UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

Fraunhofer Institut Intelligente Analyse- und Informationssysteme

Virtueller Nachrichtensprecher

Diplomarbeit

Zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Computervisualistik

vorgelegt von

Tobias Kilian

Erstgutachter: Prof. Dr. Stefan Müller
(Institut für Computervisualistik, AG Computergraphik)

Zweitgutachter: Dipl.-Phys. Christian Eckes
(Fraunhofer Institut, IAIS)

Koblenz, im Dezember 2006

Für meine liebe Frau Susanna.

Erklärung

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

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

Ort, Datum Unterschrift

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

UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Aufgabenstellung für die Diplomarbeit

Tobias Kilian

(Matr.-Nr. 201210708)

Thema: **Entwicklung eines virtuellen Nachrichtensprechers.**

Virtuelle Menschen spielen als Avatare eine immer wichtigere Rolle im Fernsehen und in den interaktiven Medien. Um eine überzeugende Erscheinung und ein realistisches Verhalten der Avatare in Echtzeit zu erreichen, sind hocheffiziente Methoden der Computergrafik notwendig. Interaktion von Menschen mit diesen Avataren ist nur in einfachen, sehr stark eingeschränkten Domänen möglich und immer noch Thema aktueller Forschungsarbeiten.

In dieser Arbeit sollen vorhandene Komponenten aus dem Forschungsprojekt "Virtual Human" in einen Prototypen integriert werden, so daß ein virtueller Nachrichtensprecher gegebene Texte möglichst realistisch vortragen kann. Dabei soll Wert auf hohe Effizienz gelegt werden, um eine interaktive Erweiterung des Systems sicherstellen zu können.

Schwerpunkte dieser Arbeit sind:

1. Anforderungsanalyse an einen virtuellen Nachrichtensprecher anhand einiger Use-Cases und möglicher Dialoge. Überprüfung der Anforderungen an eine eventuelle Verwendung in einem Virtuellen Studio.

2. Evaluierung und Auswahl einer Engine zur Umsetzung. Identifizierung der notwendigen Infrastruktur und der wiederverwendbaren Komponenten aus dem "Virtual Human"-Projekt (Charaktere + Bewegungsdaten, Hautshader, Sprachsynthese & Spracherkennung oder Gestenerkennung)

3. Konzeption

4. Realisierung des virtuellen Nachrichtensprechers: Vorlesen von Texten mit automatischer, synchroner Viseme-Ansteuerung. „Emotionen“ (Augenbrauen, Kopfbewegung) skriptbasiert und editierbar mit bereits aufgenommenen Daten aus der Tagesschau bzw. den Tagesthemen.

5. Evaluation anhand von Tagesschau/Tagesthemen-Daten, die annotiert im IMK vorliegen.

6. Dokumentation der Ergebnisse.

Die Diplomarbeit wird am Fraunhofer-Institut für Medienkommunikation in Sankt Augustin durchgeführt.

Betreuer: Christian Eckes (Fraunhofer IMK)

S. Müller

Koblenz, den 1. Mai 2006

- Prof. Dr. Stefan Müller-

1 Inhaltsverzeichnis

1 Einführung

1.1 Aufgabenstellung aus technischer Sicht

Die Darstellung virtueller Charaktere ist eine spannende Aufgabe für die Computergrafik. Die vorliegende Diplomarbeit hatte die Konzeption und Entwicklung eines virtuellen Nachrichtensprechers[1] unter Echtzeitbedingungen zum Ziel. Die offizielle Aufgabenstellung findet sich vor dem Inhaltsverzeichnis. Für mich lag der Schwerpunkt in der technischen Umsetzung.

Daher waren die primären Ziele, die ich der Aufgabenstellung entnommen habe:

1. Darstellung des Nachrichtensprechers in einer studio-ähnlichen Umgebung.
2. Einbindung eines Sprachsynthese-Systems zum Vorlesen beliebiger Texte.
3. Implementierung eines Animationssystems, das die vielfältigen Bewegungen des menschlichen Körpers darstellen kann. Dazu gehören Ganzkörperbewegungen, lippensynchrones Sprechen und Darstellung von Emotionen durch Animation der Augenpartie.
4. Möglichst realistische Darstellung von Haut mit einem Volumenstreuungsverfahren.
5. Einbindung einer Benutzeroberfläche, um Animationen einstellen zu können.
6. Speichern und Laden von Animationen.
7. Für alle Teilbereiche sind die Echtzeitbedingungen zu beachten. Hohe Bildwiederholungsraten, um die Skalierbarkeit des Systems zu gewährleisten sind erforderlich.

1.2 Nicht Teil dieser Arbeit

Der Fokus dieser Arbeit lag in der Entwicklung eines Nachrichtensprechers. Ein Avatar in einem Dialogsystem könnte mit anspruchsvollen Interaktionsmethoden wie Spracherkennung, Dialogverstehen und Gestikerkennung ausgestattet sein, was aber den Rahmen dieser Arbeit sprengen würde.

[1] wird im Folgenden kurz als „VAM“ (=engl. Virtual Anchorman) bezeichnet.

Eine Anbindung an das virtuelle Studio (VS) wurde geprüft und für „nicht ohne Weiteres möglich“ befunden, da die dort verwendete Technik auf eine hausinterne Engine beruht, die speziell auf die Bedürfnisse im VS zugeschnitten ist und mit den von VAM gestellten Anforderungen an eine Engine nicht kompatibel ist.

1.3 Inhaltsübersicht

Im nächsten Kapitel gehe ich auf die theoretischen Probleme der Charakteranimation ein. Dann werden Ganzkörpermodelle vorgestellt, Animationsverfahren gezeigt und Bewegungserfassung thematisiert. Anschließend folgen Gesichtsmodellierung und Gesichtsanimation.

Im dritten Kapitel wird die Konzeption der Arbeit dargelegt. Diese umfasst die Anforderungsanalyse, die Bestandsaufnahme und begründet meine Wahl der Grafikengine[2].

Im vierten Kapitel wird die Implementation vorgestellt. Dafür werden zunächst die verwendeten Bibliotheken erläutert und auf technische Details eingegangen. Das Gesamtsystem wird skizziert und die Funktionsweise des Hautshaders näher erklärt.

Kapitel fünf befasst sich mit der Evaluation des Gesamtsystems.

Abschließend folgen Zusammenfassung, Fazit und Ausblick im sechsten Kapitel.

Im Anhang werden die mathematischen Grundlagen Matrixtransformation, homogene Koordinaten und Quaternionen kurz erläutert.

[2]Eine Engine ist eine Sammlung von Funktionen, die dem Programmierer eine Reihe von Aufgaben zu einer bestimmten Thematik abnehmen und den Zugriff auf die Hardware erleichtern sollen.

2 Grundlagen

2.1 Warum virtuelle Charaktere ?

Heutzutage bieten virtuelle Realitäten einen großen Fundus an Werkzeugen und Möglichkeiten. Meistens bestehen sie jedoch nur aus statischen Objekten oder physikalischen Simulationen, wobei der Benutzer wenig Interaktions- und Partizipationsmöglichkeiten hat. Aktive Beteiligung erfordert Kommunikation zwischen den Teilnehmern und den Anbietern einer virtuellen Umgebung. Diese erfordert virtuelle Verkörperungen der beteiligten Personen. Innerhalb von virtuellen Umgebungen stellen virtuelle Charaktere die Schlüsseltechnologie dar, die für virtuelle Verkaufsagenten, Lehrer, Schauspieler und zur Simulation und Erforschung von menschlichem Verhalten im Allgemeinen erforderlich ist.

Virtuelle Charaktere können der Forschung und Lehre in unterschiedlichsten Disziplinen, aber auch der Unterhaltung in Computerspielen, Filmen und interaktiven Fernsehshows auf vielfältige Art und Weise dienen. Für interaktive Umgebungen sind Systeme erforderlich, die in Echtzeit Charaktere animieren können.

Das ultimative Forschungsziel ist die Simulation von virtuellen Welten, die von autonomen, intelligenten Menschen bevölkert werden, die miteinander kommunizieren, verhandeln, Freunde finden, sich zusammentun und wieder trennen, und das alles nur in Abhängigkeit ihrer Möglichkeiten, Bedürfnisse, Stimmungen, Ziele, Ängste etc. Die Handlungen und Interaktionen sollten also nicht beim Programmieren festgelegt werden, sondern das Gesamtgeschehen sollte vom Verhalten vieler autonomer Agenten abhängen, die ein gemeinsames, virtuelles Umfeld teilen.

Neben der Simulation virtueller Welten mit autonomen Agenten sind noch zwei weitere Anwendungsgebiete denkbar. Als erstes ist die virtuelle Substitution eines echten Menschen zu nennen, der seine Interessen über das Internet vertritt, in Chats, Meetings, Spielen, etc. Für solche Anwendungen wird heutzutage ein Pool von Avataren zur Verfügung gestellt, aus denen sich der Anwender einen Charakter heraussuchen darf, der ihn repräsentiert. Noch besser wäre ein Charakter, der die Stimme und das Aussehen der realen Person glaubhaft transportieren kann.

Zweitens besteht die Möglichkeit des Einsatzes als virtueller Prototyp. Zum Beispiel wäre es für Chirurgen sinnvoll, einen menschlichen Körper virtuell nachzubilden, um an ihm Operationen üben zu können, ohne dass reale Personen dadurch in Gefahr sind. Neben medizinischen Anwendungen ist auch das Militär an einer möglichst wirklichkeitsnahen Simulation von z.B. Gefechtssituationen interessiert, wie das Projekt „Santos“ [www:Santos] aus den U.S.A. anschaulich zeigt.

2.2 Geschichte der virtuellen Charaktere

Die ältesten virtuellen Figuren wurden für Ergonomieanalysen genutzt. [Thalmann et al.(2004)] Der erste virtuelle Charakter ist der „Landing Signal Officer“(LSO), der 1959 von William Fetter für Boeing entwickelt und zur Ergonomieanalyse der Instrumententafel einer Boeing 747 genutzt wurde.

1985 wurde erstmals Gesichtsanimation eingesetzt, und zwar in dem Kurzfilm „Tony de Peltrie“.

1988 konnte „Mike the talking Head“ erstmals in Echtzeit zwischen mehreren Gesichtsausdrücken interpolieren. [deGraf et al. (1988)]

Die erste virtuelle Nachrichtensprecherin war Ananova [www:Ananova] (2000). Leider ist sie schon seit 2004 nicht mehr auf der Webseite zu sehen, mit dem Hinweis, das sie „momentan weiterentwickelt wird“. Sie war in der Lage, Kurznachrichten aus Ananova's Echtzeit-Nachrichten-System synthetisch vorzulesen. Offline erstellte Videos davon konnten dann per Live-Stream angeschaut werden.

2.3 Herausforderungen virtueller Charaktere

Die Entwicklung glaubhafter virtueller Charaktere ist ein komplexes Thema, das verschiedene Forschungsgebiete beinhaltet. Eine sehr umfassende Übersicht zum Thema bieten [Thalmann et al.(2004)]. Sie unterscheiden mehrere Aspekte:

2.3.1 Repräsentation des Körpers und des Gesichtes

Die Modellierung eines Menschen ist der erste Schritt bei der Entwicklung eines virtuellen Charakters. Dies kann in einem 3D-Programm manuell geschehen, aber weniger zeitaufwändig ist die halbautomatische Rekonstruktion der Realität. Als Eingabe dienen dafür entweder zweidimensionale Fotos bzw. Videos einer Person, oder Tiefeninformationen aus einem Laser-Scanner o.ä.

Dabei werden normalerweise das Gesicht und der Körper getrennt behandelt. Da das Gesicht ungleich mehr der Identifizierung von Personen dient, ist es wesentlich sorgfältiger zu modellieren.

Ein weiteres Problem, an dem noch geforscht wird, ist die Erfassung von Veränderungen des menschlichen Körpers beispielsweise durch Alter und Fettleibigkeit.

2.3.2 Flexible Animationskontrolle

Das Ziel der Computeranimation ist die Synthese eines gewünschten Bewegungseffekts, der sich aus natürlichen Vorgängen, Wahrnehmung und Vorstellungskraft des Designers zusammensetzt. Das Animationssystem muss dem Designer also Werkzeuge zur Verfügung stellen, mit denen er diese Bewegungen aus seiner mentalen Repräsentation in die Sprache des Systems umsetzten kann. Die atomare Einheit dieser Sprache, auf der die Animationskontrolle aufbaut, ist je nach verwendeter Technik unterschiedlich. Bei einem auf Keyframes basierenden System sind das die Winkel in allen Gelenken zu jedem Zeitpunkt. Bei Verwendung von dynamischer *forward kinematic*, ist die Basisinformation eine Sammlung von Kräften und Drehmomenten. Selbstverständlich werden bei der dynamischen Simulation auch die Winkel zwischen den Gelenken berechnet, doch sind diese dann als eine abgeleitete Information zu sehen und daher nicht im Fokus der Animationskontrolle.

Unabhängig davon, ob die Bewegungsdaten aus Aufzeichnungen durch Motion Capturing bestehen, das Ergebnis einer physikalischen Simulation sind oder manuell definiert wurden, ist es wichtig, diese wiederverwenden zu können, damit die zeitaufwändige Verarbeitung vorab erfolgen kann. Sobald eine akzeptable Animation erstellt ist, kann diese dann später in Echtzeit abgespielt werden.

2.3.3 Hochwertiges Verhalten

Damit virtuelle Menschen realistisch wirken, indem sie autonom handeln, sollten sie ein Verhalten aufweisen, das menschliche Intelligenz suggeriert. Dazu müssen sie virtuelle Sensoren besitzen, mit denen sie ihre Umgebung visuell, taktil und akustisch wahrnehmen können. Aufgrund der wahrgenommenen Informationen und den Zielen des Charakters muss der Verhaltensmechanismus festlegen, welche Handlungen vollzogen werden sollen. Diese müssen dann in motorische Handlungen umgesetzt werden. Idealerweise ist das Verhalten des virtuellen Menschen spontan und unvorhersehbar und außerdem glaubwürdig, weil er Emotionen ausdrückt, die zur jeweiligen Situation passen.

2.3.4 Emotionen

Eine Emotion ist ein innerer Zustand des Charakters, der von seiner Wahrnehmung beeinflusst wird und seine Handlungen mitbestimmt. Die Äußerung von Emotionen durch Gesten und Gesichtsausdrücke sollte daher zwischen Wahrnehmung und Handlung liegen. Da die Emotionen subjektiv sind, reagiert jeder Charakter anders auf die selben Reize und handelt damit auch individuell. In [Orthony (1990)] wird ein logisches Modell vorgeschlagen, das Emotionen danach klassifiziert, ob sie sich auf eine Handlung, einen anderen Agenten oder ein Objekt beziehen. So kann die Wahrnehmung eines Objektes Einfluss haben auf die Anziehungskraft, die der Agent verspürt, und die ihn möglicherweise veranlasst, eine Handlung durchzuführen. Bei der emotionalen Haltung gegenüber anderen Agenten kommen zusätzliche Parameter wie Abscheu oder Bewunderung und Schamgefühl oder Stolz hinzu.

