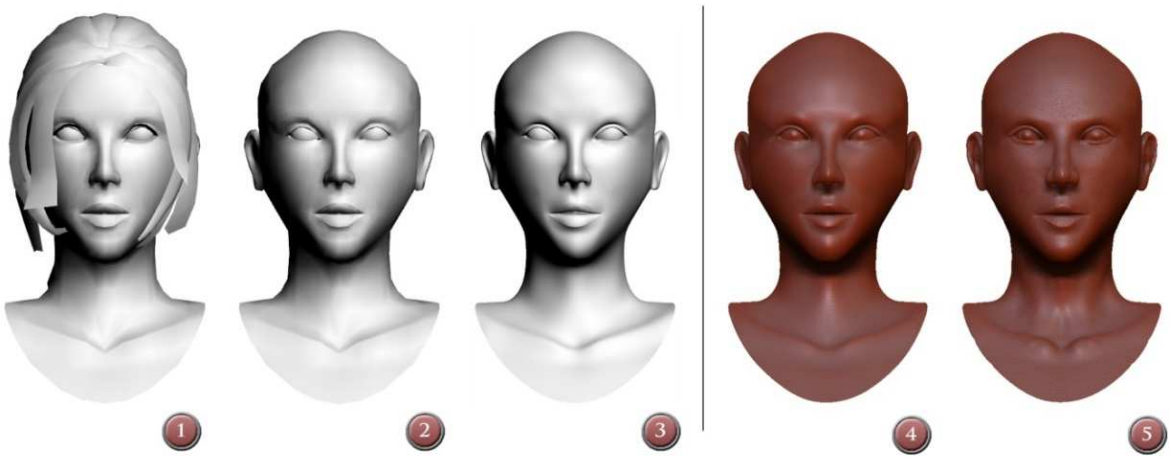


Abbildung 23: Sculpting des Kopfes

Nach dem Import in ZBrush (Abbildung 23, 4), wurde der Kopf mithilfe von Subdivisions noch 3 weitere Male unterteilt, bis die Oberfläche fein genug war, um Poren, Unebenheiten und Hautirritationen glaubwürdig über die Geometrie darzustellen. Nach der Erhöhung der Subdivisions wurde Sophies Kopf mit den zusammengestellten Pinseln in ZBrush schrittweise ausmodelliert bzw. skulptiert. Das Objekt besaß zum Zeitpunkt des Skulptierens etwa 4 Millionen Polygone. Dank der guten Performance lief die Bearbeitung in ZBrush flüssig und ohne Schwierigkeiten bei der Anzeige.

Abbildung 24: Kopfdetails

Für die Ausmodellierung des Gesichts waren einige Photovorlagen, etwas künstlerische Phantasie und Fingerspitzengefühl beim Zeichnen erforderlich. Die Bedienung von ZBrush war dabei sehr intuitiv und erstaunlich einfach. Hervorzuheben sind voralldingen die Hautstruktur mit ihren feinen Poren, die rauhen Lippen, die Linse des Auges, die Augenfalten und die Knorpel der Ohren.

2.5.4.2 Körper

Die vorzeitige Glättung nach Glättungsgruppen in 3ds Max hat sich bei Sophies Kopf als sehr praktische Methode herausgestellt, hartkantige Stellen beizubehalten. Da Sophies Körper entlang der Gürtel, Kleidung und Fingernägel geradlinige Kanten besitzt, haben wir daher dasselbe Verfahren bei dem Körperobjekt wiederholt. Die Geometrie wurde jedoch nicht zusätzlich bearbeitet. Das originale Objekt mit etwa 3000 Polygonen (Abbildung 25, 1) wurde erneut mit beibehaltenen Glättungsgruppen zweimal geglättet (Abbildung 25, 2). Mit ca. 24.000 Polyognen aus 3ds Max als OBJ exportiert, in ZBrush importiert (Abbildung 25, 3) und „gesculpted“ (Abbildung 25, 4). Nach der Glättung mit 4 Subdivisions besaß der Körper umgerechnet 24 Millionen Polygone.

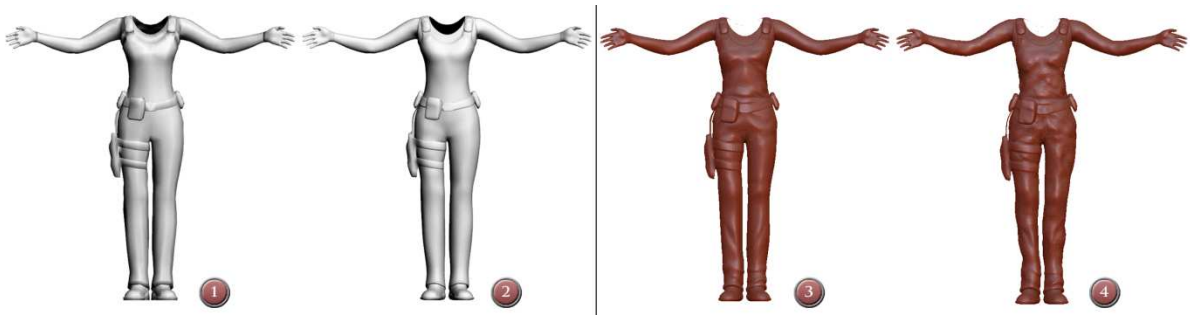


Abbildung 25: Sculpting des Körpers

Bis auf die Arme ist der größte Teil des Körpers mit Kleidung bedeckt, die überwiegend aus Leder besteht. Das Muster für die Formung des Leders entstand aus der Kombination eines Stoff- und Elefantenhautmusters, sowie einer zusätzlichem Pinsel für zerknitterte Strukturen, entlang der Taille, der Brust und im Schritt. Das Shirt unter der Lederbekleidung besitzt ein einfaches Stoffmuster, Gürtel und Taschen eine sehr glatte Lederstruktur. Die Schnürsenkel der Stiefel wurden mit dem Track-Pinsel nur grob heraus modelliert. Die Haut der Arme erhielt von uns dieselben Strukturen und Pinselformen des Kopfes. Schwierigkeiten bereiteten die Modellierung der Gürtelschnallen an Hüfte und Brust, da diese nicht „intuitiv“, organisch modelliert werden konnten, sondern praktisch mit „einem Mausklick“ korrekt zu sitzen hatten. Die oben beschriebene RectDrag- und DragDot-Funktionen halfen bei der genauen Positionierung der Displacements.

Abbildung 26: Körperdetails

2.5.5 Der ZMapper

Zu den wichtigsten Neuerungen von ZBrush 3.1 gehört das neue ZMapper-Plugin. Es ersetzt die Standardfunktion zum Export von Normal Maps, mit einem umfangreichen Werkzeug, das mit einer gewaltigen Anzahl an Einstellungen aufwartet. ZMapper wurde in der Version 3.1 in ZBrush integriert und besitzt in Revision-E zahlreiche Presets, die auf spezielle Anforderungen verschiedenerer 3D-Programme angepasst sind. Ist das Objekt also fertig skulptiert (soweit das zunächst den Anschein hatte, denn Änderungen mussten sehr oft gemacht werden), wird das Displacement auf eine Textur übertragen.

2.5.5.1 Vorbereitungen zur Generierung der Normal Map

Zur Verwendung des ZMappers müssen innerhalb des Programms zunächst einige Anforderungen erfüllt sein:

- Nur ein 3D-Tool (in ZBrush als „PolyMesh3D“ bezeichnet) kann durch den ZMapper analysiert werden ZBrush befindet sich zwar standardmäßig in dem sogenannten „Projection Master Mode“, in dem nur 3D-Objekte gezeichnet werden können, allerdings kann man auch in einen 2D-Modus wechseln, in der nur die aktuelle Ansicht von vorne bemalt wird. Diese Ansicht muss daher beim ZMapper wieder abgeschaltet werden, sollte sie angewendet worden sein.
- Vor der Projektion muss in die Ansicht mit der niedrigsten Subdivision geschaltet werden, damit das Programm die Geometrie darüber liegender Subdivisions berechnen kann.
- Zuletzt muss schließlich eine leere Textur angelegt werden, auf die die Normal Map gespeichert wird. An dieser Stelle mussten wir uns auch für eine Auflösung der Texturen entscheiden und uns auf Höhe und Breite festlegen. Wie beim Rendering des Drahtgitters der Texturkoordinaten (siehe Kapitel 2.4.3.1 auf Seite 43), entschieden wir uns bei Kopf und Körper von Sophie eine Auflösung von 2048x2048 Pixel bei zunächst 16-Bit Farbtiefe zu verwenden. Mehr Informationen zur Texturierung auf Seite 54, Kapitel 2.6.

2.5.5.2 Funktionsweise & Einstellungen

Abbildung 27: Das ZMapper Plugin in ZBrush 3.1

Nach diesen Vorbereitungen wird nun der ZMapper aktiviert. Das Programm wechselt in eine interaktive, rotierende Ansicht, in der die Auswirkungen vieler Einstellungen gleich betrachtet werden können. Der Benutzer muss sich nun mit einer Vielzahl von Optionen auseinandersetzen, um für das jeweilige Objekt das richtige Ergebnis zu erzielen. Der ZMapper erleichtert die Auswahl und die Optimierung mithilfe einiger Presets, von denen wir die für 3ds Max zunächst vorgeschlagene Konfiguration verwendeten (3ds Max, Tangent Space, Best Quality) und einige Werte nachträglich anpassten.

Eine wichtige Schlüsseleinstellung des ZMappers ist die Verwendung des „Tangent Space“, der die Normalen orthogonal an der Tangente jeder Flächennormale (dem Richtungsvektor einer Fläche) projiziert. Diese Einstellung ist die beste Option für animierte Objekte, die sich auch bewegen oder verschieben lassen. Dies ist auch die häufigste Projektionstechnik bei der Erzeugung von Normal Maps, denn die Grundlage für das Koordinatensystem des Tangent Space bilden die Texturkoordinaten, die für jede Fläche spezifiziert sind. Daher wird der Tangent Space oft auch als Texture Space bezeichnet [Hedge, 2007].

Einstellungen zu Normal & Cavity Maps

Jede Normal Map ist eine völlig „gewöhnliche" Bitmap mit RGB-Werten (meist mit 8, manchmal mit sogar 16-Bit). Standardmäßig bestimmt der Rot-Kanal einer Normal Map den X-Wert des Richtungsvektors einer Normalen. Aus der Sicht des UV-Koordinatensystems bedeutet dies, dass er sich dabei nach links oder rechts dreht. Der Grünkanal ist für den Vektor entlang der Y-Achse zuständig, und dreht sich so nach oben oder unten. Der Blaukanal zeigt in die Vertikale, quasi aus dem UV-Koordinatensystem nach oben heraus. Die RGB-Werte der Bitmap entsprechen also dem Richtungsvektor für XYZ, wobei 0% = -1, 50% = 0 und 100% = 1 bedeuten. Da die Normalen stets orthogonal auf eine Fläche zeigen, besitzt eine Normal Map im Tangent Space bei unverändertem Aussehen also die RGB-Werte [50%,50%,100%] mit dem Richtungsvektor [0,0,1]. Dies ist der Grund, warum Normal Maps im Tangent Space ihr charakteristisches Blau haben und Veränderungen auf der Oberfläche durch bunte Farbverläufe zu erkennen sind.

Zusätzlich zum Tangent Space gibt es als Auswahl den „Local-, World- oder Screen Space", die Normalen eines Objekts aus unterschiedlichen Koordinatensystemen heraus projiziert. „Local Space" kann dabei für stationäre oder bewegliche Objekte verwendet werden, die sich im Raum bewegen, aber nicht deformieren. „World Space" ist für völlig statische Objekte, wie z.B. Landschaften brauchbar. „Screen" steht für die Verwendung des Koordinatensystems aus der Sicht der Kamera.

Zusätzliche Einstellungen des ZMappers:

- ***Flip Red (X)***: Vertauscht den Rotkanal der Normal Map, sodass die linke und reichte Seite des Richtungsvektors verdreht sind. Diese Einstellung ist für manche Programme notwendig (nicht für 3ds Max).
- ***Flip Green (Y)***: Vertauscht den Grünkanal der Normal Map, sodass oben und unten gedreht werden.
- ***Swap Red & Green***: Vertauscht rot und grün, sodass das Mapping um 90 Grad rotiert.
- ***Flip Image Vertically***: Invertiert den Blaukanal, sodass die Normalen nach unten zeigen.

Zusätzlich zu den Einstellungen über die Richtungsvektoren der Normal Map, können auch zahlreiche Methoden zu ihrer Berechnung und ihrem Aussehen gemacht werden.

- ***Seam Overpaint***: Da man bei der Anzeige häufig erkennt, dass die UV-Nähte entlang der Ränder (z.B. auf schwarzem Hintergrund) einer Textur verlaufen, werden die Ränder einer „ungewrappten" Fläche um einen variablen Wert erweitert. Der Seam Overpaint nimmt den letzten Pixelwert am Rand einer Fläche und erweitert ihn nach außen, sodass die Linien der UV-Koordinaten nicht erst auf einem anderen, angrenzenden Bereich der Textur liegen. Wir haben diese Einstellung mit einem starken Wert verwendet, da bei Normal Maps sonst an vielen Stellen feine Nähte entlang der Seams sichtbar gewesen wären.
- ***Sharp, Blur***: Mithilfe von Sharp und Blur kann die Textur detaillierter oder verschwommener berechnet werden. Bei Charakteren bietet es sich an die Schärfe zu erhöhen, damit Poren und feine Details auf der Kleidung deutlicher erkennbar sind.
- ***Interpolate, Raytrace***: Als Rendering-Methode wählten wir „Raytracing", da dieses sehr viel präzisere Ergebnisse lieferte, als die grobe Annäherung mit dem „Interpolate"-Verfahren.
- ***Samples***: Die Samples pro Fläche erhöhten wir für die Genauigkeit der Berechnungen.
- ***Smooth, Subdivide***: Zusätzlich Glättung und Unterteilung der Flächen. Sinnvoll, wenn nicht ausreichend Subdivision bei der Modellierung verwendet wurden oder um die Qualität der Normal Map zu erhöhen, was bei Sophie nicht der Fall war.
- ***Inflat Hires Mesh Details***, ***Inflat Bumpmap Details***: Abhängig von ihrer Einstellung, überziehen diese beiden Methoden das Relief und die Winkel, indem sie die Strukturen künstlich aufblasen. Die erste Einstellung steuert die Inflation bevor die Bump berechnet wurde und bläht zunächst nur die Geometrie auf. Die zweite Einstellung vergrößert die Detailstruktur des Reliefs, nachdem die Glättung berechnet wurde. Es bietet sich nicht an beide in extremen Maßen zu verwenden, da oft Artefakte übrig bleiben und fehlerhafte Be-

rechnungen gemacht werden. Die Stärke und Verwendungen dieser Einstellungen bleiben also dem Künstler vorbehalten, wir haben sie jedoch nicht verwendet.

- ***Sharpen Bumpmap Details***: ZBrush kann nach der Erzeugung der Normal Bump Map Elemente mit einem zusätzlichen Filter schärfen. Dieser Effekt erzeugte in unserem Fall jedoch zu sehr pixelige Ergebnisse und wurde daher nicht verwendet.
- ***Sharpen Hires Mesh Details***: Ähnlich zur vorigen Einstellung erhöht sich hier die Schärfe des Objekts, allerdings ohne die Bump. Diese Option wurde von uns nicht verwendet.
- ***Cavity Intensity***: Eine sehr verbreitete Technik, um realistische Texturen zu erzeugen, ist die Berechnung einer Ambient Occlusion, bei der tiefe, eingedrückte Stellen einer Oberfläche dunkler erscheinen, auch wenn das Modell gleichmäßig mit diffusem Licht ausgeleuchtet wird. Schon lange im Voraus entschieden wir nach der Hinzufügung der Normal Map in 3ds Max auch eine Ambient Occlusion zu verwenden. Zusätzlich wählten wir dazu noch das Cavity Shading in ZBrush. Der ZMapper erzeugt dabei eine einfache Variante einer Ambient Occlusion, in der über ein diffuses Himmelslicht Schattierungen erzeugt werden, sondern über die Tiefe der Reliefstrukturen. Eingedrückte Stellen (z.B. Hautporen) werden somit automatisch dunkler. In Kombination mit einer richtigen Ambient Occlusion Rendering, sprich einer Map, die über ein globales Himmelslicht gleichmäßig berechnet wird, erhält der Künstler so die Freiheit Schattierungen mit hoher bzw. geringer Detaildichte miteinander zu kombinieren und deren Stärke nachträglich zu manipulieren, bevor dieser auf die Textur gespeichert werden. Wir haben die Intensität sehr hoch gestellt, um eine starke Cavity Map zu generieren.
- ***Cavity Blur***: Mit diesem Filter wird die Cavity Map nachträglich unschärfer gerendert, um Details zu verwischen. Diese Option wurde von uns nicht verwendet.
- ***Cavity Coverage***: Beeinflusst weitestgehend die Breite der Hohlräume und verstärkt den Effekt des Detailreichtums des Reliefs. Hiermit wirken Falten breiter, Poren größer und tiefer. Der Wert steuert dabei das Verhältnis zwischen Hohlräumen und glatten Stellen und wurde von uns auf ein mittleres Niveau gestellt, sodass die Schattierungen auf der Cavity Map nicht zu übertrieben wirken.

Zusätzlich zur Berechnung von Normal und Cavity Map, gibt es beim ZMapper die Möglichkeit zur Projektion. Hier können 2 Objekte (in der Regel ein fertiges High- und ein Low-Resolution Objekt – womöglich in einem anderen Programm) geladen und für die Generierung von Normal Maps oder zur Übertragung von UV-Koordinaten aufeinander projiziert werden. Da wir das High-Resolution Objekt in ZBrush skulptiert hatten und dieses dieselben UV-Koordinaten wie das Ausgangsobjekt besaß, war keine Projektion mehr notwendig.

Die Voreinstellungen, die in ZBrush für die Erzeugung der Normal und Cavity Maps zur Verfügung standen, konzentrierten sich auf die Tutorials und die Anforderungen der Programme 3ds Max und Maya, sowie auf eine optimierte Darstellung mithilfe von ATI und NVIDIA-Grafikkarten. Leider bot das User Interface von ZBrush keinen Einblick in die exakten Werte unserer Konfiguration. Auch die gespeicherte Datei unseres Presets gab keinen Aufschluss über die interne Funktionsweise des ZMappers.

Nach der Berechnung der Normal und Cavity Map werden die errechneten Bilddaten automatisch auf der erzeugten Textur abgelegt. Die errechnete Map wird jedoch stets vertikal gespiegelt auf den Texturspeicher übertragen (der Grund hierfür blieb uns verborgen). Die „Flip Vertically"-Funktion in ZBrush korrigierte diesen Fehler wieder. In ZBrush wird die aktive Textur interaktiv im Viewport angezeigt und auf das aktive Tool übertragen. Nach der Überprüfung der Maps speicherten wir diese nacheinander als PSD (Photoshop Dateiformat).

2.5.5.3 Normal Map

Der Fortschritt bei der Berechnung (etwa 20 Minuten) und das Aussehen der Normal Map kann im Viewport des ZMappers zur Laufzeit beobachtet werden. Allerdings waren zahlreiche Versuche und Feinjustierungen mancher Einstellungen nötig bis das Resultat allen Anforderungen entsprach.

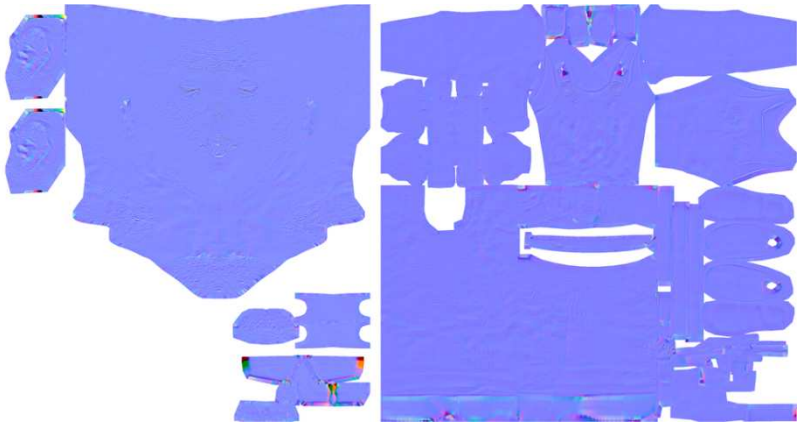


Abbildung 28: Normal Bump Maps von Kopf ohne Haare (links) und Körper.

2.5.5.4 Cavity Map

Der „kleine Bruder“ der Ambient Occlusion, der gemeinsam mit dieser erst bei der Texturierung auf Diffuse Map zum Einsatz kommt, wird (zusammen mit der Normal Map) in ZBrush berechnet. Cavity Maps haben gegenüber gewöhnlichen Ambient Occlusion Maps den Vorteil sehr viel detaillierter und körniger zu wirken, als die gleichmäßigen Schattenberechnungen einer Ambient Occlusion. Für eine glaubhafte Texturierung bietet es sich an die Strukturen der Cavity Map und die Verlaufsschatten der Ambient Occlusion miteinander zu kombinieren.

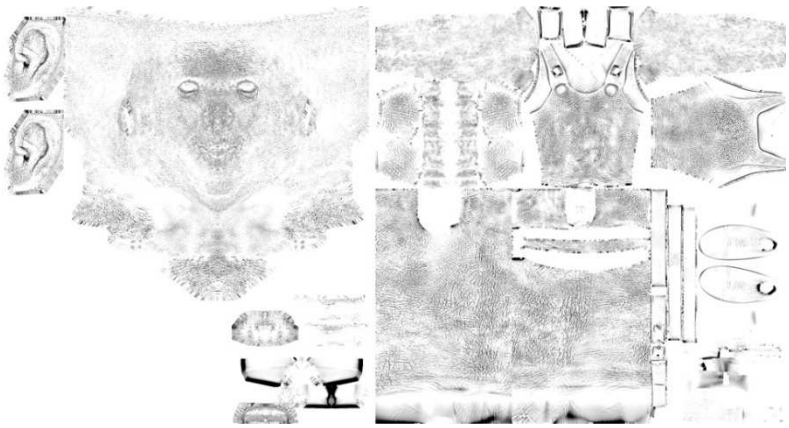


Abbildung 29: Cavity Maps von Kopf (links) und Körper

Grundsätzlich wurde jede von uns erzeugte Map in 3ds Max geladen und gleich auf Fehler überprüft. Zwar waren keine nennenswerten Probleme bei der Generierung der Normal und Cavity Map aufgetaucht, trotzdem schien es uns sinnvoll beide Maps auch über das Rendering des UV-Templates (siehe Seite 43, Abschnitt 2.4.3.1) zu legen und abzugleichen. An dieser Stelle verwendeten wir schließlich Photoshop, auf was wir im nächsten Kapitel näher eingehen.

2.5.5.5 Texture Painting in ZBrush?

Wir möchten an dieser Stelle begründen, warum wir die Painting-Funktionen von ZBrush nicht für die Texturierung in Frage kamen. Zwar bot ZBrush mit seinen Pinseln und Werkzeugen ein sehr intuitives Tool zu Bearbeitung der Geometrie, doch für eine effiziente Bemalung der Oberflächen war es nicht zu gebrauchen. Ohne die Füllwerkzeuge, Masken und unterschiedlichen Filter, wie wir sie aus Photoshop kannten, war es sehr schwer Oberflächen auch mit Strukturen und exakten Linien zu versehen. Die Kombination der gerenderten Maps, insbesondere der erzeugten Cavity Map und der späteren Ambient Occlusion, dessen Transparenz bzw. Stärke nachträglich von uns kontrolliert werden musste, war mit ZBrush nicht möglich. Insbesondere das Fehlen der Transfer Modes (Füllmethoden) von Ebenen, war für uns ebenfalls ein Grund ZBrush für die Texturierung nicht zu verwenden.

2.6 Texturierung

2.6.1 Allgemeines

Texturen sind Bilddaten (Bitmaps), die auf Oberflächen dreidimensionaler Objekte aufgetragen werden. Durch Texturen können Farben und Strukturen auf die Objekte dargestellt werden, ohne dabei zusätzliche Geometrie zu gebrauchen. Sie lassen Oberflächen detaillierter und glaubwürdiger wirken, können dabei einfach ersetzt werden und sind relativ leicht zu bearbeiten. Da in 3D-Anwendungen ausgesprochen häufig Texturen sichtbar sind, sollten diese für ein realistisches Gefühl einer Szene oder Charakters sorgfältig angepasst und bearbeitet werden. Die Texturierung genießt daher bei der Entwicklung von Spielen einen besonders hohen Stellenwert. Mit Ausnahme des Displacement Mappings wird bei der Texturierung keine Geometrie verändert. Wir beleuchten in diesem Kapitel unser Vorgehen bei der Texturierung, begründen noch einmal die Vorbereitungen der obigen Kapitel und erklären die Funktionen der einzelnen Maps.

2.6.1.1 Texturen, Maps, Shader und Materials

Als **Texturen** bezeichnet man im Allgemeinen Bilddaten, die als Oberflächen für 3D-Objekte dienen. Meist werden Photos, bemalte Bilder, kachelbare oder prozedurale (vom Computer errechnete) Strukturen als Texturen verwendet. Als **Maps** bezeichnet man Texturen, die innerhalb des 3D-Programms noch bestimmte Funktionen erfüllen. Darunter zählt nicht nur allein die diffuse Kolorierung eines Objekts, sondern auch z.B. der Glanz, die Glanzfarbe, die Transluzenz (die partielle Lichtdurchlässigkeit), das Relief (Bump), die Opazität (Lichtundurchlässigkeit) oder Eigenleuchtkraft eines Modells. Maps werden meist in einem sogenannten **Material** zusammengefasst, das wiederum einen (Software-)**Shader** besitzt, der entscheidet auf welche Weise, sprich mit welchen Algorithmen, die angewandten Maps gerendert werden sollen. Der Shader interagiert dabei zusammen mit dem umgebenden Licht und ist für die Berechnung aller Farb- und Helligkeitswerte eines Objekts verantwortlich.

2.6.1.2 Auflösung

Da Grafikkarten in ihren Speicher derzeit nur Texturen mit einer Auflösung von maximal 2048x2048 Pixeln laden können, haben wir uns entschlossen diese Auflösung auch bei der Erstellung von Sophies Maps zu verwenden. Die Vorteile hoher Auflösungen liegen auf der Hand: mehr Pixelinformationen auf der Geometrie sorgen für mehr Detailreichtum, weniger „pixlige“ Texturen und die Möglichkeit die Texturen nachträglich herunter zu skalieren, falls nicht genügend Grafikspeicher für alle Texturen zur Verfügung stehen sollte. Nachteil bei vielen hochaufgelösten Maps ist die sinkende Performance der Grafik. Die Wahl viel auf diese Texturauflösung, da Sophie damit selbst bei hohen Bildschirmauflösungen detailliert und scharf wirkt.

2.6.1.3 Beleuchtung

Das Licht spielt bei der Konfiguration des Materials und für die Bearbeitung der Maps eine wichtige Rolle. Um für die Texturierung eine adäquate Beleuchtung zu finden, haben wir uns überlegt zunächst die Einstellungen aus NOAH in 3ds Max nachzustellen. Dazu gehört eine einfache Umgebungsbeleuchtung (quasi die gleichmäßige Beleuchtung des Himmels) und ein direktes Sonnenlicht von schräg oben. Um die Farbwerte nicht zu verfälschen blieben alle Lichter normal weiß.

2.6.2 Texturrendering der Ambient Occlusion

2.6.2.1 Render-to-Texture

Die „Render-to-Texture"-Funktion oder das „Texture Baking" (Texturrendern) ermöglicht das Rendering von Maps des aktuellen Aussehens eines Objekts, anhand einer (Beleuchtungs-)Szenerie. Beim Texture Baking werden alle Licht- und Oberflächeninformationen des 3D-Modells mithilfe dessen Texturkoordinaten als neue „gebackene" Textur gespeichert. Um spätere Änderungen zu vereinfachen, kann dieses Rendering in unterschiedliche Kanäle aufgeteilt werden. So ist es unter anderem möglich, sämtliche Schatten einer Szene vorberechnen zu lassen, die Lichtsituation auf den Texturen aller Objekte zu speichern und schließlich in Echtzeit anzuzeigen. Da nun die Schatten der Lichter direkt auf den Texturen liegen, können diese abgeschaltet werden. Die Geschwindigkeit der Anzeige oder des Renderings beschleunigt sich dabei um ein Vielfaches, bei nahezu gleichem Ergebnis (abhängig von der Qualitätseinstellung und Auflösung des Renderings). In Spielen ist das Texture Baking besonders beliebt, nicht selten werden sogar die Schatten ganzer Levels allein mithilfe des Texture Bakings ausgeleuchtet.

Der Nachteil dieser äußerst beliebten und verbreiteten Technik ist seine „Unanimierbarkeit". Die Texturen veränderlicher, animierter Objekte (wie eben ein Charakter) können zwar gespeichert werden, würden aber bei einer Animation nicht mehr glaubwürdig wirken. Speicherte man z.B. die Schatten des Sonnenlichts (Lighting Map) unserer Beleuchtung auf Sophies Texturen, würde bei ihrer Drehung der Eindruck erweckt werden, dass die Sonne mit ihr gehen und plötzlich von einen anderen Seite scheinen würde. Die Schatten direkter Lichter sind also für animierte Objekte ungeeignet. Anstelle die Schlagschatten des direkten Sonnenlichts auf Sophies Textur zu speichern, haben wir uns entschlossen allein die Umgebungsschatten des gleichmäßigen Himmelslichts (Skylight) als Map zu rendern. Für dieses Verfahren gibt es einen eigenen Kanal, der beim Rendering der Map mit Mental Ray ausgewählt werden kann: die sogenannte **Ambient Occlusion**. Für die direkten Schatten im Spiel ist schließlich die OGRE-Engine verantwortlich. Diese sind also bei einem Charakter kein Teil der Texturierung.

Zur Übersicht haben wir die Kanäle aufgelistet, die in 3ds Max berechnet und als vorgerenderte Map gespeichert werden können:

- ***CompleteMap:*** Sämtliche Kanäle und Maps, sprich das ganze Erscheinungsbild eines ausgewählten Objekts werden auf eine Textur herunter gerechnet.
- ***SpecularMap:*** Nur die Glanzfarbe wird auf eine Map gespeichert.
- ***DiffuseMap:*** Die diffusen Farbwerte eines Objekts werden gerendert.
- ***ShadowsMap:*** Nur die Schatten, die ein Objekt von einem anderen erhält, werden gespeichert.
- ***LightingMap:*** Die beleuchteten Stellen werden hell gerendert, Schatten sind dunkel.
- ***NormalsMap:*** Wie in ZBrush können auch in 3ds Max Normal Map generiert werden. Die Normal Map benötigt jedoch eine Projektionshülle (Geometrie), auf die es angewendet werden kann. (Da es in 3ds Max keine Sculpting-Möglichkeiten gab, haben wir diese Funktion hier nicht benötigt)
- ***BlendMap:*** Vergleichbar mit einer CompleteMap, ohne Schatten.
- ***AlphaMap:*** Rendert nur die Transparenzen (Alphakanäle) eines Objekts.
- ***HeightMap:*** Vergleichbar mit der Cavity Map in ZBrush, benötigt eine Projektionshülle (sich selbst oder zusätzliche Geometrie) zur Berechnung einer normalisierten Höhe.
- ***Ambient Occlusion (nur Mental Ray):*** Umgebungslichtberechnung

Bevor wir in Photoshop texturierten, renderten wir in 3ds Max mit Mental Ray die Ambient Occlusion als separate Map, die oft (und fälschlicherweise) auch als „Dirt Map" bezeichnet wird.

2.6.2.2 Vorbereitungen

Damit sämtliche Details der Normal Map aus ZBrush bei der Berechnung der Ambient Occlusion in 3ds Max berücksichtigt wurden, musste das Material, mit dem Sophies Map erstellt werden sollte, die Normal Map auch natürlich beinhalten. Damit das geschieht, erstellten wir ein neutrales, glanzloses, weißes Material mit einer einzigen „Normal Bump" als Map, in der die Datei aus ZBrush (siehe Seite 52, Abschnitt 2.5.5.3) eingefügt wurde. Dazu wurde schließlich die doppelte Glättung mithilfe des TurboSmooth-Modifikators aktiviert, damit bei der Berechnung der Ambient Occlusion keine „kantigen" Schatten der „Low Poly"-Version von Sophie sichtbar wurden. Da die Presets von Mental Ray auf Geschwindigkeit und nicht auf Qualität gesetzt sind, mussten Verbesserungen einiger Einstellungen erfolgen:

- Deaktivierung aller Lichter, ausgenommen des Himmelslichts (Skylight), welches zur Berechnung des Final Gatherings notwendig ist.
- Erstellung einer Bodenfläche, die verhindert, dass das Himmelslicht Strahlen von unten her emittiert. Die Bodenfläche dient zu Reflexion der Strahlen, die nur von der oberen Hemisphäre des Himmelslichts kommen können. Da Sophie im Spiel überwiegend herumläuft (nicht die Füße hoch hebt), sind diffuse Schatten im Bereich der Füße gewollt.
- Aktivierung des Final Gatherings. Das Final Gathering ist ein Raytracing-(Strahlenverfolgungs)-Algorithmus, der von Mental Ray (einem reinen Raytrace-Renderer) für diffuse Beleuchtungs- und Schatteneffekte genutzt wird. Von der Benutzersicht ausgehend, werden im sogenannten Presampling (Vorberechnung des Final Gather Points) Strahlen in die Szene geschickt. Bei einer Kollision mit der Geometrie werden entlang der Normalen (orthogonal zu den Flächen) die Final Gather Strahlen ausgesendet. Der Renderer sendet mehr dieser Strahlen an detaillierte Stellen, weniger an großflächige Geometrie. Durch die Bestimmung der Farbe und der ungefähren Intensität eines Lichtstrahls erfolgt eine schnellere Berechnung der Ambient Occlusion, in der auch die Wechselwirkung des Lichts zwischen Objekten (das sogenannte „Bouncing" der Global Illumination) berücksichtigt werden kann. Mit der Erhöhung der Strahlen pro Punkt sinkt das Rauschen und steigt die Qualität des Renderings (Preset auf „Very High", Rays per FG Point": 1000).[13]
- Wir haben auf die Aktivierung der Global Illumination verzichtet, da es in unserer Szene (bis auf den flachen, farblosen Boden) keine weiteren, umliegenden Objekte in der vorbereiteten Szene gab. Die Ambient Occlusion sollte nur durch den Boden und Sophie selbst errechnet werden.
- Erhöhung der minimalen und maximalen Sampling Quality („4 bzw. 16 Samples per Pixel") zur Verbesserung der Kantenglättung (Antialiasing: „Box"-Filtertyp).

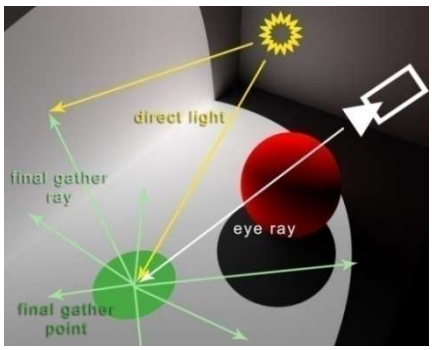


Tabelle 4: Final Gathering, Quelle: Wikipedia, ShareAlike 2.5 License, Autor: Tobias Rütten

- Reduktion des niederfrequenten Rauschens beim Final Gathering („Extremely High") mithilfe des integrierten Noise Filters.
- Verringerung der Trace Depth (Anzahl der Final Gather Rays) von 5 auf maximal 3 Reflexionen, zur Verringerung der Rechenzeit. Zu beachten ist hier, dass auch diffuse Objekte Licht reflektiere und die Beleuchtung beeinflussen, d.h. eine Szene wird bei einem geringen Wert dunkler, bei einem höheren heller.
- Verwendung der Final Gather Point Interpolation. Zur schnelleren Berechnung der Intensität eines Strahls, werden benachbarte Final Gather Punkte verwendet. Dazu wird vom Benutzer ein maximaler Pixelradius festgelegt (max. 5/min. 0).