2.3.5 Realistisches Aussehen

Für eine überzeugende Darstellung eines virtuellen Menschen sind auch eine natürlich wirkende Haut sowie möglichst dynamische Haare und Kleidung erforderlich. Kleidung wird heutzutage schon sehr gut in Echtzeit simuliert, beispielsweise von [www:Seungwoo]. Es gibt verschiedene Ansätze: analytische Lösungen, die exakt, aber rechenaufwändig sind, Approximation der analytischen Lösungen mit finiten Elementen und Simulation mit Hilfe von Partikelsystemen (siehe [www:Lander]). Allgemein lässt sich sagen, dass die Simulation von Kleidung an virtuellen Charakteren

eher ein Problem der Komplexität als der Berechenbarkeit ist. Das hängt damit zusammen, dass für die Kleidung Kollisionserkennung und Kollisionsbehandlung berechnet werden muss. Da für einen realistischen Faltenwurf eine relativ hohe Anzahl von zu testenden Elementen erforderlich ist, kommen Brute-Force Methoden nicht in Frage, da dort der Rechenaufwand quadratisch mit Anzahl der Elemente steigt. Hierarchische Methoden zur Kollisionserkennung können aber den Rechenaufwand verringern. In Echtzeitanwendungen wird versucht, mit möglichst grober Auflösung auf der CPU zu rechnen, um dann Falten in kleinerem Maßstab auf der Grafikhardware zu deformieren und mit Normal Mapping weiter optisch zu verfeinern.

Haare sind ebenso kompliziert zu simulieren wie Kleidung. Die drei Disziplinen dabei sind das Modellieren, die Berechnung einer Dynamik und das Rendering[3] von Haaren. Es gibt jeweils verschiedene Ansätze, die in [Thalmann et al.(2004)] erläutert werden.

Aufgrund der hohen Komplexität der Themen konnten Simulation von Kleidung und Haaren in dieser Arbeit nicht berücksichtigt werden. Das verwendete Modell hat statische Haare und Kleidung, denen auch keine besondere Renderingtechnik zugrunde liegt.

Hautshading: Mit einfachem Echtzeit-Shading (z.B. Phong-Blinn) erscheint Haut plastikhaft und künstlich. Um realistisches Aussehen von künstlicher Haut zu erreichen, muss das Eindringen von Licht in die oberen Schichten der Haut und das Austreten des mehrfach gestreuten Lichts von der Hautoberfläche simuliert werden, wie es z.B. [Unterberg (2004)] im Rahmen des *Virtual Human*-Projektes realisiert hat. Diese Arbeit wurde in VAM integriert. (siehe Kapitel 4.7)

Weitere Probleme bei virtuellen Charakteren, jedoch nicht im Fokus dieser Arbeit, sind die Interaktion mit der virtuellen (und in „mixed-reality“ auch mit der realen) Umwelt, die ebenfalls in [Thalmann et al.(2004)] eingehend beleuchtet werden.

[3]Rendering (engl.) = hier: computergestützte Erzeugung eines digitalen Bildes.

2.4 Modellierung des Körpers

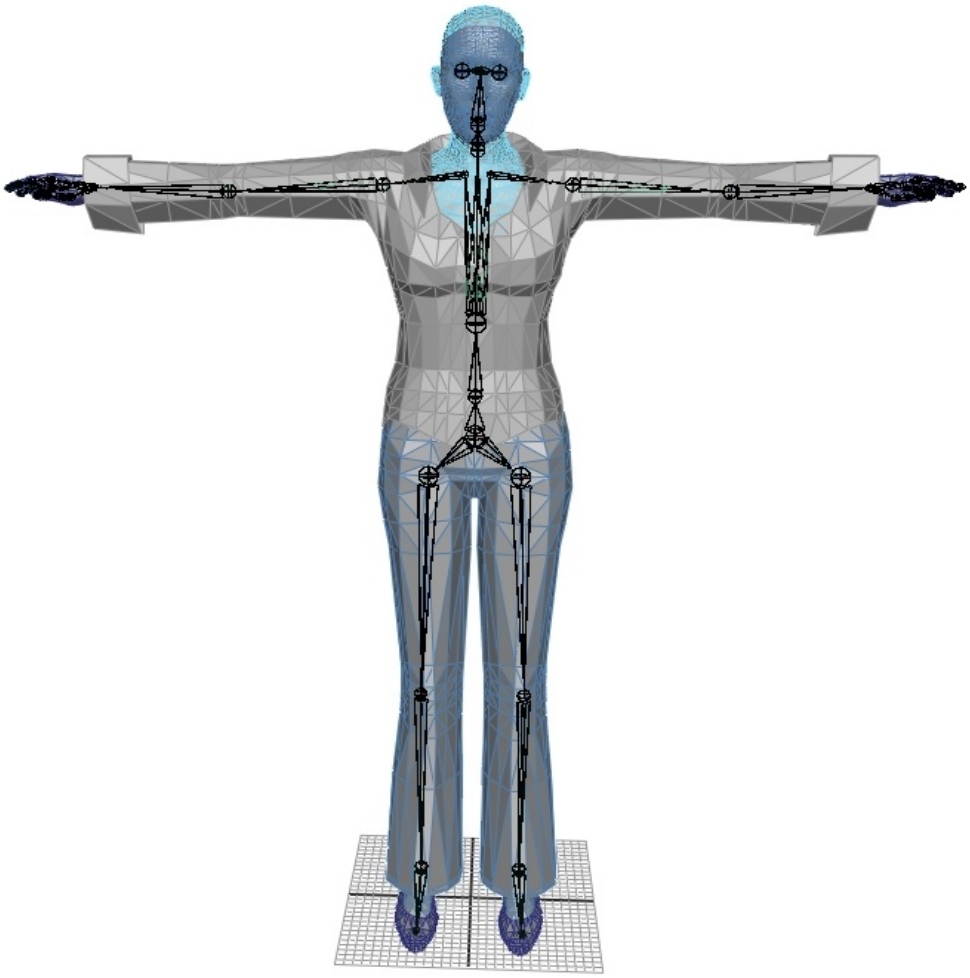


Abbildung 2.1.Bind Pose

2.4.1 Geometrische Repräsentation

3D Modelle können z.B. aus Bildern, Videos oder Laser-Scans halbautomatisch erfasst oder manuell in einem 3D-Programm modelliert werden. Die praktischste und am häufigsten verwendete Repräsentation für die Geometrie sind *polygonale Netze (Meshes)*. Sie bestehen aus vielen Eckpunkten, die durch Flächen verbunden werden. Es ist auch möglich, weiche Oberflächen mit NURBS oder Subdivision Surfaces zu beschreiben. In Echtzeitanwendungen werden diese aber nur genutzt, um polygonale Netze beliebiger Detailstufe zu erzeugen, womit das Problem der Charakteranimation immer noch besteht. Daher werden meistens, wie auch in dieser Arbeit, Dreiecksnetze verwendet.

In der Regel besteht das Modell eines virtuellen Charakters aus mehreren Teilkomponenten, die nach praktischen Gesichtspunkten gewählt werden. Die Eigenschaften der Grafik-Pipeline verlangen für Echtzeitanwendungen mindestens eine Aufteilung nach Materialeigenschaften. In Computerspielen ist es außerdem oftmals wünschenswert, einzelne Komponenten (Rüstung, Waffen, Schilde) austauschen zu können.

2.4.2 Verformung des Körpers

Um die Bewegungen eines Menschen am Computer nachzubilden, ist es erforderlich, den Bewegungsapparat zu abstrahieren. Einfache Oberflächenmodelle sind aus zwei Ebenen aufgebaut:

- Ein artikuliertes Skelett, welches die groben Bewegungen des Körpers modelliert. Ein Skelett wird durch eine hierarchische Folge von *Joints*(=Gelenken) definiert. Der Verbindung zwischen je zwei *Joints* nennt sich *Bone* (=Knochen).
- Eine externe geometrische Hülle oder Haut, deren Deformation von der unterliegenden Schicht bestimmt wird.

Bei komplexeren Modellen, meist für medizinische Simulationen, werden außerdem Muskeln und Fettgewebe berücksichtigt.

2.4.3 Rigid Deformation

Das einfachste Hautmodell ist Rigid Deformation (rigid=steif). Dabei wird jedem Knochen ein Teil des polygonalen Netzes zugeordnet. Die Hülle folgt somit zwar grob den Bewegungen des Skelettes, aber in der Nähe von Gelenken sind die Teile nicht miteinander verbunden und bilden entweder Lücken oder durchdringen sich. Rigid Deformation eignet sich daher höchstens zur Animation von Robotern.

2.4.4 Skeletal Subspace Deformation

Andere Namen für Skeletal Subspace Deformation (SSD) sind : „(Vertex) Skinning“, „Smooth binding“, „transform blending“, „matrix blending“ etc. So wird eine Technik bezeichnet, die meistens für Echtzeitanwendungen verwendet wird. Nach [Merry(2006)]

ist sie im Original nicht in der Literatur zu finden, obwohl sie von vielen Quellen benutzt wird, die sie erweitern oder verbessern.

SSD ist nach der Rigid Deformation die einfachste Art der Körperverformung. SSD geht ebenfalls von einem stark vereinfachten Skelettmodell aus, über dem eine Haut (Skin) gespannt ist, welche als polygonales Netz realisiert wird.

Jedoch geschieht beim SSD die Zuordnung zu den Knochen nicht auf Meshebene, sondern jeder Eckpunkt kann unabhängig voneinander mehreren Knochen zugeordnet sein.

Die Transformationen der einzelnen Knochen werden linear kombiniert. Ihre Anteile ergeben sich aus einer definierten Gewichtung:

$$w(V)=(w_1, w_2, w_3, .., w_n)\in\mathbb{R}, n=Anzahl\ der\ Knochen, w_1..w_n=Gewichtung\ des\ jeweiligen\ Knochens$$

Für jeden Knochen existiert eine Transformationsmatrix[4], die seine Position und Orientierung im Raum beschreibt. Die endgültige Position $p_{skinned}$ eines Vertex V kann dann wie in Formel 2.4.4.1 ermittelt werden.

$$p_{skinned}=\sum_{b=1}^{n} M(b)\cdot\begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}\cdot weight(v,b), mit$$

$$M(k)=Transformationsmatrix\ des\ Bone\ k$$
$$p=Neutrale\ Position\ des\ Vertex(bind\ bose)$$
$$weight(v,b)=Gewichtung\ von\ Vertex\ v\ für\ Bone\ b\in 0..1$$

Formel 2.4.4.1: Skeletal Subspace Deformation

So kann man eine Figur animieren, indem man nur das Skelett bewegt, während die Positionen der Eckpunkte sich aus den Zuordnungen zu den Knochen ergeben. Die Gewichte ergeben sinnvollerweise in der Summe immer eins, können theoretisch aber auch andere Werte annehmen.

Die für das SSD notwendigen Matrixtransformationen sind natürlich rechenintensiv, weshalb es wünschenswert ist, diese Berechnungen auf der Grafikkarte auszuführen um die CPU zu entlasten. Die neueren Grafikkarten erlauben diese Möglichkeit über ihre programmierbare Pixel-Pipeline.

[4]Eine Erläuterung der Transformationen mit Matrizenrechnung findet sich im Anhang.

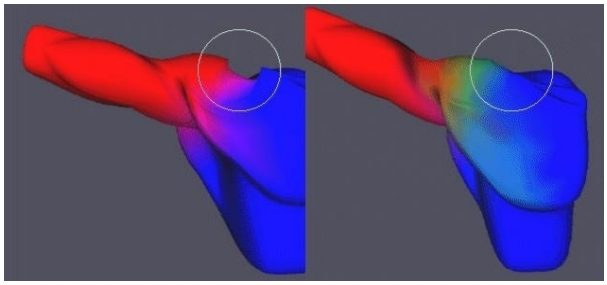


Abbildung 2.3.Skinning mit 2(links) und 3 Matrixtransformationen Quelle:ATI

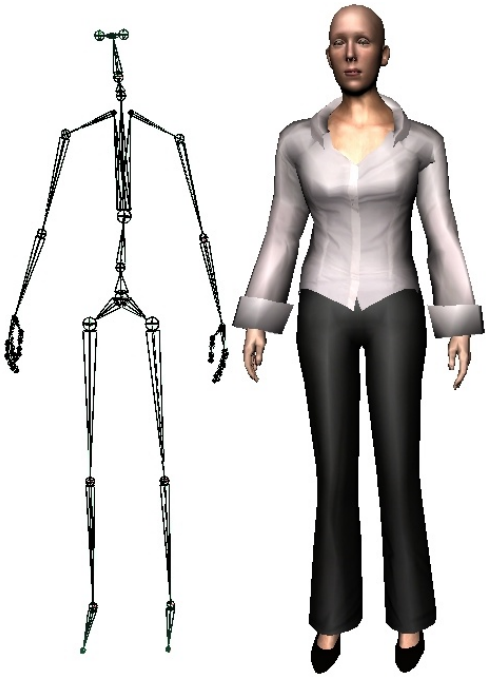


Abbildung 2.2: Skelettstruktur und entsprechend verformtes Modell

Die meisten Vertices sind nur einem Knochen zugeordnet; so bewegt sich die Nasenspitze z.B. immer nur mit dem Knochen des Kopfes. Die Punkte, die an den Gelenken liegen, müssen jedoch mehreren Knochen zugeordnet werden, um einen allmählichen Übergang zu schaffen. Das dabei auch mehr als 2 Zuordnungen benötigt werden zeigt Abbildung 2.3

Die Zuordnung erfordert ein Aufeinanderlegen von Mesh und Skelett. Dieser Prozess wird *skin binding, weighting* oder einfach nur *skinning* genannt. Die Körperhaltung, die dafür verwendet wird, nennt man *bind pose* oder *neutral pose*. In Abbildung 2.1 auf Seite 15 sieht man, dass die Arme vom Körper gestreckt sind und die Handflächen nach unten zeigen. Diese Körperhaltung wird typischerweise verwendet. Es gibt einige Algorithmen, die das *skin binding* automatisch berechnen, allerdings liefert keiner perfekte Ergebnisse, da alle auf Heuristiken basieren. Der *point-to-line-mapping*-Algorithmus ordnet jeden Vertex den ihm am nächsten stehenden Knochen zu. Beim *containment binding* werden den Knochen Hüllkörper in Form von z.B. Zylindern oder Ellipsoiden zugeordnet, und jeder Vertex wird dann dem Knochen zugeordnet, zu dessen Hüllkörper er den kleinsten Abstand hat. Auch mit sog. *Delaunay tetrahedralization* [Shin(2000)] wurde schon gearbeitet.

Die Zuordnung von Eckpunkten zu Knochen ist also bestenfalls halbautomatisch und bedarf viel manueller Korrektur, bis eine für viele Bewegungen passender Kompromiss gefunden ist.

Alle Animationen für das Skelett werden relativ zur *bind pose* notiert. Dadurch wird es möglich, mehrere Bewegungen additiv zu überlagern, so dass z.B. aus einem „Laufen“ und einem „mit-den-Armen-winken“ zu einem Winken während des Laufs wird. Durch einfache gleichmäßige Mischung der Animationen würde jeweils die andere Bewegung abgeschwächt.

Bei der additiven Überlagerung müssen die bewegten Komponenten nicht disjunkt sein. So könnte man zum Beispiel eine Animation „Hinken“ erstellen, in der lediglich ein Fuß- und Kniegelenk etwas zurückversetzt ist. Wird dies mit Laufen kombiniert, wirkt das so, als ziehe der Charakter sein Bein etwas hinterher.

SSD ist sehr weit verbreitet und hat sich in Computerspielen längst bewährt. Dynamische Bewegungen können auf Skelettebene berechnet werden. Durch Einfügen von zusätzlichen Pseudo-Gelenken in die artikulierte Struktur können sekundäre Effekte wie Atmung und Muskelkontraktion simuliert werden.

SSD bringt aber auch ernsthafte Probleme bzw. Einschränkungen mit sich: Die wesentlichen Defizite sind die Verformung des Volumens an Gelenken (Abbildung 2.5 li.) und Zusammenfallen am Gelenk durch Verdrehung eines Knochens (siehe Abbildung 2.5 re.).

In Abbildung 2.4 sieht man, wie der unerwünschte Knick an einem Gelenk (Abbildung 2.5 li.) entsteht. Die Punkte v1 und v2, die im neutralen Zustand des Zylinders übereinander liegen, sind durch die Beugung der Knochen verschoben worden. Der durch Linearkombination errechnete Endpunkt V liegt genau dazwischen, was eine unschöne Einbuchtung zur Folge hat.

Einen vollständigen Kollaps erhält man schon bei Verdrehung eines Gelenkes um 180 Grad, was im Ellenbogen durchaus anatomisch möglich ist. (siehe Abbildung 2.5 re.) Das kann durch Einfügen zusätzlicher Pseudo-Gelenke teilweise vermieden werden.

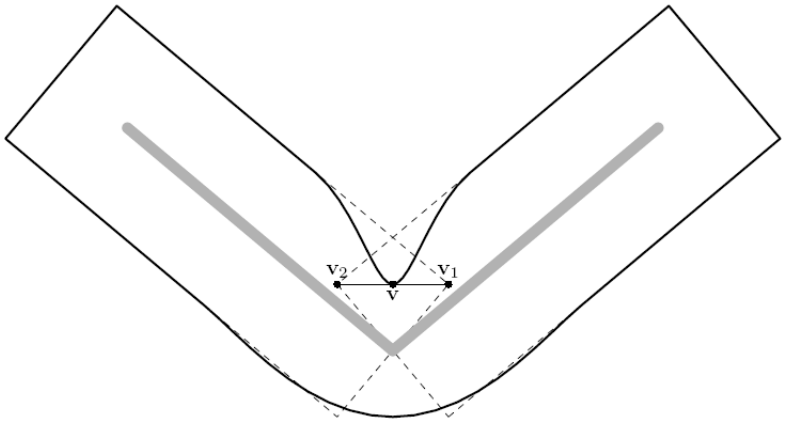


Abbildung 2.4. Knick durch Beugung beim Skinning [Merry(2006)]

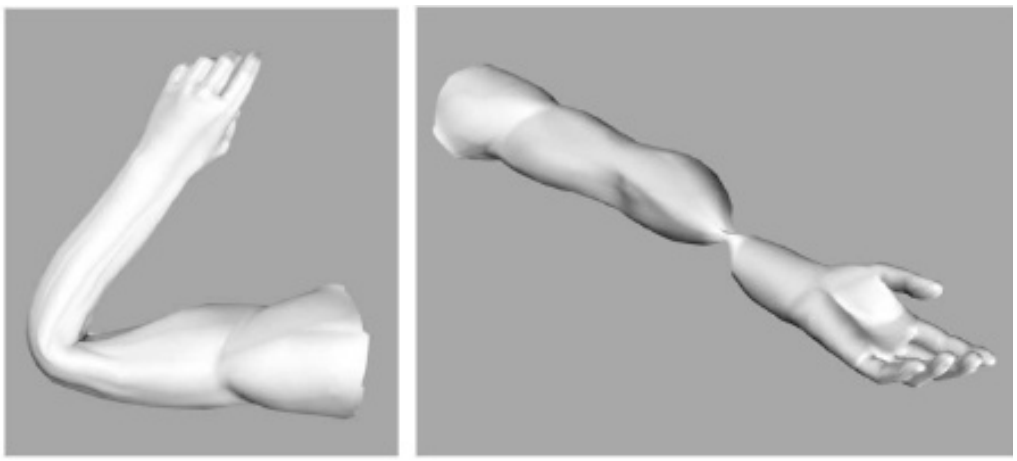


Abbildung 2.5. Probleme beim linearen Skinning[Mohr (2003)]

2.4.5 Multi-Weight-Enveloping

Eine Erweiterung von SSD zur Umgehung der Probleme nennt sich Multi-Weight-Enveloping (MWE) und wurde von [Wang et al. (2002)] entwickelt. Dabei wird zu jedem Vertex nicht nur ein Gewicht pro Knochen, sondern eine ganze Matrix von Gewichten mitgegeben. Durch die neuen Freiheitsgrade kann man die Abhängigkeit genau steuern. Man hat somit mehr Kontrollmöglichkeiten als mit SSD und kann damit die SSD-typischen Probleme umgehen. Die notwendigen Gewichte werden mit einem *least-squares solver*[5] anhand von Beispielposen ermittelt.

[5]Least squares solver (engl.) = Eine mathematische Methode, um zu einer gegebenen Datenmenge eine Funktion zu finden, die diese Daten hinreichend approximiert. Es wird versucht, den quadratischen Fehler zwischen den Daten und der Funktion zu minimieren.

2.4.6 Pose Space Deformation

Ein anderer Ansatz zur Deformierung eines Körpers ist *Deformation by Example.* Damit wird versucht, diese Verformungen zu umgehen, indem auf viele gespeicherte Schlüsselposen zugegriffen wird. [Lewis et. al (2000)] nannten ihre Technik *Pose Space Deformation*. Sie ist eine Verallgemeinerung der *Shape Interpolation* (vgl. Kapitel 2.7.2) . Die grundlegende Idee ist, dass die verwendeten Basisformen einen abstrakten Raum aufspannen, in dem neue Formen durch Interpolation oder Extrapolation entstehen. Da beim SSD die Position jedes Eckpunktes nur für die *bind pose* definiert ist, ist jede andere Körperhaltung immer eine Extrapolation. Durch Hinzufügen von weiteren Basisformen wird der gesicherte Parameterraum erweitert. Die Auswahl der Basisformen ist willkürlich, somit im Parameterraum der Skelettanimation (die Winkel zwischen den Gelenken) unregelmäßig verteilt. Damit handelt es sich um ein sog. *scattered data interpolation* Problem, das mit Hilfe von Radialen Basisfunktionen (RBF) gelöst werden kann [Thalmann et al.(2004)]. Damit können die Probleme gelöst werden, die beim SSD auftreten, wie aus Abbildung 2.6 und 2.7 hervorgeht. Allerdings bedeutet das auch, dass mit steigendem Freiheitsgrad des Skelettes auch mehr Formen vorhanden sein müssen, um alle möglichen Bewegungen adäquat ausdrücken zu können.

Abbildung 2.6. Deformierung einer Schulter mit Skeletal Subspace Deformation [Lewis et. al (2000)]

Abbildung 2.7. Deformierung mit Pose Space Deformation [Lewis et. al (2000)]

2.4.7 Animation Space

[Merry et al. (2006)] entwickelten *Animation Space*, eine hybride Lösung von PSD und MWE, indem sie zeigen, dass sich die neun Gewichte pro Eckpunkt (MWE) auf drei reduzieren lassen, die in Charakteranimationen tatsächlich benötigt werden. Welche Gewichte das sind, wird aus den Basisformen (PSD) vorab ermittelt. Die Autoren entwickelten damit nach eigenen Angaben ein neuartiges Verfahren, das mindestens so effizient auf der Grafikhardware berechnet werden kann wie SSD und zusätzlich frei ist von dessen Problemen.

2.4.8 Keyframing

Wenn eine geeignete Methode zur Verformung des Körpers gefunden wurde, stellt sich die Frage, wie Animationen definiert werden. Eine Animationssequenz besteht aus mehreren diskreten Körperhaltungen, sog. Keyframes, die nacheinander betrachtet eine Bewegung vermitteln sollen. Zu jedem Zeitpunkt kann dann durch Interpolation von zwei benachbarten Keyframes die entsprechende Körperhaltung ermittelt werden. Die Keyframes können durch Transformation aller beteiligten Gelenke in einem 3D-Programm erzeugt werden, deren Qualität jedoch von den Fähigkeiten und der investierten Zeit eines Mediengestalters abhängt. Natürliche Bewegungen sind sehr komplex und vielfältig, weshalb für überzeugendere Animationen alle Bewegungen eines realen Akteurs aufgezeichnet und später auf ein virtuelles Skelett übertragen werden.

2.5 Bewegung des Körpers

2.5.1 Geschichtliches

Die Analyse natürlicher Bewegungen fand schon Aristoteles (384-322 v.Chr.) faszinierend. Er untersuchte sie in seinem Werk 'De motu animalium' – von der Bewegung der Tiere.

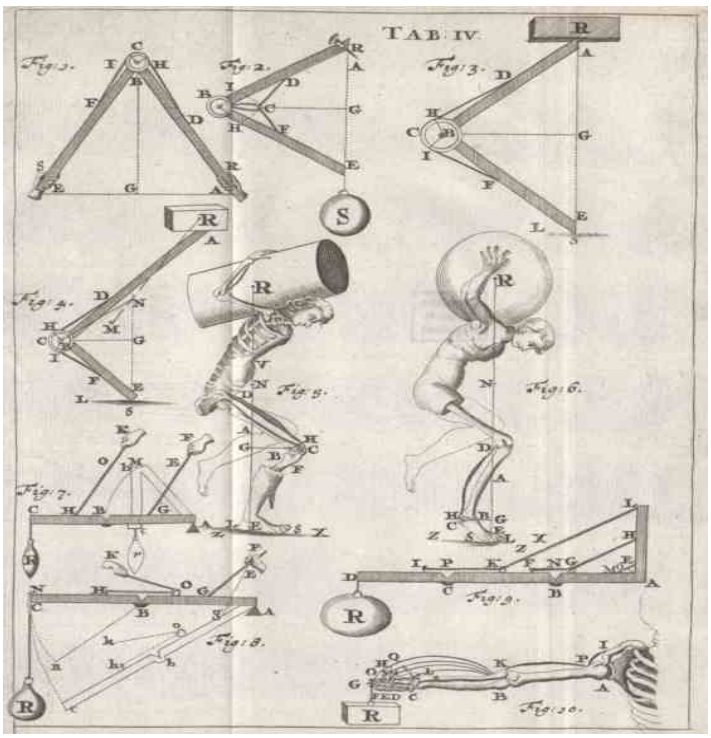


Abbildung 2.8. Der menschliche Bewegungsapparat als arbeitende Maschine [www:Borelli]

Unter gleichem Namen veröffentlichte Giovanni Alfonso Borelli (1608-1679) fast zweitausend Jahre später eine Publikation, in der er den Mensch als Arbeit leistende Skelett-Muskel-Maschine ansah. Mit „De motu animalium“ wurde Borelli zu einem frühen Vorläufer der modernen Biophysik.

Viele der namhaften Wissenschaftler in der Zeit danach beschäftigten sich ebenfalls mit Biomechanik: Newton (1642-1727), Bernoulli (1700-1782), Euler (1707-1783), Poiseuille (1799-1869), Young (1773-1829), etc. [www:xsens]

2.5.2 Kinematik

Die Kinematik beschäftigt sich mit der Animation von Objekten, ohne die Ursache der Bewegungen zu beachten. Man unterscheidet *Forward Kinematik (FK),* und *Inverse Kinematik (IK).*

Bei der FK werden, vom Wurzelknoten des Skeletts ausgehend, alle Winkel der Gelenke bestimmt. Daraus ergibt sich die Position aller Glieder. Bei IK dient als Eingabe eine gewünschte Position eines Endeffektors (z.B. der Hand). Mit algorithmischen Verfahren wird versucht, eine plausible Lösung für das mehrdeutige Problem der Winkelverteilung in den Gelenken zu finden. Das ist nötig, damit man z.B. einen Charakter ein Objekt ergreifen lassen kann. Allerdings steigt mit Anzahl der Gelenke auch die Anzahl der möglichen, aber unnatürlichen Lösungen. Um eine möglichst natürliche Animation zu erreichen, wurden deshalb von [Watt & Watt (1992)] *Constraints*[6] eingeführt, die die Anzahl der möglichen Lösungen beschränken und somit zu plausibleren Ergebnissen führen.

2.5.3 Dynamische Animation

Schon 1985 schlugen [Armstrong & Green (1985)] *physical-based modeling* zur Charaktersteuerung vor. Sie betrachten den menschlichen Körpers als dynamisches System, in dem Kräfte wirken, die Beschleunigungen verursachen, was wiederum zu Geschwindigkeitsänderungen führt und letztlich die Position eines Objektes bestimmt. Durch Simulation dieser Kräfte kann ein höherer Grad an Realismus erzielt werden. Im Gegensatz zu einer kinematischen Animation würde beispielsweise ein stehender Charakter, auf den Gravitationskräfte wirken, sofort stürzen, sofern er sich nicht in einer Gleichgewichtsposition befindet. Die Fragestellung, welche Kräfte erforderlich sind, um das Gleichgewicht zu halten, ist allerdings nicht trivial. Daher sind solcherlei Berechnungen noch nicht für Echtzeitanwendungen geeignet. Passive Systeme, die nur auf äußere Kräfte reagieren, können allerdings in Echtzeit simuliert werden, was beispielsweise bei Haaren und Kleidung Anwendung findet.

[6]Constraint (engl.) = Zwangsbedingung

2.5.4 Animation anhand aufgezeichneter Daten

Muybridge (1830-1904) war der erste, der menschliche und tierische Bewegungen systematisch photographierte. In Abbildung 2.9 sieht man eine Sequenz von seiner Studie zur Bewegung von Pferden im Galopp, womit er zeigen konnte, dass sich einige Künstler bei der Darstellung der Bewegung eines Pferdes getäuscht hatten.