[13] Mehr zum Rendern mit Final Gathering und indirekter Beleuchtung siehe: 3ds Max 9 Bible [Murdock, 2007, S. 1073]

Zu den allgemeinen Vorbereitungen für das Texturrendern in 3ds Max gehören:

- Die Vermeidung von Überlappungen der Texturkoordinaten gehört zu den wichtigsten Voraussetzungen beim Texturrendern. Zwar gibt es als zusätzliche Möglichkeit die automatische Vergabe von Texturkoordinaten (womit sämtliche Flächen getrennt und nach Größe sortiert werden) und diese Auswahl auf einen zusätzlichen Mapping Kanal zu legen. Es empfiehlt sich jedoch bei Charakteren, das ausgewählte Objekt von von Hand mit Texturkoordinaten zu versehen und einen Mapping-Kanal zu verwenden, um beim Texturieren in Photoshop die Ambient Occlusion als Layer über die Textur legen und zusammenhängende Strukturen besser erkennen und bemalen zu können.
- Wie schon bei der Berechnung von Normalen in ZBrush, bietet es sich beim Texturrendern an, die Ränder der gerenderten Map nach außen hin um einen bestimmten Wert zu erweitern. Dies ist notwendig, damit die Seams der Texturkoordinaten später in keinen unerwünschten (Rand-)Bereich der Textur fallen. In 3ds Max nennt sich diese Funktion „Padding“ (schon bekannt als Overlap der Normal Map in ZBrush), dessen Wert wir auf 4 Pixel gestellt haben.
- Die Render-to-Texture-Funktion bietet die Möglichkeit unterschiedliche Mapping Channels zu verwenden. Wir haben den einzig existierenden, den 1. Kanal, ausgewählt.
- Es wurde keine automatische Neuberechnung der Texturkoordinaten und keine Projektion durchgeführt.
- Als Output Kanal wählten wir allein die Ambient Occlusion (MR).

2.6.2.3 Rendering der Ambient Occlusion

Die Berechnung erfolgte schließlich lokal auf einem PC und benötigte etwa 7 Stunden auf einem Dual Core von Intel mit 2,4 GHz. Das Ergebnis war jedoch auf Anhieb vollkommen zufriedenstellend und wurde verlustfrei als PNG gespeichert. Auf den Maps sind die einzelnen Strukturen Sophies Oberfläche diffus schattiert und gleichmäßig ausgeleuchtet. Details der zugewiesenen Normal Map sind gut erkennbar, aber weniger deutlich als bei der Cavity Map aus ZBrush.

Abbildung 30: Ambient Occlusion (v.l.n.r.: Sophie als AO Rendering, AO des Kopfes, AO des Körpers)

2.6.3 Texturrendering der Haare

2.6.3.1 Haare in Echtzeit

Obwohl es (insbesondere von dem Grafikkartenhersteller NIVDIA) auf CUDA[14] und DirectX 11 basierende Projekte zur Berechnung von Haaren in Echtzeit (mithilfe der Grafikkarte/GPU) gibt[15], sind diese leider noch in keiner aktuellen Spiele-Engine integriert. Haare, die mithilfe von Vektoren oder Geometrie gerendert werden, besitzen aufgrund ihrer Schattierungen, Transluszenz und räumlichen Ausbreitung weitaus mehr Glaubwürdigkeit, als Haare, die nur als Texturen auf ebenen Flächen zu sehen sind.

Abbildung 31: Echtzeit-Haare mit CUDA und DirectX 11, Copyright: NVIDIA, Siggraph 2008

Da die Leistungsfähigkeit von Grafikkarten und PCs immer weiter zunimmt, ist die Darstellung von Haaren, wie sie in Abbildung 31 zu sehen sind, auch bald für zukünftige Computerspiele zu erwarten.

Für das gewöhnliche Rendering von Haaren über die CPU, gibt es in 3ds Max den „Hair and Fur"-Modifikator, der erfahrungsgemäß sehr gute Ergebnisse beim Rendering von Haaren erzielt. Leider war es damit eben noch nicht möglich Haare in Echtzeit anzuzeigen, oder diese in die OGRE-Engine oder in NOAH zu übertragen. Dies bedeutet, dass Sophies Haare weiterhin über Texturen dargestellt werden mussten.

Wir haben uns jedoch überlegt, den „Hair und Fur"-Modifikator auf andere Weise zu verwenden. Um uns das Bemalen der Haare in Photoshop zu erleichtern, verwendeten wir die Funktion dazu die Haare (und den zugehörigen Alpha-Kanal) direkt auf eine Textur zu rendern. Statt also der Berechnung der Haare in Echtzeit, haben wir diese einfach als Map gerendert.

2.6.3.2 Vorbereitungen

Das Rendering der Haare erfolgte in einer neuen Szene, da verschiedene Anpassungen nötig waren, um die Haare exakt an den Texturkoordinaten auszurichten. Als Hintergrundbild dieser Szene diente das Template der Texturkoordinaten (siehe Seite 43, Abbildung 21). Da der „Hair and Fur"-Renderer nicht mit einer orthogonalen Ansicht gerendert werden kann, mussten wir eine zentrierte Kamera zur Fixierung der Benutzeransicht verwenden. Auf der Vorlage der Texturkoordinaten des Kopfes zeichneten wir Splines auf all den Texturregionen, die für die Geometrie der Haare verantwortlich waren. Nach der Positionierung der wichtigsten Splines wird der „Hair and Fur"-Modifikator darauf angewendet und zeigt (zu einem bestimmten Prozentsatz) eine grobe Voransicht der Haare, die später zu sehen sein werden. Die Splines werden nun zu sogenannten „Guide Lines", die nachträglich natürlich noch vom Benutzer beeinflusst und verändert werden können. Zwischen diesen Kurven interpoliert der „Hair and Fur"-Modifikator eine vorgegebene Anzahl von „Hairs" und passt deren Länge und Krümmung den umliegenden „Guides" an.

Die Voransicht im Viewport war eine große Hilfe bei der Ausrichtung der Splines. Einige Anpassungen im „Hair and Fur"-Modifikator waren noch notwendig, um die Haare dichter und gepflegter wirken zu lassen. Ziel war es Sophie einen Pferdeschwanz zu geben und Strähnen über das Gesicht fallen zu lassen.

---

[14] CUDA: Computer Unified Device Architecture von NVIDIA dient zur Beschleunigung grafischer und physikalischer Berechnungen mithilfe der Grafikkarte (GPU).

[15] Vorgestellt auf der Siggraph 2008: Echtzeit Haar Rendering auf der GPU[NVIDIA Corporation, 2008], Präsentation und Hintergrundinformationen unter: http://developer.nvidia.com/object/siggraph-2008-hair.html

Zu den wichtigsten Einstellungen beim „Hair and Fur“-Modifikator gehören:

- Die Anzahl und Untergliederung der Haare (Hair Count: 10000, Segments: 50)
- Dichte, Skalierung, Schnittlänge (jeweils 100%)
- Zufällige Skalierung der Haarlängen (20%)
- Dicke der Wurzel und Spitzenhaare (0,3mm, 0,1mm)
- Gelb, Orange als Farbe der Haare mit zufälligen Farbton- (10%) und Helligkeitsvariationen (100%)
- Kein Glanz (Specular: 0), keine Lockenkräuselung (Frizz: 0)
- Wellen in den Spitzen (Kink: 3)
- Wenige Strähnen (Strands: 1)

Mehr zur Funktionsweise des „Hair and Fur“-Modifiaktors in der 3ds Max 9 Bible **Es ist eine ungültige Quelle angegeben.**.

2.6.3.3 Rendering der Haare

Da die Final Gathering und Global Illumination-Funktionen von Mental Ray beim Rendering der Haare nicht benötigt wurden (keine Wechselwirkungen mit dem Licht erwünscht), wählten wir den Standard-Scanline Renderer in 3ds Max. Die Haare werden somit nicht als Geometrie, sondern als Effekt im Image-Buffer in 3ds Max gerendert und nachträglich in das Rendering eingefügt. Da nur schattenloses Licht aus der gerendert Sicht schien, dauerte dieser Prozess verhältnismäßig kurz und lieferte für uns vollkommen zufriedenstellende Resultate.

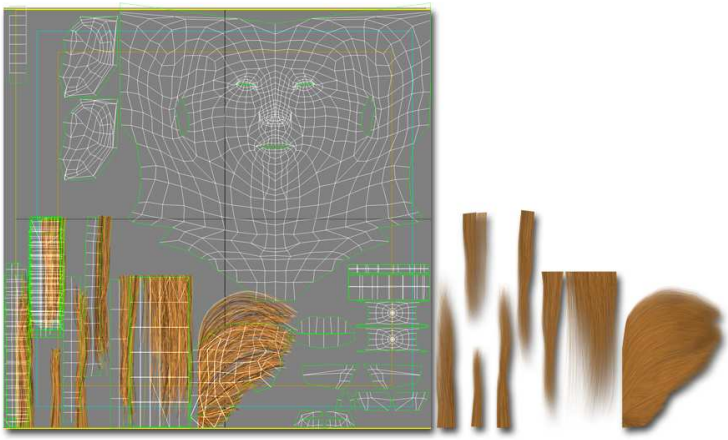


Abbildung 32: Haare (links: Anpassung der Splines an die Texturkoordinaten, rechts: Rendering der Haare)

Der wichtigste Vorteil bei der Zeichnung von Sophies Haaren in 3ds Max gegenüber Fotos oder Zeichnungen, war der zusätzliche Alpha-Kanal, der nach dem Rendern in eine PNG mitgespeichert wurde. Insbesondere im Sub-Pixel-Bereich, sprich an Stellen, an denen die Haare dünner als die gerenderten Pixel sind, schien uns der „Hair and Fur“-Effekt weitaus bessere Ergebnisse zu liefern, als ausgeschnittene Fotos oder von Hand gezeichnete Haare.

Da wir nicht darauf angewiesen sein wollten, dass andere Programme (oder das Spiel) die Hintergrundfarbe des Bildes im Verhältnis zum integrierten Alphakanal („Premultiplied Alpha“) abmischen konnten, dass z.B. die schwarze Hintergrundfarbe eines transparenten Pixels bei der Anzeige der Map wieder herausgenommen wird, wählten wir für die Hintergrundfarbe des Renderings in etwa die RGB-Farbe von Sophies Haaren. Somit waren selbst bei einer fehlenden Premultiplied Alpha keine (meist schwarze, verschwommene) „Ränder“ entlang halbtransparenter Bereiche erkennbar.

2.6.4 Texturierung in Photoshop

Zur Bildbearbeitung der Maps verwendeten wir Adobe Photoshop CS3. Aufgrund seiner umfangreichen Funktionen und Filter stellte es das ideale Werkzeug für die Texturierung von Sophie dar. Besonders die Ebenen- und Füllmethoden des Programms waren von großer Wichtigkeit für uns. Obwohl sich der folgende Abschnitt unserer Arbeit explizit mit der Software Photoshop (Version 10, CS3) von Adobe beschäftigt, möchten wir nicht auf die vielen Funktionen und Filter dieses Bildbearbeitungsprogramms eingehen, sondern uns auf eine allgemeine Übersicht bei der Erstellung der einzelnen Maps von Sophie konzentrieren.

Ebenen & Gruppen

Ein zentraler Bestandteil bei der Texturierung war die Verwendung von Ebenen, die wiederum zu Gruppen zusammengefasst werden können. Auf der Basis der Renderings der Texturkoordinaten („Wireframe"-Ebene) entstanden die Photoshop-Dateien (PSDs) von Kopf und Körper. Die Kombination von Ebenen war für uns sehr wichtig, um effektiv Bilddaten zu ersetzen, zu löschen oder neu einzufügen. Bestimmte Regionen (z.B. Mund, Augen, etc.) wurden stets zu Ebenen zusammengefasst, die unabhängig voneinander bearbeitet werden konnten.

Gruppen können die zahlreichen Ebenen in Container zusammenfassen. In unseren Dateien repräsentierte einfach jeder Container jeweils eine Map dar (Diffuse, Specular, Glossiness, Transparenz und die einfügte Normal Map). Da die Ambient Occlusion und Cavity Map nur auf die Diffuse angewendet und bearbeitet werden sollten, lagen diese separat in der „Diffuse"-Gruppe.

2.6.4.1 Diffuse Map

Von „diffuser Reflexion" spricht man im Allgemeinen, wenn Licht auf ein Objekt trifft, dass aufgrund seiner unregelmäßigen Oberflächenbeschaffenheit in alle Richtungen zerstreut wird. In der Computergrafik wird die Diffuse Map allerdings zur Darstellung von Farben verwendet und daher oft auch als „Color Map" bezeichnet. Sie ist die wohl wichtigste Textur für die Darstellung von Sophie. Die Farben und Strukturen der Diffuse Map bilden die Grundlage für die Specular und Glossiness Map (Glanz).

Bemalungstechniken

In Photoshop standen zahlreiche Pinselmuster zur Auswahl, die mit folgenden Bemalungstechniken verwendet werden konnten:

- ***Pinsel, Radierer*** – Zur gewöhnlichen Bemalung mit einem Pinselmuster mit einer bestimmten Stärke und Größe (Schatten, Make Up, Finger, Hautfalten, Adern, etc.)
- ***Kopierstempel*** – Wird dazu verwendet, Ausschnitte ein- oder mehrerer Ebenen herauszukopieren und an anderer Stelle wieder einzufügen (Gesicht, Lippen, Brust, Lederstrukturen, etc.).
- ***Pipette*** – Dieses Werkzeug hilft zur Auswahl einer bestimmten Farbe (z.B. Farbtöne der Haut).
- ***Abwedler, Nachbelichter, Schwamm*** – Werkzeuge wie sie aus der Phototechnik bekannt sind. Sie helfen verschiedene Stellen aufzuhellen, abzudunkeln oder zu bleichen (Lidschatten, Wimpern, Haare, Fingernägel, Aufhellung der Ohren an transluzenten Zonen, etc.).
- ***Verwischen, Scharfzeichnen, Weichzeichnen*** – Korrekturpinsel für die Schärfe eines Bereichs (Schuhe, Gürtel, Kleidung)
- ***Füllen*** – Um geschlossene Flächen mit einer neuen Farbe zu versehen, benötigt man dieses Werkzeug.
- ***Masken*** – Die Bereichsauswahl zur begrenzten Bearbeitung mithilfe von Masken, die direkt oder ebenfalls mithilfe von Pinselwerkzeugen gezeichnet werden können.

Verwendung von Photos

Die Suche nach einem Farbton für die Haut, die Bemalung des Gesichtes mit Make Up und die Positionierung der Farbnuancen und feinen Strukturen gehörten zu den großen Herausforderungen bei der Erstellung der Texturen. Neben der allgemein üblichen Bearbeitung mithilfe des Grafiktabletts, verwendeten wir auch die Möglichkeiten der Retusche und Maskierung von Photos. Dazu griffen wir auf ein freies verfügbares Fotoarchiv eines US-amerikanischen Models (Aneta) zurück, einige selbst photographierte Lederstrukturen und freie Texturen der Seite www.cg-textures.com.

Da durch die Texturierung sämtliche verwendete Fotos nahezu vollständig unkenntlich gemacht wurden, haben wir die Auflistung der Quellen auf eine Übersichtsauswahl der wichtigsten acht beschränkt. Abbildung 33 veranschaulicht die Regionen von Sophies Textur, an denen die Retusche besonders häufig angewendet wurde.

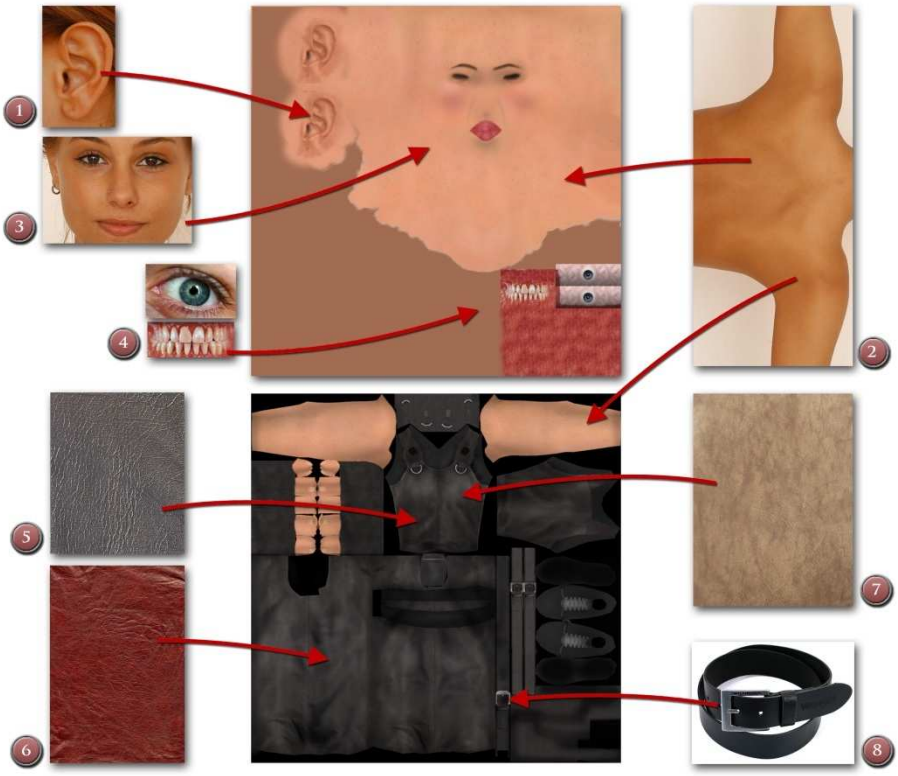


Abbildung 33: Kombination und Retusche von Photomuster für die Diffuse Map

	Photo	Rolle/Verwendung	Herkunft
1	Ohr	Ohrmuschel links und rechts	Freies Photoarchiv (Aneta, www.3d.sk)
2	Rücken	Große Hautflächen und Arme	Freies Photoarchiv (Aneta, www.3d.sk)
3	Gesicht	Lippenstruktur, Augenbrauen	Freies Photoarchiv (Aneta, www.3d.sk)
4	Augen/Zähne	Iris und Mundinnenraum	Werbung
5	Lederhaut	Abgenutzte Lederbereiche	www.cgtextures.com
6	Ledersessel	Basistextur der Kleidung	selbst photographiert
7	Stoffbezug	Schattierungen und Shirt-Muster	www.cgtextures.com
8	Gürtel	Schnalle und Schlaufe	Modezeitschrift

Tabelle 5: Verwendung und Quellen von Photos

Wichtige Erkenntnisse bei der Retusche

Beim nachträglichen Abgleich der Textur mit dem Modell der Sophie in 3ds Max kamen wir zu folgenden Erkenntnissen:

- Phototexturen wirken weitaus glaubwürdiger, als reine Pinselbemalungen.
- Gegenüber Computeranimationen müssen Spieltexturen mehr Schattierungen besitzen (sind insgesamt dunkler).
- Augen sind und bleiben der wichtigste Teil der Darstellung.
- Oft werden Verzerrungen erst mit der Anzeige der Textur im Viewport sichtbar. Daher müssen Texturkoordinaten nachträglich korrigiert werden.
- Nur sichtbare Bereiche sollten auch detailliert texturiert werden.
- Farben sollten kräftig, aber nicht zu bunt wirken.
- Photos sind auch als Vorlagen für Pinselbemalungen sinnvoll (z.B. beim Nachzeichnen von Äderchen)

Füllmethoden, Deckkraft & Kombination der Ebenen

Zur Überlagerung und Kombination der in Kapitel 2.6.2 erstellten Ambient Occlusion (Abbildung 34, 1), der in Kapitel 2.5.5.4 erstellten Cavity Map (Abbildung 34, 2) und in Kapitel 2.6.3 gerenderten Haare (Abbildung 34, 3) verwendeten wir generell die „Multiplizieren"-Füllmethode von Photoshop. Die Farbinformationen der übereinanderliegenden Kanäle werden zusammenmultipliziert. So entsteht z.B. beim Multiplizieren einer Farbe mit Schwarz ebenfalls Schwarz, d.h. die Helligkeit verringert sich mit dem Farbwert. Nur eine Multiplikation mit Weiß, lässt den Wert unverändert. Ambient Occlusion und Cavity Map lassen sich mit dieser Füllmethode komfortabel auf den bemalten Untergrund auftragen.

Andere Füllmethoden kamen nur in geringem Maße zum Einsatz, bildeten also keinen zentralen oder vollständig flächendeckenden Bestandteil einer Map. So wurde z.B. nur bei der Make Up-Ebene im Gesichtsbereich die Füllmethode „Farbton" verwendet, sodass alle darunterliegenden Pixel jeweils die Farbnuance dieser Ebene enthalten.

Hautstrukturen wurden sehr häufig auch mit der „Luminanz"-Füllmethode hervorgehoben, wobei die Ergebnisfarbe die Sättigung und Helligkeit der Füllfarbe erhält.

„Ineinander kopieren" führt eine Multiplikation und eine negative Multiplikation der Farben durch. Lichter und Tiefen der Ausgangsfarbe bleiben dabei erhalten, während Muster und Farbtöne überlagert werden. Die Ausgangsfarbe wird dabei nicht ersetzt, sondern mit der Füllfarbe gemischt, die die Lichter und Tiefen der Originalfarbe wiederspiegeln. Diese Füllmethode eignete sich für besonders auffällige Strukturen, wie z.B. Lippen oder Augenbrauen.

„Farbig abwedeln" verringert den Kontrast und hellt die Farbinformationen der einzelnen Kanäle auf. Bei einer schwarzen Füllung findet keine Änderung statt. Diese Füllmethode war besonders bei den hellen, rosigen Backen von Sophie hilfreich.

22 weitere Füllmethoden gibt es in Photoshop. Mehr als die oben genannten wurden jedoch bei der Texturierung von Sophie nicht eingesetzt und finden aus diesem Grund keine zusätzliche Erwähnung.

Die Deckkraft der einzelnen Ebenen ist für deren Sichtbarkeit verantwortlich. Nicht jede Ebene ist 100% sichtbar, daher variieren die Ebenen innerhalb unserer Maps stark mit ihrer Deckkraft.

Kombination der Ebenen zur vollständigen Diffuse Map

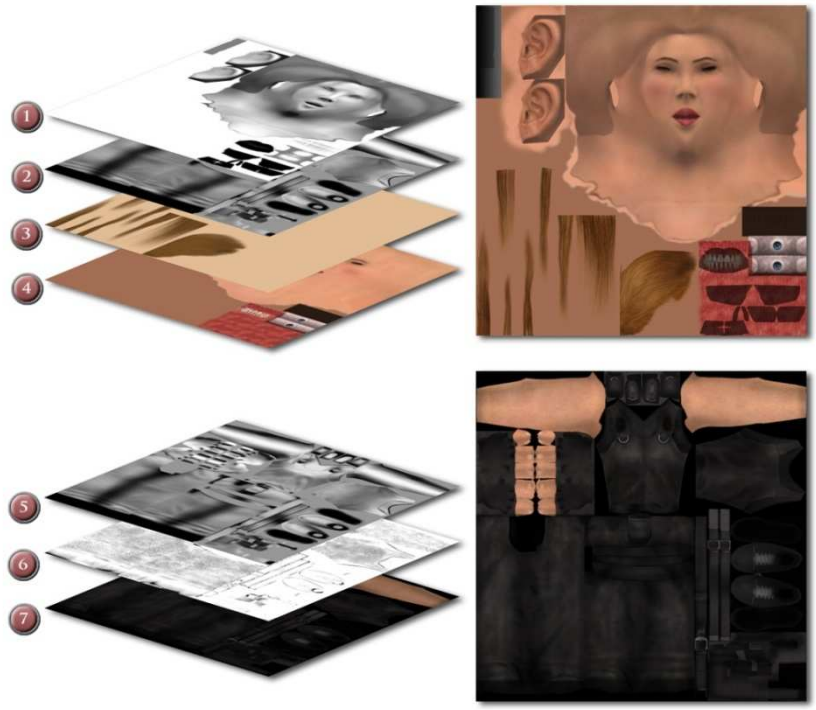


Abbildung 34: Kombination der Layers zu den Diffuse Maps (links), fertige Diffuse Maps

	Ebene	Füllmethode	Dreckkraft	Nachbearbeitet
1	Ambient Occlusion (Kopf)	Mulitplizieren	70%	Ja (ohne Haare)
2	Cavity Map (Kopf)	Mulitplizieren	30%	Nein
3	Haare (Kopf)	Normal	100% (Alpha als Maske)	Ja (Alphakanal)
4	Color Map (Kopf)	Normal	100%	Nein
5	Ambient Occlusion (Körper)	Mulitplizieren	70%	Nein
6	Cavity Map (Körper)	Mulitplizieren	30%	Nein
7	Color Map (Körper)	Normal	100%	Nein

Tabelle 6: Füllmethoden wichtiger Ebenen in Photoshop

2.6.4.2 Transparency Map

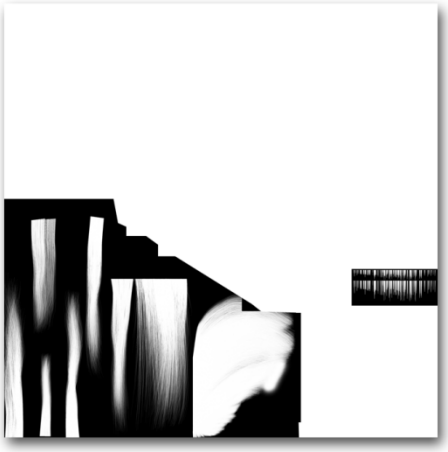


Tabelle 7: Transparency Map (Alpha Map der Diffuse Map)

Da Sophies Kopf weder Haare noch Wimpern besitzt, benötigten wir eine Opacity Map für die Transparenz der Teiloberflächen, die unsichtbar sein sollte (weiße Stellen sind sichtbar, schwarze unsichtbar). Die Transparenz der Haare gewannen wir aus der Map, die wir in Kapitel 2.6.3.3 aus 3ds Max heraus gerendert hatten.

Die Wimpern wurden manuell mit schwarz nachgezeichnet (je etwa 120 Härchen für oben und unten) und schließlich invertiert.

2.6.4.3 Specular Map

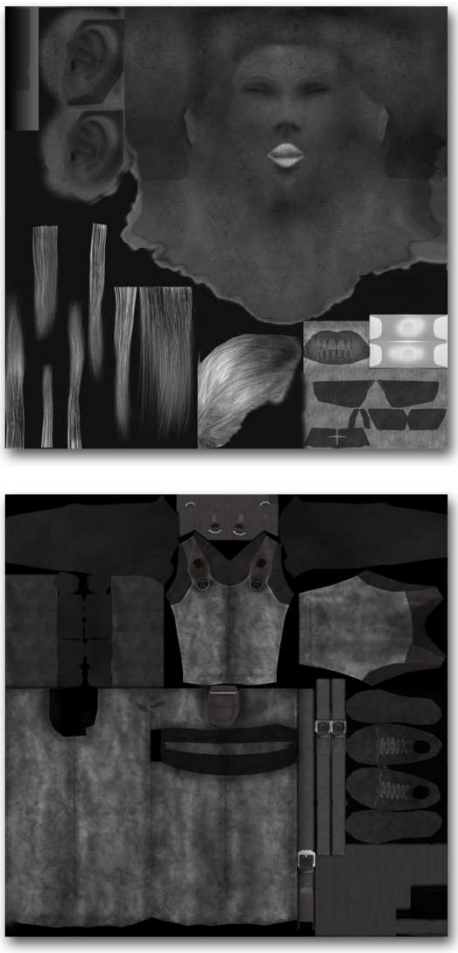


Abbildung 35: Specular Map von Kopf (oben) und Körper

Eine wichtige Eigenschaft jedes Objekts ist sein Glanz oder seine Mattheit (sprich fehlender Glanz). Um der Oberfläche eines Objekts zusätzlich zu ihrer Farbe Glanz zu verleihen, benötigt man eine sogenannte Specular Map, die die Stärke der Reflexion des Lichts festlegt und mit der Helligkeit ihrer Pixel die Stärke des Glanzes bestimmt.

Trotz des Namens (specular: spiegelnd) handelt es sich hierbei um keine Reflexion der Strahlen einer Szene (wie das beim Raytracing der Fall ist), sondern nur um eine einfache Glanzpunktberechnung des umgebenden Lichts. Wichtig ist es hierbei sich zu überlegen, wo Glanz auf der Sophie überhaupt benötigt wird.

Markante Stellen, bei denen dies der Fall war, sind Sophies Augen, ihre Lippen, die Haare, die Metalle der Gürtelschnallen, sowie auch etwas das Leder ihres Anzugs. Das bedeutet, dass die Helligkeit glatter und angefeuchteter Bereiche erhöht werden muss, um diese mit mehr Glanz zu versehen. Allerdings sollte auch darauf geachtet werden, dass die Strukturen der Diffuse Map erhalten bleiben, damit Specular und Diffuse Map übereinstimmen. Daher werden Ebenen vor ihrer Bearbeitung dupliziert, vollständig entsättigt und ihre Helligkeit über eine Tonwertkorrektur angepasst oder invertiert.

2.6.4.4 Glossiness Map

Da die Specular Map selbst allein nur für die Stärke eines Glanzpunktes verantwortlich ist, verwendet man sehr häufig als Ergänzung noch die sogenannte Glossiness (Glanz) Map. Sie bestimmt die Größe eines Glanzpunkts. Die Helligkeit ihrer Pixel bestimmt den Durchmesser. Dabei gilt: Je heller ein Bereich, desto kleiner wird seine Glanzfläche.

Bei Metall ist der Glanzpunkt z.B. sehr viel kleiner als auf der rauen Lederoberfläche von Sophies Kleidung. Daher sind die Metalle der Glossiness Map weiß. Die Haut besitzt einen durchschnittlich großen Glanzpunkt und ist daher grau auf der Textur.

Um im Spiel keine (unserer Ansicht: unnötigen) Ressourcen im Speicher unserer Echtzeitanwendung zu verbrauchen, verzichteten wir auf weitere zusätzliche Maps, die zur Helligkeit und Größe des Glanzes auch noch ihre Farbe oder Form festlegen. Ohne diese Map ist die Farbe jedes Glanzpunktes neutral weiß und kreisrund. Die sogenannte „Specular Color Map“ wird häufig eingesetzt, um farbige Glanzpunkte zu erzeugen. Die „Anisotropy Map“ bestimmt dabei die Ausrichtung des Glanzpunkts. Keine der beiden Map Formen kam bei Sophie zum Einsatz.

Abbildung 36: Glossiness Map von Kopf (oben) und Körper

2.6.4.5 Nachbearbeitungen

PNGs besitzen meist einen Alphakanal. Um Texturspeicher zu sparen, wurde der Transparenz Kanal als Alphakanal der Diffuse Map verwendet. Dazu wurde mithilfe von Photosoph einfach die Schwarzweiß-Map der Transparenz in den Alphakanal der fertigen Diffuse geladen. Die Transparenzen der PSD wurden unverändert gelassen, um nachträgliche Korrekturen ausführen zu können.

Um weiteren Texturspeicher zu schonen wurden sämtliche Maps mit 16-Bit auf 8-Bit Farbtiefe heruntergesetzt und gespeichert. Zusätzlich wurden in Photoshop die Normal Maps von Kopf und Körper etwas abgeschwächt (30%), da uns die ursprüngliche Stärke zu stark schien.

2.6.5 Material- & Shadereinstellungen

Jedes Material eines Objekts ist hierarchisch gegliedert und setzt sich aus einzelnen Elementen zusammen. Die Zusammensetzung bzw. Berechnung der Maps wird durch Shader bestimmt. OGRE verwendet für die Darstellung in Echtzeit spezielle Shaderprogramme und benötigt Informationen darüber, mit welcher Technik Objekte dargestellt werden sollen. In 3ds Max wird dies mithilfe eines zusätzlichen Materialtyps, dem **OGRE Material**, eingestellt. Für die Anzeige im Viewport oder für Renderings in 3ds Max kann zwar das Original-Max-Material in das oFusion-Material eingefügt werden, Standard-Material-Einstellungen aus 3ds Max werden beim Export jedoch nicht mitberücksichtigt. Für den Export benötigt OGRE also einen eigenen Materialbaum, der verschiedene Darstellungstechniken beinhalten kann.

Grafikkarten verschiedener Generationen unterscheiden sich in ihren technischen Möglichkeiten. Dadurch kann es vorkommen, dass gewisse Techniken von älteren Grafikkarten nicht unterstützt werden. Um dem entgegen zu wirken, kann man verschieden komplexe Techniken bereitstellen, wovon OGRE automatische die erste auswählt, die von der aktuell sich im PC befindenden Grafikkarte unterstützt wird. Zusätzlich können weiter entfernte Objekte auch einfachere Rendertechniken verwenden, da man bei ihnen den hohen Detailgrad aufgrund der Entfernung nicht mehr wahrnehmen kann. Welche Techniken verfügbar sind, hängt davon ab, ob sie von der Grafikkarte, auf der das Programm läuft, unterstützt werden. Wir haben Sophie in „NOAH“ für Gesicht, Haare und Körper nur eine OGRE-**Technique** (mit dem Abstand 0 beginnend) zugewiesen. Die Basis-Parameter der Techniques sind die sogenannten **Passes**. Diese sind für die Maps und deren Kombination durch Vertex- und Pixelshader verantwortlich. Die einzelnen Bitmaps werden schließlich durch die **Texture Units** festgelegt. Mit ihnen werden die Maps angegeben, die bei der Texturierung (siehe Seite 54, Kapitel 2.6) entstanden sind. Techniques, Passes und Texture Units können jeweils nacheinander vier Mal angewendet werden (sprich: je 4 Techniques pro Material, 4 Passes pro Technique, 4 Texture Units pro Pass).

2.6.5.1 Materialien

Sophie_Body (Körperobjekt)	**Typ & Shaderfunktion**
oFusion/OGRE Material (oFusion Material)	OGRE Material-Container
+- MAX_Sophie_Body (Standard)	Material für die Anzeige in 3ds Max
+- Diffuse Color: Diffuse (Sophie_body_diffuse.png)	Farbtextur des Körpers
+- Specular Level: Specular (Sophie_body_specular.png)	Glanzstärke
+- Glossiness Color: Glossiness (Sophie_body_glossi.png)	Glanzpunktstärke
+- Bump: Bump (Normal Bump)	Normal Bump Map
+- Normal Bump (Sophie_body_normal.png)	
+- OGRE_Sophie_Body (oFusion Technique)	Export-Material für OGRE (LoD: 0)
+- pass_1_Ambient (oFusion Pass)	Ambiente Beleuchtung zur Aufhellung
+- pass_2_NormalSpec (oFusion Pass)	Berechnung der diffusen Beleuchtung in Abhängigkeit der Normal Map und Addition zu Pass 1
+- Normal: Sophie_body_normal.png (oFusion TextureUnit)	
+- pass_3_Diffuse (oFusion Pass)	Muliplizierung des Ergebnisses von Pass 1 und 2 mit der Diffuse Map
+- pass_4_SpecGlossiNormal (oFusion Pass)	Berechnung (mithilfe der Normal, Specular und Glossiness Map) und addiert den Glanz
+- Normal: Sophie_body_normal.png (oFusion TextureUnit)	
+- Specular: Sophie_body_specular.png (oFusion TextureUnit)	
+- Glossiness: Sophie_body_glossi.png (oFusion TextureUnit)	

Tabelle 8: Materialbaum des Körpers

Alle Shadings der OGRE Passes besitzen „Gouraud-Shading“. Die Standard-Materialien (hier grau gekennzeichnet) wurden in 3ds Max mithilfe von Blinn Shadern dargestellt. Mehr zu den Shadern in 3ds Max finden sie in der „3ds Max Bible“, S. 526.[16]

Alpha Rejection

Aufgrund von Problemen bei der Darstellung mehrerer, hintereinanderliegender Transparenzen (Haare und Wimpern) erhielt der Kopf zwei Material-IDs. Ursache dieses Problem ist zunächst der Z-Buffer einer Grafikkarte, der für einen Pixel nur eine Tiefe vorschreibt. Überlagern sich mehrere Transparenzen schreibt die Grafikkarte jeden Z-Wert (in einer mehr oder weniger zufälligen Reihenfolge) in den Z-Buffer. Die Grafikkarte nimmt also irgendwann an, dass dahinterliegende Geometrie verdeckt ist. Mithilfe der Alpha Rejection kann nun aufgrund eines Referenzwertes entschieden werden, ob ein Pixel gemalt werden soll (sprich in den Z-Buffer und den Backbuffer geschrieben werden soll) oder nicht. Wir haben die „Alpha Rejection" der Haare auf größer gleich („greater equal“) eines Mittelwerts (128) gestellt, sodass Pixel mit einer Transparenz in diesem Wertebereich ausgeblendet werden. Opake Objekte bekamen keine Alpha Rejection (always pass, 0).

Damit allerdings die Umgebungslichtberechnung (Ambient), die Normal Map, Specular und Glossiness gleichermaßen den Alpha Kanal der Diffuse Map mitberücksichtigen, muss noch ein angepasster Shader geschrieben werden. Bis zum Zeitpunkt der Abgabe dieses Kapitel bezog sich die Alpha Rejection nur auf den Diffuse Pass. Die Haare besaßen damit nur die Glanzwerte des Standardmaterials, ohne Berücksichtigung der übrigen Maps.