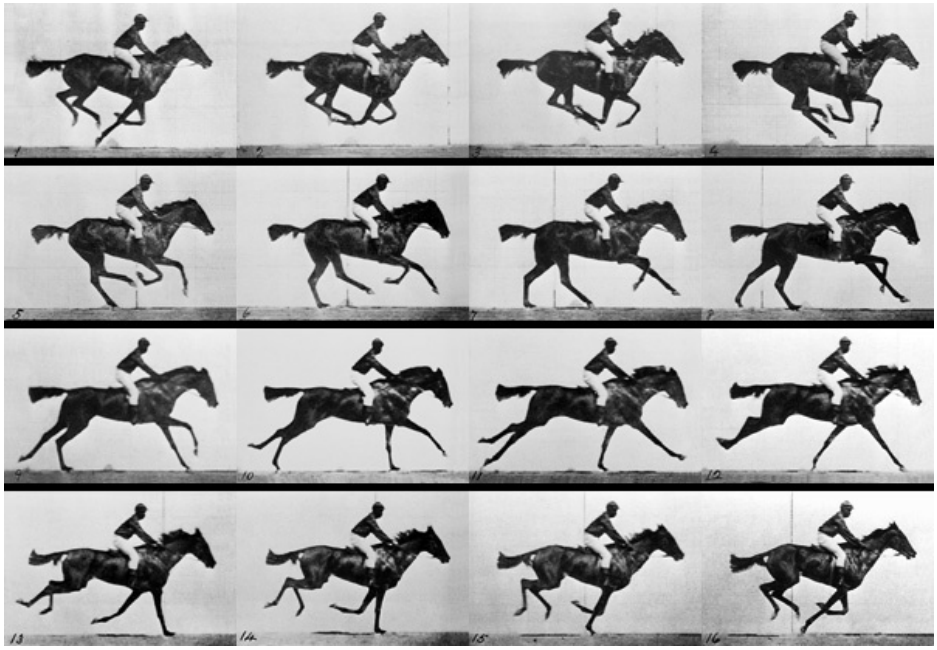


Abbildung 2.9. Bildfolge eines galoppierenden Rennpferds *[Muybridge1887]*

Muybridges Technik wurde später wissenschaftlich von Étienne-Jules Marey (1830-1904) verwendet, der als Pionier der modernen Bewegungsanalyse angesehen wird.

Abbildung 2.11. Marey's Anzug zur Bewegungsanalyse. [www:Marey]

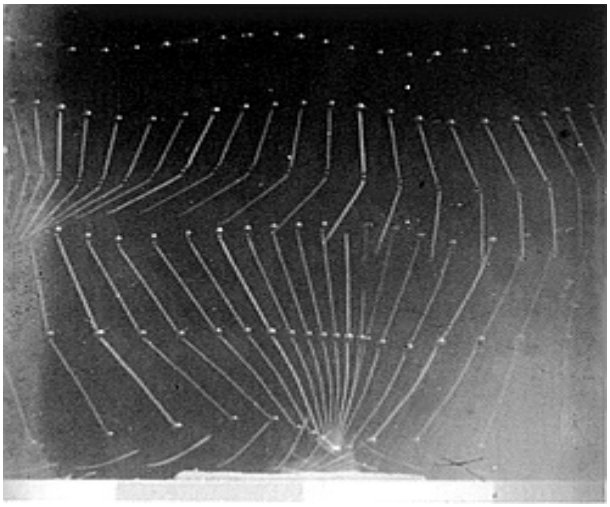


Abbildung 2.10.Die damit aufgenommenen Bewegungen. [www:Marey]

2.5.5 Rotoskopie

Diese Aufzeichnungen natürlicher Bewegung inspirierten Max Fleischer, der 1914 die Rotoskopie entwickelte. Dabei wird eine reale Szene Bild für Bild aufgenommen und anschließend von hinten auf eine matte Glasscheibe projiziert, um sie anschließend abzuzeichnen. Das Verfahren wurde in der Trickfilmindustrie vor allem für realistisch gestaltete menschliche Charaktere eingesetzt, besonders dann, wenn sehr komplexe Bewegungen wie z.B. Tanzszenen gefordert waren.

2.5.6 Modernes Motion Capturing[7]

Das Verfahren der Rotoskopie basiert auf einer zweidimensionalen Abbildung der Wirklichkeit. Moderne, computergestützte Bewegungsaufnahme benötigt aber Bewegungen in allen drei Dimensionen. Systeme, die Positionen und/oder Orientierung von Körperteilen messen, um diese direkt auf einen virtuellen Charakter zu übertragen, nennt man *direkt*, während *indirekte* Systeme die gemessenen Daten auf andere Parameter übertragen, wie z.B. bei *Mike the Talking Head* [deGraf(1988)].

In den letzten Jahren wurden viele verschiedene Technologien mit unterschiedlichen Ansätzen und Eigenschaften entwickelt.

Gemeinsames Ziel aller Motion Capture-Systeme ist die Erfassung von Position und Orientierung aller Gelenke im Raum mit möglichst vielen Bildern pro Sekunde. Das Abspielen von Daten, die mit Motion Capturing aufgenommen wurden ist ein kinematisches Animationsverfahren, weil es nicht die ursächlichen Kräfte betrachtet, sondern die letztendlichen Positionen aller Gelenke ermittelt und aufzeichnet.

Motion Capture-Systeme können klassifiziert werden in mechanische, optische, elektromagnetische, akustische und inertiale[8] Systeme und hybride Varianten.

Die folgende Tabelle 2.1 bietet eine Übersicht über diese Systeme.

Leider sind die beim Motion Capturing erfassten Bewegungsdaten noch fehlerbehaftet, weshalb sie selten ohne Nachbesserung in einem Animationsprogramm verwendet werden können.

[7] Motion Capturing (engl.) = Bewegungserfassung
[8] Inertial = trägheitsbasiert

Technologie / Beispiel	Verfahren	Eigenschaften	Messwerte
Mechanisch Waldo [Walters (1989)]	Bewegliches Exoskelett[9] wird am Körper des Actors fixiert. Potenziometer nehmen analoge mechanische Bewegung auf, die digitalisiert und in Gelenkwinkel umgerechnet wird.	Fehleranfällige Konstruktion. Stark eingeschränkte Bewegungsfreiheit. Präzise Messungen möglich.	Nur lokale Winkel.
Optisch VICON [www:Vicon]	Passiv reflektierende oder aktiv leuchtende Markierungen werden am Actor fixiert und mit mehreren (Infrarot-)Kameras verfolgt. Korrespondierende Punkte werden in 3D Punkte umgerechnet. Je 4-6 Markierungen werden zu eindeutigen Formen zusammengefasst, um Orientierung daraus berechnen zu können.	Kamerakonfiguration muss bekannt sein. Kalibrierung erforderlich. Jeder Punkt muss immer aus mindestens zwei Kameras gesehen werden. Kameras teuer. Rechenintensiv. Kleiner Aktionsradius. Gute Qualität.	Globale Position und Orientierung
Elektromagnetisch Polhemus STAR*TRAK [www:Polhemus]	Sender emittiert Magnetfeld. Sensoren am Actor messen Position und Orientierung anhand von Magnetfeldänderungen.	Keine Verdeckungsprobleme. Verzerrung durch Metall in der Umgebung. Ungenau durch Rauschen.	Globale Position und Orientierung
Akustisch Logitech Ultrasonic Tracking [www:vrdepot]	Kurze Ultraschalltöne werden von mehreren Mikrofonen aufgenommen, um Position zu ermitteln. Empfänger am Körper und Sender fest oder andersherum möglich.	Verdeckungsprobleme und Störung durch Reflexion. Hohe Latenz. Teilweise ungenau. Räumlich beschränkt, da Sender oder Empfänger fest installiert sind. 3 Sensoren für 6 DOF[10] nötig.	Globale Position.
Inertial Moven [www:Xsens]	Trägheitssensoren messen lineare und rotatorische Beschleunigung. Diese wird in (Dreh-) Geschwindigkeit und dann Position und Orientierung umgerechnet.	Beschleunigungsmessung erlaubt bessere Vorhersagen. Drifting: Da nur relative Werte erfasst werden, addieren sich Ungenauigkeiten mit der Zeit. Dadurch driften Position und Orientierung ab.	Lokale Position und Orientierung. Werte driften ab.
Hybrid[11] InterSense IS-900 (Inertial + Ultraschall) [www:InterSense]	Verfahren werden kombiniert, um optimale Lösung zu erreichen. Normalerweise Kombination aus Trägheitssensoren und einer globalen Technik, um das Driften bei Trägheitssensoren auszugleichen und trotzdem geringe Latenzen zu bekommen.	Drifting kann korrigiert werden. Geringe Latenz. Räumlich beschränkt. Keine Interferenzprobleme. 3 Sensoren für 6 DOF nötig.	Globale Position und Orientierung ziemlich genau.

Tabelle 2.1: Vergleich von Motion Capture Verfahren

[9] hier: Mechanische Konstruktion aus Metallstangen, die am Körper fixiert wird.
[10] Degrees of Freedom (engl.) = Freiheitsgrade
[11] Hybrid = ein aus unterschiedlichen Arten oder Prozessen zusammengesetztes Ganzes.

2.5.7 Markerloses Tracking

Markerlose optische Verfahren verfolgen mit Hilfe von Bildverarbeitung und *Pattern Matching* vom Benutzer oder Programmierer vorgegebene Punkte in einem Video (vgl. Abbildung 2.14). Das bedeutet praktisch keinen Mehraufwand vor der Aufnahme und ist außerdem sehr kostengünstig, da mit einer einzigen konventionellen Kamera realisierbar.

Allerdings ist die Qualität sehr stark von den Lichtverhältnissen abhängig und nicht so genau wie markerbasierte Verfahren. Mit mehreren Kameras zu arbeiten ist auch schwierig, da die nachverfolgten Punkte je nach Blickwinkel sehr unterschiedlich aussehen können.

Die Punkte müssen in jedem Bild wiedergefunden werden, da sie sonst undefiniert sind. Das schränkt die Bewegungsfreiheit stark ein. Für manche Aufnahmen z.B. zum Extrahieren von Gesichtsmerkmalen (Position der Augenbrauen, Blinzeln etc.) ist es aber durchaus sinnvoll.

Abbildung 2.13.Motion Caputuring [Jörg (2005)]

Abbildung 2.12.Die verwendeten Kameras [Jörg (2005)]

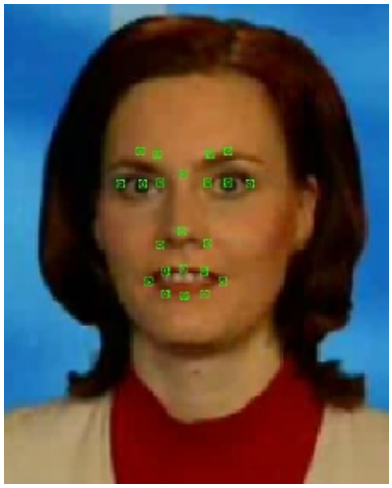


Abbildung 2.14.Makerloses Tracking [Stöcker (2004)]

2.6 Modellierung des Gesichtes

Ein virtuelles Gesicht kann per Hand in einem 3D-Programm erstellt sein. Viel interessanter ist aber das Klonen eines realen Modells. Die meisten hier vorgestellten Techniken können so ähnlich auch zum Klonen des restlichen Körpers verwendet werden. Folgende Teilaufgaben gehören zum Klonen eines Gesichtes:

- Erfassen der Form des menschlichen Gesichtes.
- Realistische hochauflösende Texturierung.
- Funktionale Informationen, um das Gesicht animieren zu können.

2.6.1 Form des Gesichtes

Im zweiten Kapitel von [Thalmann et al.(2004)] werden folgende Verfahren zum Erstellen von virtuellen Gesichtern, einem Spezialfall der 3D-Rekonstruktion eines Kopfes, vorgestellt:

- *Plastische Modelle*: Durch plastisches Modellieren eines Gesichtes aus Ton oder einem ähnlichem Material und anschließendem Digitalisieren können künstliche Charaktere interaktiv und künstlerisch gestaltet werden. Ein prominentes Beispiel ist die Figur 'Geri' von Pixar [Pixar (1998)].
- *Organisierte Fotos:* Mit halbautomatischer Merkmalserkennung lässt sich ein generisches Kopfmodell geometrisch verändern, um dem aufgenommen Gesicht ähnlicher zu werden. Dieser Prozess geht schnell, ist aber wegen der wenigen relevanten Merkmale recht ungenau.
- *Arbiträre Fotos:* Mit A-priori-Wissen in einer großen Datenbank von bereits eingescannten Gesichtern und ein paar Stunden Rechenzeit, finden [Blanz & Vetter (1999)] ein Modell, das der Eingabe in Form eines oder mehrerer Fotos sehr ähnlich ist. Dazu wird ein statistisches Modell verwendet, das auf 200 mit einem Laser Scanner aufgenommenen Köpfen beruht.
- *Tiefeninformationen*: Mit verschiedenen Techniken ist es möglich, Tiefeninformationen zu erfassen. 3D-Laser Scanner [Blanz & Vetter (1999)] scheinen die praktikabelste Möglichkeit zu sein, die Form eines Gesichtes zu erfassen. Allerdings ist der Prozess in seiner Komplexität nicht mit der Aufnahme eines Fotos vergleichbar, da viele Nachbearbeitungsschritte nötig

sind, um ein brauchbares Modell zu erhalten. Laser-Scanner sind teuer, liefern dafür hochauflösende Modelle, aber ohne funktionale Informationen.

- *Stereoskopie*: Aus stereoskopischen Aufnahmen lassen sich ebenfalls 3-D Punkte errechnen [Lee et al. (2000)]. Allerdings sind diese Daten ungenau, da sich korrespondierende Punkte in beiden Bildern nur an markanten Stellen finden lassen.
- *Videosequenzen*: Aus Videoaufnahmen lassen sich ebenfalls 3D-Daten rekonstruieren. Je mehr Frames eine Sequenz hat, desto besser wird dabei das Ergebnis. [Guenter et al. (1998)] markierten das Gesicht vor der Aufnahme. [Fua (2000)] entwickelte eine Technik, die ohne diese Markierungen auskommt, indem sie a-priori-Wissen in Form eines generischen Modells nutzt. Das Problem liegt in der Minimierung des Fehlers nach einer Rückprojektion. Mit dem generischen Modell lassen sich Zwangsbedingungen einführen, die die Komplexität des Minimierungsproblems reduzieren.

Abbildung 2.15: ShapeSnatcher von Eyetronics
(Digitalkamera mit Gitterblitz)

- *Streifengenerator:* Eine elegante und kostengünstige Lösung besteht darin, ein reguläres Streifen- oder Gittermuster auf das Gesicht einer Person zu projizieren, während es aus einem, oder mehreren Positionen mit einer Kamera aufgenommen wird. Die Verzerrung des Gitters ermöglicht es, die Tiefeninformationen zu berechnen. Die Firma Eyetronics entwickelten Shape*Snatcher* [www:Eyetronics], ein kommerzielles System, das auch für das Modell in dieser Arbeit verwendet wurde.

2.6.2 Texturierung

Zu einem realistischen Modell gehört neben der Form auch die Farbe. Dazu werden normalerweise orthogonal aufgenommene Bilder, von vorne und von der Seite zusammengefügt und farblich aneinander angepasst, um möglichst hochauflösende Texturen für alle Stellen zu erhalten.

Wenn Photos oder Videoaufnahmen zum Erfassen des Modells verwendet wurden, ist schon bekannt, wo ein Punkt in der Textur auf das 3-D Modell abgebildet werden muss. Bei Laserscans oder ähnlichem muss man diese sog. *Texturkoordinaten* per Hand definieren.

Abbildung 2.16: Kamera mit Polfilter zum Erfassen von Texturen

2.6.3 Funktionale Informationen

Damit das eingescannte Gesicht andere Gesichtsausdrücke zeigt, müssten einige Eckpunkte verschoben werden. Alle möglichen Gesichtsausdrücke für einen neuen Kopf per Hand zu erzeugen, wäre sehr mühsam. Daher benutzt man in der Regel nicht direkt die eingescannten Daten, sondern ein bereits artikuliertes Kopfmodell, dessen Topologie vereinfacht und für die spätere Animation sinnvoll verteilt ist, und verschiebt dessen Eckpunkte so, dass es den eingescannten Daten entspricht. Nach [Thalmann et al.(2004)] muss man dazu die beiden Modelle aufeinander legen und kann dann z.B. mit der Dirichlet Free Form Deformation (DFFD) die nötige Verschiebung der Punkte des generischen Kopfmodells berechnen.

2.7 Gesichtsanimation

Die natürlich wirkende Animation des Gesichtes stellt für die Computergrafik eine besondere Herausforderung dar. Das Gesicht ist der Körperteil, der die meiste Aufmerksamkeit des Betrachters bekommt. Über Gesichtsausdrücke werden Gefühle transportiert und nur, wenn die Gesichtsausdrücke zur jeweiligen Situation passen, kann ein virtueller Charakter Lebendigkeit suggerieren. Die Verformungen des Gesichtes basieren auf vielen Muskeln und weichem Gewebe und nicht auf Skelettbewegungen wie bei der Ganzkörperanimation. Zur effizienten Animation müssen auch beim Gesicht die Bewegungen abstrahiert gesehen werden. Mehrschichtige Verfahren, die einzelne Muskeln und Fettgewebe simulieren (z.B. [Parke & Waters (1996)]) sind aufgrund der hohen Komplexität längst nicht echtzeitfähig.

Viele Animationsverfahren beziehen sich auf eines der theoretischen Modelle, um die vielfältigen Bewegungen eines menschlichen Gesichtes abstrahiert beschreiben zu können.

Das älteste Modell ist das *Facial Action Coding System* (FACS) [Ekman (1978)]. Es wurde entwickelt, um Gesichtsausdrücke zu untersuchen. Die Verformungen des Gesichtes, werden in 46 nicht zerlegbare Action-Units (AU) eingeteilt, deren Ausprägungen jeden beliebigen Gesichtsausdruck beschreiben können.

Ein neuerer Standard ist im MPEG-4 Modell enthalten, das 84 *feature points*[12] definiert, deren Position im Raum zur vollständigen und eindeutigen Beschreibung eines Kopfes ausreichen. Zur Animation werden 68 *Facial Animation Parameters* (FAP) eingeführt, die jeweils einen oder mehrere Punkte relativ zu Bezugsgrößen im Modell verschieben. Umfangreiches Material über MPEG-4 findet sich in [Pandzic & Forchheimer (2002)]

2.7.1 Skelettanimation

Da Muskelmodelle zu komplex für Echtzeitberechnungen sind, liegt es nahe, bei der Gesichtsanimation auch das in Kapitel 2.4 beschriebene Verfahren der Skelettanimation einzusetzen. Dazu werden im Gesicht zusätzliche *Joints* angebracht, denen die umliegenden Vertices zugeordnet werden. Durch Drehung dieser *Joints* lassen sich Eckpunkte im Gesicht verschieben.

[12]Feature points (engl. Merkmalspunkte) =Markante Punkte im Gesicht, wie z.B. Nasenspitze oder Mundwinkel

Diese Animationstechnik bietet vielfältige Möglichkeiten, vor allem ist sie geeignet zur Verwendung mit *Facial Motion Capture*, weil die Bewegungen der einzelnen *Joints* nicht voneinander abhängig sind und direkt an Markierungen im Gesicht gebunden werden können.

Problematisch ist aber die große Anzahl an *Joints,* die verwendet werden müssen, um möglichst viele Ausdrücke erfassen zu können. Das erschwert eine zuverlässige Gewichtung sehr und führt außerdem zu erhöhtem Rechenaufwand.

2.7.2 Shape Interpolation

Die populärste Methode der Gesichtsanimationen ist die *Shape Interpolation*. Ein *Shape* beschreibt einen diskreten Gesichtsausdruck. Andere gebräuchliche Ausdrücke sind „Morph Target“, „key-shape“, „shape key“, oder „Pose“. Diese Gesichtsausdrücke können gemischt werden, was „shape interpolation“, „blending“, „morphing“, „Geometrie Interpolation“ oder „per-vertex Animation“ genannt wird.

Man unterscheidet dabei zwischen „average-“ und „additive-“ Blending. Beim *average blending* wird der Durchschnitt der einzelnen Shapes ermittelt, um das Gesamtgesicht zu erzeugen. Dadurch heben sich aber Effekte gegenseitig auf. Daher verwendet man normalerweise *additives Blending*: alle *Shapes* werden relativ zu einem neutralen Gesicht, dem „default“- oder „neutral“-Shape gespeichert, mit ähnlichem Zweck wie die *binding Pose* bei der Skelettanimation. Beim Blending werden alle Shapes wieder anteilig auf das neutrale Gesicht aufaddiert. Damit lassen sich z.B. die Augen unabhängig vom Mund animieren.

Dafür ist es erforderlich, dass alle *Shapes* die gleiche Topologie haben, damit jeder Eckpunkt seinem Partner im neutralen *Shape* eindeutig zugeordnet werden kann.

Die Animation geschieht, wie beim Skelett, mit Keyframe Interpolation; jeder Keyframe definiert eine Gewichtsverteilung für die Blendshapes, die einen konkreten Gesichtsausdruck bestimmt. Zwischen je zwei Ausdrücken wird linear interpoliert. Dadurch lassen sich einfache Animationen erstellen. Nachteilig ist aber die lineare Abhängigkeit der einzelnen Ausdrücke und die Tatsache, dass die Ausdrucksstärke davon abhängt, wie viele Blendshapes der Kombination zugrunde gelegt werden. Da die Blendshapes exakt die gleiche Topologie haben müssen, um die Linearkombination

zu ermöglichen, steigt der Speicherverbrauch proportional zur Anzahl der eingesetzten Blendshapes und deren Anzahl an Vertices.

2.8 Sprachsynthese

2.8.1 Phoneme

Ein Phonem ist die theoretische Repräsentation eines Lautes und die Basiseinheit der menschlichen Sprache. Aus ihnen werden Silben gebildet, die wiederum zu Wörtern zusammengefasst werden.

[Bußmann (2002)] definiert Phoneme auf Seite 510 als

„kleinste, aus der Rede abstrahierte, lautliche Segmente mit potentiell bedeutungsunterscheidender (distinktiver) Funktion."

Natürliche Sprachen bestehen aus ca. 10-80 unterschiedlichen Phonemen. In der deutschen Sprache gibt es etwa 40.

2.8.2 Spracherzeugung beim Menschen

Natürliche Sprache wird von den Sprechorganen erzeugt. Die Atmungsorgane erzeugen beim Ausatmen einen Luftstrom, der beim Sprechen durch die im Kehlkopf befindliche Stimmritze zu einem Ton wird. Dieser Grundton bestimmt die Höhe und Stärke der Sprache, wird aber erst in den Lautformungsorganen zu Phonemen geformt.

2.8.3 Künstliche Spracherzeugung

Sprachsynthese ist der Versuch, eine menschliche Stimme zu erzeugen, die möglichst verständlich und natürlich klingt. Eine umfassende Übersicht mit Hörbeispielen gängiger Systeme findet man unter [www:ttssamples].

Das Ziel der sog. *Text-To-Speech* (*TTS*)-Systeme ist das Vorlesen eines beliebigen Textes. Dazu muss der Text in mehreren Schritten verarbeitet werden.

Bei der Textvorformatierung wird der Text, in dem Zahlen, Sonderzeichen, Abkürzungen etc. vorkommen können, in eine einheitliche Form gebracht. Schwierigkeiten bereiten außerdem phonetische Mehrdeutigkeiten (ein Graphem steht

für mehrere Phoneme, z.B. V in Vater und Vase) und graphemische Mehrdeutigkeiten (ein Phonem wird durch mehrere Grapheme abgebildet, z.B. /f/ in Fisch, Vater, Photo).

2.8.4 Grapheme-Phoneme-Mapping

Im nächsten Schritt werden die Grapheme in Phoneme umgewandelt. Das kann geschehen durch grammatische Umformung mit vielen Regeln, die von der jeweiligen Sprache abhängig sind, oder durch Lexika, in denen Wörter mitsamt ihrer phonetischen Übersetzung gespeichert sind.

Viele Systeme verwenden zusätzliche Ausnahme-Lexika, falls diese von allgemeinen Regeln abweichen. Das Ergebnis dieses Schrittes ist eine Folge von Phonemen.

2.8.5 Prosodie

Eine synthetische Sprache, die durch einheitliches Aneinanderreihen von Phonemen gleicher Länge und Stärke erzeugt wird, klingt sehr unnatürlich und mechanisch. Dabei fehlt nämlich die *Prosodie*, ein Begriff aus der Phonetik, der folgendermaßen definiert wird:

„Gesamtheit spezifischer sprachlicher Eigenschaften wie Akzent, Intonation, Quantität, (Sprech-) Pausen. Prosodie bezieht sich im Allgemeinen auf Einheiten, die größer sind als ein einzelnes Phonem. Zur Prosodie zählt auch die Untersuchung von Sprechtempo und Sprechrhythmus.“

- [Bußmann (2002)], Seite 542

Zur Erzeugung von Prosodie werden regelbasierte [Klatt (1979)], wissensbasierte und hybride Verfahren verwendet. [Jokisch et al. (1998)].

In [Klatt (1979)] findet sich eine weit verbreitete Formel zur Berechnung der Tondauer, dessen Parameter experimentell erfasst werden müssen und mit neuronalen Netzen gelernt werden können.[Jokisch et al. (1998)].

Intonationen, wie beispielsweise das Erhöhen des Tons am Ende einer Frage, werden von [Mixdorff & Fujisaki (1995)] behandelt. Eine Evaluation zur Prosodie findet sich in [Sonntag (1999)].

2.8.6 Signalsynthese

Der letzte Schritt der Verarbeitungskette in einem TTS-System ist die Erzeugung des Signals. Man unterscheidet *regelbasierte* und *konkatenative* Synthese. Zu den *regelbasierten* Systemen zählt man die *artikulatorischen* Systeme, die den natürlichen Sprechvorgang mit hohem Rechenaufwand und mäßigem Erfolg physikalisch simulieren sowie die *formantbasierten* Systeme. Letztere erzeugen das Ausgangssignal als mathematische Funktion über Frequenz, Dauer und Höhe eines Phonems. Jedem Phonem werden dabei charakteristische Frequenzen zugeordnet, die sog. *Formanten*. Dadurch wird das Signal durch eine mathematische Funktion angenähert, was ziemlich unnatürlich klingt.

Konkatenative Systeme verketten einzelne Stücke aufgenommener Sprache hintereinander. Einfachste Form ist die *Diphon*-Synthese, die für jedes mögliche Paar von Phonemen auf einen gespeicherten Ton zurückgreift. Analog dazu werden bei der *Triphon*-Synthese Kombinationen von je drei Phonemen gespeichert.

2.8.7 Unit-Selection

Bei der *Unit Selection [Hunt & Black (1996)]* wird dieses Verfahren verallgemeinert. Dazu wird ein großer Sprachkorpus benötigt, der aus Phonemen, Morphemen, Silben, Wörtern und sogar ganzen Sätzen bestehen kann. Diese Segmente werden aus aufgenommener Sprache mit einem Spracherkenner extrahiert und manuell nachbearbeitet. Für jedes Segment wird ein Index erstellt, der aus Metainformationen wie der fundamentalen Tonhöhe, der Dauer, der Position in der höheren Ebene und den Nachbarsegmenten berechnet wird. Zur Laufzeit kann für jedes Phonem dieser Index berechnet und mit einem Entscheidungsbaum die optimale Kette von Segmenten aus der Datenbank ausgewählt werden. Das Sprachsignal wird dann durch Zusammensetzen der Elemente bzw. Units unter Zuhilfenahme von Glättungs- und Koartikulationsfiltern synthetisiert. Damit kann man realistische Sprache erzeugen, jedoch hängt die Qualität stark vom verwendeten Korpus ab, der nur semi-automatisch erstellt werden kann und bei zunehmender Größe außerdem viel Speicherplatz verbraucht.

2.8.8 Phoneme-Viseme Mapping

Ein Visem ist ein Gesichtsausdruck und bezieht sich auf den sichtbaren Zustand der Sprechorgane.

Eine Herausforderung bei der Animation eines sprechenden Gesichtes besteht darin, zu jedem Zeitpunkt das „passende“ Gesicht zu generieren. Vereinfachend geht man meistens davon aus, dass jedem Phonem ein Visem eindeutig zugeordnet werden kann, auch wenn tatsächlich Mehrdeutigkeiten existieren. Diese Zuordnungen nennt man Phoneme-Viseme-Mapping.

2.8.9 Klonen von Sprache

Will man natürliche, aufgenommene Sprache auf einen virtuellen Charakter übertragen, ist es notwendig, die Phoneme aus dem Sprachmitschnitt mitsamt Zeitmarken zu extrahieren. Das geschieht mit einem Spracherkenner, der bei bekanntem Text mit sog. *Forced Alignment [Rabiner (1989)]* gute Ergebnisse liefern kann.

2.9 Programmierbare Grafikpipeline

2.9.1 Standard Grafikpipeline

In der Computergrafik beschreibt die Grafikpipeline den Weg von der dreidimensionalen, vektoriellen Beschreibung einer Szene zum zweidimensionalen, gerasterten Bild. Dafür muss jeder Vertex mit Matrizen mehrfach transformiert werden (vgl. Abbildung 2.17).

Ein Modell besteht aus mehreren Vertices, deren Koordinaten in einer Datei festgehalten werden können. Daher nennt man das Koordinatensystem, in dem diese angegeben sind *model space (Objektkoordinatensystem)*.

In einer Szene werden aber mehrere Objekte an verschiedenen Positionen mit unterschiedlicher Orientierung platziert. Die Transformation, die dies bewerkstelligt, nennt sich *world transformation,* das Koordinatensystem entsprechend *world space*.

Da die Szene aber aus einer bestimmten Kameraposition aus betrachtet werden soll, muss sie mit der *view transformation* in den *view space* gebracht werden. Da es letztlich gleich ist, ob man eine Kamera oder die ganze Szene in die entgegengesetzte Richtung transformiert, wird die Kamera in den Ursprung gesetzt und *model-* und *view-space* zusammengefasst. Mit der *modelview transformation* gelangt man nun also von Modellkoordinaten in homogene 3D-Koordinaten relativ zur Kamera (sog. *Eye coordinates*).

Anschließend folgt die *projection transformation,* die die intrinsischen Eigenschaften der Kamera (perspektivisch oder orthogonal) enthält.