Sophie_Head (Kopfobjekt)	**Typ & Shaderfunktion**
Mulit/Sub-Object Material	ID:1 Gesicht, ID:2 Haare
(1) : oFusion/OGRE Material (oFusion Material)	OGRE Material-Container
+- MAX_Sophie_Head (Standard)	Material für die Anzeige in 3ds Max
+- Diffuse Color: Diffuse (Sophie_head_diffuse.png)	Farbtextur des Körpers
+- Specular Level: Specular (Sophie_head_specular.png)	Glanzstärke
+- Glossiness Color: Glossiness (Sophie_head_glossi.png)	Glanzpunktstärke
+- Bump: Bump (Normal Bump)	Normal Bump Map
+- Normal Bump (Sophie_head_normal.png)	
+- OGRE_Sophie_Head (oFusion Technique)	Export-Material für OGRE (LoD: 0)
+- pass_1_Ambient (oFusion Pass)	Ambiente Beleuchtung zur Aufhellung
+- pass_2_NormalSpec (oFusion Pass)	Berechnung der diffusen Beleuchtung in Abhängigkeit der Normal Map und Addition zu Pass 1
+- Normal: Sophie_head_normal.png (oFusion TextureUnit)	
+- pass_3_Diffuse (oFusion Pass)	Muliplizierung des Ergebnisses von Pass 1 und 2 mit der Diffuse Map
+- pass_4_SpecGlossiNormal (oFusion Pass)	Berechnung (mithilfe der Normal, Specular und Glossiness Map) und addiert den Glanz
+- Normal: Sophie_head_normal.png (oFusion TextureUnit)	
+- Specular: Sophie_head_specular.png (oFusion TextureUnit)	
+- Glossiness: Sophie_head_glossi.png (oFusion TextureUnit)	
(2) : oFusion/OGRE Material (oFusion Material)	OGRE Material-Container
+- MAX_Sophie_Head (Standard)	Material für die Anzeige in 3ds Max
+- Diffuse Color: Diffuse (Sophie_head_diffuse.png)	Farbtextur des Körpers

[16] Mehr über den Blinn-, Phong- und weitere Shader in 3ds Max siehe: 3ds Max 9 Bible [Murdock, 2007, S. 526]

+- Specular Level: Specular (Sophie_head_specular.png)	Glanzstärke
+- Glossiness Color: Glossiness (Sophie_head_glossi.png)	Glanzpunktstärke
+- Opacity: Opacity (Sophie_difffuse_png)	Alpha Kanal der Diffuse
+- Bump: Bump (Normal Bump)	Normal Bump Map
+- Normal Bump (Sophie_head_normal.png)	
+- OGRE_Sophie_Head (oFusion Technique)	Export-Material für OGRE (LoD: 0)
+- pass_1_Ambient (oFusion Pass)	
+- pass_2_NormalSpec (oFusion Pass)	
+- Normal: Sophie_head_normal.png (oFusion TextureUnit)	
+- pass_3_Diffuse (oFusion Pass)	Diffuse Farbe mit Alpha Rejection „greater equal“ 128 ohne Verwendung von Shader
+- pass_4_SpecGlossiNormal (oFusion Pass)	
+- Normal: Sophie_head_normal.png (oFusion TextureUnit)	
+- Specular: Sophie_head_specular.png (oFusion TextureUnit)	
+- Glossiness: Sophie_head_glossi.png (oFusion TextureUnit)	

Tabelle 9: Materialbaum des Körpers

Shader-Einstellungen

Mithilfe unseres Materialbaumes verwendet der oFusion-Exporter bzw. dessen spezielles Material beim Export nach OGRE die angewandten "Pass"-Maps und die verschiedene Shader- und Material-Einstellungen. Zu den Pass-Einstellungen gehören:

- ***Pass Basic Parameters:*** Für die Grundeinstellungen der Materialparameter (z.B. Diffuse, Ambient und Specular Color). Zusätzlich wird hier festgelegt, ob der Farb- oder Alphakanal der Quelle bei der Darstellung verwendet werden soll (Src Blend, Dest Blend).
- ***Shading:*** z.B. Gouraud, Phong oder Flat
- ***Alpha rejection:*** always pass, less, less equal, equal, not equal, greater equal, greater mit Wert.
- ***Light Settings:*** Beleuchtungseinstellungen, Anzahl der Lichter, die auf einen Pass angewendet werden.
- ***Fog:*** In wie weit das Material in Nebel verschwinden kann (kein Veränderungen)

Der wichtigste Punkt bei der Konfiguration eines OGRE-Materials ist schließlich die Anwendung von Vertex- und Pixel-Shader-Programmen. Da keiner der vorgefertigten Shader von oFusion imstande war Specular, Bump, Glossiness, Diffuse Map und Umgebungsbeleuchtung zusammenzurechnen, fertigten wir mehrere eigene Shader an. Diese wurden nach der Programmierung in das oFusion-Verzeichnis der Shader geladen und schließlich auf die Materialien von Sophie angewendet. oFusion bietet dabei die Möglichkeit Shader interaktiv im Viewport von 3ds Max anzuzeigen. Aufgrund der Verwendung unterschiedlicher OGRE -Versionen bei der Programmierung von oFusion und unserer eigenen Shader, kam es zu einigen Fehlern bei der Darstellung im oFusion-Viewport in 3ds Max. Diese ignorierten wir und exportierten die Materialien. Dabei wurden die Shader nicht automatisch mitgespeichert, sondern müssen separat mit ins Zielverzeichnis kopiert werden.

Eine weitere, unangenehme Schwierigkeit lag bei der Speicherung der Materialeinstellung. Zahlreiche Einträge, die eigentlich variabel bleiben sollten (z.B. Licht- und Farbeinstellungen aus 3ds Max) wurden fest mit in die exportierten Materialien von OGRE gespeichert. Diese mussten nach dem Export aus den Einstellungen der .material-Dateien wieder manuell heraus gelöscht werden.

Vollständig korrigierter Programmcode der aus oFusion exportierten Material-Dateien:

Sophie_Body.mesh.material

```
material OGRE_Sophie_Body {
  technique {
    pass {
      ambient 0.5 0.5 0.5 1

      vertex_program_ref AmbientVP_1_1 {}
      fragment_program_ref AmbientFP_1_1 {}
    }
     pass {
      iteration once_per_light
      diffuse 0.8 0.8 0.8 1
      scene_blend add

      vertex_program_ref DiffNormalLightVP_2_0 {}
      fragment_program_ref DiffNormalLightFP_2_0 {}

      texture_unit {
        texture_alias Sophie_Body_Normal
        texture sophie_body_normal.png
      }
    }
    pass
    {
      lighting off
      scene_blend modulate

      vertex_program_ref TextureVP_1_1 {}
      fragment_program_ref TextureFP_1_1 {}

      texture_unit
      {
          texture_alias Sophie_Body_diffuse
          texture sophie_body_diffuse.png
      }
    }
    pass
    {
      iteration once_per_light
      specular 0.5 0.5 0.5 1 20
      scene_blend add

      vertex_program_ref SpecMapNormalLightVP_2_0 {}
      fragment_program_ref SpecMapNormalLightFP_2_0 {}
    }

    texture_unit {
      texture_alias Sopie_Body_Normal
      texture sophie_body_normal.png
    }

    texture_unit {
      texture_alias Sopie_Body_Spec
      texture sophie_body_specular.png
    }

    texture_unit {
      texture_alias Sopie_Body_Glossi
      texture Sophie_head_glossi.png
    }
  }
}
```

Sophie_Head.mesh.material

```
material Sophie_Head/OGRE_Sophie_Head {
  technique {
    pass {
      ambient 0.5 0.5 0.5 1
      vertex_program_ref AmbientVP_1_1 {}
      fragment_program_ref AmbientFP_1_1 {}
    }
    pass {
      iteration once_per_light
      diffuse 0.8 0.8 0.8 1
      scene_blend add

      vertex_program_ref DiffNormalLightVP_2_0 {}
      fragment_program_ref DiffNormalLightFP_2_0 {}

      texture_unit
      {
        texture_alias Sophie_Head_Normal
         texture Sophie_head_normal.png
      }
    }

    pass {
      lighting off
      scene_blend modulate

      vertex_program_ref TextureVP_1_1 {}
      fragment_program_ref TextureFP_1_1 {}

      texture_unit {
        texture_alias Sophie_Head_diffuse
        texture Sophie_head_diffuse.png
      }
    }

    pass {
      iteration once_per_light
      specular 0.5 0.5 0.5 1 20
      scene_blend add

      vertex_program_ref SpecMapNormalLightVP_2_0 {}
      fragment_program_ref SpecMapNormalLightFP_2_0 {}

      texture_unit {
        texture_alias Sopie_Head_Normal
        texture Sophie_head_normal.png
      }

      texture_unit {
        texture_alias Sopie_Head_Spec
        texture Sophie_head_specular.png
      }

      texture_unit {
        texture_alias Sopie_Head_Glossi
        texture Sophie_head_glossi.png
      }
    }
  }
}

material Sophie_Head/OGRE_Sophie_Hair
{
  transparency_casts_shadows on
  technique {
    pass {
      alpha_rejection greater_equal 128
      cull_hardware none
      cull_software none

      texture_unit {
        texture_alias Diffuse
        texture Sophie_head_diffuse.png
      }
    }
  }
}
```

Tabelle 10: Überarbeiteter Material Programmcode für OGRE

Da weder oFusion, noch der LexiViewer imstande waren, unsere programmierte Shader korrekt auszuführen, verwendeten wir unser eigenes Tool (die Anwendung „OgreIK“), zur Darstellung und Überprüfung der Material- und Shader-Einstellungen. Der Material-Code ist (wie das Material in 3ds Max) noch hierarchisch gegliedert und berücksichtigt noch nicht die erweiterten Möglichkeiten der Vererbung aus OGRE 1.6.

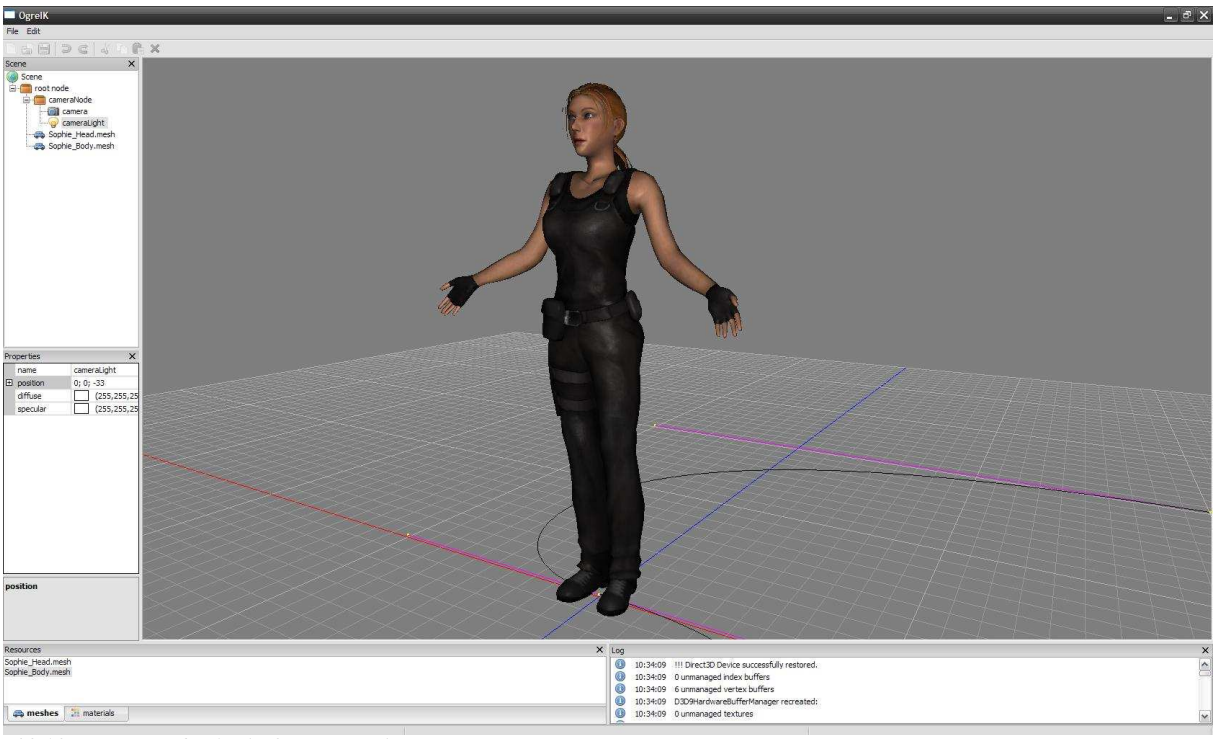


Abbildung 37: Das finale Shading von Sophie in „OgreIK“

Unsere Anwendung (kurz: „OgreIK“) war zum Zeitpunkt der Abgabe des ersten Teils dieser Ausarbeitung, imstande Shader-Programme auszuführen. Wie in Abbildung 37 zu sehen, wurden alle Texturen der exportierten Daten von Kopf und Körper, mithilfe der korrigierten Material-Einstellungen aus Tabelle 10 richtig dargestellt. Mehr über Vertex- und Pixel-Shader-Programme und die Funktionsweise unserer eigenen Shader auf Seite 127, Kapitel 3.7.1.

2.7 Animation

2.7.1 Allgemeines

Durch die lateinischen Herkunft des Wortes *animare*, könnte man annehmen, dass wir Sophie in diesem Kapitel nun vollständig „zum Leben erwecken". Dieses Kapitel beschäftigt sich jedoch nur zum Teil mit der Charakteranimation, wie sie später in der Echtzeitanwendung zu sehen sein wird. Wir möchten im folgenden Abschnitt besonders die Vorbereitungen hervorheben, die für die Animierung erforderlich sind.

Als Animation bezeichnet man im weitesten Sinne „bewegte Bilder", die der Mensch in einer Sequenz ab etwa 24 Bildern pro Sekunde als ruckelfrei bzw. flüssig wahrnimmt. Die klassische Methode bei der Computeranimierung, ist die sogenannte Keyframe-Animation, auf die wir später näher eingehen werden (siehe Seite 98, Kapitel 3.2.3). Sie ist die Basis, auf der dann die Echtzeitanimierung beruht. Bevor ein Charakter animiert werden kann, benötigt man ein System, dass für eine Bewegung verantwortlich ist. Bislang besteht Sophie aus zwei statischen Objekten (Kopf und Körper), deren Geometrie sich ohne weiteres nicht deformieren lässt. Analog zur realen Welt, benötigt man also unter der Haut des Charakters, ein verborgenes, zusammenhängendes Skelett. Ein Modifikator deformiert und animiert schließlich die Geometrie des Körpers mithilfe dieses Skeletts (Skeleton).

Steht ein solches System auf zwei Beinen, nennt man es schlicht Biped (Zweibeiner). Handelt sich nur um einzelne, verbundene Glieder, spricht man von Bones (Knochen). Die Gelenkwinkel von Biped und Bones können mithilfe der sogenannten Inversen Kinematik (IK) kontrolliert werden. Der Vorteil eines IK-Systems liegt in der einfachen Animierbarkeit seiner Endeffektoren (den sogenannten IK-Solvern). Bewegt man z.B. nur einen Anfasser eines Bipeds (Hände oder Füße), werden die Winkel der übrigen Gelenke, die zu den Anfasser führen, mithilfe der IK berechnet. Diese müssen somit nicht separat animiert werden. Winkel- und Rotationsbegrenzungen (Limits) der Gelenke verhindern Überdehnungen (z.B. bei Knie und Ellenbogen) oder ein falsches Verhalten beim Drehen.

2.7.2 IK –System

2.7.2.1 Biped

Das Biped gibt es in 3ds Max als eigenständiges, zusammenhängendes Skelettsystem mit einer festgelegten IK und Verbindungshierarchie (Bindings). Es besitzt alle wichtigen Gliedelemente (auch als Bones bezeichnet) des menschlichen Körpers und eine große Auswahl an Features, kann aber aufgrund seiner vielen Funktionen in kein gewöhnliches Bones-System umgewandelt, sondern nur durch Bones ergänzt werden. Die Stärken des Bipeds liegen in seiner einfachen Animierbarkeit, dem Figure-Modus für nachträgliche Änderungen und Einstellungen am Skelett, dem Footstep-Modus zur Animierung von Laufbewegungen, sowie den „Twists" für die bessere Deformierung bei der Verdrehung von Gelenken.

Da wir uns aufgrund der technischen Besonderheiten des Bipeds nicht sicher waren, ob dieses zuletzt exportierbar ist, haben wir zunächst einige Tests mit den Exportern für OGRE (siehe Seite 81, Kapitel 2.8) durchgeführt, bevor wir uns entschlossen haben, das Biped bei Sophie tatsächlich zu verwenden. Im weiteren Verlauf der Animierung hat sich diese Entscheidung als richtig erwiesen. Eine Alternative hierzu wäre gewesen, das Biped mit einem klassischen Bones-System nachzubauen.

Die Hierarchie des Bipeds beginnt beim Biped-Point, dem Ursprungspunkt des Skeletts, der die Position, Orientierung und Schwerpunkt des Körpers bestimmt. Darauf folgen der Hüft- (Pelvis), die Rückenknochen (Spine) und die Beine (Legs), am letzten Spine (3 Rückenwirbel bei Sophie) hängen die Schultern (Clavicle), sowie Nacken (Neck) und Kopf (Head). Das Biped besitzt 4 IK-Solver (an je beiden Händen und Füßen). Sie kontrollieren die Winkel und die Orientierung der Gelenke an Ober- und Unterarm (Fore-/UpperArm) bzw. Ober- und Unterschenkel (Thigh/Calf). An jeder Hand gibt es fünf Finger mit je 3 Glieder. Da an den Füßen die Zehen nicht sichtbar sind, haben wir einen einzelnen „Zeh" hinzugefügt, um den Knick beim Auftreten der Schuhe zu animieren.

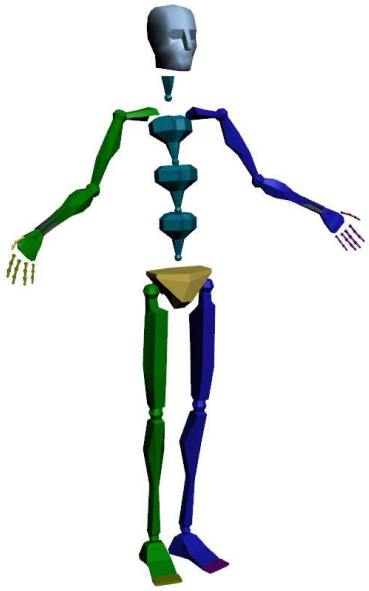


Abbildung 38: Sophies Biped IK-System

Ein besonders wichtiges Feature des Bipeds in 3ds Max sind die sogenannten Twists. Bei der Rotation des Handgelenks entstehen aufgrund der Entfernungen der Bones unschöne Deformationen bei der Drehung. „Twists" sind Bones, die sich entlang eines Knochens aufteilen und nacheinander der Drehung eines Gelenks annähern. Mithilfe dieser zusätzlichen „Drehknochen" verdrillt sich die umgebende Haut nicht um eine Achse, sondern verteilt sich (bei entsprechender Gewichtung) gleichmäßig z.B. über den ganzen Unterarm. Wir haben uns entschieden für die beiden Unterarme je 3 Twists zu verwenden, damit Sophie gleichmäßig ihre Handgelenke drehen kann. Die Unterarmknochen blieben Bestandteil des Skeletts, wurden aber bei der Gewichtung der Scheitelpunkte (siehe Seite 73) nicht berücksichtigt.

Das Biped bietet eine Symmetrierung von Elementen an, sodass die Posen der beiden Seiten des Skeletts gespiegelt werden konnten. Die Positionierung der Glieder erfolgte zusammen mit der Neuausrichtung der Geometrie der Arme. Da sich aufgrund von Verzerrungen die Schultergelenke nicht sauber drehen lassen konnten, musste Sophies Mesh und Skelett in eine lockere, offene Haltung übergehen (siehe oben).

2.7.2.2 Bones

Wir haben normale Bones für die Animierung des Gesichts und der Haare von Sophie verwendet. Daher ist der Begriff „Bones" etwas irreführend, rein technisch sind unsere sogenannten „Facial Bones" natürlich jedoch mit denen des Bipeds identisch. Um die Augen und Augenbrauen, ihre Lider, ihre Wangen, ihren Mund und ihre Haare zu animieren, verwendeten wir keine Morphings, sondern eben Bones, um die Bewegung von Gesichtsmuskel zu animieren. Die einzelnen „Knochen" wurden daher sorgfältig im Gesicht positioniert und ausgerichtet. Verbunden wurden die Bones mit dem „Head"-Element des Bipeds. Die Symmetrierung der Gesichtshälften erfolgte mit einer Kopie der Bones auf die gegenüberliegende Seite und der manuellen Eingabe negierte Koordinaten entlang der X-Achse.

Abbildung 39: Bones in Sophies Gesicht

2.7.3 Skinning/Rigging

Damit das „virtuelle Skelett“ eines Charakters auch imstande ist, einen Körper zu bewegen und zu deformieren, benötigt er Informationen darüber, inwiefern seine Scheitelpunkte dem Skelett folgen sollen. Es muss daher ein „Table“ angelegt werden, der bestimmt, welcher Scheitelpunkt der Geometrie mit welchem Knochen des Skeletts verbunden sein soll. Eine Gewichtung der einzelnen Punkte bestimmt dann, wie stark diese der Bewegung der Bones folgen sollen. Das beeinflusst schließlich die Art und Weise der Deformierung des ganzen Körpers, die auch erheblichen Einfluss auf die Glaubwürdigkeit eines Charakters hat. Diesen Prozess nennt man „Skinning“ bzw. „Rigging“ und wird natürlich aufwendiger, je mehr Scheitelpunkte ein Körper hat. Für das Skinning gibt es in 3ds Max den „Skin“-Modifikator und dessen (interaktive) Werkzeuge, mit denen der „Weight Table“ editiert werden kann.

Damit keine unnötige Arbeit bei diesem sehr arbeitsintensiven Prozess entsteht, sollte festgelegt werden, welche Elemente des Skeletts für die Deformierung überhaupt in Frage kommen. Nicht alle Bones müssen automatisch auf die Liste des Weight Tables gesetzt werden. Die Anzahl der Bones, denen ein Körper folgen kann, sollte so gering wie möglich gehalten werden, damit später im Spiel keine unnötigen Performance-Einbußen entstehen. Zudem sind viele Elemente nur Helfer, die beim Export und in der Spiele-Engine keine Rolle spielen.

Da Kopf und Körper getrennte Objekte waren, der Charakter aber bei der Animation nach „einem Objekt“ auszusehen hatte, war es notwendig, dass beide Skin-Modifikatoren die Hierarchie bis zum Ursprungspunkt des Bipeds „kannten“ (Sophie Biped) und diese von einem gleichen Anfangspunkt für das Skelett ausgehen konnten. Nicht alle Bones wurden für die Scheitelpunkte von Kopf und Körper gewichtet. Sie waren aber Bestandteil des Modifiaktors, damit die Hierarchie der Knochen und das Objekt korrekt positioniert werden konnte.

Wir haben eine Tabelle erstellt, die festlegt, welche Elemente des Skeletts für das „Skinning“ von Kopf und Körper in Frage kamen. Sogenannte „Childs“ bzw. „Endeffektoren“, die bei der Animation nur zur Ausrichtung eines übergeordneten Bone dienen (z.B. „Sophie Bone R EyeLidTop02“) und Bones, die in den Skin-Modifikatoren beider Objekten nicht gelistet sind, werden bei der Zählung (siehe unten) nicht mitberücksichtigt.

	Elementname	Biped/Bone	Skinning: Kopf	Skinning: Körper
1	Sophie Biped	Biped	ja	ja
2	+- Sophie Biped Pelvis	Biped	ja	ja
3	+- Sophie Biped Spine	Biped	ja	ja
4	+- Sophie Biped Spine1	Biped	ja	ja
5	+- Sophie Biped Spine2	Biped	ja	ja
6	+- Sophie Biped Neck	Biped	ja	ja
7	+- Sophie Biped Head	Biped	ja	ja
8	+- Sophie Bone R EyeBall01	Bone	ja	nein
9	+- Sophie Bone R EyeBall02	Bone	nein	nein
10	+- Sophie Bone R EyeLidBottom01	Bone	ja	nein
11	+- Sophie Bone R EyeLidBottom02	Bone	nein	nein
12	+- Sophie Bone R EyeLidTop01	Bone	ja	nein
13	+- Sophie Bone R EyeLidTop02	Bone	nein	nein
14	+- Sophie Bone L EyeBrow01	Bone	ja	nein
15	+- Sophie Bone L EyeBrow02	Bone	nein	nein
16	+- Sophie Bone R EyeBrow01	Bone	ja	nein
17	+- Sophie Bone R EyeBrow02	Bone	nein	nein
18	+- Sophie Bone L Wangs01	Bone	ja	nein
19	+- Sophie Bone L Wangs02	Bone	nein	nein
20	+- Sophie Bone R Wangs01	Bone	ja	nein
21	+- Sophie Bone R Wangs02	Bone	nein	nein
22	+- Sophie Bone L UpperLip01	Bone	ja	nein
23	+- Sophie Bone L UpperLip02	Bone	nein	nein
24	+- Sophie Bone R MouthWinkle01	Bone	ja	nein
25	+- Sophie Bone R MouthWinkle02	Bone	nein	nein
26	+- Sophie Bone HairStrands01	Bone	ja	nein
27	+- Sophie Bone HairStrands02	Bone	nein	nein
28	+- Sophie Bone L MouthWinkle01	Bone	ja	nein
29	+- Sophie Bone L MouthWinkle02	Bone	nein	nein
30	+- Sophie Bone BottomLip01	Bone	ja	nein
31	+- Sophie Bone BottomLip02	Bone	nein	nein
32	+- Sophie Bone L EyeLidBottom01	Bone	ja	nein
33	+- Sophie Bone L EyeLidBottom02	Bone	nein	nein
34	+- Sophie Bone L EyeLidTop01	Bone	ja	nein
35	+- Sophie Bone L EyeLidTop02	Bone	nein	nein
36	+- Sophie Bone HairPony01	Bone	ja	nein
37	+- Sophie Bone HairPony02	Bone	nein	nein

38	+- Sophie Bone R EyeBall01	Bone	ja	nein
39	+- Sophie Bone R EyeBall02	Bone	nein	nein
40	+- Sophie Biped L Clavicle	Biped	ja	ja
41	+- Sophie Biped L UpperArm	Biped	nein	ja
42	+- Sophie Biped L Forearm	Biped	nein	ja
43	+- Sophie Biped L Hand	Biped	nein	ja
44	+- Sophie Biped L Finger0	Biped	nein	ja
45	+- Sophie Biped L Finger01	Biped	nein	ja
46	+- Sophie Biped L Finger02	Biped	nein	ja
47	+- Sophie Biped L Finger1	Biped	nein	ja
48	+- Sophie Biped L Finger11	Biped	nein	ja
49	+- Sophie Biped L Finger12	Biped	nein	ja
50	+- Sophie Biped L Finger2	Biped	nein	ja
51	+- Sophie Biped L Finger21	Biped	nein	ja
52	+- Sophie Biped L Finger22	Biped	nein	ja
53	+- Sophie Biped L Finger3	Biped	nein	ja
54	+- Sophie Biped L Finger31	Biped	nein	ja
55	+- Sophie Biped L Finger32	Biped	nein	ja
56	+- Sophie Biped L Finger4	Biped	nein	ja
57	+- Sophie Biped L Finger41	Biped	nein	ja
58	+- Sophie Biped L Finger42	Biped	nein	ja
59	+- Sophie Biped L ForeTwist	Biped	nein	ja
60	+- Sophie Biped L ForeTwist1	Biped	nein	ja
61	+- Sophie Biped L ForeTwist2	Biped	nein	ja
62	+- Sophie Biped R Clavicle	Biped	ja	ja
63	+- Sophie Biped R UpperArm	Biped	nein	ja
64	+- Sophie Biped R Forearm	Biped	nein	ja
65	+- Sophie Biped R Hand	Biped	nein	ja
66	+- Sophie Biped R Finger0	Biped	nein	ja
67	+- Sophie Biped R Finger01	Biped	nein	ja
68	+- Sophie Biped R Finger02	Biped	nein	ja
69	+- Sophie Biped R Finger1	Biped	nein	ja
70	+- Sophie Biped R Finger11	Biped	nein	ja
71	+- Sophie Biped R Finger12	Biped	nein	ja
72	+- Sophie Biped R Finger2	Biped	nein	ja
73	+- Sophie Biped R Finger21	Biped	nein	ja
74	+- Sophie Biped R Finger22	Biped	nein	ja
75	+- Sophie Biped R Finger3	Biped	nein	ja
76	+- Sophie Biped R Finger31	Biped	nein	ja
77	+- Sophie Biped R Finger32	Biped	nein	ja
78	+- Sophie Biped R Finger4	Biped	nein	ja
79	+- Sophie Biped R Finger41	Biped	nein	ja
80	+- Sophie Biped R Finger42	Biped	nein	ja
81	+- Sophie Biped R ForeTwist	Biped	nein	ja

82	+- Sophie Biped R ForeTwist1	Biped	nein	ja
83	+- Sophie Biped R ForeTwist2	Biped	nein	ja
84	+- Sophie Biped L Thigh	Biped	nein	ja
85	+- Sophie Biped L Calf	Biped	nein	ja
86	+- Sophie Biped L Foot	Biped	nein	ja
87	+- Sophie Biped L Toe0	Biped	nein	ja
88	+- Sophie Biped R Thigh	Biped	nein	ja
89	+- Sophie Biped R Calf	Biped	nein	ja
90	+- Sophie Biped R Foot	Biped	nein	ja
91	+- Sophie Biped R Toe0	Biped	nein	ja
	Anzahl gelisteter Bones:		**25**	**59**

Tabelle 11: Verwendung des Skinning

Die maximale Anzahl an Bones pro Objekt, die wegen der begrenzten Anzahl von Shader-Registern einer Grafikkarte festgelegt ist (siehe Seite 27, Kapitel 2.1.12), lag bei etwa 70 Stück. Aus Tabelle 11 wird ersichtlich, dass diese Anzahl weder bei Kopf, noch bei Körper überschritten wurde.

Da die OGRE-Engine nicht mehr als 4 Bone-Elemente pro Scheitelpunkt unterstützt, musste der Weight Table von uns manuell überprüft und korrigiert werden. Besaß ein Punkt mehr als 4 Einträge, wurden nacheinander die Gewichte der Bones mit den geringsten Werten auf 0 zurückgesetzt, sodass diese keinen Einfluss mehr auf seine Bewegung nahmen.

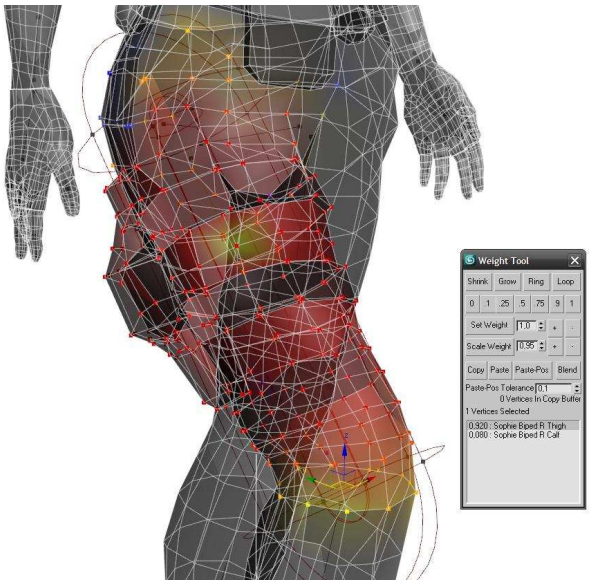


Abbildung 40: Rigging mit Envelopes

Envelopes

Hilfreich bei der Vergabe von Gewichtungen, sind „Envelopes“. Die zylindrischen „Gewichtungshüllen“ mit kugelförmigen Enden und ihrem kontrollierbaren „Grenzschalen“, die festlegen, wie sehr die Scheitelpunkte innerhalb dieses Einflussbereichs um einen Bone herum folgen sollen. Bei der Überschneidung zweier Hüllen entstehen verteilte Gewichtungen, die durch die Größe der minimalen und maximalen Grenzhülle der Envelopes bestimmt werden. Mithilfe einer farbigen Anzeige im Viewport in 3ds Max können somit große Bereiche des Körpers einfach ausgewählt und zugweiesen werden. Die Envelopes helfen jedoch nur bei der groben Vergabe von Gewichtungen und sind keine Hilfe bei schwierigen Stellen.

Paint Weights

Das Paint-Weight-Tool des Skin-Modifikators ist ein etwas präziseres Werkzeug für die Vergabe von Gewichtungen. Mithilfe eines Pinsels können hier interaktiv die Gewichtungen mit einem bestimmten Wert und durch eine festgelegt Größe „aufgemalt“ werden. Diese intuitive und einfache Methode ist jedoch trotzdem häufig noch zu ungenau, um sämtliche Scheitelpunkte mit den richtigen Gewichtungen zu versehen.

Manuelle Vergabe von Gewichten

Um für jeden Scheitelpunkt die zugehörigen Elementen und Werte zu finden, hilft meist nur die manuelle Vergabe von Gewichten. Pro Punkt wird für den ausgewählten Knochen eine Gewichtung bestimmt, die im Weight Table

gespeichert wird. Da es sich bei Sophie um eine ungeglättete Low-Poly-Figur handelt, hielt sich der Aufwand hierbei einigermaßen in Grenzen. Ein hilfreiches Tool war das Weight Tool des Skin-Modifikators, das uns mit zahlreichen Selektionstechniken bei der Auswahl von Scheitelpunkten helfen konnte.

Spiegelung von Gewichten

Der „Mirror-Mode“ des Skin-Modifikators ermöglicht es die Envelopes und Gewichtungen einer Seite auf eine andere zu spiegeln. Mithilfe eines Thresholds wird auf der gegenüberliegenden Seite der eingestellten Spiegelebene (an der X-Achse) überprüft, ob ein symmetrischer Scheitelpunkt oder Bone existiert. Findet die Funktion einen entsprechenden „Partner“ können die Werte kopiert werden. Da Sophie an einigen Stellen (Bauch, Oberschenkel, Hose) keine Symmetrie besaß, konnte hier auch keine Spiegelung erfolgen, sodass beide Seiten einzeln gewichtet werden mussten.

Kopf & Körper

Da Kopf und Körper von Sophie getrennte Objekte waren, damit auch getrennte Skin-Modifikatoren besaßen und separat gewichtet werden mussten, kam das Problem auf, dass bei Sophies Arm- oder Kopfbewegungen Lücken zwischen beiden Objekten entstanden. Die Lösung lag darin die Scheitelpunkte der Ränder beider Objekte, an der gleichen Position zu halten. Dies geschah mithilfe identischer Gewichtungen für jeden Punkt und Bones entlang der dieser Grenzen.

Test Animationen

Häufig war es notwendig die Gelenke des Biped oder die der Bones neu zu positionieren und neu auszurichten, da diese nicht von Anfang an optimal an Sophies Körper angepasst waren. Zur Überprüfung und Korrektur der Gewichtungen, war es notwendig Sophie in einigen Grundzügen zu animieren. Hierbei ließen wir Sophie einige akrobatische Posen, wie auch Standard-Laufbewegungen durchführen. Mit extremen Posituren war es uns möglich die Gewichtungen auf spezielle, grenzwertige Körperhaltungen hin zu testen und zu optimieren.

2.7.4 Skeleton Animationen

Die Animierung von Sophie geschieht zwar zuletzt innerhalb der eigentlich 3D-Echtzeitanwendung, basiert jedoch auf den Animationen, die in der (Animations-)Software 3ds Max erstellt werden. Das gesamte Verfahren zur interaktiven Animation wird in Kapitel 3 genauer erklärt. Für den Animator gilt zunächst zu wissen, dass nicht alle Bones des Skeletons zur Laufzeit animiert werden. Die Bones werden in der Anwendung je nach Bedarf ein- und ausgeschaltet, sprich die in 3ds Max erstelle Keyframe-Animation wird entweder beibehalten oder durch eine dynamische Bewegung ersetzt. Für eine glaubhafte Animierung, war also eine Basis erforderlich, damit in der Anwendung zusätzlich dynamische Bewegungen hinzugefügt werden konnten. Wir entschieden uns als erste Grundlage, einen sogenannten „Walk Cycle“ zu erstellen, die wiederholbare Animation einer Laufbewegung, sowie ihre zugehörige Anfangs- und Endphase. Zu den Animationen, die später nicht abgeschaltet werden, gehören z.B. die Arm- und Schulterbewegungen oder das Mitschwingen der Haare (Pferdeschwanz & Strähnen).

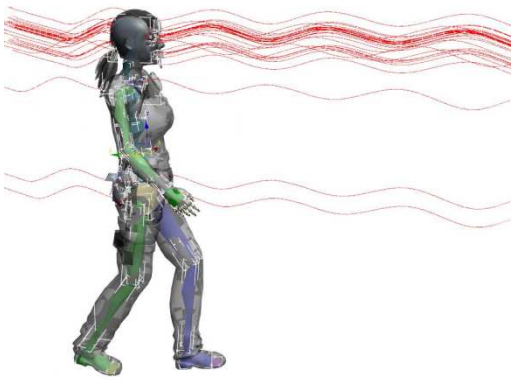


Abbildung 41: Wellenbewegung der Bones

Footsteps

Ein nützliches Werkzeug, bei der Animation von Geh- und Laufbewegungen in 3ds Max, sind die „Footsteps“ des Bipeds, die sich im Footstep-Modus positionieren lassen. Diese verschiebbaren Ankerpunkte legen fest, wie und in welche Richtung das Skelett zu laufen hat. Ähnlich wie Fußabdrücke im Schnee, zeigen die Footsteps an, wohin ein Charakter läuft und wie er sich bewegt. Wir haben dieses Tool, trotz einiger Komplikationen und Probleme in der Handhabung, als sehr hilfreich empfunden und für die Erstellung von Laufanimationen verwendet.

Damit nach einer Gehbewegung zusätzlich Animationen eingefügt werden können, muss der Footstep-Modus nach seiner Verwendung wieder deaktiviert und in eine Keyframe-Animation verwandelt werden. Dies bedeutet, dass die „Fußabdrücke“ und die Möglichkeit diese zu positionieren und auszurichten verschwindet, und nur noch mithilfe von Keyframes animiert werden kann.

In Place Mode

Der In Place Modus des Bipeds verhindert, dass ein Skelett quasi „davonrennt“. Statt es auf dem Boden entlang laufen zu lassen, bewirkt die In Place Funktion, dass das Biped sich stets auf der Stelle bewegt, während der Boden unter dessen Füßen „wegrollt“. Dieser Modus muss aktiviert sein, wenn Bewegungen für den Export animiert werden. So wird sichergestellt, dass nur die Höhe und nicht die X- und Y-Koordinaten des Skelett-Ursprungs verändert werden. Damit lässt sich die Figur in der Anwendung besser positionieren und einfacher steuern.

Motion Mixer

Der Motion Mixer in 3ds Max hilft die erstellten Animationen miteinander zu kombinieren und (in unserem Fall) in eine logische Reihenfolge zu bringen. Wir benutzen den Motion Mixer als Übersichtswerkzeug bei der Erstellung des Walk-Cycles, besonders um Anfangs- und Endbild einer Keyframe-Animation zu vergleichen.

Animierung

Die Liste an Animationen eines Charakters ist davon abhängig, was für das Spiel später benötigt wird. Zum Standardrepertoire menschlicher Animationen gehören meist Stehen, Gehen, Laufen, Blinzeln, Atmen, diverse Mundbewegungen, usw. Da diese Bewegungen für fast alle Menschen oder Charaktere eines Spiels zutrifft, werden Animationen oft getrennt behandelt (sprich allgemein animiert), sodass sie auf sämtliche Charaktere angewendet werden können. Wie bei uns liegt der Fokus dieser Arbeit jedoch sehr oft nur auf dem Hauptcharakter eines Spiels. Die Möglichkeiten solche Bewegungen auszubauen und zu variieren, gehen dabei ins Unendliche. Das Prinzip bleibt dasselbe: Die Künstler (Animatoren) animieren Bewegungen und Posen, die für das Spiel voraussichtlich benötigt werden. Diese werden dann im Spiel auf die Charaktere (oder Kreaturen) angewendet.

Abbildung 42: Laufanimation in 10-Frames-Schritten

Man unterscheidet zwischen wiederholbaren Animationen und Übergangsanimationen. Da sich der menschliche Körper ständig in „Bewegung“ befindet und leblos wirkt, wenn er statisch ist, sollten häufige Animationen wiederholt abspiel- und kombinierbar sein. Atembewegungen, Blinzeln, Kopfbewegungen und die Fortbewegung sind dabei gute Beispiele. Animationen zwischen zwei Loops, bezeichnet man als „Transitions“. Loops sind sehr schwer zu erstellen, da Anfang- und End-Frame einer solchen Animation identisch sein und trotzdem einen glaubwürdige Bewegung ausführen müssen.

Statt zahllose Animationen zu fertigen, die zwar für immer mehr Glaubwürdigkeit, aber auch sehr viel Aufwand sorgen und in der Anwendung viele Skelettreihen durch die Interaktion „überschrieben“ bzw. neuanimiert werden, entschlossen wir uns weiterhin auf die dynamische Animierung zu konzentrieren. Eine kleine Liste vorgefertigter

Keyframe-Animationen war die Basis, um diese Technik zu testen und zu verbessern. Die Animationen wurden bis zur Abgabe dieser Ausarbeitung nach und nach ergänzt.

Keyframes & Zeit

Der Zeitindex in 3ds Max wird beim Export in Frames angegeben. Die Animierung erfolgt mit 30 Bildern pro Sekunde. Da eine 3D-Anwendung je nach Szene eine schwankende FPS-Rate besitzt und in einer Echtzeitanwendung Frames unterschiedlich schnell berechnet werden, müssen Animationen innerhalb der Anwendung einem unabhängigen Zeitindex gehorchen. Daher wird dieser beim Export automatisch in die normale Zeitrechnung (Millisekunden, Sekunden, etc.) umgewandelt und beim Abspielen einer Animation durch die Systemzeit berechnet.

Animationsliste

Die Liste von Skeleton Animationen zum Zeitpunkt der Abgabe:

Nr.	Animation	Start	End	Dauer	Loop
1	Sophie_Walk_Start	0	45	45	nein
2	Sophie_Walk_Cycle	45	73	28	ja
3	Sophie_Walk_End	73	110	37	nein
4	Sophie_Eye_Blinked	110	115	5	ja
5	Sophie_Breath_Calm	115	150	35	ja
6	Sophie_Look_around	150	250	100	nein

Tabelle 12: List von Skeleton Animationen

2.7.5 Vertex Animationen

„Vertex Animationen“ oder allgemein auch „Morph Animationen“ sind eine Alternative zur bisher angewandten Skeleton Animation. Hier werden die Scheitelpunkte nicht über ein oder mehrere Bones animiert, sondern direkt an ein Ziel, zum sogenannten „Morph Target“ hin, verschoben. Der Vorteil von Vertex Animationen ist die freie Animierbarkeit aller Scheitelpunkte. Mithilfe von Morphings lassen sich Körper frei deformieren, ohne dabei auf die Struktur einer IK angewiesen zu sein. Der Nachteil freier Vertex Animationen ist die lange Berechnungszeit bei einer großen Anzahl von Scheitelpunkten und der Speicherverbrauch durch die Verwendung vieler Morph Targets.

In OGRE gibt es zwei unterschiedliche Typen:

- Bei „Morph Animationen“ werden die Bewegungen zwischen dem Ausgangsmodell und nur einem „Target“ linear über Keyframes animiert. Das „Blending“ zwischen zwei Keyframes bzw. die Bewegung der Scheitelpunkte werden ohne Gewichtungen interpoliert.
- Mithilfe von „Pose Animationen“ können mehrere Morph Targets gleichzeitig ineinander geblendet werden. Auf der Basis eines Offsets und mithilfe gewichteter Vertex Daten lassen sich z.B. die unterschiedlichen Animationen im Gesicht oder die Deformierung von Muskeln miteinander kombinieren, auch wenn sich das Modell noch zusätzlich mithilfe eines IK-System bewegt.

Selbst bei dem immensen Einsatz von Bones (in unserem Fall sogar in Sophies Gesicht), können Vertex Animationen sehr hilfreich sein, um Bewegungen und Deformierungen zu schaffen, die nicht mehr mithilfe einer IK animiert werden können. Um Animationen mit mehreren Morph Targets und gemeinsam mit der Skeleton Animation verwenden zu können, haben wir uns für die „Pose Animationen“ entschieden und sie einfach als Ergänzung zur IK betrachtet. Da Vertex Animationen von der Modellierung ihrer Targets abhängig sind und diese nicht dynamisch berechnet werden können, waren sie von geringerem Interesse für uns. Es ist jedoch möglich die Gewichtung, sprich das Blending zu einem Target hin, dynamisch ohne eine Keyframe-Animation zu steuern. Wir haben uns diese

Option offen gehalten und eine Reihe von Pose Animationen erstellt, die im Laufe der (dynamischen) Animierung zum Einsatz kommen können. In 3ds Max werden Vertex Animationen mithilfe des „Morpher"-Modifikators realisiert.

Morpher

Damit Morphing Targets zugewiesen werden können, müssen sie dieselbe Anzahl von Scheitelpunkten besitzen wie das Ursprungsobjekt. Aus diesem Grund wird in der Regel jedes Morphing Target vom Originalobjekt kopiert, und für die entsprechende Zielpose modelliert. Bei der Modellierung dürfen dabei keine Scheitelpunkte hinzugefügt oder gelöscht werden. Besitzt ein Objekt eine andere Anzahl Scheitelpunkte, als das Originalobjekt, kann dieses nicht in die Liste des Morpher-Modifikators aufgenommen werden. Der Aufwand der Modellierung von Morphing Targets steigt natürlich, je mehr Zielposen es gibt.

Jedes Morphing Target erhält nach seiner „Neu-Modellierung" den Namen seiner Funktion (z.B. „Expression Angry"). Die einzelnen Objekte werden dann in die Kanäle des Morpher-Modifikators auf dem Originalobjekt geladen. Die Morphings können nun gewichtet (0-100%) und miteinander kombiniert werden. Morph Animationen müssen nicht mit Keyframes animiert sein, sondern stehen bereits jetzt (nach dem Export) in OGRE zur Verfügung.

Abbildung 43: Morphing Targets für den Kopf

Animationsliste

Kopf	Körper
Expression Angry	Muscle R Calf
Expression Sad	Muscle L Calf
Expression Suprise	Muscle R Thigh
Expression Not Amused	Muscle L Thigh
Expression Happy	Muscle R Forearm
Expression Smile	Muscle L Forearm
Expression Giggling	Muscle R UpperArm
Expression Mouth Open	Muscle L UpperArm
Expression Closed Eyes	
Expression Clever	

Tabelle 13: Morph Targets für Kopf und Körper

Wie in der Liste zu sehen ist, haben wir Kopf und Körper mit Pose Animationen versehen. Das Kopf-Objekt erhielt dabei einige emotionale Ausdrücke (Expressions), die sich nicht mit der existieren IK realisieren ließen (ausgenommen das Blinzeln der Augen). Beim Körper konzentrierten wir uns auf die Bewegung kontaktierender Muskeln (Muscle), die sich verdicken, wenn ein Arm oder ein Bein gestaucht wird.

In 3ds Max gibt es zur automatisierten Anwendung von Morphing Targets den sogenannten „Skin-Morpher"-Modifikator, der es erlaubt die Gewichtung der Morphings mit dem Winkel zwischen zwei Bones zu verknüpfen. Somit können z.B. bei jeder Armbewegung automatisch auch die Bewegung ihrer Muskeln animiert werden. Diese Funktion stand in OGRE leider nicht zu Verfügung.

2.8 Export/Import nach OGRE

2.8.1 Allgemeines

Um dreidimensionale Objekte aus einer 3D-Anwendung in ein lesbares Format für die Engine eines Spiels zu bringen, müssen die Daten mithilfe eines Programms oder Plugins konvertiert bzw. exportiert werden. Dies bedeutet für uns, dass von 3ds Max (oder einer entsprechenden 3D-Software) Sophies Geometrie und Texturen für die OGRE-Engine bereitgestellt werden müssen. Dafür benötigt man eines der frei verfügbaren OGRE-Exporter-Plugins. Wichtig war uns natürlich, dass dieser zusammen mit den Geometriedaten auch Texturkoordinaten, Materialien und auch die erstellten Animationen exportiert. Wir haben daher einige Exporter installiert und ausprobiert. Zur Überprüfung der Daten benötigten wir darüberhinaus noch einen Viewer, eine zusätzliche Applikationen, welche die exportierte Sophie mithilfe der OGRE-Engine darstellt. Unsere Applikation („OgreIK“) gehörte ebenfalls dazu und wird in Kapitel 3 näher beschrieben.

2.8.2 Vorbereitungen

Im Laufe der Charaktererstellung kann sich in einer Szene in 3ds Max allerhand Objekte ansammeln, die für die Verwendung in unserer Anwendung oder im Spiel nicht mehr zu gebrauchen sind. Diverse Lichter, der Boden, Helfer und Modellierungsvorlagen sollten aus der Szene entfernt oder versteckt werden, sodass der Exporter diese nicht mitberücksichtigt.

2.8.3 Der LexiExporter

Zu den schnellen, einfacheren Exporten zählt der LexiExporter, der neben der Geometrie auch Materialien exportiert. Die Fähigkeiten dieses Exporters stießen jedoch an ihre Grenzen, als dieser selbst bei unanimierten Bones überflüssige NodeTracks[17] vergab, welche für die Performance und für die spätere Konfiguration bei der dynamischen Animierung des Skeletts nicht zu gebrauchen waren.

2.8.4 Der OGRE-Exporter

Der OGRE-Exporter war für 3ds Max 2008 leider nicht initialisierbar („Failed to initalize“, keine zulässige Win32-Anwendung). Das Problem ließ den Exporter weder in der 32- noch in der 64-Bit Version des Programms starten. Auf Anfrage riet man uns auf 3ds Max 2009 umzusteigen, was uns allerdings gegen Ende des Projekts nicht mehr sinnvoll schien.

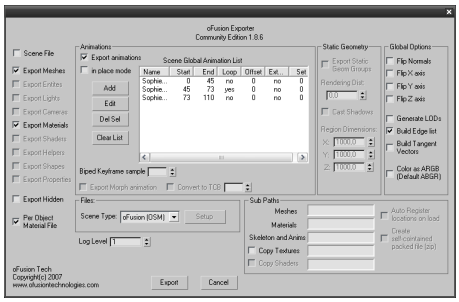


Abbildung 44: oFusion Exporter für 3ds Max

[17] NodeTracks sind Sequenzen, die mithilfe von Keyframes animiert werden. Durch die einzelnen Nodes (Bones) kann OGRE schließlich Transformationshierarchien bauen.

2.8.5 LexiView

Der LexiViewer ist ein praktisches Werkzeug, um den exportierten Charakter mithilfe der OGRE-Engine darzustellen. Das Programm ist in der Lage Geometrie und Texturen anzuzeigen und sämtliche Animationen abzuspielen, die in 3ds Max indiziert und exportiert worden sind. Zusätzlich bietet es die Option einzelne Animationen im Loop wiederholt abzuspielen. Der Viewer war eine große Erleichterung bei der Überprüfung der exportierten Daten. Er half uns zahlreiche Probleme zu erkennen und im späteren Verlauf zu vermeiden.

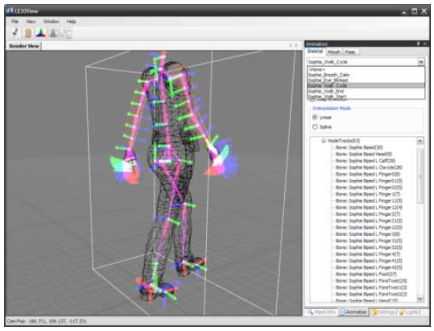


Abbildung 45: LexiViewer mit Skelett-Animation

Nachteil an LexiView war, dass es separate Vertex- und Pixel-Shader-Programme nicht ausführen konnte. Dies bedeutet, dass Specular- und Normal-Bump Map in LexiView nicht angezeigt werden konnten. Die Objekte mussten ohne Material exportiert werden, um Überprüfungen der Animation zu machen.

2.8.6 oFusion

Der umfangreichste und wichtigste Exporter, der uns zur Verfügung stand und schließlich auch zur Anwendung kam, war der oFusion-Exporter, der das Problem überflüssiger NodeTracks in Skeleton Animationen behob. Mit dem oFusion war es möglich Loops zu definieren, Animationsbereiche übersichtlich festzulegen und Materialien in 3ds Max für OGRE zu definieren. oFusion ersetzt die bestehenden Bitmaps durch eigene Maps und bereitet sie mit zusätzlichen Einstellungen für den Export vor.

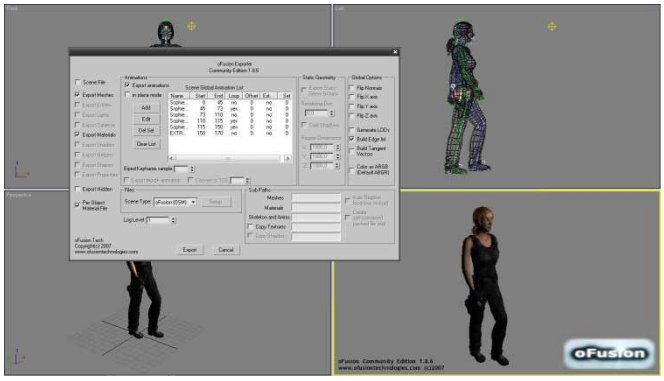


Abbildung 46: Export und Darstellung von Sophie in oFusion

oFusion bietet als Plugin in 3ds Max einen eigenen Viewport-Renderer an, der auf OGRE basiert. Dieser unterstützt die Darstellung von Geometrie, Texturen, Animationen und die Anzeige seiner Vertex- und Pixel-Shader-Programmen. Eigene Vertex- und Pixel-Shader-Programme wurden (aufgrund unterschiedlicher Versionen von OGRE) falsch dargestellt. Die Anzeige unterstützt dabei auch nicht mehr als ein Licht. Weiterer Nachteil des oFusion-Exporters war die Vergabe unnötiger Material-und Licht-Informationen in die Material-Datei, die nachträglich bearbeitet werden musste (siehe Seite 69, Tabelle 10). Die fehlende Unterstützung der 64-Bit Version von 3ds Max 2008 empfanden wir ebenfalls als störend.

2.8.7 Abschluss

Nach dem Export der Daten und der Korrektur aller Material-Dateien, standen das 3D-Modell, die Texturen und Animationen von Sophie anderen Anwendungen und schließlich für „Die Stadt Noah“ zur Verfügung. Spezielle Anpassungen ihrer IK und Animierung folgen in Kapitel 3. Sophies „Entstehung“ war somit vollendet.

2.9 Fazit

Ebenso wie Computeranimationen befinden sich aktuelle Spiele, unserer Ansicht nach auf dem Weg zum Uncanny Valley. Zwar wirken 3D-Spielfiguren immer glaubwürdiger und dank vielfältiger Möglichkeiten und Techniken immer realer, doch erst eine hundertprozentige Verwechselbarkeit zwischen einem echtem und einem virtuellen Menschen, verhindert ein befremdliches Gefühl beim Ansehen. Wir haben uns einige Techniken angesehen, die manche dieser Probleme in Zukunft womöglich lösen werden.

2.9.1 Mögliche Lösungen zur Verbesserung des Realismus

Es gibt verschiedene Ansätze das Uncanny Valley zu umgehen und die zwar sehr geringe, aber entscheidende Differenz zwischen Realität und Virtualität zu überwinden oder ganz zu meiden. Wie am Anfang erwähnt, umgehen viele Animationsstudios (Pixar, Sony, Dreamworks) das Uncanny Valley geschickt durch comichafte (meist verniedlichte, tierähnliche) Charaktere, die vom Zuseher erst gar nicht mit Menschen verglichen werden.

Damit aber auch virtuelle Menschen nicht mehr von richtigen unterschieden werden können, müssen noch zahlreiche technische Probleme gelöst werden. Einige dieser Lösungen sind unverhältnismäßig zeit- und arbeitsintensiv, oder bedürfen weiterer Entwicklungen, die auch wiederum mehr Rechenleistung erforderlich machen. Dies gilt für gerenderte Computeranimationen und ganz besonders für Echtzeitanwendungen. Wir möchten kurz einige dieser Faktoren zusammenfassen, die wohl in Zukunft einige Verbesserungen erfahren dürften.

2.9.1.1 Raytracing zur Beleuchtung und Animation

Realistische Beleuchtungsmodelle sind wohl eine Grundvoraussetzung für mehr Realismus und Glaubwürdigkeit virtueller Welten. Viele 3D-Echtzeit-Umgebungen wirken trotz immer mehr Leistung heutiger Computer noch immer nicht realistisch. Die lange Zeit verwendeten Shadow Maps erzeugen teilweise heute noch Artefakte und Flackern bei animierten Objekten oder veränderlichen Lichtern. Hochaufgelöste Bilder mit hohem Kontrastumfang (HDRIs) werden schon jetzt zur Beleuchtung von Szenen in 3D-Echtzeitanwendungen eingesetzt, wobei die Berechnung von globalen Beleuchtungsmodellen noch in den Kinderschuhen steckt.

Mehr Rechenleistung ist also nötig, um Schatten, Lichtbrechungen, Spiegelungen und indirekte Beleuchtungseffekte photorealistisch und in Echtzeit darzustellen. Interessant beim Raytracing ist auch die genaue Kollisionberechnung zwischen der Geometrie und der polygongenauen Treffererkennung, die bei Simulationen und interaktiven Animationen für mehr Genauigkeit sorgt **Es ist eine ungültige Quelle angegeben.**. Mithilfe von Bewegungszersetzung (Motion Decomposition) können durch Raytracing-Algorithmen auch dynamisch animierte Modelle genauer berechnet und dargestellt werden **Es ist eine ungültige Quelle angegeben.**.

2.9.1.2 Physikalische Deformation der Haut

Die Haut und die Kleidung von Charakteren wird heutzutage mithilfe von IK-Systemen und der Gewichtung einzelner Vertices deformiert. Doch um völlig realistische Bewegungsabläufe zu schaffen, muss man ihre ganze physische Struktur und Beschaffenheit berücksichtigen. Muskeln, Sehnen und Knochen unter der Haut verändern dessen Aussehen, ebenso wie kleine Falten, Irritationen oder Fremdeinwirkung von außen. Um Bewegungsabläufe realistisch wirken zu lassen, müssen diese nicht mithilfe eines virtuellen Knochensystems deformiert, sondern physi-

kalisch korrekt **simuliert** werden (ähnlich wie Kleidung oder Haare). Solche Simulationen sind sehr aufwendig und werden noch selten in Computeranimationen angewendet[18].

2.9.1.3 Volumetrische Körper

Seit ihrer Entstehung konzentriert sich die Computergrafik weitestgehend auf die Darstellung von Oberflächen, die im Computer zwar dreidimensional modelliert, aber nur zweidimensional texturiert werden. Da es jedoch in der Wirklichkeit unter einer lichtdurchlässigen Oberfläche feste Strukturen gibt (z.B. Muskeln, Adern und Knochen unter der Haut), müssten sämtliche Körper, die ein festes Volumen aufweisen auch mit einem solchen modelliert werden. Derzeit werden volumetrische Körper in ihrem Innern nicht modelliert, sondern mithilfe der Oberfläche, um sie herum durch rechenintensive Algorithmen berechnet. Diese Technik nennt sich Sub-Surface-Scattering[19] (kurz SSS) und lässt einfache, lichtdurchlässige Körper wie Wachskerzen, Früchte oder Baumblätter realistisch aussehen. Die Berechnung erfolgt nachdem Licht in eine Materie eingedrungen ist, doch komplexe dreidimensionale Strukturen (wie der Mensch) sehen trotz dieser Technik noch nicht vollkommen realistisch aus, da die Muster unterschiedlicher Dichte, Struktur und Zusammensetzung unter der Haut (Fingerknochen, Knorpel am Ohr) von diesem Verfahren noch nicht berücksichtigt werden.

2.9.1.4 Mikrostrukturen auf Oberflächen

Eine neuere Technik, die aufgrund ihrer aufwendigen Berechnung bislang kaum zum Einsatz kam, aber durch ihre vielen Vorteile immer mehr an Bedeutung gewinnt, ist das sogenannte Microdisplacementmapping. Praktisch eine Erweiterung des gewöhnlichen Displacementmappings (Verschiebung bestehender Scheitelpunkte durch eine Grau-Textur mit Höheninformationen), verändert es nicht nur die Scheitelpunkte einer Oberfläche, sondern erzeugt mithilfe einer hochaufgelösten Textur zusätzlich neue Strukturen (sprich neue Scheitelpunkte und Polygone), die sehr viel schärfer und detaillierter wirken, als gewöhnliche Höhenverschiebungen. Diese Technik wird mit der Zeit wohl das veraltete Bumpmapping ablösen, das nur die Normalen einer Textur verschiebt, anstatt auch wirklich die Geometrie zu verändern. Durch Microdisplacement könnten sehr viel feinere Strukturen und Unebenheiten der menschlichen Haut realistisch dargestellt werden[20].

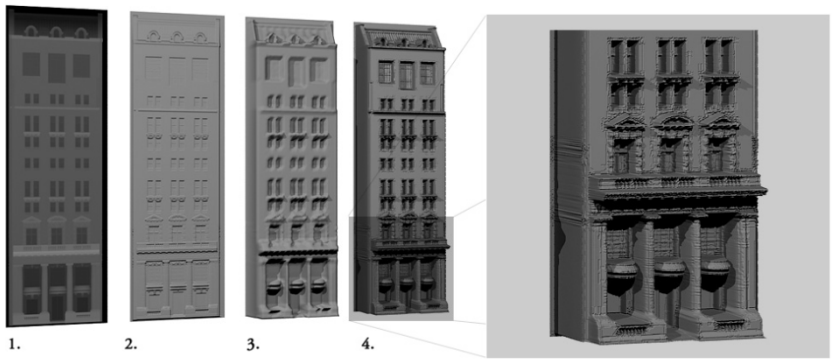


Abbildung 47: Microdisplacement: 3ds Max (vray-Renderer)

[18] In dem US-amerikanischen Spielfilm „10.000 BC“ wurden die Bewegungen der Haut von Mammuts mithilfe solcher Simulationen berechnet. Diese sind aber aufgrund des Fells der Tiere nicht sichtbar. Sie sorgen lediglich für eine glaubwürdigere Bewegung der Haare.

[19] zu deutsch: Volumenstreuung

[20] Anmerkung: Aufgrund begrenzter Ressourcen derzeitiger Computersysteme gibt es für 3D-Anwendungen in Echtzeit (darunter natürlich Computerspiele) derzeit noch keine Möglichkeit Microdisplacement darzustellen. Hier bleiben nur Normal- und Parallaxmapping als Optionen, die im Verlauf dieses Kapitels näher erläutert werden.

1. Textur: Höheninformationen einer Grau-Textur auf eine flache Ebene
2. Klassische Bumpmap/Normalmap: Feine unebene Strukturen auf einer flachen Geometrie
3. Klassische Displacementmap: Höhenverschiebung existierender Scheitelpunkte (Formveränderung)
4. Microdisplacement: Erzeugung neuer, exakter Geometrie mithilfe derselben Map

2.9.1.5 Haare

Trotz zahlreicher Fortschritte im Bereich der Haar- und Fellsimulationen, stellt (vor allem das menschliche) Haar immer noch eine sehr große Herausforderung dar. Die Grafikkünstler sehen sich dabei vor allem mit folgenden Schwierigkeiten konfrontiert:

- Modellierung komplexer Frisuren: z.B. Dauerwelle, Hochsteckfrisuren, Locken, etc.
- Körperbehaarung: feine Haare auf der Haut, Bärte, Brusthaare
- Shader: realistische Schattierung der Haare durch Färbungen, Transluzenz der Haare
- Simulation: z.B. Kollision von Haaren untereinander, sowie mit Körpern, realistisches Verhalten bei Wind und Schwerkraft

Für die genannten Punkte gibt es praktische und theoretische Lösungsansätze[21]. Diese sind aufgrund begrenzter Ressourcen und des noch mangelnden Verständnisses für das genaue Verhalten von Haaren noch nicht realisierbar, werden aber in den nächsten Jahren zu erwarten sein.

2.9.1.6 Motion Capturing & Keyframeanimationen

Realistische Bewegungen von Charakteren sind in Echtzeit- und Computeranimationen ebenfalls ein großes Problem. Zwar lassen sich mithilfe von Motion Capturing und klassischer Keyframe-Animierung zahlreiche Bewegungen realisieren, jedoch stoßen die Animatoren bei sehr kleinen und kurzen, aber auffallend wichtigen Veränderungen des Körpers auf Probleme.

Der menschliche Körper verändert mit seinen zahlreichen Muskeln (besonders im Gesicht) in kürzester Zeit sein Aussehen. Animationen im sogenannten Subframe-Bereich (sprich Bewegungen, die schneller verlaufen, als das 1/25 einer Sekunde) bereiten dabei sehr große Schwierigkeiten, da gewöhnliche Kameras oder Motion Capturing Systeme nicht so schnell aufzeichnen können. Alle menschlichen Bewegungen einer Animation müssten mit Hochgeschwindigkeitskameras aufgenommen und (z.B. über Rotoskopie) wiedergegeben werden, was jedoch unvorstellbar viele Bewegungsdaten und Arbeitsaufwand bedeuten würde. Für minimale Bewegungen (vor allem bei Muskelpartien, Augenbewegungen und Hautirritationen z.B. Stirnrunzeln) sind derzeitige Motion-Capturing-Systeme noch zu ungenau und unflexibel.

2.9.1.7 Zusätzliche Effekte

Bereits aus Computeranimationen bekannt und derzeit auch schon teilweise in Echtzeit umgesetzt, sorgen Bewegungsunschärfe, Tiefenunschärfe, Linsenverzerrungen und atmosphärische Aufwirbelungen für mehr Glaubwürdigkeit. Leistungsfähige Partikeleffekte, wie z.B. mit der CUDA-Technologie von NVIDIA realisiert, sorgen z.B. für realistischeres Feuer, Explosionen, Regen, Schnee oder Schmutz.

[21] Anmerkung: Techniken, die sich derzeit noch nicht auf 3D-Anwendungen übertragen lassen. Hier werden Haare mit Geometrie und transparenten Texturen realisiert.

2.9.2 Ergebnis

Wir haben Sophie im Laufe ihrer Entstehung natürlich immer wieder mit anderen Spielfiguren oder echten Menschen verglichen. Durch unsere Bemühungen wirkt Sophie zwar echter und glaubwürdiger, als viele virtuelle Spielfiguren vor ihr – aber immer noch nicht wie ein echter Mensch. Die vorgestellten Techniken zur Modellierung und Animation, die auch mehr Leistung und bessere Hardware zur Darstellung benötigen, ermöglichen einen steigenden Grad der Glaubwürdigkeit von Charakteren. Trotz (oder gerade wegen) der Interaktionsmöglichkeiten, sind sie aber noch nicht verwechselbar mit Menschen aus der Wirklichkeit.

2.9.3 Ausblick

Wir sind im Laufe unserer Arbeit zu der Ansicht gelangt, dass nicht nur die Begrenzungen technischer Mittel bei der Darstellung und Animation in Echtzeit dafür verantwortlich sind, dass sich Spiele auf dem Weg in das Uncanny Valley befinden. Auch die Erwartungshaltung des Nutzers, uneingeschränkt mit seinem Charakter interagieren zu können, ist teilweise dafür verantwortlich, dass kein hundertprozentiges Realismusgefühl spürbar ist. Das Unbehagen beim Betrachten wird durch fehlende Freiheiten bei der Interaktion verstärkt. Das Uncanny Valley zu überwinden, wird also bei interaktiven Echtzeitanwendungen eine sehr viel größere Herausforderung werden, als dies bei anderen Medien der Fall ist. Wir sind aber zuversichtlich, dass Künstler und Programmierer eines Tages gemeinsam das Uncanny Valley überwinden werden.

Wir betrachten das Ende von Sophies Entstehungsprozesses mit ein wenig Wehmut und hoffen, dass diese Ausarbeitung mit dazu beiträgt, dass sie und „Die Stadt NOAH“ in Zukunft weiter entwickelt werden. Unserer Einschätzung nach hat besonders die Persönlichkeit der Erfinder maßgeblichen Einfluss auf Erschaffungsprozess virtueller Charaktere. Man erlebt bei der Entstehung eines Charakters eine unvergleichliche Entwicklung mit, welche nicht nur die Figur, sondern auch die Erfinder verändert. Wir haben Sophie das Laufen beigebracht. Gehen kann sie nun alleine und auf ihrer Reise wünschen wir alles Gute.

„Ein Charakter ist ein vollkommen gebildeter Wille.“.

Es ist eine ungültige Quelle angegeben.

Abbildung 48: Sophie Faber, gerendert in 3ds Max

3 Darstellung und interaktive Animation eines virtuellen Charakters in Echtzeit

3.1 Einführung

Eine virtuelle 3D-Welt, in der sich nichts bewegt, wirkt in der Regel recht leblos. Die Ansicht des Betrachters auf diese Welt, wirkt ohne Bewegungen unnatürlich und befremdlich. Um dies zu ändern kann mittels Animationen einer solchen Welt mehr „Leben“ eingehaucht werden.

Dabei gibt es einerseits die Möglichkeit, die Position, Ausrichtung, Größe und Form von Objekten innerhalb der Welt zu ändern. Andererseits können aber auch die Materialeigenschaften, die das Aussehen der Oberflächen der Objekte bestimmen, manipuliert werden.

Worum geht es?

In unserer Arbeit soll es hauptsächlich um Ersteres gehen, da hier im Speziellen Animationstechniken für menschliche bzw. menschenähnliche Charaktere vorgestellt und anhand aktueller Computerspiele analysiert werden. Die Animation von Oberflächenparametern spielt hierbei, wenn überhaupt, eine eher untergeordnete Rolle. Zum einen liegt das daran, dass es in dieser Arbeit um die Darstellung und Animation von Charakteren in dreidimensionalen Welten geht, und zweidimensionale Charakterdarstellungen somit außen vor bleiben. Zum anderen daran, dass die Darstellung von Charakteren in 3D Welten mit sogenannten Billboards[22] durch die enorme Rechenleistung moderner Heim-PCs inzwischen veraltet ist. Im Verlauf des folgenden Unterkapitels 3.1.2, welches sich mit der Geschichte der Animationstechniken für Charaktere beschäftigt, wird jedoch kurz darauf eingegangen. Die restlichen Anwendungsfälle für Oberflächenanimationen bei Charakteren beschränken sich auf

- Simulation von kleineren Details. Beispiele hierfür sind die Blickrichtung der Augen, der Augenlider, dem Mund usw. Diese werden jedoch mehr und mehr durch den erhöhten Detailgrad hinfällig, den heutige PCs bereits schon jetzt darstellen können. Die Details werden heutzutage oft durch Geometrie dargestellt und müssen nicht durch gemalte oder vorberechnete Bilder simuliert werden.
- exotische Spezialfälle wie z.B. Science-Fiction Anzügen oder Ähnlichem.

Bei den Animationstechniken für menschliche bzw. menschenähnliche Charaktere wird es hauptsächlich darum gehen, wie die dreidimensionale Form des Charakters geschickt so verändert werden kann, damit der Eindruck entsteht, dass sich der Charakter realistisch auf seinen Füßen durch die Welt bewegt. Dabei wird besonderes Augenmerk auf die Vermeidung von Fehlern im Bewegungsablauf gelegt. Beispiele hierfür sind

- physisch und physikalisch unmögliche Bewegungen, wie z.B. blitzartige Veränderung der Position oder/und Ausrichtung von Füßen oder Beinen bzw. des gesamten Charakters.
- sich ständig wiederholende, gleichaussehende und an die Umgebung schlecht angepasste Animation, die den Charakter sehr unglaubwürdig und sich bestenfalls roboterartig fortbewegen lässt.

Diese tragen sehr dazu bei, den wahrgenommenen Realismus in virtuellen Welten zu verringern und stören - in gewissem Ausmaß - den Benutzer dabei, sich in eine solche Welt, bzw. in die Rolle, die die Simulation versucht dem Benutzer darzustellen, hineinzufinden. Daher sollen in dieser Arbeit verschiedene Techniken vorgestellt werden, die zur Verbesserung und/oder Vermeidung dieser Probleme dienen und mit dem derzeitigen Stand der Technik möglich sind.

Ergänzend zur Animation von Charakteren wird am Ende der Arbeit auf die Darstellung der Oberfläche von Charakteren, also ihrer Haut und der Kleidung, die sie tragen, eingegangen.

---

[22] simple, meist rechteckige Fläche mit einem Bild von beispielsweise einem Charakter oder einem Baum darauf, die automatisch auf den Beobachter ausgerichtet werden

Didaktische Reihenfolge

Die Reihenfolge, in der die verschiedenen Themen in diesem Kapitel angesprochen werden ist wie folgt: Zuerst wird in diesem Unterkapitel 3.1 eine Einführung in das Thema und die damit verbundene Problematik gegeben. Außerdem wird ein Rückblick auf ältere Animationstechniken gegeben, um daraufhin in Unterkapitel 3.2 zu den aktuellen Animationstechniken für Charaktere überzuleiten. Dazu zählen zum einen die Simulation von Knochen (engl. **Bones**) und eine Möglichkeit zur Oberflächenverformung, mit der beispielsweise Muskelkontraktionen dargestellt werden können. Diese Techniken bilden die Grundlage für spätere Unterkapitel.

Im folgenden Unterkapitel 3.3 wird der Einsatz der verschiedenen Animationstechniken anhand einigen, auf dem Markt erhältlichen Computerspielen gezeigt und analysiert. Dabei wird speziell auf das visuelle Ergebnis und eventuelle Schwächen bzw. Probleme der verschiedenen Techniken eingegangen.