Bei der *Perspektivischen Division* werden alle Werte durch die homogene Koordinate geteilt. Damit ist man wieder im nichthomogenen dreidimensionalem Raum, dem sog. *Kanonischen Volumen.* In diesem Volumen liegen alle Punkte im Sichtfeld der Kamera im Bereich von -1 bis 1.

Mit der *Viewport Transformation* werden diese Koordinaten schließlich auf die Fensterkoordinaten in Pixel umgerechnet.

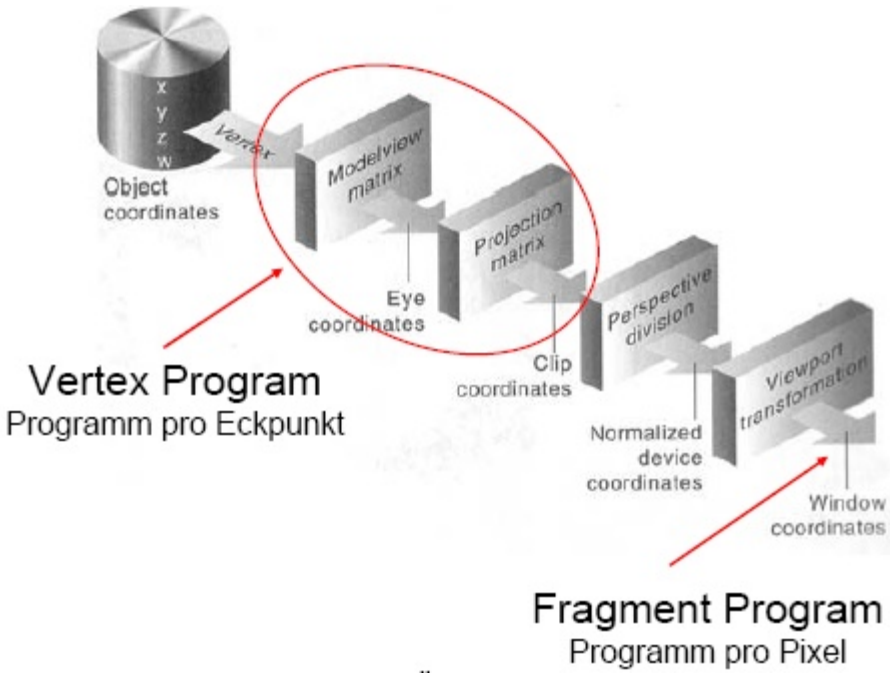


Abbildung 2.17.OpenGL-Pipeline und Einordnung der Hardwareprogramme [Müller & Grosch (2003)]

2.9.2 Programmierbare Pipeline

Die programmierbare Grafikpipeline erlaubt es, Teile der Standardpipeline zu ersetzten. Üblicherweise wird dabei ein *Vertex Program* mit einem dazugehörigen *Fragment Program* auf die Grafikkarte geladen.

Man kann diese Programme in verschiedenen Shader-Sprachen definieren. Die ersten Shader-Sprachen basieren auf Assembler, ähnlich den frühen Sprachen für die CPU, nur mit stark eingeschränktem Befehlssatz. Danach wurden die drei wichtigsten Hochsprachen entwickelt: Die *High Level Shader Language* (HLSL), die Microsoft mit DirectX8 einführte, *C for Graphics* (CG) der Firma nVIDIA [www:nvidia] und die mit OpenGL 2.0 standardisierte *OpenGL Shading Language* (GLSL) [Rost (2004)], die ursprünglich von 3DLabs [www:3DLabs] entwickelt wurde. Gemeinsam ist allen Sprachen eine syntaktische Ähnlichkeit zur Programmiersprache ANSI-C. Die Shader-Sprachen unterliegen im Gegensatz zur CPU-Programmierung aber erheblichen Einschränkungen, was die Anzahl der möglichen Instruktionen und den erlaubten Speicheraufwand angeht.

2.9.3 Vertex Programme

In Abbildung 2.17 kann man sehen, wie ein benutzerdefiniertes *Vertex Program* in die Grafikpipeline eingreift. Das Vertex Programm wird für jeden Vertex (Eckpunkt) der Geometrie aufgerufen. Er hat einen begrenzten Satz an Eingabeparametern, in denen Position, Normale, Texturkoordinaten etc. per Vertex etc. gespeichert werden können. Außerdem kann es auf sog. *uniforme* Parameter zugreifen, also auf Variablen, die aus Sicht des Vertex Programms global sind, weil sie für alle Vertizes gelten. Beispielsweise können Licht- und Kamerapositionen so in das *Vertex Programm* übertragen werden.

Normalerweise wird ein *Vertex Programm* die Transformation der *Vertexparameter* von Objektkoordinaten (model space) ins kanonische Volumen vornehmen und evtl. Texturkoordinaten berechnen. In der schematischen Darstellung (in Abbildung 2.18) wird die Aufgabe des *Vertex Programms* mit *Geometrie Operations* (Geom. Ops.) angegeben. Die für die Transformationen benötigten Matrizen sind uniforme Parameter und werden dem Programm von der Grafik-API[13] zugänglich gemacht. Das *Vertex*

[13]API (engl. application programming interface) = eine Schnittstelle, die von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird

Program hat eine begrenzte Anzahl möglicher Ausgabeparameter, die durch Interpolation in Eingabeparameter für das *Fragment Programm* umgewandelt werden.

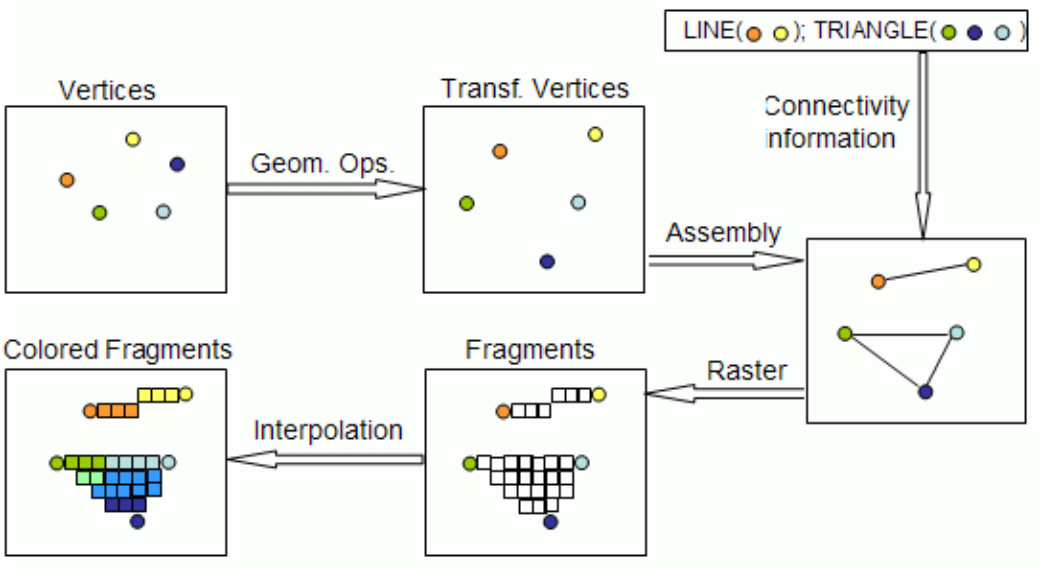


Abbildung 2.18. Schematische Darstellung der Grafikpipeline[www:lighthouse]

2.9.4 Fragment Programme

Nach der Rasterisierung, die von der Grafikkarte durchgeführt wird, wird für jedes zu zeichnende Fragment (Pixel) das Fragment Programm ausgeführt, dessen Aufgabe es ist, die Beleuchtung zu berechnen, also den Farbwert für den Pixel zu ermitteln. Dazu kann das *Fragment Programm* zusätzlich zu den interpolierten Werten aus dem *Vertex Programm* und den uniformen Parametern auch auf Texturen zugreifen.

2.10 Bump Mapping

Bump Mapping ist ein Verfahren, dass verwendet wird, um Details zu modellieren, die kleiner sind, als die verwendete Geometrie. In einer Textur werden feine Unterschiede festgehalten, die beim Rendern verwendet werden um die Oberflächennormale zu verändern oder zu ersetzen.

Bei einem Lambert-Strahler hängt der diffuse Anteil des Lichtes von dem Skalarprodukt der Normale mit dem Lichtvektor ab. Der Lichtvektor beschreibt die Richtung des einfallenden Lichtes. Für eine ebene Fläche bedeutet das eine gleichmäßige Lichtverteilung, wie aus Abbildung 2.6 ersichtlich wird.

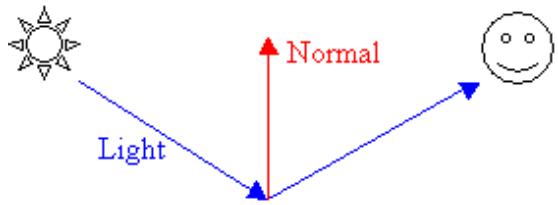


Abbildung 2.19. Flache Oberfläche [www:gamedev]

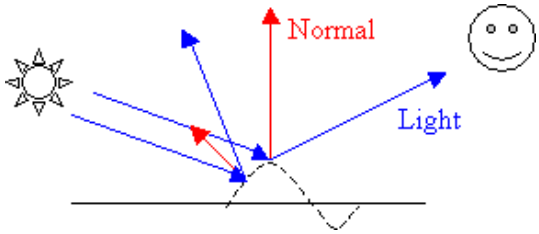
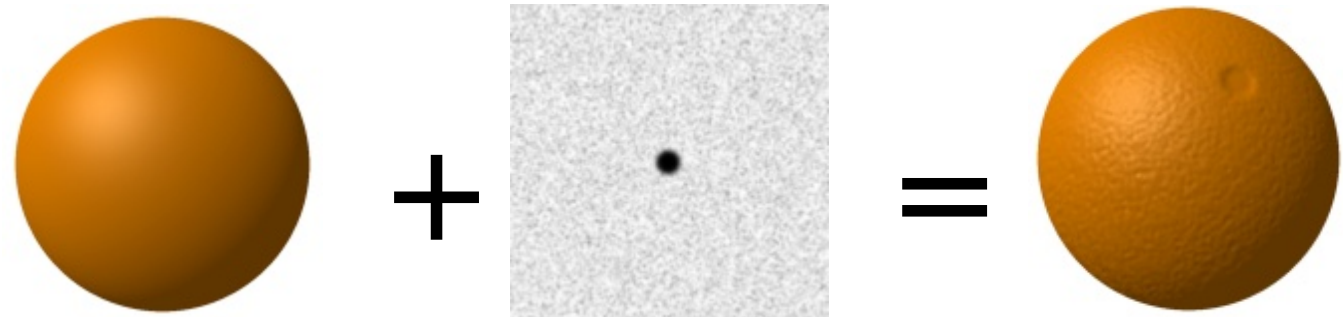


Abbildung 2.20. Oberfläche mit Struktur [www:gamedev]

Wenn die Oberfläche aber leichte Unebenheiten aufweist, wird die Normale, je nach Position auf der Fläche, in eine etwas andere Richtung zeigen (siehe Abbildung 2.7).

2.10.1 Höhenkarten

Eine intuitive Art, die Normale eines Objektes mit einer Textur zu verändern, ist die Verwendung von Höhenkarten (heightmaps). [Blinn (1978)] Dabei wird eine Textur benutzt, die die Höhe der Oberfläche in Richtung ihrer Normale als Grauwert kodiert. Beim Rendern kann die Normale entsprechend verändert werden, so dass die Geometrie detailreicher wirkt, als sie tatsächlich ist:



Kugel *Höhenkarte* *Orange*

Quelle: [www:wikipedia]

2.10.2 Normal Mapping

Beim sog. *Normal Mapping* werden keine Höhendaten benutzt, sondern die Koordinaten der Normale in RGB-Farbwerten kodiert. Eine Möglichkeit ist die Notation der Normalen im uniformen *object space*, der für alle Eckpunkte konstant ist.

Bei der Notation im *tangent space* (auch *texture space* genannt) dagegen, werden die Normalen relativ zu den geometrischen Normalen gespeichert. Das ist der Raum der von Normale, Tangente und Binormale aufgespannt wird. Die Tangente wird typischerweise in Richtung einer Texturkoordinate angenommen. Die Binormale ergibt sich aus dem Kreuzprodukt von Normale und Tangente.

Bei der Lichtberechnung im *fragment program* ist zu beachten, dass alle verwendeten Größen im selben Bezugssystem vorliegen und ggf. in diesen überführt werden müssen.

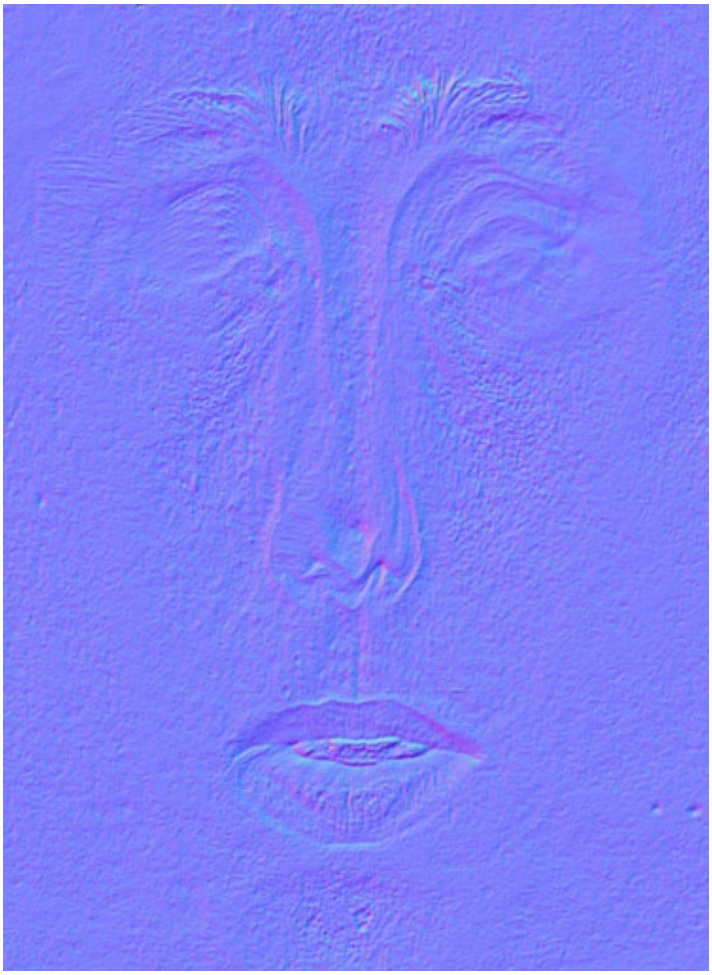


Abbildung 2.21.Ausschnitt der verwendeten Normalmap (tangent space)

3 Konzeption

3.1 Anforderungsanalyse

Die Aufgabe für diese Diplomarbeit war: „Ein **Charakter** soll in Echtzeit **animiert** werden. Er soll seinen Körper bewegen, **Texte vorlesen** (TTS), **synchron** dazu die **Lippen bewegen** und **Emotionen (Augenbrauenbewegung)** zeigen können. Ein hoher Grad an **Realismus** ist wünschenswert. Die **Engine** zur Umsetzung soll selbst gewählt werden."