Unterkapitel 3.4 erläutert daraufhin die Mathematik hinter Bézierkurven. Zusammen mit Unterkapitel 3.5, in dem das Prinzip und die effiziente Implementierung von Inverser Kinematik beschrieben werden, bilden diese beiden Unterkapitel die Grundlage für das darauffolgende Unterkapitel:

Das Unterkapitel 3.6 beschäftigt sich mit dem Errechnen von Fußspuren, auf welchen Charaktere sich bewegen können. Sie bilden Fixpunkte, die von den Füssen des Charakters beim Beschreiten des Pfades erreicht werden müssen. Dazu wird der Bewegungsablauf für die Knochen unterhalb der Hüfte benötigt. Eine Möglichkeit zur Berechnung dieses Bewegungsablaufs wird in dem genannten Unterkapitels vorgestellt.

Unterkapitel 3.7 beschäftigt sich daraufhin mit der realitätsnahen Darstellung der Haut und der Kleidung eines Charakters. Dazu gibt es zu Beginn einen Einblick in die Funktionsweise heute üblicher 3D-Chips, da dies die Grundlage bildet.

Das Thema des letzten Unterkapitel 3.8 ist schließlich die Umsetzung der in Kapitel 3.6 und 3.7 beschriebenen Ansätze.

3.1.1 Allgemeines

Folgende Grundkenntnisse der Algebra werden dabei vorausgesetzt:

- **Vektor- und Matrizen-Rechnung**. Vektoren werden in der Computergrafik dazu verwendet, Positionen und Richtungen zu speichern. Sie können mittels einer affinen Abbildung (welche durch eine Matrix dargestellt werden kann) unter anderem rotiert, skaliert, und transformiert werden. Außerdem kann eine Projektion eines dreidimensionalen Vektorraums auf einen zweidimensionalen Vektorraum ebenfalls durch eine Matrix beschrieben werden. Die Grundlagen hierfür lassen sich aus mehreren Büchern zum Thema Computergrafik entnehmen, wie beispielsweise [Tremblay, 2004]
- **Quaternionen-Rechnung.** Ein Quaternion ist die Erweiterung der reellen Zahlen, ähnlich der komplexen Zahlen. Allerdings besteht es aus insgesamt vier Komponenten. Mit seiner Hilfe lassen sich insbesondere Rotationen im dreidimensionalen Raum besonders gut darstellen. Die Grundlagen und detailliertere Erklärung von Quaternionen finden sich ebenfalls in einigen Büchern zum Thema Computergrafik, wie [Tremblay, 2004]

3.1.2 Geschichte

Durch die rasante Entwicklung der Leistungsfähigkeit von Heim-PCs in den letzten Jahren, speziell von CPU und GPU, haben sich die (technischen) Möglichkeiten zur Darstellung von Animationen in interaktiven 3D-

Anwendungen stark gewandelt bzw. erweitert. In diesem Kapitel werden die verschiedenen Verfahren chronologisch sortiert beschrieben. Die derzeit aktuellen Verfahren zur Animation (speziell von Charakteren) wird in Unterkapitel 3.2 beschrieben.

3.1.2.1 Sprites

Die ersten Computerspiele sind aufgrund der damals noch recht geringen Rechenleistung zweidimensional. Entsprechend sind auch die darin vorkommenden Charaktere in zwei Dimensionen gehalten – sie bestehen lediglich aus einer Reihe von Bildern, die den Charakter in verschiedenen Posen zeigt und einigen, die als Übergang zwischen diesen fungieren.

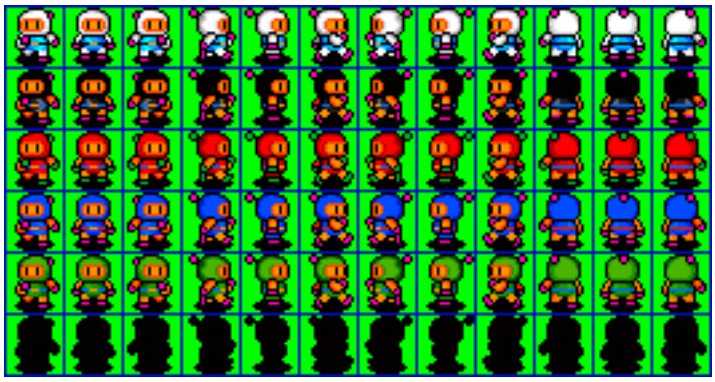


Abbildung 49: Sprites aus dem Spiel „Bombermaaan" Quelle: http://bombermaaan.sourceforge.net Autor: Thibaut Tollemer

Der Name Sprite (engl. Gespenst, Elfe, Kobold) rührt daher, dass die ersten Generationen von PC- und Konsolen-Hardware so wenig Rechenleistung hatten, dass sie nicht in der Lage waren, den Hintergrund schnell genug neu aufzubauen, um eine für das menschliche Auge flüssige Animation entstehen zu lassen. Damalige Grafikchips boten jedoch die Möglichkeit, kleinere Grafikobjekte nachträglich mittels eingebauten Hardwarefunktionen auf dem Bild zu positionieren bevor bzw. während es an den Monitor übertragen wurde. Dadurch waren diese Objekte nicht im Grafikspeicher[23] zu finden, sondern sie „geisterten" ausschließlich auf dem Bildschirm umher. Selbst als durch darauffolgende Hardware-Generationen genug Rechenleistung zur Verfügung stand und diese Optimierung unnötig wurde, behielten animierte und sich über den Bildschirm bewegende Objekte diesen Namen bei. [Wikipedia (Sprite (Computergrafik)), 2008]

Vertreter dieser Generation von Hardware waren unter anderem:

- C64
- Amiga
- Game Boy
  (Sprites: 8×8 Pixel oder 8×16 Pixel groß, max. 40 Stück) [Wikipedia (Game Boy), 2008]

In dreidimensionalen Computerspielen taucht vereinzelt ebenfalls noch der Begriff Sprite auf. Gemeint ist damit meist ein flaches Objekt (z.B. ein Dreieck oder Viereck), das im Raum positioniert wird und mit einer Oberflächentextur versehen ist, die ggf. animiert ist. Während in den ersten dreidimensionalen Computerspielen mit Hilfe von Sprites noch relativ viele Objekte wie Charaktere, Bäume, Feuer usw. dargestellt wurden, nimmt bis heute die Bedeutung von Sprites kontinuierlich ab, da immer mehr Objekte mit „echter" Geometrie dargestellt werden.

[23] Speicher, in dem das aktuell auf dem Bildschirm dargestellte Bild in Form eines zweidimensionalen Arrays gespeichert ist. Die Größe des Arrays entspricht dabei der Bildgröße und beinhaltet je Speicherzelle den Farbwert des jeweiligen Pixels.

Heutzutage spielen Sprites eigentlich nur noch bei Partikelsystemen eine Rolle, die zur Darstellung von (vielen) winzigen, sich bewegenden Objekten, wie beispielsweise Regentropfen oder Schneeflocken dienen.

3.1.2.2 Morph Target Animation

Als durch die wachsende Rechenleistung von Heimcomputern dreidimensionale Simulationen und Spiele in Echtzeit darstellbar wurden, wurden neue Techniken für die Animation von Objekten benötigt. Diese Objekte waren nun (durch Oberflächengeometrie dargestellt) plastisch und ließen sich nicht mehr durch ein Bild repräsentieren. Die simpelste Art solche Objekte zu animieren, nennt sich „Morph Target Animation". Dabei wird das Objekt in mehreren Posen (engl. **Targets**) gespeichert, d.h. jeder Vertex[24] erhält pro Pose eine bestimmte Position. Außerdem wird zu jeder Pose zusätzlich ein Zeitindex gespeichert, der angibt, zu welchem Zeitpunkt sich das Objekt genau in dieser Pose befindet. Dies wird zusammen als *Keyframe* bezeichnet. Ordnet man mehrere Keyframes in der Zeitachse hintereinander an, so ergibt sich eine Animation.

Soll das Objekt zu einem Zeitpunkt dargestellt werden, der nicht genau auf einen Keyframe fällt, so kann die Form des Objekts anhand der beiden angrenzenden Keyframes berechnet werden. Dies geschieht durch lineare Interpolation zwischen den Posen der angrenzenden Keyframes entsprechend der Position des Zeitpunkts auf der Zeitachse.

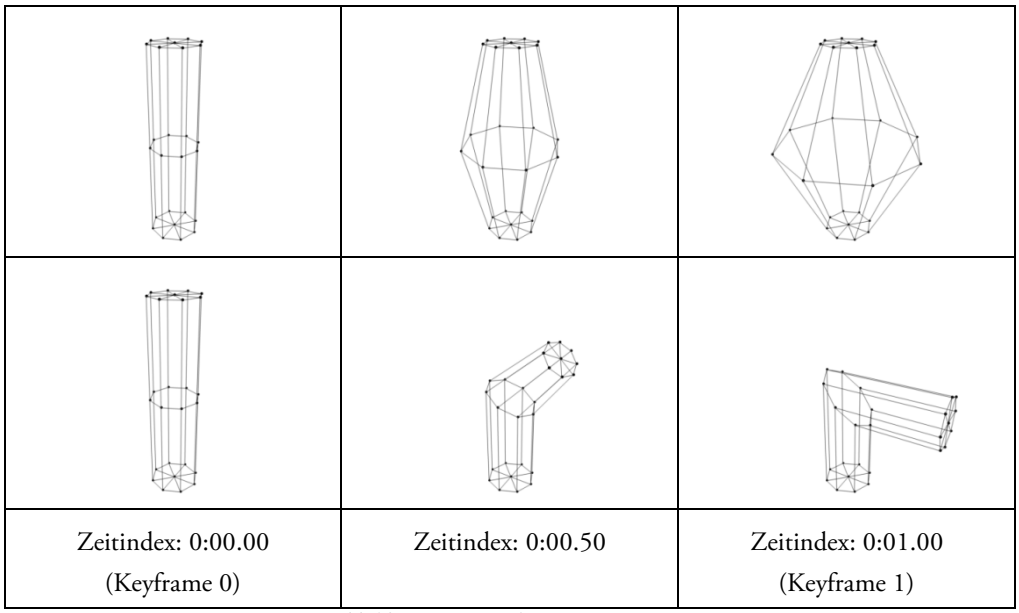


Abbildung 50: Morph Target Animation

Die Morph Target Animation ist leicht zu implementieren, allerdings wächst der Speicherbedarf und der Rechenaufwand linear mit dem Detailgrad an (jeder Vertex des Modells benötigt für jede Pose eine Position, und die aktuelle Position muss für jedes Bild berechnet werden). Damit eignet sie sich gut für simple Modelle, die aus wenigen Vertices bestehen. Für den heutzutage üblichen Detailgrad ist sie jedoch meist ungeeignet.

Ein weiterer Nachteil dieser Technik ist, dass sie gelegentlich nicht das gewünschte Ergebnis liefert, wenn das Objekt zu einem Zeitpunkt zwischen zwei Posen dargestellt werden soll. So verkürzen sich beispielsweise durch Rotationen die Strecken zwischen den Vertices deutlich, was im Falle eines Armes oder Beines eines Charakters sehr unnatürlich aussieht. Dies liegt daran, dass die Vertices zwischen den Posen linear interpoliert werden und außer Start- und Endposition keinerlei Information für die Interpolation vorliegt (siehe Abbildung 50, untere Zeile).

[24] auch Scheitelpunkt genannt, siehe Kapitel 2.3.1

3.1.2.3 Hierarchien und Skelette

Mehr Flexibilität bietet die Hierarchische Animation. Dabei wird das Objekt in mehrere starre Teile unterteilt, welche in einer baumartigen Hierarchie organisiert werden. Jeweils untergeordnete Teile können dabei relativ zum übergeordneten Teil transformiert werden[25].

Auch hier kommen Keyframes zum Einsatz, allerdings bestehen diese zusätzlich zum Zeitindex nur aus den Positionen und Ausrichtungen der Teile, aus denen das Objekt zusammengesetzt ist. Dadurch benötigt der Keyframe einer Hierarchischen Animation in der Regel wesentlich weniger Speicherplatz, als ein Keyframe einer Morph Target Animation. Die Position eines Vertices zu einem Zeitpunkt zwischen zwei Keyframes kann ebenfalls durch Interpolation errechnet werden. Allerdings stehen nun auch Informationen über die Rotation bereit, sodass eine andere Form der Interpolation genutzt werden kann, wie beispielsweise die spherical Interpolation (siehe Kapitel 3.2.3). Die einzelnen Teile des Objekts behalten dadurch auch bei Rotationen ihre Länge bei, statt sich, wie bei der Morph Target Animation, zu verkürzen. Dies verursacht jedoch unschöne (und meist nicht beabsichtigte) Spalten im Objekt (siehe Abbildung 51). Dieser Effekt kann kaschiert werden, indem man den Teilen eine passende Form gibt, sodass sie sich gegenseitig überlappen. Allerdings fällt diese Überlappung meist während einer Rotation beider Teile zueinander auf, da sich das eine Teile in das andere hineinzuschieben scheint.

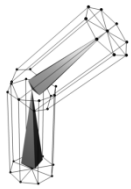
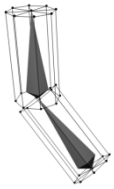
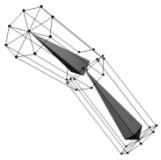
			
Grundposition	Oberer Bone 45° um X-Achse gedreht	Unterer Bone 45° um X-Achse gedreht	Oberer Bone 45°um Y-Achse gedreht

Abbildung 51: Hierarchical Animation

3.1.3 Animationen in interaktiven Anwendungen

Animationen in interaktiven Anwendungen können in mehrere Kategorien aufgeteilt werden:

3.1.3.1 Statische Animation

Die einfachste Form von Animation in interaktiven Anwendungen sind die statischen Animationen. Sie sind meist vorberechnet oder werden im Voraus von einem Grafiker erstellt. Dadurch können sie nur sehr eingeschränkt auf Einflüsse der virtuellen Welt reagieren. Man könnte sie mit einem Tonband oder einer CD vergleichen: Man kann die Animationen mit beliebiger Geschwindigkeit abspielen, bei Bedarf auch rückwärts und man kann sie stoppen. Aber inhaltlich bleibt die Animation unverändert.

Beispiele für diese Kategorie sind meist Dekorationen, die im Hintergrund stehen und unwesentlich zum Geschehen der virtuellen Welt beitragen bzw. nicht direkt vom Benutzer beeinflussbar sind, wie

- Windräder, Mühlräder, Ölbohrtürme usw.
- sich im Wind wiegende Bäume

[25] Oft wird dabei nur Rotation und Translation unterstützt

- applaudierende Zuschauer

3.1.3.2 Dynamische Animation

Dynamische Animationen hingegen reagieren auf (unvorhersehbare) Einflüsse des Benutzers und der virtuellen Welt, wie zum Beispiel

- Kollision mit anderen Objekten
- Simulation von Schwerkraft, Wind, Reibung, Trägheit usw.

Da sich daraus eine sehr große bis unendliche Zahl an Möglichkeiten ergibt, wie ein Objekt sich bewegt, kann meistens keine Animation vorberechnet werden. Sie wird daher meist „just in time" vom Programm generiert.

Typische Beispiele hierfür sind

- starre, bewegliche Objekte wie Kisten, Fässer, Steine usw.
- Flüssigkeiten, bzw. deren Oberfläche
- flexible Objekte wie Stahlfedern, Seile, Sprungbretter usw.
- leicht verformbare Objekte zum Beispiel aus Gummi oder Blech
- komplex zusammengesetzte Objekte wie z.B. Hängebrücken oder Radaufhängungen bei Fahrzeugen

3.1.3.3 Charaktere

Charaktere stellen besondere Anforderungen an die Animation ihrer Bewegungen. Dies sieht man beispielsweise daran, wie schwer heutzutage die physikalisch korrekte Berechnung von Bewegungsabläufen für menschenähnliche Roboter ist. Diese Roboter haben häufig Schwierigkeiten ihr Gleichgewicht zu halten bzw. sich zu orientieren. Daher funktionieren sie meist nur in genau definierten Umgebungen. Ähnliche Probleme tauchen in virtuellen Welten auf, wenn menschenähnliche Charaktere dynamisch animiert werden sollen. Zwar muss hier der Bewegungsablauf nicht zwangsweise auf physikalische Gesetze Rücksicht nehmen, da die Simulation selbiger einfach weggelassen werden kann (wodurch Probleme wie Gleichgewicht, Bodenhaftung usw. entfallen). Werden genannte Aspekte jedoch bei der Berechnung der Animation völlig ignoriert, ist dies meist vom Benutzer erkennbar, da der Bewegungsablauf dann sehr künstlich bzw. unrealistisch wirkt.

Da eine realistische Simulation zu kompliziert und rechenaufwendig ist und eine simple Lösung zu einem schlechten visuellen Ergebnis führen würde, muss ein Mittelweg gefunden werden, der die existierenden, praktikablen Möglichkeiten zur Animation nutzt und dabei das bestmögliche visuelle Ergebnis erzielt.

Ziel unserer Arbeit ist es, einen solchen Mittelweg zu finden. Dabei wird versucht, die Vorteile von statischer Animation mit der von dynamischer Animation zu verbinden, um sowohl sehr viel gestalterische Kontrolle über das Aussehen zu erhalten und gleichzeitig eine möglichst große Anpassungsfähigkeit auf äußere Einflüsse zu erhalten. Besonderes Augenmerk wird dabei auf die Animation der Beine und Füße beim Fortbewegen wie z.B. beim Schleichen, Gehen oder Rennen gelegt.

3.2 Aktuelle Animationstechniken für Charaktere

Da Charaktere wesentlich komplexere Bewegungsabläufe besitzen als andere Objekte, stellen sie größere Herausforderungen an die Animation, als beispielsweise Türen oder Fenster. Zudem erfordern sie eine wesentlich größere Zahl an unterschiedlichen Animationen. So kann ein Charakter typischerweise stehen, gehen und rennen bzw. sitzen oder liegen, um nur die wichtigsten Zustände zu nennen. Hinzu kommt jeweils mindestens eine Animation um zwischen zwei Zuständen zu wechseln, denkbar sind auch mehrere. So sieht die Animation eines Charakters, der sich auf ein Bett zum Schlafen legt anders aus, als die Animation eines Charakters, der gerade ohnmächtig zu Boden fällt. Auch kann es sinnvoll sein, mehrere Animationen für denselben Zustand bzw. denselben Übergang zwischen zwei Zuständen zu bieten, insbesondere wenn der Benutzer diesen voraussichtlich oft zu sehen bekommt, um etwas Abwechslung zu schaffen. Das Programm kann dadurch zufällig eine Animation auswählen, wodurch der Benutzer, genug Variation vorausgesetzt, die Wiederholung einzelner Animationen nicht mehr bemerkt und der Charakter so insgesamt lebendiger wirkt.

Eine Möglichkeit diese Wiederholungen in Bewegungsabläufen zu vermindern bzw. vollständig zu verhindern ist es, die Bewegungsabläufe dynamisch zu errechnen bzw. vorhandene, statische Animationen dynamisch abzuändern, sobald sie benötigt werden. So kann der Bewegungsablauf der Situation angepasst werden, indem Einflüsse, wie beispielsweise die Bodenneigung beim Gehen, berücksichtigt werden. Auch die Kollision mit anderen Objekten kann dabei eine Rolle spielen: Wird sie nicht beachtet, so fällt ein Charakter, der in der Nähe einer Wand ohnmächtig wird, durch die Wand hindurch. Zur Abhilfe kann eine Physiksimulation eingesetzt werden, um Kollisionen zwischen Gegenständen und Charakteren zu erkennen und eine Reaktion zu errechnen. Diese kann in die Animation mit einfließen und die Bewegung der betroffenen Gliedmaßen entsprechend abändern. Das kann leicht bewerkstelligt werden, indem die Knochen eines Charakters simuliert werden. Kapitel 3.2.1 stellt eine Technik vor, mit der dies möglich ist.

Ein weiterer Aspekt ist die Animation des Gesichts eines Charakters. Da Menschen sehr genau auf ein Gesicht achten und schon kleinste Bewegungen in diesem Bereich wahrnehmen, ist eine detailgetreue Animation hier besonders wichtig, um einen realitätsnahen Eindruck zu erzeugen (siehe auch „Das Uncanny Valley“, Kapitel 2.1.4). Hier ergeben sich ebenfalls zahlreiche Übergänge zwischen verschiedenen Zuständen, wie beispielsweise Emotionsausdrücken wie Freude, Trauer, Misstrauen, Angst usw. Hinzu kommen weitere, davon unabhängige Animationen, wie zum Beispiel die Bewegung der Augenlider oder die Mundbewegung beim Sprechen. Da sich die von der Animation betroffenen Gesichtsregionen überschneiden können, ist ein spezieller Ansatz notwendig, um diese zu einer endgültigen Animation zu kombinieren. In Kapitel 3.2.2 wird ein solcher Ansatz vorgestellt.

Beide Ansätze haben gemeinsam, dass die Animation durch Keyframes beschrieben wird. Ein Keyframe stellt den (Animations-)Zustand eines Objekts zu einem bestimmen Zeitpunkt dar. Soll das Objekt jedoch zu einem Zeitpunkt dargestellt werden, der nicht genau dem Zeitpunkt eines Keyframes entspricht, so muss der Zustand mit Hilfe von Interpolation aus den beiden umliegenden Keyframes errechnet werden. Auf diese Weise werden weiche Übergänge zwischen den Zuständen der Keyframes ermöglicht. Kapitel 3.2.3 zeigt einige Methoden der Interpolation.

Kapitel 3.2.4 beschreibt anschließend, wie die in Kapitel 3.2.1 und Kapitel 3.2.2 vorgestellten Techniken kombiniert werden können, das heißt auf demselben Objekt gleichzeitig zum Einsatz kommen können. Das ist zum Beispiel bei der Animation eines Charakters notwendig, bei dem der Körper mit Hilfe der zuerst vorgestellten Technik animiert wird, während die Animation von Gesicht und Muskeln mittels der zweiten Technik dargestellt wird. So soll beispielsweise die Animation des Gesichts weiterhin korrekt dargestellt werden, wenn der Kopf gedreht wird.

3.2.1 Skeletal Animation

Skeletal Animation ist eine Weiterentwicklung der Hierarchischen Animation (siehe Kapitel 3.1.2.3). Sie baut auf deren mathematischen Grundlagen auf, mit dem Unterschied, dass das zu animierende Objekt nicht in starre Teile getrennt werden muss. An Stelle der starren Teile treten **Knochen** (im englischen **Bones** genannt), die hierarchisch zusammengesetzt, ein **Skelett** (engl.: **Skeleton**) ergeben. Dabei „erben" sie entsprechend der Teile der Hierarchischen Animation jeweils die Transformation der ihnen übergeordneten Knochen. Eine besondere Rolle spielt dabei der in der Hierarchie an oberster Stelle stehende Knochen. Er wird als **Root-Bone** bezeichnet und besitzt keine übergeordneten Knochen. Daher kann er direkt zur Positionierung des gesamten Skeletts (und damit des Charakters) benutzt werden, da alle untergeordneten Knochen seiner Transformation folgen.

Das Skelett ist in der Anwendung nicht sichtbar (außer es wird beispielsweise zu Debug-Zwecken durch Hilfsobjekte dargestellt). Es dient lediglich als Hilfskonstrukt mit welchem der Charakter „verformt" werden kann. Dies geschieht, indem die Transformationen der Knochen auf die Geometrie, die die Oberfläche des Charakters definiert, angewandt werden. Es wird allgemein auch als Skinning oder Rigging bezeichnet (siehe Kapitel 2.7.3).

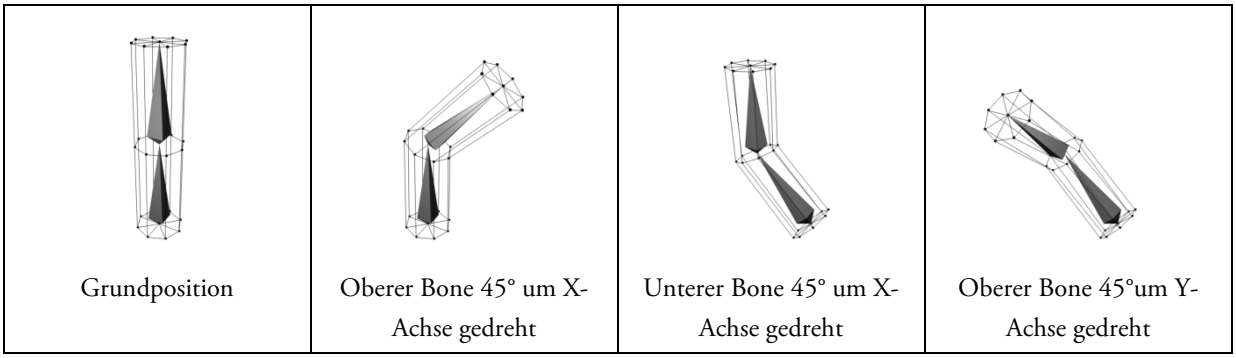


Abbildung 52: Skeletal Animation

Die Vorteile der Skeletal Animation bestehen darin, dass durch eine geeignete Konfiguration von Bones und der Weight Table häufig das gewünschte Ergebnis erreicht werden kann. Insbesondere bei der Animation von Charakteren kommt die Simulation von Knochen zur Deformation der Haut der Realität sehr nahe. Der Rechenaufwand ist jedoch höher, als bei der Hierarchical Animation, da pro Vertex vorwiegend mehrere Transformationen (der Knochen) gewichtet und summiert werden müssen. Dies kann allerdings von aktuellen Grafikkarten durch Einsatz eines Vertex-Shaders übernommen werden, um die CPU zu entlasten. Fast immer ist der Mehraufwand dabei minimal, da die Grafikkarte auch ohne Skeletal Animation bereits alle Vertices transformieren muss, um sie an die gewünschte Stelle in der virtuellen Welt zu rücken.

Dadurch, dass die Skeletal Animation auf heutiger Hardware mit geringem Mehraufwand möglich ist, und meist sehr gute Ergebnisse liefert, spricht nichts gegen ihren Einsatz bei der Entwicklung neuer virtuellen Welten.

3.2.1.1 Twist Bones

Twist Bones dienen zur besseren Darstellung der Oberfläche, wenn zwei aufeinanderfolgende Knochen zueinander verdreht sind. Es handelt sich um eine Eigenheit der Animationssoftware 3D-Studio Max von Autodesk und funktioniert, indem für einen gegebenen Knochen eine wählbare Anzahl von kleineren Knochen eingefügt wird, die den gegebenen Knochen in Segmente unterteilen. Diese Segmente werden dem gegebenen Knochen in der Hierarchie untergeordnet und Twist Bones genannt (siehe Abbildung 53).

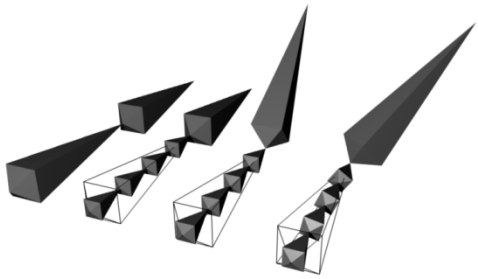


Abbildung 53: Twist Bones

Wird der dem gegebenen Knochen folgende Knochen relativ zu dem gegebenen Knochen verdreht, so wird diese Drehbewegung gleichmäßig auf die Twist Bones verteilt. Die Twist Bones verhalten sich im Bezug auf die Weight Table wie gewöhnliche Knochen, d.h. Vertices können mittels Einträgen in der Weight Table einem (oder mehreren) Twist Bones (teilweise) zugeordnet werden. So kann beispielsweise die Verformung der Haut eines Armes besser simuliert werden, wenn die Hand sich dreht.

Da die Twist Bones eine 3D-Studio Max spezifische Eigenheit sind, müssen sie bei dem Erstellen einer dynamischen Animation berücksichtigt werden und entsprechend ihrem übergeordneten Knochen (und dem ihm nachfolgenden Knochen) transformiert werden. Bei statischen Animationen ist dies nicht notwendig, da die Information über ihre Ausrichtung während der Animation bereits beim Exportieren in den Keyframes gespeichert wird.

3.2.2 Pose Animation

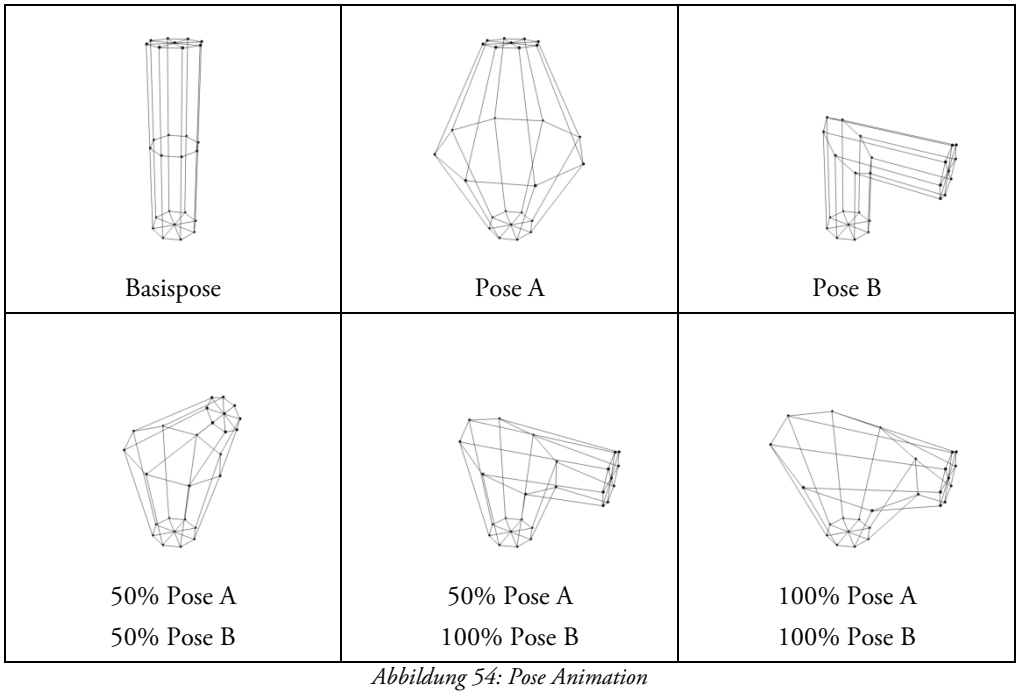


Abbildung 54: Pose Animation

Die Pose Animation stellt eine Weiterentwicklung der Morph Target Animation dar (siehe Kapitel 3.1.2.2). Sie funktioniert, ähnlich wie die Morph Target Animation, indem das Objekt in mehreren Posen gespeichert wird. Allerdings wird dabei für jeden Vertex nicht die absolute Position gespeichert, sondern die Positionsänderung jedes Vertex relativ zu seiner Basisposition (welche durch die Basispose definiert wird). Dadurch lassen sich mehrere Posen kombinieren, indem nacheinander die Positionsänderungen für jeden Vertex auf die Basisposition addiert werden.

Zudem lassen sich die Posen durch einfache Multiplikation gewichten. Somit sind die verschiedenen Posen voneinander unabhängig kontrollierbar und lassen sich beliebig zu einem neuen Endergebnis kombinieren. Allerdings ist die Pose Animation dadurch rechenaufwendiger, als die Morph Target Animation, da gegebenenfalls mehrere Posen mit dem Gewichtungsfaktor multipliziert und anschließend summiert werden müssen.

3.2.3 Keyframes und Interpolation

Keyframes stellen eine Möglichkeit dar, den Ablauf von Animationen zu speichern. Sie kommen (neben älteren Animations-Techniken) sowohl bei der Skeletal- als auch bei der Pose-Animation zum Einsatz. Alle Keyframes besitzen stets einen Zeitindex, welcher den Zeitpunkt auf der Zeitachse der Animation definiert, für den der Keyframe das Aussehen des Objekts eindeutig bestimmt. Die Informationen, die hierfür benötigt werden, variieren von Animations- zu Animationstechnik, meist bestehen sie jedoch aus einer Reihe von Positionen oder/und Transformationen.

Tritt der Fall ein, dass die Form eines Objekts zu einem Zeitpunkt bestimmt werden soll, der nicht genau durch einen Keyframe definiert wird, so muss zwischen zwei Keyframes *interpoliert* werden. Dies passiert beispielsweise, wenn die Anwendung die Animationen nicht in festen Zeitabständen aktualisiert, oder die Keyframes der Animationen nicht an einem festen Zeitraster ausgerichtet sind. Durch die Interpolation wird ein neuer Keyframe errechnet, der das Aussehen des Objekts zum gewünschten Zeitpunkt bestimmt. Der Zeitindex dieses neuen Keyframes ergibt sich aus dem gewünschten Zeitpunkt. Die restlichen Informationen des Keyframes müssen mit einem der folgenden Interpolationsverfahren berechnet werden:

- Die konstante Interpolation ist das einfachste Verfahren. Mit ihrer Hilfe kann jede Information interpoliert werden. Sie kopiert dazu die Information des vorhergehenden Keyframes und belässt diese, unabhängig vom gewünschten Zeitpunkt, unverändert. Daraus folgen sprunghafte Änderungen der Form des zu animierenden Objekts - es befindet sich nie in einem Übergangszustand. Diese Art der Interpolation ist nur in Spezialfällen nützlich bzw. erwünscht.
- Die lineare Interpolation interpoliert zwischen Informationen, die sich mittels eines skalaren Faktors multiplizieren lassen (wie beispielsweise Positions- und Transformationsänderungen). Dieser Faktor errechnet sich aus dem gewünschten Zeitpunkt relativ zu den Zeitpunkten der umgebenden Keyframes. Für die Implementierung einer linearen Interpolation siehe [Tremblay, 2004]. Die Form des zu animierenden Objekts ändert sich beim Einsatz von linearer Interpolation gleichmäßig von der Form aus dem vorhergehenden Keyframe, hin zu der Form des nachfolgenden Keyframes. Rotationsbewegungen werden dabei meist „abgekürzt“, da der direkte Weg genommen wird, wodurch sich fast immer ungewollte Verformungen ergeben.
- Die sphärisch-lineare Interpolation (engl.: *spherical linear interpolation*, kurz *slerp*) eignet sich zur Interpolation von Rotationen. Sie verwendet dazu denselben Faktor, wie die lineare Interpolation und basiert auf dem Einsatz von Quaternionen zur Darstellung der Rotation. Für eine mögliche Implementierung siehe [Tremblay, 2004]. Die Form des zu animierenden Objekts ändert sich durch ihren Einsatz nicht, da sie nur für Drehbewegungen gedacht ist. Dabei dreht sie das Objekt mit einer linearen Winkelgeschwindigkeit. Sie bildet somit eine gute Ergänzung zur linearen Interpolation von Keyframes, um deren Nachteil zu beheben.
- Das *dual quaternion linear blending* (kurz *DLB*) eignet sich, um Rotationen und Translationen zu interpolieren. Dabei kommen Quaternionen mit dualen Zahlen zum Einsatz. Für die Erklärung der Funktionsweise siehe [Kavan, Collins, O'Sullivan, & Zara, 2006]. Sie ist besonders für die Interpolation einer Skeletal Animation geeignet, da sich die Knochen des Skeletts mit Hilfe von Rotationen und Translationen beschreiben lassen. DLB behebt außerdem Darstellungsprobleme bei der Kombination mehrerer Animationen.

3.2.4 Verbinden von Skeletal Animation und Pose Animation

Besonders bei der Animation von Charakteren ist es oft notwendig, dass Skeletal Animation und Pose Animation gemeinsam eingesetzt werden sollen, um die Vorteile beider Verfahren optimal nutzen zu können. So eignet sich die Skeletal Animation gut, um den Körper des Charakters insgesamt zu animieren, während die Pose Animation bestens geeignet ist, um kleine Details darzustellen, wie der Animation des Gesichts oder der Muskeln. Auch noch kleinere Details wie beispielsweise die Simulation von Gänsehaut sind in naher Zukunft denkbar.

Die Herausforderung besteht darin, beide Animationstechniken so zu kombinieren, dass am Ende ein stimmiges Ergebnis herauskommt. Da die Pose Animation in der Regel Details darstellt, die vom Skelett beeinflusst werden sollen, wird diese zuerst auf das Objekt angewandt. Anschließend wird die Skeletal Animation auf das verformte Objekt angewandt. Dadurch ist sichergestellt, dass die Details der Haut weiterhin den Knochen folgen. Würde man die umgekehrte Reihenfolge wählen, so hätte dies zur Folge, dass die Pose Animation auf eine Art und Weise angewandt wird, als sei das Objekt nicht verformt worden. So würde ein sich schließendes Augenlid eines Charakters, der seinen Kopf in den Nacken legt, sich unabhängig von der Lage des Kopfes nach unten bewegen (also in diesem Fall in den Kopf hinein).

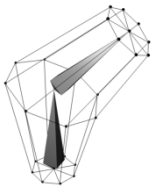
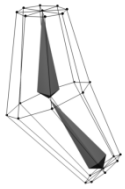
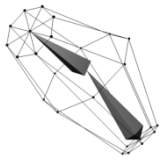
			
Grundposition	Oberer Bone 45° um X-Achse gedreht	Unterer Bone 45° um X-Achse gedreht	Oberer Bone 45°um Y-Achse gedreht

Abbildung 55: Pose und Skeleton Animation kombiniert

Bei der Kombination von Pose und Skeletal Animation ergibt sich die Möglichkeit, die Gewichtung der Pose Animation anhängig vom Winkel zweier Knochen zu der Skeletal Animation zu machen. Dafür wird ein Faktor oder eine Kurve benötigt, welche(r) für einen gegebenen Winkel die Gewichtung der Pose Animation liefert. Während der Animation des Objekts kann für jeden Zeitpunkt der Winkel bestimmt werden, um eine dynamische Pose Animation zu generieren. Dies ist zum Beispiel nützlich für die Simulation von Muskelkontraktionen und der sich daraus ergebenden Wölbung der Haut: Je stärker beispielsweise ein Charakter seinen Arm anwinkelt, desto mehr treten die Muskeln seines Oberarmes zum Vorschein.

3.3 Charakteranimationen in Computerspielen

3.3.1 Doom (1993)

Doom ist eine Computerspielreihe, deren erster Teil am 10. Dezember 1993 von id Software veröffentlicht wurde. Der erste Teil setzte damals technisch neue Maßstäbe für Heim-Computerspiele, da es das erste Spiel war, das größere dreidimensionale Levels flüssig auf der damaligen PC-Hardware (Intel 386) darstellen konnte.

Der Spieler wird während des Spiels in die Rolle eines Space-Marines versetzt, der sich nach einem missglückten Forschungsexperiment gegen Kreaturen aus der Hölle wehren muss. Ihm stehen dazu im Verlaufe des Spiels mehrere Waffen zur Verfügung (z.B. Motorsäge, Pistole, Schrotflinte usw.).

Abbildung 56: Doom

Da das Spiel in der Ego-Perspektive spielt, sieht der Spieler in der 3D-Ansicht von seinem Charakter nur die Hand, die die derzeitig ausgewählte Waffe hält. Diese und die Kreaturen, gegen die sich der Spieler wehren muss, werden dabei durch (dreidimensionale) Sprites dargestellt, die dem Spieler immer zugewandt sind. Sie werden, wenn der Spieler ihnen näher kommt, relativ grob dargestellt. Außerdem fällt beim Spielen schnell auf, dass es sich bei den Gegnern eigentlich nur um flache „Scheiben“ handelt.

Die Animationen werden durch simples Austauschen des Sprite-Bildes realisiert. Dies geschieht jedoch nur mit einer geringen Frequenz, so dass die Kreaturen sehr stockend in ihren Bewegungen wirken.

Die für die damalige Zeit sehr moderne Darstellung von Charakteren wirkt heutzutage sehr unglaubwürdig, da Anhand vieler Details erkannt werden kann, dass der Charakter nur eine vom PC errechnete Simulation ist.

3.3.2 Grand Theft Auto – Serie (1997-2008)

Die Grand Theft Auto Serie ist eine der erfolgreichsten Serien von Videospielen, zu der neben einigen Ablegern inzwischen vier Hauptteile gehören. Insgesamt gibt es derzeit neun Teile welche vom Rockstar North (bis 2001 als DMA Design bekannt) entwickelt wurden, sechs davon sind für den PC erhältlich.

Technisch unterschiedlich sind besonders der zweite, dritte und vierte Teil, weswegen sie im Folgenden getrennt behandelt werden. Allen gemeinsam ist jedoch die Rolle, in die der Spieler schlüpft und die Handlung mit der er konfrontiert wird: Man spielt einen (teils namenlosen) männlichen Charakter, der meist bereits vorbestraft ist. Dieser versucht sein „Glück“ in einer amerikanischen Großstadt, wo er schnell auf Seinesgleichen trifft. So findet sich durch

Erfüllen von Aufträgen von Kleinkriminellen schnell Kontakt zur Mafia oder ähnlichen Verbrecherorganisationen, welchen man sich anschließen kann. Auch von ihnen erhält man Aufträge, deren Erfolg meist mit Waffen und/oder Geld belohnt wird. Dabei kann sich der Spieler jedoch frei innerhalb und außerhalb der Stadt bewegen, in der freiwillige Zusatzaufgaben auf ihn warten. So kann man beispielsweise mit einem Taxi Personen befördern, mit einem Feuerwehrauto Brände bekämpfen usw. Die Spielwelt versucht dabei so realistisch wie möglich zu sein und dem Spieler nichts zu verbieten. Deshalb muss eine Vielzahl von Aktionen berücksichtigt werden, die der Spieler unternehmen kann. Die Animation dieser Aktionen ist somit ein wesentlicher Bestandteil des Spiels.

3.3.2.1 Grand Theft Auto II

Der zweite Teil der Serie erschien 1999 für PC, Playstation, Dreamcast und Gameboy Color. Er wird, ebenso wie der erste Teil aus einer zweidimensionalen Perspektive gespielt, aus der der Spieler die Spielwelt von oben sieht. Während die Häuser bereits durch dreidimensionale Geometrie simuliert werden, werden die Charaktere und die Autos noch durch Sprites dargestellt. Da man die Charaktere meist von oben sieht, werden keine aufwändigen Animationen benötigt. Das Spiel kommt somit einer geringen Zahl von Animationen aus, welche die Charaktere beim Stehen und Gehen mit verschiedenen Waffen in der Hand darstellen. Die Charaktere sind ausschließlich dann von vorne zu sehen, wenn sie mit dem Rücken auf dem Boden liegen. Dies ist beispielsweise nach einem Verkehrsunfall der Fall. Allerdings reicht hierfür ein einzelnes Bild (statt einer Animation) aus.

Abbildung 57: Grand Theft Auto II

Die Animation von Charakteren in Grand Theft Auto II wirkt dennoch sehr stimmig, da durch die Vogelperspektive geschickt große Teile des Charakters versteckt werden, ohne dass dies für den Spieler auffällig ist.

3.3.2.2 Grand Theft Auto III

Der dritte Teil der Serie erschien am 22. Oktober 2001 zunächst für die Playstation 2, die Umsetzung für den PC war erst ab dem 21. März 2002 erhältlich. Über ein Jahr später, im November 2003 folgte schließlich die Umsetzung auf die XBox. Er ist der zuerst erschienene Teil einer Trilogie, jedoch spielt die Story chronologisch an letzter Stelle.

Abbildung 58: Grand Theft Auto III

Völlig neu war die dreidimensionale Spielwelt, in der sich der Spieler nun typischerweise aus der Verfolgerperspektive sah (andere Perspektiven konnten optional während des Spiels gewählt werden, waren meist jedoch spielerisch wenig sinnvoll). Ebenfalls neu war, dass die Story des Spiels nun teilweise durch Zwischensequenzen in Spielgrafik erzählt wurde. Beides zusammen bedeutete wesentlich höhere Anforderungen an die Animationen, als es noch im Vorgänger der Fall war. So war nun erstmals das Gesicht der Charaktere sichtbar, und musste passend animiert werden. Hierfür wurde die Morph Target Animationstechnik (siehe Kapitel 3.1.2.2) verwendet, um vorgefertigte Animationssequenzen abzuspielen. Für die Animation des restlichen Körpers wurde die zu diesem Zeitpunkt schon leicht veraltete Hierarchical-Animation-Technik (siehe Kapitel 3.1.2.3) angewendet, bei dem der Körper aus getrennten, starren Teilen zusammengesetzt wurde. Dadurch wirkten die Charaktere teilweise wie Plastik-Puppen, deren Glieder sich durch Gelenke bewegen lassen (sichtbar auf Abbildung 58 an der Schulter). Dies ist allerdings der einzige Punkt, der negativ an den Animationen auffällt, denn die Animationen (für die Zwischensequenzen) selbst, waren durch aufwändige Handarbeit erstellt worden und wirkten dadurch sehr stimmig und natürlich.

Außerhalb der Zwischensequenzen bestand das Problem, dass die Position der Objekte, mit denen der Spieler interagiert, wie der Griff einer Autotür, vom Charakter aus gesehen, variabel ist, und somit keine genau passende Animation vordefiniert werden kann.

3.3.2.3 Grand Theft Auto IV

Dieses Problem wird in Grand Theft Auto IV, dem neunten Teil der Serie, durch den Einsatz von NaturalMotion's Euphoria behoben (siehe Kapitel 3.3.6). Auch die Animationstechnik wurde (bereits mit dem vierten Teil der Serie) auf die aktuell gebräuchliche umgestellt – der Skeletal Aniamtion. Dadurch wirken die Charaktere nicht mehr wie Puppen. Somit ist der derzeit aktuelle Teil der Serie, der am 29. April 2008 für Playstation 3 und XBox 360 veröffentlicht wurde, animationstechnisch auf dem aktuellsten Stand. In der PC-Version, welche am 3. Dezember 2008 folgte, wurden nachträglich noch einige Details, wie das Aussehen von Explosionen verbessert.

Insgesamt forderte der stark erhöhte Detailgrad der Spielwelt und der Charktere die Mitarbeit von über 1.000 Personen, die in irgendeiner Weise an dem Projekt beteiligt waren, davon allein 150 Entwickler. Die Produktionskosten werden vom Produzenten Leslie Benzies auf etwa 100 Mio. US-Dollar geschätzt. [Bowditch, 2008]

Abbildung 59: Grand Theft Auto IV: Skeletal Animation

Durch den stark erhöhten Detailgrad der Charaktere im Vergleich zu den Vorgängern stellte auch die Facial Animation der Charaktere ebenfalls höhere Anforderungen als je zuvor. Daher beauftragte Rockstar North die Firma image metrics mit der Erstellung der Gesichtsanimationen für die über 80 Charaktere im Spiel, die in den Zwischensequenzen auftauchen. Insgesamt stellte image metrics unter Verwendung einer eigenentwickelten, proprietären Software innerhalb von 6 Monaten über 300 Minuten Animationsmaterial her. [image metrics, 2008]

Abbildung 60: Grand Theft Auto IV: Facial Animation und Einsatz von NaturalMotion's euphoria

NaturalMotion's euphoria übernimmt für Grand Theft Auto IV die Berechnung der dynamischen Animationen zur Laufzeit. Es kann neben dem simplen Greifen von Objekten auch die Auswirkung von Schlägen und Schüssen auf einen Charakter realitätsnah darstellen, indem es die Muskeln und das Nervensystem des Charakters simuliert. Auch die Auswirkungen eines Sprungs aus einem fahrenden Autos kann mit Hilfe von euphoria simuliert werden. Dabei versucht euphoria die Arme des Charakters während des Abrollens in eine, den Kopf schützende Position, zu bringen, wie es ein Mensch in einer solchen Situation reflexartig tun würde (siehe Abbildung 60, rechts).

Grand Theft Auto IV demonstriert durch den Einsatz von aktuellster Technik, was heutzutage bei der Animation von Charakteren möglich ist. Durch den sowohl großen technischen Aufwand, als auch durch den großen Arbeitsaufwand, der bei der Animation betrieben wurde, wirken die Charaktere äußerst authentisch, was unter anderem dafür sorgte, dass dieses Spiel (unter anderem) wegen der realistischen Darstellung von Gewalt in die Kritik geriet.

3.3.3 Baphomet's Fluch - Der schlafende Drache (2003)

Baphomet's Fluch - Der schlafende Drache (engl. Originaltitel: „Broken Sword: The Sleeping Dragon") ist der dritte Teil einer Adventure-Serie, die von Revolution Software entwickelt wurde. Es ist im Gegensatz zu seinen beiden Vorgängern jedoch der erste Teil, der in Echtzeit-3D-Grafik dargestellt wird. Es erschien zeitgleich im Oktober 2003 in je einer Version für PC, Playstation 2 und XBox.

Der Spieler übernimmt während des Spiels abwechselnd die Kontrolle über den US-Patentanwalt George Stobbart und der französischen Journalistin Nicole Collard (meist nur kurz „Nico“ genannt). Während George im Kongo zu Beginn des Spiels über einen mysteriösen Mordfall stolpert, recherchiert Nico die genauen Umstände eines ebenso mysteriösen Mordfalls in Paris. Dies führt beide zusammen und es wird klar, dass hinter den beiden Mordfällen mehr steckt, als sich zuerst vermuten ließ. Um das Geheimnis dieses Rätsels zu lösen, reist der Spieler mit den Protagonisten an verschiedene Schauplätze rund um die Welt, um dort durch Aufsammeln von Gegenständen und Beweisen, dem Führen von Gesprächen und Lösen von Rätseln, mehr über die Mordfälle und deren Zusammenhänge zu erfahren.

Abbildung 61: Baphoments Fluch

Gesteuert wird das Spiel ausschließlich mit der Tastatur, durch die der Spieler die Charaktere bewegen und Aktionen auslösen kann. Er sieht dabei die Protagonisten immer aus dem Blickwinkel einer dritten Person. Dadurch wurden speziell an die Animation große Anforderungen gestellt, da die visuelle Darstellung und Animation der Protagonisten viel zum Charme des Spiels beitragen und das hohe Level der vorherigen Teile gehalten werden sollte.

Da die Story die möglichen Aktionen der Protagonisten und die Positionen der Objekte, mit denen sie interagieren können, genau vorgibt, konnte eine große Zahl von speziell angepassten Animationen im Voraus durch Computergrafiker erstellt werden. Zusammen mit dem Einsatz von Skeletal Animation (siehe Kapitel 3.2.1) konnten daher die Anforderungen erfüllt werden: George und Nico greifen Objekte beispielsweise sehr exakt und hantieren mit ihnen auf eine sehr natürliche Art und Weise (siehe Abbildung 61)

3.3.4 Gothic-Serie (2001-2006)

Gothic ist eine Serie von Computerspielen, in dem der Spieler einen namenlosen Helden spielt. Der erste Teil erschien bereits am 15. März 2001, aktuell ist derzeit der dritte Teil (Erscheinungsdatum: 13. Oktober 2006). An dem vierten Teil wird derzeit gearbeitet. Die ersten drei Teile wurden von Piranha Bytes entwickelt, der in Entwicklung befindliche vierte Teil wird jedoch nach einem Streit zwischen den Entwicklern und dem Publisher JoWood von Spellbound Entertainment produziert.

In allen Teilen der Serie steuert der Spieler den Helden aus der Verfolgerperspektive. Er erkundet mit ihm die große Spielwelt, sammelt Gegenstände, die ihm auf seiner Reise nützlich erscheinen, spricht mit den Einwohnern und erfüllt die von ihnen gestellten Aufgaben, um so Erfahrung zu sammeln. Hintergrund ist dabei ein Krieg zwischen den Orks und den Menschen. Schauplatz des ersten Teils ist die Minenkolonie auf der Insel Khorinis, der einzigen Stelle im Königreich der Menschen, wo magisches Erz zum Schmieden von Waffen abgebaut werden kann. Die Minenkolonie wird von einer magischen Barriere geschützt, die die Flucht der darin gefangenen Sträflinge (und damit auch des Spielers) verhindern soll. Diese geriet durch (zu Beginn des Spiels) unbekannten Gründen größer als geplant, weshalb auch die Magier, die sie erschufen, zu Mitgefangenen wurden. Mit ihrer Hilfe gelingt es dem Spieler

zum Ende des ersten Teils die Barriere zu zerstören. Der zweite Teil der Serie knüpft direkt daran an und spielt auf der gesamten Insel, auf der nun, nach dem Zusammenbruch der Barriere etwas Chaos herrscht. Jedoch deutet sich zu Beginn dieses Teils eine noch sehr undeutlich beschriebene Bedrohung an, die der Spieler besiegen muss, da er der Auserwählte dafür sei. Nach dem Meistern vieler kleinerer Aufgaben und der Hauptaufgabe des zweiten Teils segelt der Held mit einem Schiff Richtung Festland, wo sich unter anderem das Königreich Myrtana befindet. Dies ist zusammen mit 2 weiteren Regionen Schauplatz des dritten Teils der Serie. Dort gerät der Held nach dem Anlegen mitten zwischen die Fronten des immer noch andauernden Kriegs und muss sich früher oder später entscheiden, welcher Seite seine Gunst gilt, da nicht immer alle Aufgaben beider Seitenfür ihn lösbar sind.

Abbildung 62: Gothic 3: Gesten

Einen Großteil des Spiels verbringt der Held neben dem Erkunden der Landschaft mit dem Führen von Gesprächen und Einsammeln von Gegenständen. Dafür wurde von den Entwicklern ein Satz von (statischen) Animationen bereitgestellt, die währenddessen abgespielt werden. So gestikulieren der Held und sein Gesprächspartner meist (einigermaßen) passend zum Gespräch. Allerdings ist die Anzahl dieser Animationen nicht sehr groß, sodass immer wieder dieselben Animationen auffallen (Abbildung 62 zeigt zwei dieser Gesten).

Abbildung 63: Gothic 3: Magischer Stein in der Hand des Helden

Da in der Spielwelt eine riesige Anzahl von Gegenständen bereitsteht, die der Held einsammeln kann oder mit denen er interagieren kann, und nicht vorhersehbar ist, aus welcher Position und Richtung er das tut, ist die jeweilige statische Animation die verwendet wird, oft unpassend. So berührt die Hand des Helden meistens nicht das Objekt, welches er einsammelt, bis er es, wie durch Zauberhand plötzlich in seinen Händen hält. Ab hier stimmt die Position des Objekts sehr genau, da das Objekt nur noch der statischen Animation der Hand folgen muss (siehe Abbildung 63). Die Position der Hand lässt sich dabei durch den Einsatz der „Skeletal Animation"-Technik sehr einfach bestimmen.

Insgesamt betrachtet wirkt die Animation der Charaktere in der Gothic-Serie durch die Ungenauigkeiten bzw. die physikalischen Unstimmigkeiten teilweise unrealistisch. Hier würde ein dynamisches System sicherlich Abhilfe

schaffen, jedoch entschieden sich die Entwickler wohl aus Zeit- und Kostengründen gegen ein solches. Auch könnten damit die ständigen Wiederholungen der exakt gleich ablaufenden Animationen vermieden werden.

3.3.5 Assassin‘s Creed (2007)

Assassin’s Creed ist ein Videospiel, das zeitgleich für mehrere Plattformen von Ubisoft Montreal mit großem Aufwand entwickelt wurde. Vier Jahre lang arbeiteten 300 festangestellte Mitarbeiter an der Umsetzung. Am 15. November 2007 erschien es für Playstation 3 und Xbox 360. Eine Version für den PC folgte am 10. April 2008.

Das Spiel handelt von einem Assassinen namens Altaïr, der im Jahre 1191 nach Christus gelebt hat, und seinem entfernten Nachfahren namens Desmond Miles, welcher in naher Zukunft als Barkeeper seinen Unterhalt verdient. Zu Beginn des Spiels wird Desmond von den Mitarbeitern der Firma Abstergo Industries entführt und gezwungen, die in ihm gespeicherten Erinnerungen seines Vorfahren mit Hilfe eines speziellen Geräts, dem Animus, zu durchleben. Abstergo Industries erhofft dadurch Informationen zu erhalten, die zum Auffinden eines alten, verschollenen Artefakts führen, welches Altaïr seinerzeit aus den Händen der Templer eroberte.

Die Erinnerungen von Desmond handeln dabei von Missionen von Altaïr, die er zum Wiedererlangen seiner Ehre zu erfüllen hatte. Der Spieler schlüpft in die Rolle von Desmond und durchlebt durch den Einsatz des Animus die Missionen von Altaïr. Jede Mission gliedert sich dabei in die Abschnitte Erkundung, Attentat und Flucht, die der Spieler nacheinander absolvieren muss. Dabei steht ihm eine große Bewegungsfreiheit zur Verfügung: Man kann beispielsweise beinahe jeden Turm hinaufklettern, um sich eine Übersicht über die Umgebung zu verschaffen. Informationen über die Zielperson lassen sich unter anderem durch den Taschendiebstahl einer Notiz erlangen. Wird man dabei erwischt, so kann es zu einem Kampf mit den Stadtwachen kommen. Der Spieler kann durch geschicktes Untertauchen in der Menschenmenge dem Kampf entgehen, genauso nach einem Attentat.

Abbildung 64: Assassin's Creed

Insgesamt stellt dies nur eine kleine Auswahl der Möglichkeiten dar, die dem Spieler zur Verfügung stehen. Daraus ergibt sich ein riesiger Bedarf an Animationen für die Charaktere des Spiels, den selbst Ubisoft Montreal zu Beginn der Entwicklung etwas unterschätzt hat. So entschieden sich die Entwickler zu diesem Zeitpunkt gegen den Einsatz von vollständig dynamisch generierten Animationen für die Charaktere, um mehr Kontrolle über Animationen zu

behalten, um eine bestimmte Qualität sicherstellen zu können. Allerdings stellte es sich wenig später als eine Entscheidung heraus, die sehr viel Aufwand verursachte: So wurden allein für Altaïr insgesamt 12.000 Animationen von Hand erstellt, die das Spiel als Grundlage zur Berechnung der Animation benutzt [Tory, 2007]. So blendet das Spiel abhängig vom Zustand des Charakters mehrere Animationen übereinander, um den endgültigen Bewegungsablauf zu erhalten. Selbst kleine Details spielen dabei eine Rolle, Altaïr konzentriert sich beispielsweise deutlich sichtbar auf seine Hand während des Taschendiebstahls (siehe Abbildung 64, rechts oben), oder es wird ein leichtes Humpeln in die Animation des Gehens eingeblendet, wenn er am Fuß verletzt ist.

Abbildung 65: Assassin's Creed: Gesten und Cloth Simulation

Auf die Körpersprache während der Dialoge wurde besonderen Wert gelegt. Die Charaktere gestikulieren passend während eines Gespräches. Damit es besonders zur Geltung kommt, wurde ein spezielles Kamera-System eingebaut, mit dem der Spieler während eines Dialogs zwischen verschiedenen Blickwinkeln wählen kann. Zusätzlich wird während des gesamten Spiels die Bewegung der Kleidung dynamisch simuliert (*Cloth Simulation*), da diese sich nicht (bzw. nur mit großem Aufwand), wie die Charaktere durch ein Skelet animieren lässt.

Abbildung 66: Assassin's Creed: Ungenauigkeit durch Interpolation

Alles in allem sind die Animationen in Assassin's Creed durch den sehr hohen Einsatz von Handarbeit sehr gelungen. Sie lassen den Charakter sehr natürlich und lebendig wirken, da selbst kleine Details dargestellt werden. Allerdings lässt sich bei genauer Betrachtung der Nachteil von vorgefertigten Animationen noch erkennen: So ergeben sich immer wieder kleine Ungenauigkeiten, beispielsweise rutschen Altaïrs Füße leicht auf dem Boden hin und her während er geht und die Position der Hand stimmt beim Greifen nicht genau (siehe Abbildung 66). Solche Ungenauigkeiten wären nur durch den Einsatz von dynamisch generierten Animationen zu verhindern gewesen.

3.3.6 NaturalMotion endorphin / euphoria

endorphin ist die erste Software, mit der die Bewegung eines Charakters dynamisch berechnet werden kann. NaturalMotion verwendet dafür den Begriff „*Dynamic Motion Synthesis*“ (DMS) und bezeichnet es als einen Durchbruch auf dem Gebiet der Charakteranimation.

Die Software basiert auf der Forschung der Universität von Oxford und wird von einem eigens dafür gegründeten Unternehmen (NaturalMotion) entwickelt und vertrieben. Sie ermöglicht die Erzeugung von Charakteranimationen in einer bisher unerreichten Qualität in Echtzeit. Die Qualität wird durch die Kombination von künstlicher Intelligenz und der Simulation der Knochen und Muskeln eines Charakters einschließlich der Kräfte, die auf sie wirken, erreicht. Kollisionen mit anderen Objekten können ebenso berechnet werden und in die Animation einfließen.

Kern der Anwendung bilden sogenannte anpassungsfähige Verhaltensmuster (engl. *Adaptive Behaviours*). Sie beschreiben das Verhalten des Charakters und lassen sich durch zahlreiche Parameter konfigurieren. So gibt es beispielsweise ein Verhaltensmuster für einen Footballspieler, der, wenn er einem anderen Footballspieler nahe kommt, diesen automatisch attackiert. Außerdem können Posen vorgegeben werden, die der Charakter versucht, begrenzt durch seine physikalischen Möglichkeiten, einzunehmen. Auch kann die errechnete Animation mit einer existierenden, vorgefertigten Animation gemischt werden. Jedoch gibt es derzeit noch kein Verhaltensmuster, um einen Charakter dynamisch generiert gehen zu lassen. Die Entwicklung eines solchen Verhaltensmusters dürfte einen sehr großen Aufwand darstellen, da die Software von NaturalMotion sämtliche physikalische Einflüsse der virtuellen Welt auf den Charakter simuliert. Allerdings steht ein Verhaltensmuster zur Verfügung, das den Charakter veranlasst, sein Gleichgewicht zu halten. Gibt man diesem Charakter einen Stoß, so torkelt er einige Schritte in die entsprechende Richtung.

Eine fertige Animation kann in jede größere Animationssoftware exportiert werden, daher eignet sich die Software besonders für die Erstellung von vorgefertigten Animationen, wie sie zum Beispiel bei der Produktion eines Computeranimationsfilms benötigt werden. Unter http://www.naturalmotion.com/endorphin.htm kann eine kostenlose, eingeschränkte Version von endorphin heruntergeladen werden (endorphin learning edition).

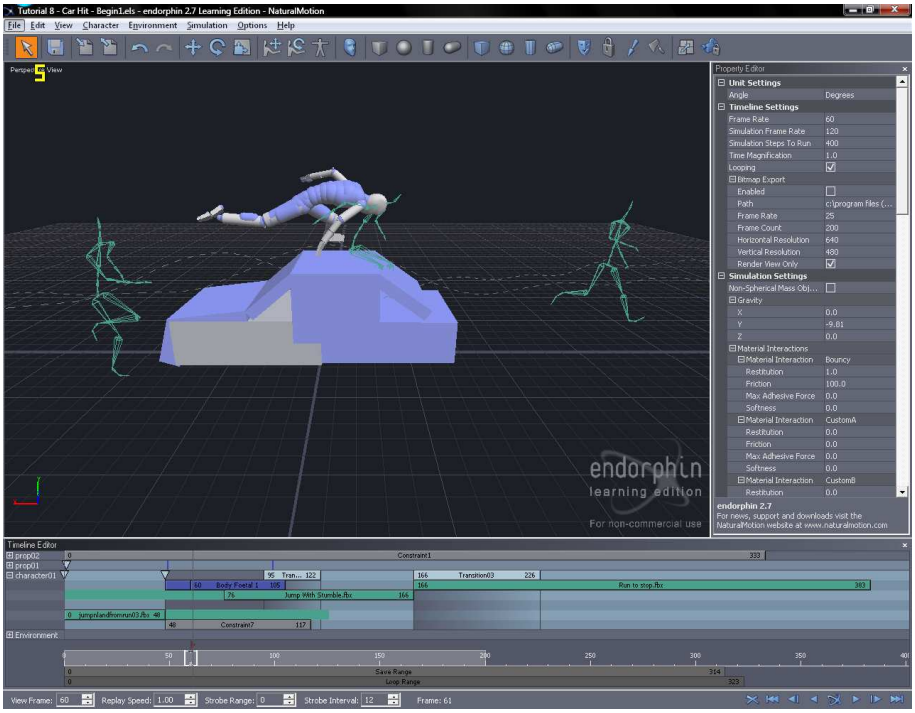


Abbildung 67: Natural Motion endorphin learning edition

Für interaktive Anwendungen, wie beispielsweise Computerspiele kündigte NaturalMotion das inzwischen erhältliche euphoria im März 2006 an. Es basiert auf demselben Prinzip wie endorphin, ist aber keine eigenständige Software, sondern eine Bibliothek, die zur Einbindung in bestehende Projekte gedacht ist. Es wird mit einem Satz von Werkzeugen ausgeliefert, mit dem die dynamischen Animationen erstellt, konfiguriert und getestet werden können. euphoria läuft auf dem PC und Playstation 3- bzw. XBox 360-Konsolen.

3.4 Bézierkurven

Kurven sind in vielen Gebieten der Computergrafik nützlich. Im einfachsten Fall dienen sie dazu, beliebige Parameter über die Zeit fließend zu ändern. So kann die Wiedergabegeschwindigkeit einer statischen Animation durch eine Kurve gesteuert werden, denkbar wäre dies beispielsweise bei einem Windrad, um Windböen vorzutäuschen. Die Möglichkeiten, die sich hierbei ergeben, sind so gewaltig, dass jede größere Animationssoftware in der Regel einen eigenen Editor zum Bearbeiten von Animationskurven mitbringt.

Ein weiteres Einsatzgebiet von Kurven ist die Definition von Geometrie. So lassen sich zum einen Flächen durch Kurven beschreiben. Bekannte Beispiele hierfür sind unter anderem das Bézier-Dreieck und Bézier- bzw. NURBS-Flächen. Auch können Kurven als Extrusionspfad für beliebige, zweidimensionale Formen dienen, d.h. sie beschreiben den Pfad, an dem eine zweidimensionale Form entlang gezogen wird, um ein dreidimensionales Objekt zu erhalten. Straßen und gewundene Röhren lassen sich auf diese Weise leicht erzeugen. Eine weitere Möglichkeit zur Erzeugung von Geometrie, ist mit einer Kurve den Querschnitt eines Rotationskörpers zu beschreiben. Dreht man diesen Querschnitt einmal um sich selbst, erhält man ein dreidimensionales Objekt wie beispielsweise ein Sektglass.

Kurven können aber auch als Transformationspfad für Objekte dienen. So kann beispielsweise die Flugbahn eines Flugzeugs simuliert werden, das eine Kurve fliegt. Die Ausrichtung des Flugzeugs kann dabei von der Kurve abgeleitet werden (siehe Kapitel 3.4.1.2).

Im Gebiet der Computergrafik sind mehrere Typen von Kurven gebräuchlich: Die simpelste Art einer Kurve dürfte sicherlich die Geradengleichung y = m * x + b darstellen, jedoch gibt es darüber hinaus noch weitere (komplexere) mathematische Definitionen, um Kurven zu beschreiben. Dieses Unterkapitel wird jedoch ausschließlich Bézierkurven behandeln, da diese die Grundlage für Kapitel 3.6 bildet. Bézierkurven sind für ihre einfache Handhabung und Implementierung bekannt und daher bei Anwendungsentwicklern sehr beliebt. Im Falle dieser Arbeit werden sie eingesetzt, um Fußspuren entlang eines durch Bézierkurven definierten Pfades zu generieren (siehe Kapitel 3.6.2).

Geschichtliche Entwicklung

Die Bézierkurve wurde fast zeitgleich von Pierre Bézier und Paul de Casteljau in den 60er Jahren entdeckt. Beide waren für die Entwicklung von CAD-Software für den Automobilbau tätig - Pierre Bézier war Entwickler bei Renault während Paul de Casteljau für Citroën arbeitete. Die Bézierkurve wurde damals entwickelt, um Karosserieformen mit Hilfe von Computern zu gestalten. [Wikipedia (Bézierkurve), 2008]

3.4.1 Definition

Eine Bézierkurve n-ten Grades ist für $t \in [0,1]$ wie folgt definiert:

$$C(t) = \sum_{i=0}^{n} B_{n,i}(t) P_i$$

wobei

- $P_0 ... P_n$ gegebene Punkte seien, die das sogenannte Kontrollpolygon bilden.
- $B_{n,i}(t) = \binom{n}{i} t^i \, (1-t)^{n-i}$ das i-te Bernsteinpolynom n-ten Grad ist.

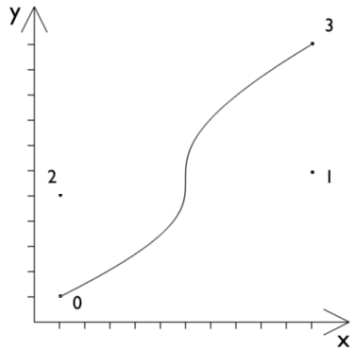


Abbildung 68: Beispiel einer Bézierkurve dritten Grades

Bézierkurven besitzen folgende Eigenschaften:

- Sie verlaufen immer durch den ersten und den letzten Kontrollpunkt
- Sie liegen innerhalb der konvexen Hülle des Kontrollpolygons
- Eine affine Transformation der Kurve lässt sich durch Transformation der Kontrollpunkte berechnen
- Die Tangente der Kurve an ihren Endpunkten entspricht immer der Richtung zum nächstgelegenen Kontrollpunkt, Pfade ohne Knick lassen sich dadurch leicht aus mehreren Bézierkurven erzeugen
- Jeder Kontrollpunkt beeinflusst die gesamte Kurve
- Für eine große Zahl von Kontrollpunkten werden sie numerisch instabil (durch die hohe Potenzierung einer Zahl kleiner Null im Term der Bernsteinpolynome)
- Sie können keine konischen Kurven repräsentieren (z.B. einen Kreisausschnitt oder andere Kegelschnitte). Dies ist nur durch eine Erweiterung, den *rationalen Bézierkurven,* möglich:

$$C(t) = \frac{\sum_{i=0}^{n} B_{i,n}(t)\, w_i\, P_i}{\sum_{i=0}^{n} B_{i,n}(t)\, w_i}$$

3.4.1.1 Bernsteinpolynome

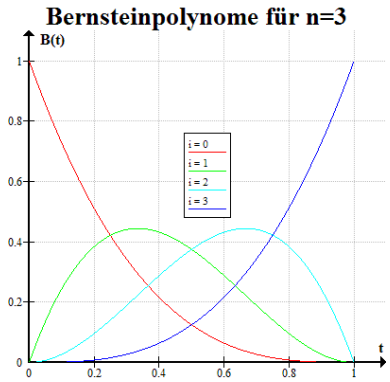


Abbildung 69: Bernsteinpolynome für n=3

Die Bernsteinpolynome fungieren bei Bézierkurven als sogenannte Gewichtungsfunktionen, d.h. sie bestimmen, wie stark ein Kontrollpunkt die Kurve „anzieht“. Die Bernsteinpolynome bilden eine Zerlegung der Eins, d.h. addiert man alle i Werte, die man für ein gegebenes $n \in \mathbb{N}$ und $t \in [0,1]$ erhält, so erhält man genau 1. Dadurch liegt die Bézierkurve immer innerhalb der konvexen Hülle des Kontrollpolygons.

3.4.1.2 Tangente

Die Tangente einer Bézierkurve lässt sich durch ihre Ableitung nach t berechnen:

$$C'(t) = n\sum_{i=0}^{n-1} B_{n-1,i}(t)(P_{i+1} - P_i)$$

3.4.2 Berechnung der Länge einer Bézierkurve

Um die Länge einer Bézierkurve zu bestimmen, ist der einfachste Ansatz der, dass man die Position einer bestimmten Anzahl von Punkten auf der Kurve berechnet. Dies geschieht, indem man in die Bézierkurvengleichung für t Werte gleichen Abstands zwischen 0 und 1 einsetzt. Anschließend ermittelt man, angefangen vom ersten Punkt, den Abstand zum jeweils nächsten Punkt und summiert diesen auf. Speichert man die Zwischenergebnisse in einer Tabelle, so lässt sich auch die Kurvenlänge an einem Punkt, bestimmt durch ein beliebiges $t \in [0,1]$, errechnen (sowie die Kurvenlänge zwischen zwei Punkten, die durch zwei verschiedene Werte für t bestimmt werden, siehe Listing 1).

3.4.2.1 Berechnung der Parameters t für eine gegebene Kurvenlänge s

Da sich t nicht proportional zur Kurvenlänge verhält, muss man einen Wert für t berechnen, wenn man die Position des Punktes bestimmen will, an der die Kurve eine bestimmte Länge s erreicht. Dieser kann mit der in Kapitel 3.4.2 beschriebenen Tabelle auf einfache Art und Weise bestimmt werden: Die Tabelle wird nach der ersten Zeile durchsucht, in welcher der in ihr gespeicherte Wert größer als s ist. t lässt sich daraufhin aus der Zeilennummer und der Anzahl der Zeilen (bzw. Punkten, in die die Kurve geteilt wurde) berechnen (siehe Listing 1).