Daraus ergeben sich folgende Anforderungen:

3.1.1 Wahl einer Grafikengine

Zur Darstellung der Grafik soll eine Engine gewählt werden. Sinnvollerweise eine, die Charakteranimation unterstützt und dabei möglichst effizient ist.

3.1.2 Charakter/-animation

Die Modelle liegen im Maya Format vor. Ein Exporter von Maya in das engine-spezifische Format ist erforderlich. Der Charakter sollte seinen ganzen Körper sowie das Gesicht unabhängig voneinander bewegen können. Das Gesicht muss unabhängig voneinander Viseme und Augenbrauenbewegungen darstellen können.

3.1.3 Texte vorlesen (Text-To-Speech)

Ein Text-To-Speech System ist einzubinden und für die Sprachsynthese zu verwenden. Die Viseme-Ansteuerung sollte von diesem System unabhängig sein, damit das System eventuell auch für aufgenommene Audiodaten erweitert werden kann.

3.1.4 Synchronisation der Lippen

Der Charakter soll synchron zur generierten Sprache die Lippen bewegen. Dazu sind die Phoneme des TTS-Systems in geeignete Viseme zu überführen.

3.1.5 Realismus

Der Hautshader von Jörg Unterberg soll in diese Arbeit integriert werden. Die nötigen Vorberechnungen müssen übernommen werden, und der Shader muss mit der Engine kompatibel gemacht werden, was Aktivierung und Parameterübergabe beinhaltet.

Realistische Bewegungen des Kopfes während des Sprechens (Nicken etc.) sind wünschenswert. Blinzeln und In-die-Kamera-schauen helfen ebenfalls, den Realismusgrad zu erhöhen.

3.1.6 Validierung

Der virtuelle Nachrichtensprecher soll Inhalte aus der Tagesschau bzw. den Tagesthemen nachsprechen, um die Glaubwürdigkeit der Gestik und Mimik anhand von Umfragen überprüfen zu können.

3.2 Bestandsaufnahme

3.2.1 Engine

Es wird zur Zeit eine hausinterne Grafikengine „Avango“ (basierend auf SGI Performer und OpenGL) genutzt, die jedoch nicht auf Charakteranimation ausgelegt ist und auch keine Grafikkartenprogrammierung vorsieht. Daher muss eine bessere Engine gefunden werden.

3.2.2 Charakter / Charakteranimation

Die Modelle für die Anwendung werden vom IMK gestellt. Es existieren z.Z. drei Modelle, davon eine weibliche und zwei männliche Figuren. Sie liegen im Maya Format vor. Die Maya Dateien enthalten sämtliche Geometrie, artikuliert mit einer Skelettstruktur sowie Bewegungsdaten für letztere. Die Bewegungen wurden mit Motion Capturing aufgezeichnet. Neue Bewegungen könnten im Studio nachträglich aufgenommen werden.

Außerdem gibt es jeweils einen artikulierten Kopf, aus dem sich die benötigten Gesichtsausdrücke generieren lassen.

3.2.3 Text-To-Speech System (TTS)

Als Text-To-Speech (TTS) System steht die Bibliothek ScanSoft RealSpeakSolo in der Version 4 mit Handbuch zur Verfügung.

3.2.4 Realismus

Zur realistischeren Simulation der Beleuchtung von Haut kann auf die Diplomarbeit von Jörg Unterberg zugegriffen werden. Sie ist in C++ für Linux geschrieben und benutzt das nVidia-SDK und CG-Shader zur Darstellung. Der Shader soll in OGRE integriert werden. Die Aktivierung des Shader-Codes und die Übergabe der nötigen Parameter muss von OGRE übernommen werden.

3.3 Design-Entscheidungen

3.3.1 Wahl einer Grafikengine

Die Wahl der Grafikengine sollte selbständig getroffen werden. Ich entschied mich dafür, nur freie Produkte zu untersuchen, da diese

1. qualitativ oft mit teuren kommerziellen Produkten mithalten können,
2. Quellcode zur Verfügung stellen, was das Verständnis und die Fehlersuche enorm verbessert,
3. untereinander besser vergleichbar sind.

Die Tabelle 3.1 zeigt das Ergebnis meiner Internetrecherche. Die Angaben beziehen sich auf flüchtige Informationen, wie sie in Foren oder den Homepages der jeweiligen Engines zu finden sind, und erheben daher nicht den Anspruch auf Vollständigkeit und Korrektheit, sind aber für die Wahl einer Engine ausreichend.

Aufgrund der gesammelten Informationen habe ich mich für OGRE (Object-Oriented Graphics Rendering Engine) entschieden, weil diese Engine die meisten der nötigen Features anbietet:

- Skelettanimation mit Skinning auf der Grafikhardware.
- Pose Blending(siehe Kapitel 2.7), auch auf der GPU.
- Ein eigenes Format, (binär und .xml umrechenbar) mesh.xml für Geometriedaten und skeleton.xml für Skelettdefinition und Animation.
- Einen Exporter für Maya, der allerdings noch keine Blendshapes unterstützt.
- Möglichkeiten zur GPU-Programmierung in mehreren Shader-Sprachen.

	OGRE	CrystalSpace	OpenSG	nVIDIA SDK	OpenScene-Graph
Plattform	Windows, Linux, MacOS	Windows, Linux, MacOS	Irix, Linux, Windows	Windows, Linux 32 und 64 bit	Windows, Linux, MacOS, Solaris, ..
Pipeline	OpenGL + DirectX	OpenGL + Software	OpenGL	OpenGL	OpenGL
Shader	Cg, GLSL, HLSL	Cg, ARB	Cg, GLSL	CgFX 1.4	Cg, GLSL
Skripting	➔Material ➔Compositor ➔Particle ➔Overlay	nein	nein	nein	"Lua scripting provided via OsgVortex plugin"
Meshes	Milkshape3D, 3D Studio Max, Maya, Blender , Wings3D	Blender, Milkshape, Maya, Cal3d, 3DS, Quake MDL and Quake II MD2.	VRML97, OBJ, OFF, RAW + extern via Cal3D	Nur eigene Formate + VRML/WRL import.	lwo, .obj, .3ds, Direct X ..
Animationen	hardware skinning und pose blending	via Cal3D	via Cal3D	??	via Cal3D oder ReplicantBody (externe Plugins)
Physik	„Bindings for ODE, Novodex and Tokamak"	via ODE	nein	nein	OsgVortex physics plugin
GUI	Ja (via CEGUI)	ja	nein	nein	??
Sound	nein	ja	nein	nein	via integrated OpenAL-Plugin
Netzwerk	nein	via VOS	ja	nein	nein
Community	7633 Forum-Mitglieder > 30 Projekte	506 Member, >100 Contributors	232 Member 11 Contributors	nur 171 Topics bzgl. SDK	>1000 Member >100 Projekte
Bonus	➔Skripting, ➔mehrere Schattenverfahren, Kompositions-Effekte wie motion-blurring, ➔Community	➔Environment Mapping ➔Lens Flares ➔ Billboarding ➔Trigger ➔ Particle System ➔Sky, Mirror	Clustering	➔Schnell, stereo, ➔Multipass antialiasing ➔Shadow map support ➔Level of detail handling	Schnell
Malus	Keine?	Langsam?	nur „Doxygen" Dokumentation	Viele Demos laufen bei mir (ATI Grafik) nicht	Angeblich Schlechte Doku

Tabelle 3.1: Vergleich von freien Grafikengines

4 Realisierung

4.1 Bibliotheken - OGRE

OGRE steht für Object-Oriented Graphics Rendering Engine. Es handelt sich dabei um ein Open-Source Projekt, das im Jahr 2001 von Steve Streeting ins Leben gerufen wurde. Ein wichtiger Schwerpunkt bei der Entwicklung von OGRE lag für die Entwickler in der Objekt-Orientierung. Gerade bei großen Softwareprojekten ist dies ein Ansatz der vielfach verfolgt wird, damit das Projekt übersichtlich bleibt. So wird es mit den Konzepten der Objektorientierung *Kapselung* und *Vererbung* möglich, Implementierungsdetails von Systemkomponenten auf unterschiedlichen Plattformen unter einem gemeinsamen *Interface* zu verstecken. Der Benutzer, muss dann nur das Interface lernen und sich nicht darum kümmern, auf welcher Plattform und mit welcher Grafik-API sein Programm läuft. OGRE ist momentan unter Windows und Linux lauffähig und unterstützt OpenGL sowie DirectX. Durch Kapselung von der Grafik-API muss der Programmierer aber gar keine OpenGL- oder DirectX-Befehle mehr direkt schreiben, weil alle Funktionen im Interface *RenderSystem* abgekapselt sind.

Ebenso verhält es sich mit dem *SceneManager*, dessen Aufgabe es ist, die Ressourcen der Szene zu verwalten und zu entscheiden, wann welche Objekte gerendert werden sollen. Da sich je nach Anforderung an die Szene unterschiedliche Optimierungen empfehlen, ist es sehr von Vorteil, dass die konkrete Implementierung des *SceneManagers* austauschbar ist.

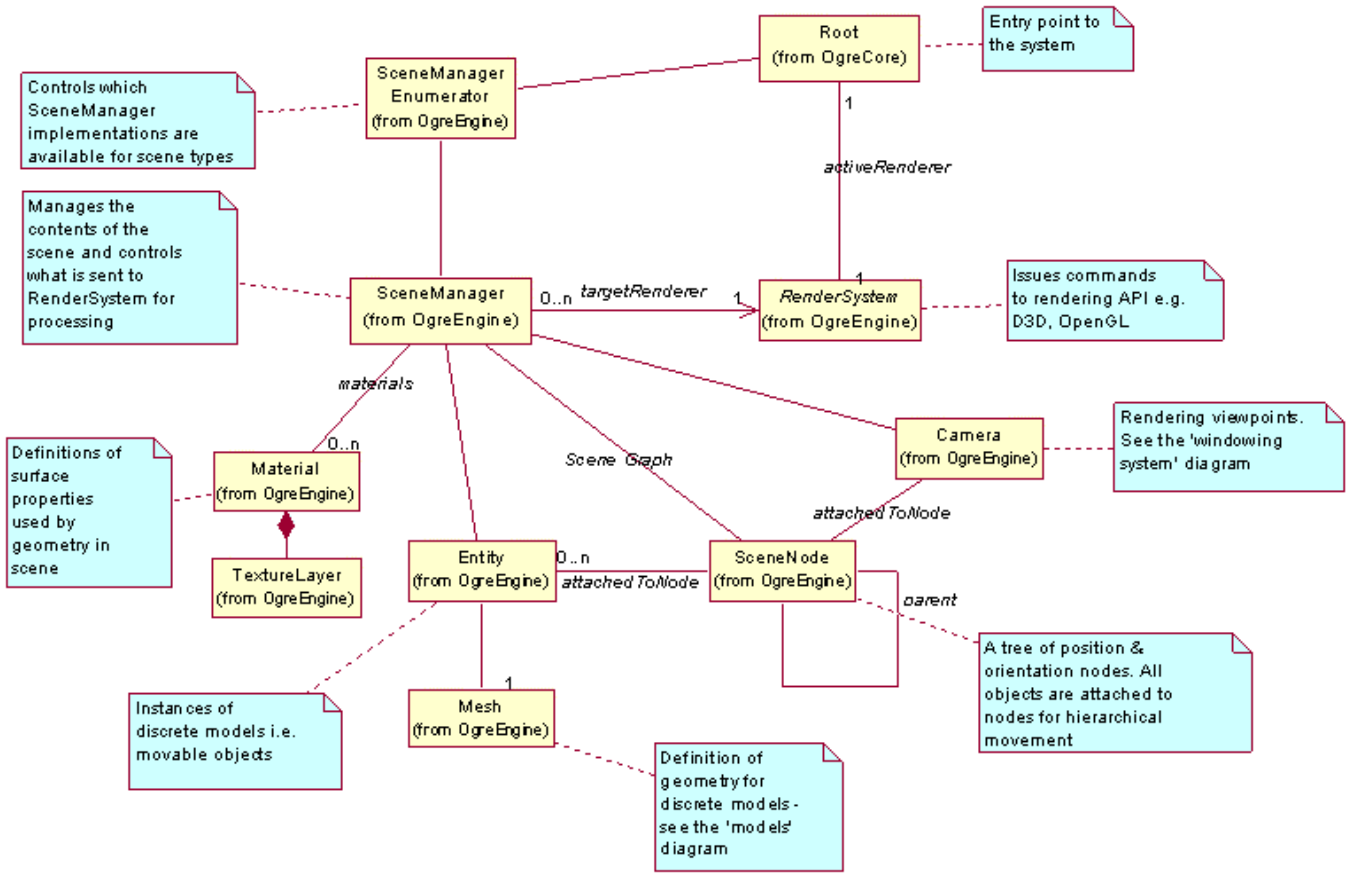


Abbildung 4.1 OGRE Architektur (Quelle: www.ogre3d.org/manual)

4.1.1 Architektur

Abbildung 4.1 zeigt die Architektur von OGRE. Eintrittspunkt für das System ist die Komponente *Root*. Sie ist ein Singleton, d.h. es existiert immer nur eine Instanz der Klasse. *Root* besitzt alle anderen Singletons im System. Dazu gehören *Manager* und *Factories*. Diese *Design Pattern* wurden von [Gamma et. al (1994)] eingeführt. Eine *Factory* ist eine Klasse, die Instanzen von Objekten erzeugen kann, ohne dass der Aufrufer die genaue Unterklasse kennen muss. Das ist sehr hilfreich, wenn Ressourcen aus Dateien geladen werden sollen, ohne dass genau bekannt ist, welchen Typs diese Ressourcen sind.

4.1.2 Szenegraph

Der Szenegraph wird vom *SceneManager* verwaltet und besteht aus hierarchisch verknüpften *SceneNodes*. Diese haben Informationen über Positionierung und Größe

der ihnen zugewiesenen Objekte. Die *SceneNode* wird verwendet, um zu entscheiden, ob und ggf. in welcher Qualität (Level-of-Detail) ein Objekt gerendert werden muss.

4.1.3 Meshes und Entities

In OGRE wird zwischen der Definition eines Modells und seiner Instanz im Szenegraph unterschieden. Ein *Mesh*, das in mehrere *Submeshes* aufgeteilt sein kann, legt die Topologie und die Materialeigenschaften eines Modells fest. Außerdem speichert es die Skelettstruktur und sämtliche Animationen, die für das Modell zur Verfügung stehen.

Eine *Entity* wird aus einem *Mesh* erstellt. Sie wird - analog zum *Mesh* - ebenfalls in kleinere Objekte, die *Subentities* aufgeteilt. Es besteht eine Eins-zu-n-Beziehung zwischen *Mesh* und seinen *Entities*, d.h. ein Modell kann beliebig oft in der Szene vorkommen, ohne dass deshalb alle Geometriedaten mehrfach gespeichert werden müssten. Eine *Entity* speichert den aktuellen Zustand in Bezug zu seinem *Mesh*, etwa welche Animationen aktiviert sind und zu welchem Zeitpunkt einer Animation sich die *Entity* befindet. Außerdem kann die *Entity* alle Eigenschaften des *Meshs* für sich lokal überschreiben, so dass verschiedene Varianten desselben *Meshs* erstellt werden können.