Der Wert für t lässt sich auch durch den Einsatz von numerischer Integration berechnen, allerdings lieferte eine testweise Implementierung der Romberg Integration bei vergleichbarem Rechenaufwand ungenauere Werte, als der gerade beschriebene Algorithmus. [Eberly, 2000]

```
 1  function calculateLengthTable(Curve curve, int detail)
 2      Real lengthTable[detail]
 3      lengthTable[0] = 0
 4
 5      Vector3 last = curve.getPosition(0)
 6      for(int i = 0..detail - 1)
 7          Real t = i / detail – 1
 8          Vector3 current = curve.getPosition(t)
 9
10          lengthTable[i] = lengthTable[i – 1] + length(current – last)
11          last = current
12
13      return lengthTable
```

```
14 end function
15
16 function getLength(Curve curve, int detail, Real lengthTable[detail])
17     return lengthTable[detail – 1]
18 end function
19
20 function getLengthFromT(Curve curve, int detail, Real lengthTable[detail],
                          Real t)
21     if(t <= 0)
22         t = 0
23     if(t >= 1)
24         t = 1
25     t = t * (detail – 1)
26
27     int i = t
28     Real l = lengthTable[i]
29     if(i < (detail – 1))
30         l += (t – i) * (lengthTable[i + 1] – lenghtTable[i])
31     return l
32 end function
33
34 function getLengthFromT2(Curve curve, int detail, Real lengthTable[detail],
                           Real t0, Real t1)
35     return getLengthFromT(curve, detail, lengthtable, t1) -
         getLengthFromT(curve, detail, lengthtable, t1)
36 end function
37
38 function getTFromLength(Curve curve, int detail, Real lengthTable[detail],
                          Real length)
39     Real t = length / getLength(curve, detail, lengthTable);
40     if(t <= 0)
41         return 0
42     if(t >= 1)
43         return 1
44
45     int iMin = 0, iMax = detail - 1
```

```
46      int i = t * (detail – 1)
47      while((iMax – iMin) > 1)
48          Real l = lengthTable[i]
49          if(l < length)
50              iMin = i
51          else if(l > length)
52              iMax = i
53          else
54              iMin = iMax = i
55              break
56
57          i = (iMax - iMin) / 2 + iMin
58
59      Real lMin = lengthTable[iMin]
60      Real lMax = lengthTable[iMax]
61      length = length - lMin
62
63      t = ((length / (lMax – lMin)) + iMin) / (detail – 1)
64      return t
65  end function
```

Listing 1 Berechnung der Länge von Bézierkurven

3.5 Inverse Kinematik

3.5.1 Grundlagen

Die Kinematik (vom griechischen Wort *kinema* stammend, dt. „Bewegung"), beschäftigt sich mit der Berechnung der Bewegungen von Punkten und Körpern im Raum. Sie definiert dabei deren Größen wie Geschwindigkeit, Beschleunigung und die Strecke, die von einem Punkt oder Körper in einer bestimmten Zeit zurückgelegt wird. Punkte sind dabei im dreidimensionalen Raum durch drei Koordinaten definiert. Da sie voneinander unabhängig sind und zusammen die Lage des Punktes im Raum eindeutig beschreiben, spricht man in diesem Zusammenhang auch von **Freiheitsgraden**. Körper besitzen darüber hinaus drei weitere Freiheitsgrade, die die Rotation des Körpers im dreidimensionalen Raum eindeutig bestimmen.

Teilgebiete der Kinematik sind unter anderem die direkte Kinematik und die inverse Kinematik. Beide beschäftigen sich mit sogenannten **Kinematischen Ketten**. Diese Ketten beschreiben eine Anzahl von Körpern, die durch Gelenke miteinander verbunden sind, wie beispielsweise einen Industrieroboter. Das letzte Glied der Kette (an welchem bei einem Industrieroboter das Werkzeug angebracht ist) wird **Endeffektor** genannt. (Im englischen wird der Bezug zur Robotik deutlicher, hier wird er als Tool Center Point, kurz TCP bezeichnet.)

Gelenke

Die Gelenke bilden die Schnittstelle zwischen den Körpern, die sie verbinden. Sie können theoretisch bis zu sechs Freiheitsgrade besitzen; jeweils drei für Translation und Rotation. Es werden jedoch selten Gelenke benötigt, die mehr als drei Freiheitsgrade besitzen oder Translationen zulassen. Die Gelenke eines menschlichen Körpers beispielsweise erlauben ausschließlich Rotationen. Diese können, wie jeder Freiheitsgrad eines Gelenks, eingeschränkt sein. Diese Beschränkungen werden auch als **Constraints** bezeichnet. Sie modellieren physikalische Gegebenheiten, durch die sich ein Gelenk nur bis zu einem gewissen Anschlag drehen oder bewegen kann.

Direkte Kinematik

Die direkte Kinematik beschäftigt sich mit der eher trivialen Aufgabe, die Position und Ausrichtung des Endeffektors aus den Gelenkparametern zu berechnen. Sie ist für Kinematische Ketten immer eindeutig lösbar. Die Gelenke lassen sich durch Rotations- und Translationsmatrizen darstellen, wodurch die Lage des Endeffektors sich durch simple Matrizenmultiplikation errechnet lässt, wie sie auch in der Computergrafik zum Einsatz kommt.

Inverse Kinematik

Die inverse Kinematik (**IK**) ist die Umkehrrechnung zur direkten Kinematik. Mit ihrer Hilfe ist es möglich, die Gelenkparameter so zu bestimmen, dass der Endeffektor sich in einer gegebenen Position (und optional auch Ausrichtung) befindet. Diese Aufgabe ist nicht trivial, da sich je nach Kinematischer Kette und gewünschter Position (und Ausrichtung) des Endeffektors keine, eine oder mehrere Lösungen (auch Konfigurationen genannt) ergeben können.

Bezug zur Computergrafik

Kinematische Ketten können neben der Robotik auch im Bereich der Computergrafik nützlich sein: So besteht ein Skelet, wie es bei der Skeletal Animation (siehe Kapitel 3.2.1) zum Einsatz kommt, ebenfalls aus einer Reihe von Körpern, welche durch Gelenke miteinander verbunden sind. So lässt sich beispielsweise beginnend von jedem Knochen eine Kinematische Kette zum Root-Bone definieren, durch die mit Hilfe der Direkten Kinematik die

Position des Knochens relativ zum Root-Bone bestimmt werden kann. Umgekehrt können mittels Inverser Kinematik die Gelenkwinkel bestimmt werden, um den besagten Knochen in eine bestimmte Position zu bringen.

3.5.2 Lösungsmethoden

Um die (Rotations- und Translations-)Parameter der Gelenke einer Kinematischen Kette zu bestimmen, bietet die Inverse Kinematik mehrere Lösungsansätze. Sie lassen sich in analytische und numerische Methoden gliedern:

- Analytische Methoden versuchen eine gültige Konfiguration durch Lösen von Gleichungssystemen oder Einsatz von geometrischen Funktionen zu lösen. Allerdings können sie in der Regel nicht für beliebige Kinematische Ketten eingesetzt werden, sondern beschränken sich beispielsweise auf zwei oder drei Gelenke oder sind nicht leicht zu implementieren.
- Numerische Methoden versuchen hingegen sich in mehreren Schritten (*Iterationen*) einer Lösung anzunähern. Sie sind meist leichter zu implementieren und bieten die Möglichkeit, Rechenzeit auf Kosten der Genauigkeit zu sparen, weshalb sie für interaktive Anwendungen wesentlich besser geeignet sind. Beispiele für numerische Lösungsansätze sind der Cyclic Coordinate Descent Algorithmus [Weber, 2002] und die Jacobian Transpose Methode [Spoerl, 2004]

3.6 Folgen von Fußspuren

In diesem Kapitel geht es um die Animation von Charakteren, die sich zu Fuß fortbewegen, indem sie beispielsweise schleichen, gehen oder rennen. Hierbei soll speziell auf die Animation eines gehenden Charakters eingegangen werden. Die anderen Gangarten lassen sich leicht durch Anpassung der Animationsparameter davon ableiten.

Während ein virtueller Charakter geht, bewegt er sich auf einem Pfad durch eine virtuelle Welt, der in der Regel nicht vorherbestimmt werden kann. Daher ist es meist nicht möglich, eine akkurate Animation für den Bewegungsablauf des Charakters im Voraus zu erstellen. Sollen in einer solchen Situation dennoch vorgefertigte Animationen zum Einsatz kommen, müssen entweder Ungenauigkeiten in Kauf genommen werden oder es müssen strenge Einschränkungen getroffen werden. Beispielsweise können die Bewegungsmöglichkeiten des Charakters stark limitiert werden, indem eine Reihe von Aktionen definiert wird, mittels denen der Charakter sich bewegen kann, wie:

- Beginne zu gehen (schleichen oder rennen)
- Gehe (schleiche oder renne) einen Schritt nach vorne (hinten, links oder rechts)
- Drehe um 45° (90° oder 180°) nach links (oder rechts)
- Springe aus dem Stehen (schleichen, gehen oder rennen)
- Bleibe aus dem Gehen (schleichen, rennen heraus) stehen

Zusammen sind dies bereits 29 Aktionen, mit deren Hilfe der Charakter stehen und sich in 3 verschiedenen Geschwindigkeiten bewegen kann. Für jede Aktion muss dabei eine eigene statische Animation angefertigt werden. Dennoch ist der Charakter in seiner Bewegungsfreiheit sehr eingeschränkt: Er kann sich - (fast) wie eine Schachfigur auf einem Schachbrett – nur um ein ganzzahliges Vielfaches seiner Schrittlänge voran bewegen und sich nur in einem bestimmten Raster drehen. Außerdem bekommt der Anwender ständig dieselben Animationen zu sehen. Alles in allem wirkt ein so animierter Charakter sehr mechanisch. Allerdings kann dadurch, dass die Bewegungsfreiheit in solche kleine, genau definierte Stückchen geteilt wird, die Animation mit hundertprozentiger Genauigkeit im Voraus erstellt werden.

Will man mehr Bewegungsfreiheit für den Charakter ohne eine explosionsartig große Anzahl von Animationen, so ist dies möglich, indem der Charakter nicht exakt zur Animation passend bewegt wird. Er kann beispielsweise langsamer oder schneller vorwärts bewegt werden und mehr oder weniger gedreht werden, als es die entsprechende Animation vorsieht. Dies hat jedoch zur Folge, dass die Füße auf dem Boden in einer seltsamen Art und Weise zu rutschen scheinen. Eine andere Möglichkeit ist es, die Bewegungen passend zur Animation zu belassen und die Animationen nur zu einem Teil abzuspielen. Soll der Charakter beispielsweise nur einen halben Schritt nach vorne gehen, so spielt man die Animation nur bis zur Hälfte ab. Allerdings passt dann die Animation für den Übergang zwischen Gehen und Stehen nicht mehr nahtlos und die Gliedmaßen des Charakters wechseln sprungartig die Position und Ausrichtung im Raum. Dieser Effekt kann zwar durch Interpolation vermindert, jedoch nicht vollständig versteckt werden. Außerdem tritt meist durch die Interpolation wieder der zuerst genannte Effekt auf.

Die einzige Möglichkeit, diese Nachteile zu vermeiden, ist die Animation dynamisch zur Laufzeit zu berechnen. Ein Ansatz hierfür soll im Folgenden vorgestellt werden.

3.6.1 Wegfindung

Wird ein Charakter in einer virtuellen Welt nicht oder nicht direkt vom Benutzer gesteuert, so muss der Computer einen Pfad berechnen, der den Charakter von seinem Ausgangspunkt an sein Ziel bringt. Dabei müssen Hindernisse wie beispielsweise Gegenstände beachtet werden. Da diese, genauso wie Ausgangs- und Zielpunkt meist eine unvorhersehbare Position besitzen, muss die Berechnung des Wegs zur Laufzeit erfolgen. Wegfindungsalgorithmen übernehmen diese Aufgabe. Ihr Ziel ist es, einen möglichst kurzen Weg um die Hindernisse herum zu finden. Sie sind daher eng mit einem Problem der Graphentheorie verwandt, einen möglichst kurzen Pfad zwischen zwei gegebenen Knoten zu finden und stellen ein eigenes Forschungsgebiet dar. Eine gute Einführung in das Thema bieten die Unterlagen des Seminars "3D Pathfinding", welches im Juni 2006 an der TU München gehalten wurde. In ihnen findet man auch eine mögliche Implementierung eines Wegfindungsalgorithmuses [Kastenholz, 2006].

3.6.2 Generieren einer Fußspur

Berechnung eines Pfades mit Hilfe von Bézierkurven

Meistens ist der Weg, den der Wegfindungsalgorithmus errechnet sehr eckig, da er nur aus einigen Wegpunkten besteht, die durch gerade Segmente miteinander verbunden sind. Bewegt sich ein Charakter entlang eines solchen Pfades, so würde er an jedem Wegpunkt sprunghaft seine Ausrichtung ändern. Alternativ kann er an jedem Wegpunkt kurz stehen bleiben, um sich in die Richtung des nächsten Wegpunktes zu drehen. Beide Möglichkeiten sehen jedoch einem Bewegungsablauf eines Menschen, welcher an Hindernissen vorbei einen Zielpunkt ansteuert, nicht sehr ähnlich. Ein Mensch läuft, sofern genügend Platz vorhanden ist, in mehr oder weniger engen Kurven um die Hindernisse herum. Um dieses Verhalten zu imitieren und den Bewegungsablauf weniger roboterhaft aussehen zu lassen, wird zwischen jeweils zwei Wegpunkten eine Bézierkurve mit vier Kontrollpunkten gelegt.

Der erste und der vierte Kontrollpunkt einer solchen Bézierkurve entsprechen den beiden Wegpunkten zwischen denen sie liegt. Da die Bézierkurve durch diese Maßnahme direkt durch die Wegpunkte hindurch führt, läuft der Charakter nicht in bzw. durch ein Hindernis, welches in der Nähe des Wegpunkts liegen könnte. (Hindernisse sind meistens in der Nähe von Wegpunkten, da viele Wegfindungsalgorithmen die Wegpunkte dicht entlang von Hindernissen generieren).

Der zweite und dritte Kontrollpunkt der Bézierkurve ergeben sich anhand der Ausrichtung, die der Charakter am jeweiligen Wegpunkt haben soll, und der Schrittweite mit welcher der Charakter den Pfad entlang schreitet. Die Ausrichtung

- am ersten Wegpunkt des Pfades ergibt sich dabei aus der Richtung, in die der Charakter gerichtet ist, wenn er den Pfad betritt. Steht der Charakter bereits an diesem Punkt (was der typische Fall sein dürfte), so entspricht sie der Richtung, in der sich der Charakter zu diesem Zeitpunkt befindet.
- am letzten Wegpunkt des Pfades ergibt sich aus der Richtung, in die der Charakter nach Erreichen seines Zieles gerichtet sein soll. Zwar könnte der Charakter sich nach Erreichen des Ziels noch in die gewünschte Richtung drehen, jedoch kennt ein Mensch meist sein Ziel und weiß, wie er dort stehen will, weshalb er ebenfalls beim darauf Zugehen typischerweise eine (kleine) Kurve einplant.
- an den Wegpunkten zwischen dem ersten und dem letzten Wegpunkt ergibt sich aus den Richtungen, in die der jeweils vorherige bzw. nachfolgende Wegpunkt des Pfades liegt. Sie wird so gewählt, dass der Charakter an dem betroffenen Wegpunkt bereits zur Hälfte vom vorherigen weggedreht und dem nächsten zugewandt ist. Sie lässt sich wie in Listing 2 beschrieben berechnen.

```
1  Pos0 = Wegpunkt[i-1].position
2  Pos1 = Wegpunkt[i].position
3  Pos2 = Wegpunkt[i+1].position
4
5  DirPrev = normalize(Pos1 – Pos0)
6  DirNext = normalize(Pos2 – Pos1)
7
8  Wegpunkt[i].direction = normalize(-DirPrev + DirNext)
```

Listing 2 Berechnung der Ausrichtung an den inneren Wegpunkten eines Pfades

Der zweite bzw. dritte Kontrollpunkt der Bézierkurve liegt ein Vielfaches der Schrittlänge vom Wegpunkt entfernt, an der die Bézierkurve startet bzw. endet. Er liegt vom diesem Punkt aus gesehen in bzw. entgegen der für diesen Wegpunkt berechneten Ausrichtung (siehe Listing 3). Als guter Faktor hat sich dabei das Doppelte der Schrittlänge herausgestellt. Ein kleinerer Faktor erzeugt kleinere, schärfe Kurven während ein größerer Faktor zu großen Schleifen an Wegpunkten führt, an welchem der vom Wegfindungsalgorithmus generierte Pfad einen scharfen Knick macht. An Wegpunkten, an dem der Pfad kein oder nur einen geringen Knick macht, muss der Faktor so gewählt werden, dass sich die Summe der sich aus dem Faktor ergebenden Abstände kleiner ist, als der Abstand zwischen den Wegpunkten. Anderenfalls ist die Bézierkurve in sich verschlungen (siehe Abbildung 70, rechts unten).

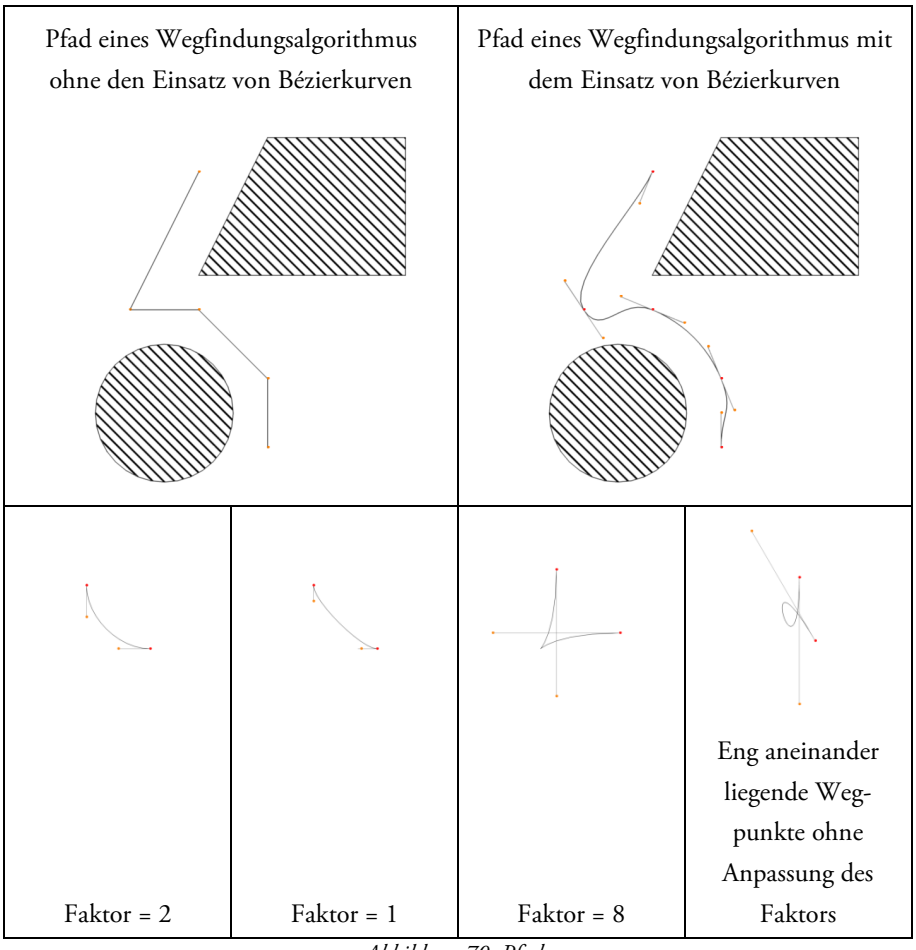


Abbildung 70: Pfade

```
f = 2 * Schrittweite
d = length(Wegpunkt[i+1].position – Wegpunkt[i].position)
w = dotProduct(Wegpunkt[i+1].direction – Wegpunkt[i].direction)
if(arccos(w) > cos(Grenzwinkel) && d < (2 * f))
    f = d / 2

Pos0 = Wegpunkt[i].position
Pos1 = Wegpunkt[i].position + (f * Wegpunkt[i].direction)
Pos2 = Wegpunkt[i+1].position – (f * Wegpunkt[i+1].direction)
Pos3 = Wegpunkt[i+1].position

kurve = BezierCurve(Pos0, Pos1, Pos2, Pos3)
```

Listing 3 Berechnung der Kontrollpunkte der Bézierkurve zwischen zwei Wegpunkten

Ausrichtung der Fußspur entlang des Pfades

Da der Pfad nun gegeben ist, auf dem der Charakter sich bewegt, können nun die **Fußstapfen** berechnet werden, die der Charakter am Boden hinterlassen wird. Die Fußstapfen werden benötigt, um die Bewegung der Füße und Beine des Charakters zu berechnen, während er sich fortbewegt. Die Position der Stapfen ist in der Regel fest, außer der Charakter bewegt sich auf beispielsweise vereisten Untergrund und rutscht. Dieser Spezialfall wird allerdings nicht im Rahmen dieser Arbeit behandelt.

Ein Fußstapfen wird definiert durch:

- die Position des **Fuß-Knochens** (z.B. „Sophie Biped L Foot“) und des **Zeh-Knochens** (z.B. „Sophie Biped L Toe0“) während der Fuß des Charakters auf dem Fußstapfen steht. Die Position des Fuß-Knochens liegt dabei innerhalb des Fersen, etwas oberhalb des Bodens. Die Position des Zeh-Knochens befindet sich an der Spitze des Fußes, nur knapp oberhalb des Bodens.
- die Pfadlänge zum Startpunkt des Pfades.
- den Zeitpunkt und die Zeitspanne, die der Fuß eines Charakters auf dem Fußstapfen verweilt, während er den Pfad entlang schreitet.

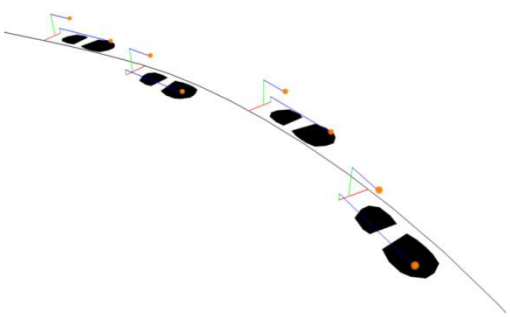


Abbildung 71: Fußstapfen mit Offsetpositionen der Knochen

Die Anzahl der Fußstapfen entlang des Pfades ergibt sich aus der Länge des Pfades, geteilt durch die Schrittlänge, mit der der Charakter den Pfad entlang schreitet. Die Länge des Pfades errechnet sich aus der Summe der Längen der Bézierkurven (siehe Kapitel 3.4.2) und die Schrittlänge ist abhängig von der Größe des Charakters und der Gangart, mit der sich der Charakter bewegt. Sie lässt sich entweder aus einer vordefinierten, statischen Animation, die einen Bewegungszyklus darstellt, ableiten[26] oder sie wird manuell für die verschiedenen Gangarten festgelegt, falls keine entsprechenden vorgefertigten Animationen zur Verfügung stehen.

Die Anzahl der Fußstapfen wird gerundet, um statt eines kleinen Schrittes am Ende des Pfades gleichmäßig verteilte Schritte mit einer leicht verkleinerten bzw. vergrößerten Schrittlänge zu erhalten. Diese *angepasste Schrittlänge* errechnet sich durch die Umkehrrechnung (Pfadlänge geteilt durch gerundete Anzahl von Schritten). Sie wird für die Positionierung der Fußstapfen und die Berechnung der Bewegung benötigt. Um die Fußstapfen entlang des Pfades auszurichten, müssen die Positionen, die sie definieren, bestimmt werden. Sie ergeben sich aus einer gemeinsamen Basisposition je Fußstapfen und einer getrennten Offsetposition des jeweiligen Knochens. Diese Offsetposition wird auf die Basisposition aufsummiert. Die Basispositionen liegen dabei direkt auf dem Pfad, jeweils eine *angepasste Schrittlänge* entfernt. Die Offsetpositionen ergeben sich aus den Positionen, die der Fuß-Knochen und der Zeh-Knochen haben, wenn der Charakter am Nullpunkt des Koordinatensystems steht und die Füße den Boden berühren. Insgesamt ergeben sich so vier Offsets, zwei je Fuß (siehe Abbildung 71 und Listing 4, calculateBonePosition).

3.6.3 Bewegungsablauf

3.6.3.1 Bewegung von Beinen und Füßen

Die Beine sind das Bindeglied zwischen den Füssen und dem Rest des Körpers eines Charakters. Während die Füße beinahe jede beliebige Ausrichtung entsprechend der Bodenneigung und Laufrichtung annehmen können, befindet sich der Oberkörper beim Gehen normalerweise in einer aufrechten Position. Außerdem kann es beim Laufen zu Höhenunterschieden zwischen dem linken und dem rechten Fuß kommen (beispielsweise wenn der Charakter den Fuß anhebt, um ihn nach vorne zu bewegen oder er sich auf unebenem Untergrund bewegt). Die Beine müssen deshalb sowohl die Rotations- als auch Translations-Unterschiede relativ zur Hüfte ausgleichen.

Bewegung des Ober- und Unterschenkels

Für die Animation der Beine ergibt sich eine unendliche Anzahl von Möglichkeiten, weil sich die Füße, je nach Untergrund, Gangart und Bewegungspfad des Charakters, relativ zur Hüfte gesehen, unvorhersehbar bewegen. Somit kann für die Ausrichtung und Position der Beine keine Aussage im Voraus getroffen werden. Diese werden deswegen mit Hilfe von Inverser Kinematik (siehe Kapitel 3.5) errechnet. Der Oberschenkelknochen und das Schien- bzw. Wadenbein bildet dabei eine Kinematische Kette mit entsprechender Rotationsbeschränkung[27] für das Kniegelenk. Als Endeffektor fungiert der Fußknochen. Mit dessen Hilfe kann der Fuß in jede physikalisch mögliche Position und Ausrichtung relativ zur Hüfte gesehen, gebracht werden. Die Position und Rotation des Ober- und Unterschenkels können daraufhin mit Hilfe eines IK-Algorithmus bestimmt werden.

Bewegung der Füße

Damit der Charakter einer Fußspur folgt, wie sie in Kapitel 3.6.2 beschrieben wird, müssen seine Füße von Fußstapfen zu Fußstapfen bewegt werden, während der Oberkörper dem Pfad folgt. Der Weg, den ein Fuß dabei nimmt,

[26] durch die Bestimmung der Entfernung, die sich der Fuß nach dem Hochheben nach vorne bewegt (siehe Kapitel 3.6.3)

[27] diese wird manuell durch den Grafiker des virtuellen Charakters festgelegt.

wird zum einen durch die Zeit, die der Fuß auf einem Fußstapfen verweilt und zum anderen durch zwei Animationskurven bestimmt. Jeweils eine Animationskurve beschreibt dabei die Bahn, die der Fuß-Knochen bzw. der Zeh-Knochen zurücklegt, während der Charakter seinen Fuß beim Gehen (einer geraden Strecke) nach vorn bewegt. Um diese Animationskurven zu erhalten, wird eine vorgefertigte, statische Animation des Skelettes benötigt, wie es zwei Schritte vorwärts geht (ein sogenannter **Walk-Cylce**). Diese Animationskurven werden bei jedem Schritt an die Krümmung des Pfads angepasst, den der Charakter zurücklegt.

Bestimmung der Animationskurven

Um eine Animationskurve aus dem Walk-Cycle zu extrahieren, wird für jeden Keyframe der Skeletal Animation die Position des Fuß- und des Zeh-Knochens relativ zur Position des Root-Bones des Skelets bestimmt und zusammen mit dem Zeitindex des Keyframes gespeichert (dies ergibt einen sogenannten **Keypoint**). Liegt die Höhe der errechneten Position unterhalb eines Grenzwertes, so wird der Keypoint ignoriert (d.h. nicht zu der Menge der Keypoints hinzugefügt, welche die Animationskurve definieren), da sich der Fuß zu diesem Zeitpunkt auf dem Boden befindet. Der Grenzwert kann von der Höhe abgeleitet werden, die der Fuß- bzw. Zeh-Knochen hat, wenn der Charakter mit beiden Füssen fest auf dem Boden steht. Ein solcher Fall ergibt sich beispielsweise zu Beginn der Animation, welche das Skelett des Charakters beim Loslaufen darstellt (siehe Listing 4, getAnimationCurve).

Da sich der Root-Bone des Charakters während des Walk-Cycles nicht vorwärts bewegt, muss seine Vorwärtsbewegung simuliert werden. Dazu muss die Schrittlänge des Charakters, falls sie nicht vorgegeben ist, errechnet werden. Sie ergibt sich aus dem Abstand zwischen dem Anheben und dem Absetzen des Fußes und entspricht somit der Länge des Vektors zwischen dem ersten und dem letzten Keypoint der Animationskurve. Selbiger Vektor bestimmt zugleich die Richtung, in der sich der Charakter beim Gehen bewegt. Die Bewegungsgeschwindigkeit kann nun errechnet werden, indem man die Schrittlänge durch den Zeitunterschied zwischen dem ersten und dem letzten Keypoint teilt. Anschließend multipliziert man die Bewegungsgeschwindigkeit mit der Bewegungsrichtung, um einen Bewegungsvektor zu erhalten. Dieser wiederum wird mit dem Zeitindex des jeweiligen Keypoints multipliziert und auf die Position des Keypoints aufsummiert. (siehe Listing 4, applyForwardMovement)

Damit die Animationskurve später der Krümmung des Pfades, den der Charakter während einem Schritt zurücklegt, angepasst werden kann, muss sie in eine „normalisierte“ Form gebracht werden. Dazu wird die Animationskurve in Laufrichtung so skaliert, dass sie der Schrittlänge von genau einer Einheit entspricht. Dadurch kann sie später mittels einer simplen Multiplikation in die gewünschte Schrittlänge gebracht werden. Außerdem muss der Offset, des zur Animationskurve gehörigen Knochens, abgezogen werden. Er entspricht genau dem Offset, der bereits zur Berechnung der Fußstapfen einer Fußspur benötigt wurde (siehe Kapitel 3.6.2). Das ist notwendig, weil der Offset zum einen bereits in der Position der Fußstapfen enthalten ist, zum anderen wird durch diese Maßnahme verhindert, dass die Position des Fuß-Knochens und des Zeh-Knochens ebenfalls der Krümmung der Kurve folgen. (Der Fuß dreht sich durch die Fußspur passend zur Krümmung der Kurve, aber es soll verhindert werden, dass der Fuß selbst die Form der Kurve annimmt). (Siehe Listing 4, normalizeCurve).

Um die Animationskurve der Krümmung des Pfades folgen zu lassen, müssen die Positionen der Keypoints angepasst werden. Dazu werden die Pfadlängen des vorhergehenden und nachfolgenden Fußstapfens benötigt, zwischen der die Animationskurve liegt. Aus ihrer Differenz berechnet sich die Schrittlänge dieses Schritts, mit welcher die Animationskurve in Laufrichtung skaliert wird. Anschließend können die Keypoints entsprechend ihrer Distanz in Laufrichtung auf dem Pfadabschnitt zwischen den beiden Fußstapfen verteilt werden. Dazu berechnet man für jeden Keypoint die Position auf dem Pfadabschnitt zwischen den Fußstapfen, welche dieselbe Distanz zum hinteren Fußstapfen hat, wie der aktuelle Keypoint vom ersten Keypoint der Animationskurve in Laufrichtung entfernt liegt. Die eben berechnete Position ist die Basisposition, auf die der Offset des Fuß- und Zeh-Knochens addiert wird, die bereits in Kapitel 3.6.2 berechnet wurde. Dieser Offset muss jedoch vorher noch an die Ausrichtung des Pfades an der Basisposition angepasst werden. Dazu wird mittels der Tangente und einem Vektor, der nach oben zeigt, eine Orthonor-

malbasis berechnet, aus der sich die Rotationsbewegung ableiten lässt, mit welcher der Offset transformiert werden muss (siehe Listing 4, getPathPosition).

Timing der Animation

Durch die Fußstapfen und der an den Pfad angepassten Animationskurven zwischen den Fußstapfen, sind die Bewegungsbahnen der Füße des Charakters definiert. Die Fußstapfen beinhalten einen Zeitindex, der den Zeitpunkt und die Verweildauer des Fußes auf ihm bestimmt (siehe Kapitel 3.6.2). Der Zeitindex, der in den Keypoints der Animationskurven gespeichert ist, gibt die nach dem Verlassen des Fußstapfens vergangene Zeit an. Durch diese Informationen lässt sich Position des Fußes zu jedem Zeitpunkt genau zu bestimmen.

```
 1  function calculateBonePosition(Bone bone, Animation anim, Time t)
 2      Matrix4x4 mat = Matrix4.IDENTITY
 3      while(bone != null)
 4          Matrix4 temp = bone.getTransformationMatrix()
 5
 6          Keyframe kf = anim.getInterpolatedKeyframe(bone, t)
 7          if(kf != null)
 8              temp = temp * kf.getTransformationMatrix()
 9
10          mat = temp * mat
11          bone = bone.getParentBone()
12
13      return mat * Vector3.ZERO
14  end function
15
16  function getForwardMovement(AnimationCurve curve)
17      int last = curve.getKeypointCount() - 1
18      return curve.getKeypoint(last).position - curve.getKeypoint(0).position
19  end function
20
21  function normalizeCurve(AnimationCurve curve, Vector3 offset)
22      Vector3 fwd = getForwardMovement(curve)
23      Real scale = 1 / length(fwd)
24      Matrix3x3 scaleMat =
            Matrix3x3.makeScaleMatrix(scale, 1, 1) *
            Matrix3x3.makeRotMatrixFromDirs(Vector3.UNIT_X, normalize(fwd))
25
```

```
26      AnimationCurve normalizedCurve
27      for(int i = 0..(curve.getKeypointCount() – 1))
28          Keypoint kp = curve.getKeypoint(i)
29
30          kp.position = kp.position – offset
31          kp.position = scaleMat * kp.position
32          normalizedCurve.appendKeypoint(kp)
33
34      return normalizedCurve
35  end function
36
37  function applyForwardMovement(AnimationCurve curve)
38      int points = curve.getKeypointCount() - 1
39      Vector3 fwd = getForwardMovement(curve)
40      Real time = curve.getKeypoint(points).time - curve.getKeypoint(0).time
41      Real factor = length(fwd) / time
42
43      Curve ret
44      for(int i = 0..points)
45          Keypoint kp = curve.getKeypoint(i)
46          kp.position = kp.position + kp.time * factor * fwd
47          ret.appendKeypoint(kp)
48
49      return ret
50  end function
51
52  function getAnimationCurve(Bone bone, Animation walkCycle,
                               Animation startWalk, Real delta)
53      Vector3 offset = calculateBonePosition(bone, startWalk, 0)
54      Real h += offset.y + delta
55
56      AnimationCurve curve
57      int frames = walkCycle.getKeyFrameCount(bone)
58      for(int f = 0..frames - 1)
59          Keyframe kf = anim.getKeyframe(f)
```

```
60
61          Keypoint kp
62          kp.position = calculateBonePosition(bone, walkCycle, kf.time)
63          Kp.time = kf.time
64
65          if(kp.position.y < h)
66              continue
67          curve.appendKeypoint(kp)
68
69      curve = applyForwardMovement(curve)
70      curve = normalizeCurve(curve, offset)
71
72      return curve
73  end function
74
75  function getPathPosition(Path path, Real l, Vector3 offset, Vector3 up)
76      Vector3 pos = path.getPosition(l)
77
78      Vector3 x = path.getTangent(l)
79      Vector3 z = crossProduct(x, normalize(up))
80      Vector3 y = crossProduct(z, x)
81
82      pos = pos + offset.x * x + offset.y * y + offset.z * z
83      return pos
84  end function
```

Listing 4 Bestimmung der Animationskurven

3.6.3.2 Bewegung der Hüfte und des Oberkörpers

Die Bewegung der Hüfte und des Oberkörpers werden beim Gehen des Charakters durch die statische Animation des Walk-Cycles bestimmt. Die Animation wird mit den Schritten des Charakters synchronisiert, indem ihre Abspielgeschwindigkeit angepasst wird. Der Faktor für die Abspielgeschwindigkeit errechnet sich aus dem Zeitindex, welcher in den Fußstapfen gespeichert ist.

Obwohl oben genannter Ansatz schon zu recht guten Ergebnissen führt, können bessere Ergebnisse erzielt werden, wenn die Animation des Oberkörpers ebenfalls dynamisch generiert wird. Dazu kann NaturalMotion's euphoria verwendet werden (siehe Kapitel 3.3.6).

3.6.3.3 Bewegung des Kopfes

Für den Kopf hingegen kann durch einfachere Mittel eine dynamische Animation berechnet werden. In der Spielwelt lassen sich beispielsweise Hotspots definieren, die die Blicke des Charakters auf sich lenken können, während er sich bewegt. Dafür kann ein System entwickelt werden, das anhand einiger Kriterien einen Hotspot auswählt, den der Charakter in einer gegebenen Situation anvisiert. Für die Animation des Kopfes des Charakters lässt sich aus der Position des Kopfes und der Position des Hotspots die Blickrichtung bestimmen. Der Kopf kann daraufhin durch Einsatz der Skeletal Animation in diese Richtung gedreht werden.

Auch kann der Gemütszustand des Charakters (sofern er von der Applikation bekannt ist oder von ihr simuliert wird) die Animation des Kopfes beeinflussen. So kann der Charakter „den Kopf hängen lassen“, wenn er schlecht gelaunt oder müde ist.

3.7 Darstellung eines Charakters in Echtzeit

3.7.1 Funktionsweise aktueller Grafikkarten

Damit die virtuelle Welt, inklusive ihrer Charaktere, auf dem Bildschirm für den Benutzer sichtbar wird, muss sie *gerendert* werden. Dies geschieht bei interaktiven Anwendungen in der Regel durch eine Grafikkarte, deren Chip (auch GPU genannt) auf die Berechnung von dreidimensionalen Bildern optimiert ist. Heutige Grafikkarten sind auf die Darstellung von Punkten, Linien und Dreiecken optimiert. Mit ihrer Hilfe lässt sich die Oberfläche jedes beliebigen, dreidimensionalen Körpers annähern. Zur Darstellung einer solchen Oberfläche auf dem Bildschirm transformiert eine GPU zunächst die geometrischen Grundformen entsprechend der Position und Ausrichtung des dargestellten Körpers (relativ zum Betrachter). Anschließend werden die Grundformen auf eine zweidimensionale Ebene projiziert, um ihre Position auf dem Bildschirm bestimmen zu können. Sie können sie *rasterisiert* werden. Um die Farbe, der sich dabei ergebenden Pixel, zu bestimmen, kommen sogenannte Materialen zum Einsatz. Sie beschrieben die Oberflächeneigenschaften des Körpers, wie beispielsweise die Farbe der Oberfläche oder die Art und Stärke der Lichtreflektion.

Shader

Bei neueren Generationen von Grafikkarten werden die Materialien unter anderem durch kleine Programme definiert, die während des Renderns auf der GPU ausgeführt werden. Diese werden **Shader** genannt. Mit ihrer Hilfe lässt sich neben der Farbe auch die Form der Oberfläche beeinflussen. Mit zukünftigen Generationen von Grafikkarten kann zusätzlich der Detailgrad der Oberfläche mit Hilfe der Shader bestimmt werden.

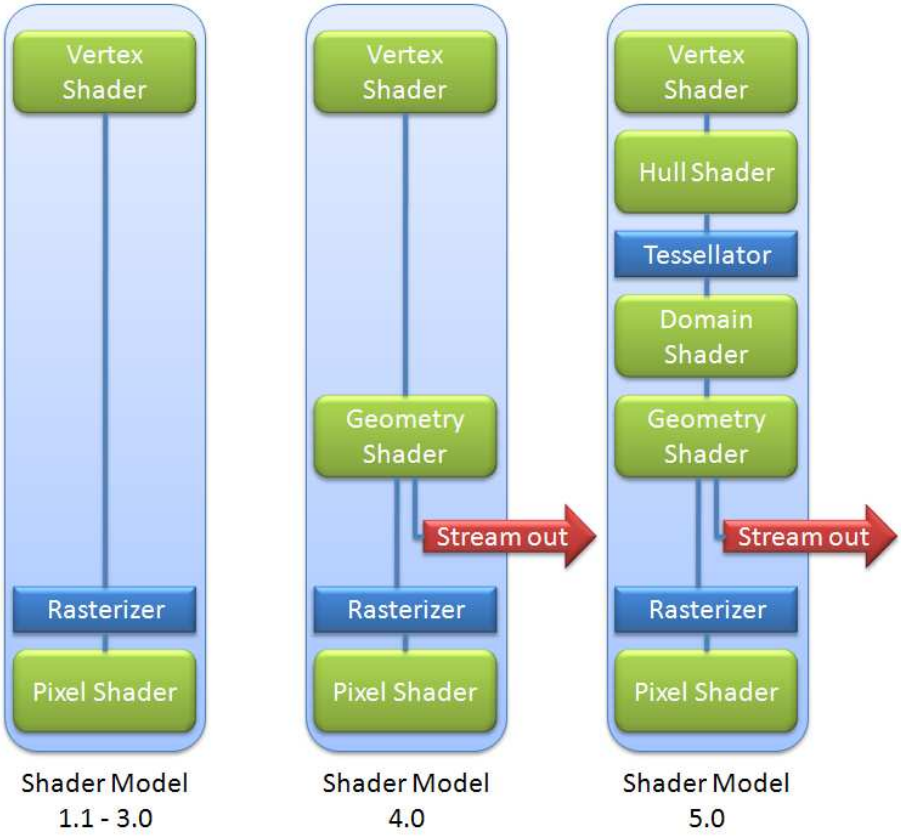


Abbildung 72: Renderpipeline der verschiedenen Shader-Model-Versionen

Die Möglichkeiten, die die Shader einer Grafikkarte haben, werden dabei von der Version des *Shader Models* bestimmt. Sie gibt (neben technischen Einschränkungen) an, welche Kategorien von Shadern vorhanden sind (siehe Abbildung 72). Die Kategorie eines Shaders bestimmt dabei seinen Aufgabenbereich:

- **Vertex Shader** werden dazu eingesetzt, die Vertices der geometrischen Grundformen zu transformieren. Ausserdem können aus den Daten eines Vertices Informationen berechnet werden, die den nachfolgenden Shadern zur Verfügung gestellt werden können.
- **Pixel Shader** berechnen die Farbe der Pixel nach der Rasterisierung. Dazu können sie sowohl auf die Daten, die von übergeordneten Shadern berechnet und zur Verfügung gestellt werden, als auch auf Texturen[28] zugreifen.
- **Geometrie Shader** können zusätzliche Vertices erzeugen. Dabei übernehmen sie die Berechnung deren Parameter, wie beispielsweise der Position.
- **Hull Shader** bieten zusammen mit den **Domain Shadern** Unterstützung für paramterisierte Kurven und Flächen, wie beispielsweise Bézier-Kurven oder NURBS-Flächen. Hull Shader transformieren die Kontrollpunkte dieser Kurven bzw. Flächen und berechnen für die Kurve bzw. jede Kante einer Fläche ein Tesselationsfaktor. Diese Daten übergibt er dem Tesselator, welcher entsprechend dem Tesselationsfaktor eine bestimmte Anzahl von Vertices erzeugt. Die Parameter dieser Vertices können im Domain Shader berechnet werden. Dem Domain Shader stehen dazu die (transformierten) Kontrollpunkte zur Verfügung.

Zusätzlich zu neuen Shader-Kategorien wurde ab dem Shader Model 4.0 die Möglichkeit geschaffen, die von den Shadern berechneten Daten bereits vorzeitig in den Grafikkartenspeicher zu schreiben, um als Grundlage für einen anderen Render-Durchlauf zu dienen. Dies wird als **Stream-Out** bezeichnet.

Aktuell ist derzeit das Shader Model 4.0. Grafikkarten, die das Shader Model 5.0 unterstützen, sind zurzeit noch in Entwicklung und daher noch nicht erhältlich. Selbst Grafikkaraten mit Unterstützung für das Shader Model 4.0 sind noch nicht sehr verbreitet. Aus diesem Grund wurde im Rahmen unserer Arbeit Shadercode geschrieben, welcher von Grafikkarten, die mindestens zu Shader Model 2.0 oder höher kompatibel sind, ausgeführt werden kann. (Das Shader Model 3.0 bietet gegenüber dem Shader Model 2.0 nur geringe Erweiterungen, die zur Animation des Charakters nicht benötigt wurden).

3.7.2 Darstellung und Beleuchtung der Oberflächen eines Charakters

Zur Darstellung der Oberfläche des Charakters wurden im Rahmen des zweiten Kapitels einige Texturen vorgestellt, die verschiedene Aspekte der Oberfläche beschreiben. Diese Aspekte wurden bereits im Kapitel 2.5 und Kapitel 2.6 erläutert. Damit sie jedoch von Grafikkarten in Echtzeit dargestellt werden können, mussten wir für die Shader der Grafikkarte ein entsprechendes Programm entwickeln.

Hauptaufgabe dieses Programms ist die Berechnung des Lichts bzw. die Berechnung der Anteile des auftreffenden Lichts, die von der Oberfläche des darzustellenden Körpers reflektiert wird. Da in der Computergrafik die physikalisch korrekte Simulation von Licht viel zu rechenaufwändig wäre, gibt es *Beleuchtungsmodelle*, die das Verhalten von Licht vereinfacht beschreiben. *Globale Beleuchtungsmodelle* beschreiben dabei die Ausbreitung von Licht in der virtuellen Welt, während sich *lokale Beleuchtungsmodelle* mit der Darstellung von Oberflächen befassen. Für die Implementierung des Shader-Programms benötigten wir somit ein lokales Beleuchtungsmodell. Unsere Wahl fiel auf das Blinn-Phong-Beleuchtungsmodell: Es ist die optimierte Version des Phong-Beleuchtungsmodells, welches trotz einiger Nachteile häufig verwendet wird. Dadurch sind Grafiker meist bereits mit den Parametern vertraut. Sie bestehen aus vier Faktoren: den drei Faktoren für den **ambienten**, **diffusen** und **specularen** (= ideal spiegelnden)

[28] Für die Definition von Texturen, siehe Kapitel 2.6

Anteil des Lichts, welches von der Oberfläche reflektiert wird und einem Faktor, der die Rauheit der Oberfläche beschreibt. Die genauen Details des Phong-Beleuchtungsmodells können beispielsweise unter [Wikipedia (Phong-Beleuchtungsmodell), 2008] nachgeschlagen werden.

Technische Umsetzung

Der Speicher, in dem der Vertex Shader Daten für den Pixel Shader abgelegen kann ist sehr klein, wodurch nur wenige Daten während eines Durchlaufs übergeben werden können. Da zur Berechnung des reflektierenden Lichts von jeder Lichtquelle die Position benötigt wird, müssten bei einer großen Anzahl von Lichtquellen jedoch viele Daten übergeben werden. Damit die Zahl der Lichtquellen, die einen Charakter gleichzeitig beeinflussen können, nicht eingeschränkt werden muss[29], wird der Charakter in mehreren Durchgängen (engl. Pass) gerendert. Dadurch kann eine beliebige Anzahl von Lichtern den Charakter beleuchten.

Allerdings müssen die Render-Durchgänge zum Berechnen der diffusen und spiegelnden Reflektion des Lichts entsprechend oft wiederholt werden, wodurch mit zunehmender Anzahl von Lichtern jedoch mehr und mehr Rechenzeit benötigt wird. Außerdem müssen durch die separaten Berechnungen der diffusen und spiegelnden Komponenten die Berechnung der ambienten Komponente und die Berechnung des Einflusses der diffusen Textur jeweils in einen separaten Durchgang ausgegliedert werden. Dadurch werden bei einer einzelnen Lichtquelle bereits vier Durchgänge benötigt. Jede weitere Lichtquelle benötigt zwei zusätzliche Durchgänge. Folgende Tabelle zeigt den Quellcode für den Pixel- und Vertex-Shader, den wir für die verschiedenen Render-Durchgänge entwickelt haben:

Vertex Shader	Pixel Shader
Ambient Pass	

`void Ambient_vp_1_1 (`
`  in  float4  iPosition : POSITION,`
`  out  float4  oPosition : POSITION,`
`  out  float4  oColor  : COLOR0,`
`  const uniform float4x4  cWorldViewProj,`
`  const uniform float4  cAmbient)`
`{`
`  oPosition = mul(cWorldViewProj, iPosition);`
`  oColor = cAmbient;`
`}`
	`void Ambient_fp_1_1 (`
	`  in  float4  iColor  : COLOR0,`
	`  out  float4  oColor  : COLOR)`
	`{`
	`  oColor = iColor;`
	`}`

Diffuse Pass	

`void DiffNormalLight_vp_2_0(`
`  in  float4  iPos  : POSITION,`
`  in  float3  iNormal  : NORMAL,`
`  in  float2  iTexCoord  : TEXCOORD0,`
`  in  float3  iTangent  : TEXCOORD1,`
`  out  float4  oPos  : POSITION,`
`  out  float4  oColor  : COLOR0,`
`  out  float2  oTexCoord  : TEXCOORD0,`
`  out  float3  oNormal  : TEXCOORD1,`
`  out  float3  oTangent  : TEXCOORD2,`
`  out  float4  oLightVec  : TEXCOORD3,`
`  const uniform float4  cLightPos,`
`  const uniform float  cLightAttenuation,`
`  const uniform float4x4  cWorldMat,`
`  const uniform float4x4  cWorldViewProjMat)`
`{`
`  float4 shadowCoord;`
`  float lightDist;`
`  float lightRange;`
`  float4 lightVec;`
`  float4 lightVecWS;`
 	
`  lightVec.xyz =`
`  cLightPos.xyz - (iPos.xyz * cLightPos.w);`
`  lightVec.w = 0;`
`  lightVecWS = mul(cWorldMat, lightVec);`
`  lightDist = length(lightVecWS.xyz) * cLightPos.w;`
`  lightRange = cLightAttenuation.x;`
 	
`  oPos = mul(cWorldViewProjMat, iPos);`
`  oColor = lightDist < lightRange ?`
`  float4(1.0, 1.0, 1.0, 1.0) :`
	`void DiffNormalLight_fp_2_0(`
	`  in  float4  iColor  : COLOR0,`
	`  in  float2  iTexCoord : TEXCOORD0,`
	`  in  float3  iNormal  : TEXCOORD1,`
	`  in  float3  iTangent  : TEXCOORD2,`
	`  in  float4  iLightVec : TEXCOORD3,`
	`  out  float4  oColor  : COLOR,`
	`  const uniform float4  cLightDir,`
	`  const uniform float4  cDiffuseColor,`
	`  const uniform float4  cLightAttenuation,`
	`  const uniform float4  cSpotLightParams,`
	`  const uniform sampler2D  normalMap : TEXUNIT0)`
	`{`
	`  float3 normal = iNormal;`
	`  float3 tangent = iTangent;`
	`  float3 binormal = cross(tangent, normal);`
	`  float3x3 tangentSpaceMat =`
	`  float3x3(tangent, binormal, normal);`
	
	`  float3 lightVecTS =`
	`  mul(tangentSpaceMat, iLightVec.xyz);`
	`  float3 lightDirTS = normalize(lightVecTS);`
	
	`  float distance = iLightVec.w;`
	
	`  float3 perturbedNormal =`

[29] bei aktuellen Shader-Modell-Versionen auf etwa zwei bis drei, je nach Größe der restlichen Daten, die übergeben werden müssen

```
        float4(0.0, 0.0, 0.0, 0.0);
    oTexCoord = iTexCoord;
    oNormal = iNormal;
    oTangent = iTangent;
    oLightVec.xyz = lightVec.xyz;
    oLightVec.w = length(lightVecWS.xyz) * cLightPos.w;
}
```

```
        tex2D(normalMap, iTexCoord).rgb;
    perturbedNormal = expand(perturbedNormal);
    perturbedNormal = normalize(perturbedNormal);

    float nl = dot(perturbedNormal, lightDirTS);

    float4 color;
    color = max(0.0, nl) * cDiffuseColor;

    float attenuation;
    attenuation = cLightAttenuation.y;
    attenuation += cLightAttenuation.z * distance;
    attenuation += cLightAttenuation.w *
        (distance * distance);
    attenuation = 1 / attenuation;

    float3 lightDir = normalize(iLightVec.xyz);
    float angle;
    angle = dot(lightDir, -cLightDir.xyz);

    float falloff;
    falloff = (angle - cSpotLightParams.y) /
        (cSpotLightParams.x - cSpotLightParams.y);
    falloff = saturate(falloff);
    falloff = pow(falloff, cSpotLightParams.z);
    falloff = cSpotLightParams.y > 0 ? falloff : 1;

    oColor = color * attenuation * falloff * iColor;
}
```

Specular Pass

```
void SpecMapNormalLight_vp_2_0(
    in    float4    iPos               : POSITION,
    in    float3    iNormal            : NORMAL,
    in    float2    iTexCoord          : TEXCOORD0,
    in    float3    iTangent           : TEXCOORD1,
    out   float4    oPos               : POSITION,
    out   float4    oColor             : COLOR0,
    out   float2    oTexCoord          : TEXCOORD0,
    out   float3    oNormal            : TEXCOORD1,
    out   float3    oTangent           : TEXCOORD2,
    out   float3    oEyeVec            : TEXCOORD3,
    out   float4    oLightVec          : TEXCOORD4,
    const uniform float4               cEyePos,
    const uniform float4               cLightPos,
    const uniform float4               cLightAttenuation,
    const uniform float4x4             cWorldMat,
    const uniform float4x4             cWorldViewProjMat)
{
    float4 shadowCoord;
    float lightDist;
    float lightRange;
    float4 lightVec;
    float4 lightVecWS;

    lightVec.xyz =
        cLightPos.xyz - (iPos.xyz * cLightPos.w);
    lightVec.w = 0;
    lightVecWS = mul(cWorldMat, lightVec);
    lightDist = length(lightVecWS.xyz) * cLightPos.w;
    lightRange = cLightAttenuation.x;

    oPos = mul(cWorldViewProjMat, iPos);
    oColor = lightDist < lightRange ?
        float4(1.0, 1.0, 1.0, 1.0) :
        float4(0.0, 0.0, 0.0, 0.0);
    oTexCoord = iTexCoord;
    oNormal = iNormal;
    oTangent = iTangent;
    oEyeVec = cEyePos.xyz - iPos.xyz;
    oLightVec.xyz = lightVec.xyz;
    oLightVec.w = length(lightVecWS.xyz) * cLightPos.w;
}
```

```
void SpecMapNormalLight_fp_2_0(
    in    float4    iColor    : COLOR0,
    in    float2    iTexCoord : TEXCOORD0,
    in    float3    iNormal   : TEXCOORD1,
    in    float3    iTangent  : TEXCOORD2,
    in    float3    iEyeVec   : TEXCOORD3,
    in    float4    iLightVec : TEXCOORD4,
    out   float4    oColor    : COLOR,
    const uniform float4     cLightDir,
    const uniform float4     cSpecularColor,
    const uniform float4     cLightAttenuation,
    const uniform float4     cSpotLightParams,
    const uniform sampler2D  normalMap   : TEXUNIT0,
    const uniform sampler2D  specularMap : TEXUNIT1,
    const uniform sampler2D  glossiMap   : TEXUNIT2)
{
    float3 normal = iNormal;
    float3 tangent = iTangent;
    float3 binormal = cross(tangent, normal);
    float3x3 tangentSpaceMat =
        float3x3(tangent, binormal, normal);

    float3 eyeVecTS = mul(tangentSpaceMat, iEyeVec);
    float3 lightVecTS =
        mul(tangentSpaceMat, iLightVec.xyz);

    float3 eyeDirTS = normalize(eyeVecTS);
    float3 lightDirTS = normalize(lightVecTS);

    float distance = iLightVec.w;

    float3 halfDirTS = normalize(eyeDirTS + lightDirTS);

    float3 perturbedNormal =
        tex2D(normalMap, iTexCoord).rgb;
    perturbedNormal = expand(perturbedNormal);
    perturbedNormal = normalize(perturbedNormal);

    float nl = dot(perturbedNormal, lightDirTS);
    float nh = dot(perturbedNormal, halfDirTS);

    float specularPower =
        tex2D(glossiMap, iTexCoord).r * 255.0;

    float4 lightCoef = lit(nl, nh, specularPower);

    float specularFactor =
        tex2D(specularMap, iTexCoord).r;

    float4 color;
    color =
        lightCoef.z * specularFactor * cSpecularColor;

    float attenuation;
    attenuation = cLightAttenuation.y;
    attenuation += cLightAttenuation.z * distance;
    attenuation += cLightAttenuation.w *
        (distance * distance);
    attenuation = 1 / attenuation;

    float3 lightDir = normalize(iLightVec.xyz);
    float angle;
    angle = dot(lightDir, -cLightDir.xyz);
```

```
    float falloff;
    falloff = (angle - cSpotLightParams.y) /
        (cSpotLightParams.x - cSpotLightParams.y);
    falloff = saturate(falloff);
    falloff = pow(falloff, cSpotLightParams.z);
    falloff = cSpotLightParams.y > 0 ? falloff : 1;

    oColor = color * attenuation * falloff * iColor;
}
```

Color-Map Pass

```
void Texture_vp_1_1 (
    in    float4    iPosition : POSITION,
    in    float4    iTexCoord : TEXCOORD0,
    out   float4    oPosition : POSITION,
    out   float4    oTexCoord : TEXCOORD0,
    const uniform float4x4    cWorldViewProj)
{
    oPosition = mul(cWorldViewProj, iPosition);
    oTexCoord = iTexCoord;
}
```

```
void Texture_fp_1_1 (
    in    float4    iTexCoord           : TEXCOORD0,
    out   float4    oColor              : COLOR,
    const uniform sampler2D texture     : TEXUNIT0)
{
    oColor = tex2D(texture, iTexCoord);
}
```

Tabelle 14: Shadercode zur Darstellung von Charakteren

3.8 Umsetzung

3.8.1 Ziele

Wie bereits in der Einleitung zu dieser Arbeit erwähnt, soll eine Anwendung entwickelt werden, in welcher die gezeigten Konzepte zur dynamischen Animation eines Charakters zur Anwendung kommen. Als Grundlage hierfür soll die quelloffene OGRE-Engine verwendet werden. Mit Hilfe der Anwendung soll es möglich sein, Fußspuren zu generieren und nachträglich zu manipulieren. Zu jeder Zeit soll dabei testweise ein Charakter darauf entlang gehen können. Außerdem sollen alle Informationen des Pfades und der Fußstapfen dargestellt werden können, um die Fehlersuche zu vereinfachen.

Die Anwendung soll später zu einem vollständigen Editor für das Spiel „Die Stadt NOAH“ ausgebaut werden können, mit dem Levels erstellt und bearbeitet werden können. Dazu ist die Implementation verschiedener Dokumenttypen und Ansichten vorzusehen, da ein Level aus mehreren Typen von Resourcen besteht (z.B. 3D-Modelle, Materialien, Skripte, Soundeffkte usw.), die betrachtet und teilweise editert werden sollen.

Da OGRE sowohl für Windows, Linux und Mac OS X verfügbar ist, soll die Anwendung idealerweise ebenfalls auf diesen Platformen lauffähig sein. Zur Entwicklung der Bentzerschnittstelle soll daher wxWidgets verwendet werden, da dieses ebenfalls quelloffen und kompatibel zu besagten Plattformen ist.

3.8.2 OgreIK

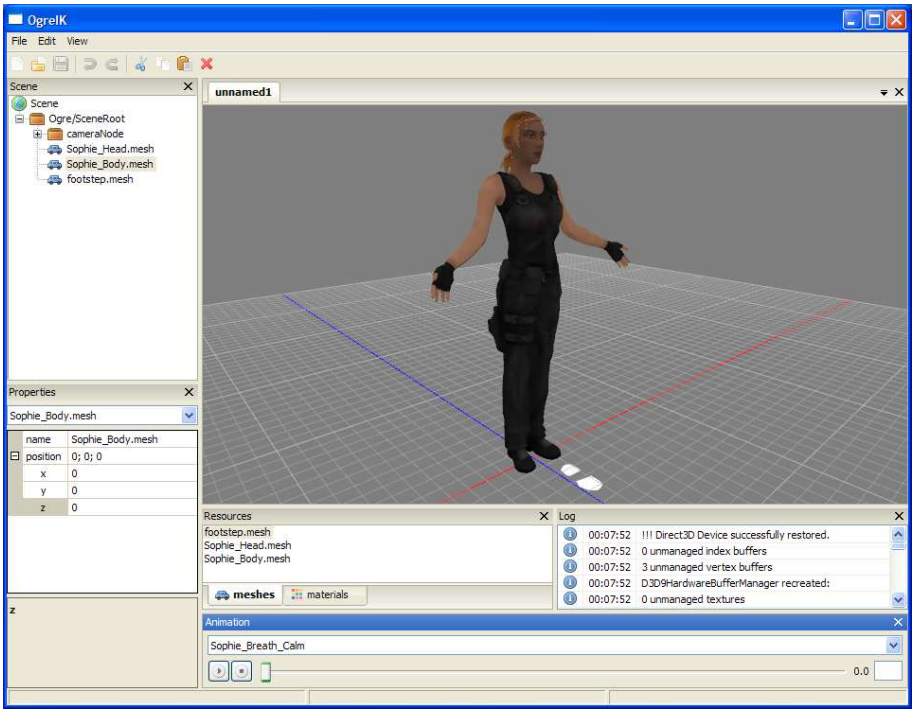


Abbildung 73: OgreIK

Der erste Anlauf zur Umsetzung dieser Anforderungen war *OgreIK*. Bereits bei der Implementierung der Oberfläche traten einige Bugs auf, für die ich jedoch durch Debuggen und der Recherche im Internet Workarounds fand. So funktionierte beispielsweise das Tree-Control nicht richtig, wenn man ein Item, um es an eine andere Stelle im Baum

zu platzieren, zunächst dupliziert und anschließend das Alte löschen will. Beide Items (obwohl sie bereits an verschiedenen Stellen im Baum sichtbar waren) erhielten dieselbe ID, wodurch das alte Item nicht eindeutig identifiziert werden konnte und wxWidgets immer zufällig eines von beiden löschte. Durch Probleme dieser Art verzögerte sich die Entwicklung mehrmals um einige Tage.

Als sich eines Tages herausstellte, dass die Implementierung des MVC-Patterns in wxWidgets etwas anders funktionierte als angenommen, und dies große Probleme verursachen würde, mussten größere Teile bereits geschriebenen Programmcodes überarbeitet werden. Dazu legte ich ein neues Projekt an, um die Übersicht über den Programmcode zu behalten.

3.8.3 Anima

Der zweite Anlauf, der zu einer überarbeiteten Version von OgreIK führen soll, wurde auf den Namen *Anima* getauft. Einige Workarounds und viel Know-How konnte dabei aus dem Quellcode von OgreIK übernommen werden, weshalb die Entwicklung zu Beginn schneller voran ging, als die Entwicklung von OgreIK. Dennoch wurde einige (nicht eingeplante) Zeit für die Neuentwicklung des fehlerhaften Programmcodes benötigt, bis Anima funktionstechnisch denselben Stand erreicht hatte wie OgreIK.

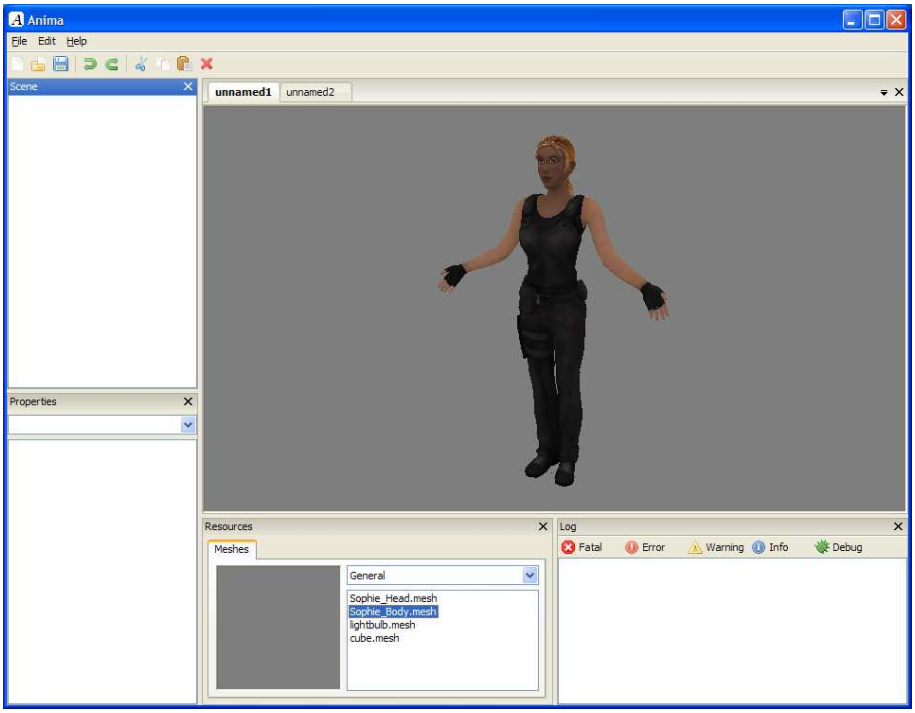


Abbildung 74: Anima

Da inzwischen die Zeit bis zum Abgabetermin unserer Arbeit knapp wurde und Anima noch nicht so weit entwickelt war, dass mit der Implementation der Algorithmen für die dynamischen Animation des Charakters begonnen werden konnte, entschloss ich mich dazu, die Entwicklung von Anima auf Eis zu legen.

3.8.4 Footsteps

Weil die Entwicklung von OgreIK bzw. Anima nicht rechtzeitig abgeschlossen werden konnte, und ich so keine Anwendung hatte, mit deren Hilfe ich meine Ideen zur Generation der Fußspuren und der dynamischen Animation austesten konnte, entschloss ich mich kurzfristig doch noch, eine minimalistische Anwendung zu schreiben, mit der

ich die Plausibilität meiner Konzepte überprüfen konnte. Sie verzichtet auf den Einsatz von wxWidgets und bietet daher nur eine sehr eingeschränkte Benutzerschnittstelle. Die meisten Ideen testete ich aufgrund der knappen Zeit mit Hilfe des Debuggers, in welchem ich ohne die Implementation von Ein- oder Ausgaberoutinen Werte ändern bzw. auslesen konnte.

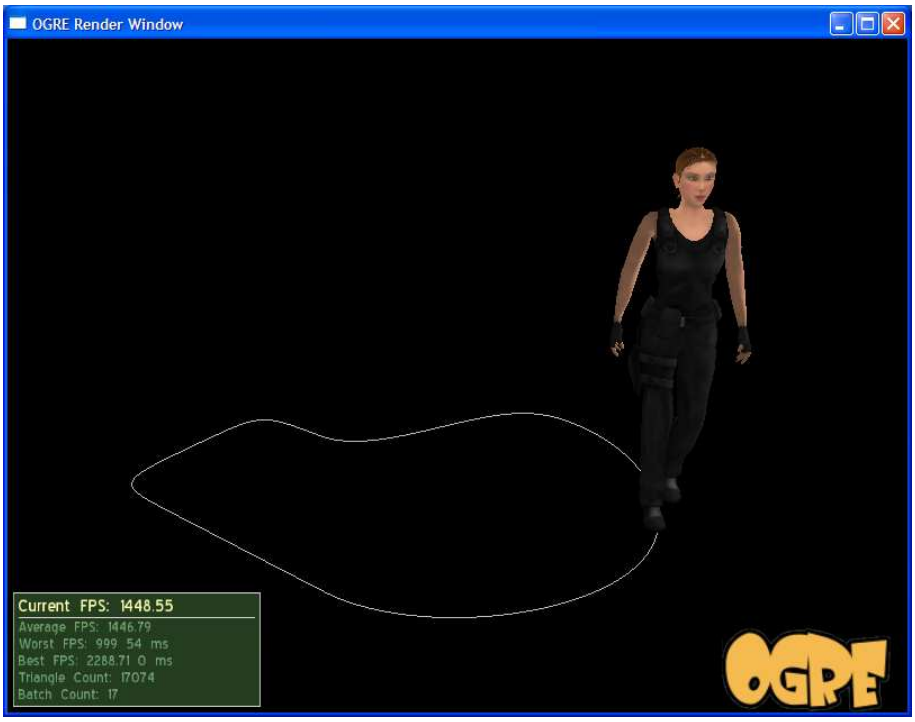
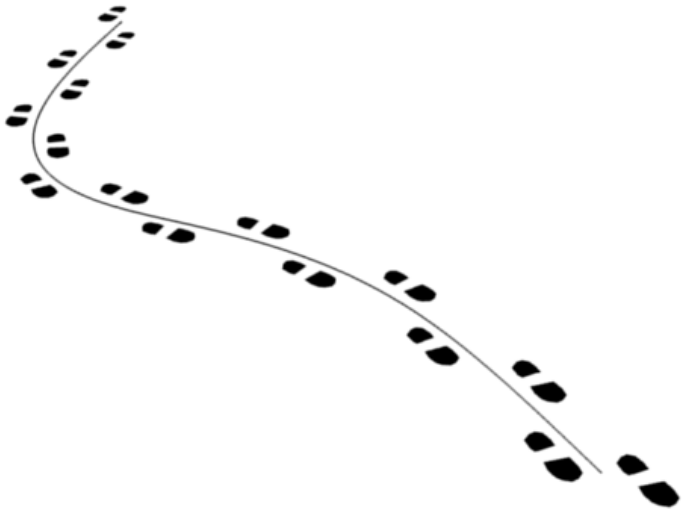


Abbildung 75: Footsteps



Quellenverzeichnis

Literatur

Bartle, R. A. (2003). *Designing Virtual Worlds.* London: New Riders Games.

Bowditch, G. (27. April 2008). *Grand Theft Auto producer is Godfather of gaming*. Abgerufen am 4. Januar 2009 von TimesOnline : http://www.timesonline.co.uk/tol/news/uk/scotland/article3821838.ece

Eberly, D. H. (2000). *3D Game Engine Design.* Morgan Kaufmann.

Haake, G. (22. Mai 2007). *Lara Croft wird zehn*. (SPIEGELONLINE, Herausgeber) Abgerufen am 3. Juni 2008 von Lara Croft wird zehn: http://www.spiegel.de/netzwelt/spielzeug/0,1518,484248,00.html

Hedge, S. (16. Juli 2007). *Gamasutra - Messing with Tangent Space*. Abgerufen am 17. September 2008 von http://www.gamasutra.com/view/feature/1515/messing_with_tangent_space.php

Hogarth, B. (2001). *Hoghart's Zeichenschule.* (W.-G. Publications, Hrsg.) New York: TASCHEN GmbH.

image metrics. (2008). *Grand Theft Auto IV*. Abgerufen am 4. Januar 2009 von http://image-metrics.com/project/grand-theft-auto-iv

Image Metrics. (2008). *Image Metrics*. Abgerufen am 16. September 2008 von Making Of Emily (Video): http://www.imagemetrics.com

Kastenholz, D. (8. Juni 2006). *3D Pathfinding.* Abgerufen am 4. Januar 2009 von http://3dpathfinding.homeunix.org/

Kavan, L., Collins, S., O'Sullivan, C., & Zara, J. (2006). *Dual Quaternions for Rigid Transformation Blending.* Abgerufen am 4. Januar 2009 von https://www.cs.tcd.ie/publications/tech-reports/reports.06/TCD-CS-2006-46.pdf

MacDorman, K. F. (2005). *Androids as an Experimental Apparatus: Why is there an Uncanny valley and can we exploit it?* Abgerufen am 22. Juni 2008 von http://www.androidscience.com/proceedings2005/MacDormanCogSci2005AS.pdf

Mori, M. (1970). *Bukimi no tani (the uncanny valley).* Energy.

Noah Team. (11. Januar 2007). *Noah Wiki*. Abgerufen am 3. Juli 2008 von http://www.gratis-wiki.com/Projekt_Noah/

NVIDIA Corporation. (9. 11 2008). *Real Time Hair*. Abgerufen am 28. September 2008 von http://developer.nvidia.com/object/siggraph-2008-hair.html

SPIEGEL Online. (26. August 2008). *Spiegel Online*. Abgerufen am 16. September 2008 von http://www.spiegel.de/netzwelt/tech/0,1518,573038,00.html

Spoerl, M. (2004). The Jacobian Transpose Method for Inverse Kinematics. In A. Kirmse, *Game Programming Gems 4* (S. 193-204). Charles River Media.

Tory, E. (16. Oktober 2007). *Assassin's Creed - Animations Interview*. Abgerufen am 4. Januar 2009 von http://www.gametrailers.com/player/26509.html?type=flv

Tremblay, C. (2004). *Mathematics for Game Developers.* Premier Press.

Volkswagen AG. (17. Juni 2008). *Volkswagen*. Abgerufen am 17. Juni 2008 von http://www.volkswagen.de

Weber, J. (2002). Constrained Inverse Kinematics. In D. Treglia, *Game Programming Gems 3* (S. 192-198). Charles River Media.

Wikimedia (Lara Croft). (3. Juni 2008). Abgerufen am 3. Juni 2008 von Wikipedia: http://de.wikipedia.org/wiki/Lara_Croft

Wikipedia (Bézierkurve). (23. Dezember 2008). Abgerufen am 4. Januar 2009 von http://de.wikipedia.org/wiki/Bézierkurve

Wikipedia (Game Boy). (24. Oktober 2008). Abgerufen am 11. November 2008 von http://de.wikipedia.org/wiki/Game_Boy#Details

Wikipedia (Phong-Beleuchtungsmodell). (25. Oktober 2008). Abgerufen am 4. Januar 2009 von http://de.wikipedia.org/wiki/Phong-Beleuchtungsmodell

Wikipedia (Sprite (Computergrafik)). (25. Oktober 2008). Abgerufen am 11. November 2008 von http://de.wikipedia.org/wiki/Sprite_(Computergrafik)

Yello Strom GmbH. (17. Juni 2008). *Yello Strom*. Abgerufen am 17. Juni 2008 von http://www.yellostrom.de

Stichwortverzeichnis