4.1.4 Materialien

Materialien und ihre Eigenschaften werden in OGRE-spezifischen Material-Dateien (siehe Text 1) definiert. Diese XML-konformen Dateien werden von der Engine beim Laden von *Meshes* aus .mesh-Dateien automatisch mit eingelesen und den entsprechenden *Submeshes* zugeordnet. Ein Material besteht aus mehreren *Techniques* (engl. Techniken), die je nach Möglichkeiten der Zielhardware unterstützt werden oder nicht. Wenn die bestmögliche Technik nicht unterstützt wird, wird die zweitbeste verwendet usw.

Eine *Technique* setzt sich aus einem oder mehreren *Passes* zusammen. Ein *Pass* ist ein Durchlauf der Rendering-Pipeline, entspricht also dem einmaligen Rendern eines Objektes.

Die eigentliche Materialdefinition wird auf Ebene der *Passes* vorgenommen. Sie beinhaltet Beleuchtungsparameter, wie diffuses, ambientes und spekulares Licht, sowie Einbindung von Texturen, die für das Material verwendet werden. Außerdem kann sie

Vertex- und Fragment- Shader einbinden und ihnen Parameter übergeben. Wenn mehrere *Passes* verwendet werden, kann man festlegen, wie die Ergebnisse der einzelnen Renderschritte kombiniert werden sollen.

```
material TestMaterial
{
   technique // Shader-Unterstützt
   {
      pass // Erster und hier auch einziger Renderschritt
      {
         vertex_program_ref MeinVertexProgramm // Referenz zum Vertex-Shader
         {
            // Automatisches Übergeben von Parametern, hier Lichteigenschaften
            param_named_auto lightPosition light_position 0
            param_named_auto lightDiffuseColour light_diffuse_colour 0
            // Manuelles Übergeben von Parametern
            param_named lightMultiplier float 1.0
            ...
         }
         texture_unit
         {
            // Einbindung einer Textur aus einer Datei
            texture Texture.jpg
            ... // Festlegen von Eigenschaften der Textur, wie z.B Filterung
         }
      }
   }
   technique software // Standard-Pipeline
   { ... }
}
```

Text 1: Beispiel des OGRE .material Dateiformates

4.1.5 Datenformat von OGRE: Mesh und Skeleton

In OGRE werden Charaktermodelle in Mesh(.mesh) und Skeleton(.skeleton) Dateien gespeichert. Mit dem beiliegenden Tool „OgreXMLConverter“ können diese binär gespeicherten Daten in menschenlesbares XML (siehe Text 2) oder wieder zurück konvertiert werden. Im Mesh-Format können sämtliche Geometrien definiert werden, inklusive Definition von Blendshapes und Animationen der letzteren, und im Skeleton-Format werden dann die Skelette mitsamt ihren Animationen hinterlegt.

Die Hierarchie beginnt mit dem Wurzelknoten *<mesh>*. Unter diesem findet sich der optionale Knoten *<sharedgeometry>* (gemeinsam verwendete Geometrie), der die Definition des von den Submeshes gemeinsam genutzten *<vertexbuffer>* eröffnet. Ein Vertexpuffer besteht aus einer Menge von Vertices, deren Reihenfolge nicht willkürlich ist, weil ihre Position innerhalb der Menge - also der Index - für die Definition der *Faces* vonnöten ist. Ein Vertex wird über seine Position bestimmt.

```
<mesh>
  <sharedgeometry vertexcount="4321">
    <vertexbuffer positions="true" normals="true" texture_coords="1">
      <vertex>
        <position x="-30.1255" y="-1.13188" z="46.8038"/>
        <normal x="0.00818188" y="-0.585702" z="0.810485"/>
        <texcoord u="0.195065" v="0.865848"/>
      </vertex>
      ... // 4320 weitere Vertices
    </vertexbuffer>
  </sharedgeometry>
  <boneassignments>
    ... // >= 4321 Boneassignements für SharedGeometry
  <boneassignments>

  <submeshes>
    <submesh material="K" use32bitindexes="true" usesharedvertices="true"
        operationtype="triangle_list">
      <faces count="123">
        <face v1="3" v2="0" v3="2"/>
      ... // 122 weitere Faces
      </faces>
    </submesh>
    <submesh material="K" use32bitindexes="true" usesharedvertices="false"
        operationtype="triangle_list">
      <faces count="456">
        ... // 456 Faces
      </faces>
      <geometry vertexcount="1234">
        <vertexbuffer positions="true" normals="true" texture_coords="1">
          ... // 1234 Vertices
        </vertexbuffer>
      </geometry>
      <boneassignments>
        <vertexboneassignment vertexindex="0" boneindex="0" weight="0.3"/>
        <vertexboneassignment vertexindex="0" boneindex="1" weight="0.7"/>
          ... // >= 1234 Boneassignements
      </boneassignments>
    </submesh>
    ... // beliebig viele weitere Submeshes
  </submeshes>

  <skeletonlink name="Character.skeleton"/>

  <poses> // =blendshapes
    <pose target="submesh" index="0" name="augen_zu">
           <poseoffset index="0" x="0.00000" y="0.00000" z="0.02842" />
           ... // alle Vertices, die verschoben werden müssen.
    </pose>
      ... // beliebig viele weitere Blendshapes
  </poses>
  <animations>
    // Pose animations
    BEISPIEL BESORGEN
  </animations>
</mesh>
```

Text 2: Beispiel des OGRE .mesh Dateiformates

Optionale Eigenschaften sind Oberfächennormale, Texturkoordinate und Farbe. Nach der *<sharedgeometry>* folgt der Knoten *<submeshes>*, der aus beliebig vielen *<submesh>*-Knoten besteht. Ein Submesh besteht aus mehreren *<faces>*. Ein *<face>* wird definiert als Folge von Indizes v1 bis vn, deren Anzahl vom Renderingmodus *operationtype* bestimmt ist. Die Eigenschaft *material* definiert, welches Material diesem Submesh zugewiesen werden soll. Das Material selbst wird, wie im vorangegangenen Abschnitt erläutert, in einer seperaten .material Datei spezifiziert. Bei einem *<submesh>* mit der Eigenschaft *usesharedvertices=true* beziehen sich die Indizes bei der Definition der Faces auf den gemeinsam genutzten Vertexpuffer. Alternativ kann das Submesh seinen eigenen Vertexpuffer referenzieren. Dieser wird unterhalb des *<geometry>*-Knotens definiert. Nach jedem Vertexpuffer benötigt man im Falle von Skelettanimation auch die Zuordnung zu den Bones auf Vertexebene, welche unterhalb des Knotens *<boneassignments>* stattfindet. Die Zuordnung zum Skelett erfolgt in der .Mesh Datei durch das XML-Tag *<skeletonlink>*. Der Abschnitt *<poses>* dient zum Definieren von Blendshapes. Mit den Eigenschaften *target* und *index* von *<pose>* wird festgelegt, auf welches Submesh sich die Pose bezieht. Es folgen beliebig viele *<poseoffset>*-Elemente, in denen die Abweichung eines Vertex zum neutralen Mesh festgehalten wird. Unter *<animations>* können die Posen animiert werden.

4.1.6 Animationen

OGRE ermöglicht Animationen mit Hilfe von *Keyframe Interpolation*. Dazu werden Keyframes definiert, die einen bestimmten Zustand beschreiben. Diese Keyframes werden in Tracks eingefügt und damit zeitlich dimensioniert.

Ein *Track* kann den Wert einer bestimmten Variable, die Transformation eines Knoten im Szenegraph, oder die Gewichtung von Blendshapes in Abhängigkeit von der Zeit beeinflussen. Eine Animation kann aus mehreren *Tracks* bestehen, die unterschiedliche Objekte betreffen. Eine Ganzkörperanimation besitzt also für jeden bewegten *Bone* im Skelett einen eigenen *Track*, kann aber über ein einziges *Animation*-Objekt angesteuert werden. Wenn mehrere Animationen ein Objekt beeinflussen, werden die Werte je nach Gewicht der Animation interpoliert.

4.2 Weitere Bibliotheken

4.2.1 Scansoft RealSpeak

RealSpeak ist ein Text-To-Speech (TTS) System, mit dem man Texte in Sprache umwandeln kann. Es basiert auf Unit Selection (siehe Kapitel 2.8.7). Die Ansteuerung erfolgt asynchron, d.h. RealSpeak läuft in einem eigenem *Thread* und informiert das aufrufende Programm über seinen Zustand, indem es vorher vereinbarte Funktionen aufruft. Für welche Ereignisse ein solcher sog. *Callback* erwünscht ist, kann der Aufrufer festlegen. Es gibt die Möglichkeit, sich phonem-, wort- und satzweise informieren zu lassen, sowohl am Anfang wie am Ende einer Sequenz.

Es gibt zwei Modi, in dem RealSpeak operieren kann. Im Online-Modus werden die Audiodaten in Echtzeit erzeugt und direkt an die Soundausgabe gesendet, im Offline-Modus dagegen werden sie für den kompletten Eingabetext auf einmal generiert und dann in eine Audiodatei geschrieben.

Aus den erzeugten Phonemen kann man sich mit der eingebauten Funktion *GetLipSyncInfo* folgende Gewichte zur Viseme-Ansteuerung geben lassen:

- mouthWidth – Breite der Mundöffnung.
- mouthHeight – Höhe der Mundöffnung.
- lipTension – Spannung auf den Lippen (nicht verwendet).
- upperTeethVisible – obere Zahnreihe sichtbar.
- lowerTeethVisible – untere Zahnreihe sichtbar.

Diese Steuerungsparameter wurden direkt für die Animierung des Gesichtes verwendet. Dafür wurden fünf Basisausdrücke so gewählt, so dass sie jeweils den extremen Ausprägungen einem der oben genannten Parameter entsprechen.

4.2.2 OpenAL

OpenAl (Open Audio Library) ist eine Bibliothek zur Soundausgabe. Sie wurde verwendet, um die von RealSpeak generierten Audiodaten im Offline-Modus abzuspielen.

4.2.3 CEGUI

CEGUI ermöglicht die Darstellung und Interaktion mit einer grafischen Oberfläche innerhalb eines 3D-Fensters. Es ist plattformunabhängig und kann in OGRE eingebunden werden. Sämtliche Eigenschaften der grafischen Elemente sind serialisierbar[14], so dass der hierarchische Aufbau der Oberfläche aus einer zur Laufzeit geladenen Datei geschieht und somit im Falle von Änderungen von einer Neuübersetzung unabhängig ist. Kernkomponente ist die Klasse *Window*, von der alle grafischen Elemente abgeleitet werden. Der *WindowManager* lädt diese Komponenten aus einer Quelldatei und positioniert sie auf der Oberfläche. Die Positionierung von Objekten kann sowohl relativ zur hierarchisch übergeordneten Komponente, als auch absolut geschehen.

Die Kommunikation mit der Anwendung erfolgt bidirektional. Die Anwendung muss die Oberfläche informieren, wenn Benutzereingaben gemacht wurden, also z.B. Mausklicks. Diese entscheidet, ob ein Event ausgelöst wurde – z.B. Drücken eines Buttons. Wenn für den Button *Callbacks* definiert wurden, werden diese aufgerufen, so dass die Anwendung auf einer höheren Ebene der Abstraktion auf das Ereignis reagieren kann. So wird zum Beispiel aus einem - für die Anwendungslogik nicht verwertbaren – Mausklick auf Pixel(x,y) ein semantisches Ereignis, wie *ABBRECHEN*, *OK* oder ähnliches.

4.3 Datenerfassung und Verarbeitung

4.3.1 Datenerfassung

Das Gesicht des verwendeten Charakters wurde mit dem ShapeSnatcher von [www:Eyetronics] aufgenommen. Es handelt sich dabei um ein Gitterscanner-System das Kalibration, Gittererkennung und 3D-Rekonstruktion durchführt:

Der ShapeSnatcher arbeitet mit einer Kamera (Canon EOS D60) und einem Blitzlicht (Canon Speedlite 550EX), das durch ein feinmaschiges Muster in strukturiertes Licht umgewandelt wird. Es werden mehrere Aufnahmen eines Objekts gemacht. Diese werden anschließend zusammengesetzt, so dass eine sehr genaue Rekonstruktion

[14]serialisieren = hier: Umwandeln eines Objektes in einen linearen Datenstrom zur Speicherung in einer Datei.

entsteht. Manche Teile der Rekonstruktion müssen manuell unterstützt werden, z.B. die ungefähre Ausrichtung der Teilrekonstruktionen zueinander.

Für das verwendete Modell wurden vier bis fünf Bilder vom Gesicht gemacht und dann rekonstruiert.

Zwei Tools der Stanford University kamen dabei zum Einsatz: Mehrere Patches[15] wurden mit Hilfe von Scanalyze [www:scanalyze] registriert und miteinander verbunden. Eventuell auftretende Löcher wurden mit volfill [www:volfill] gefüllt.

Die Textur wird beim ShapeSnatcher mit einem Polfilter aufgenommen, mit dem man den diffusen Anteil des Lichtes vom spekularen Anteil, welcher von der Blickrichtung abhängig und daher im Programm zu berechnen ist, trennen kann [Unterberg (2004)]. Eine zylindrische Platzierung der Textur auf das Modell wird ebenfalls vom ShapeSnatcher geliefert.

Wie in [Osipa (2003)] beschrieben, ist es sinnvoll, die Topologie des Meshes so zu wählen, dass die Kanten der Polygone sich an den anatomischen Gegebenheiten orientieren. Zudem ist eine höhere Auflösung in Regionen mit hoher Artikulation im Gegensatz zu eher unbeweglichen Teilen des Gesichtes wünschenswert. Auf das sehr hoch aufgelöste Modell (9000 - 20000 Dreiecke) wurde mit Hilfe der Texturkoordinaten die Geometrie des animierbaren generischen Modells übertragen, so dass das resultierende Modell eine ähnliche Struktur besitzt. Da nur die äußere Hülle so übertragen werden konnte, musste an Mund, Augen, Nase und Ohren innere Geometrie modelliert werden. Da nun die Geometrie dem schon animierbaren generischen Modell entsprach, konnten die extremen Ausdrücke für die Animationen übertragen werden und für die Erzeugung der Blendshapes (siehe Kapitel 2.7) genutzt werden.

Der Körper wurde manuell von einem Kollegen bei den Mediengestaltern im Haus erstellt.

Die Bewegungen des Skelettes wurden im hausinternen Studio mit einem selbstgebauten Anzug und optischem Tracking aufgenommen. Dazu wurden fünf Infrarot-Kameras von ART eingesetzt. Die Bewegungen der Hände wurden gleichzeitig mit Cybergloves (Datenhandschuhen) aufgenommen, da optisches Tracking wegen Ungenauigkeiten und Verdeckungsproblemen dafür nicht geeignet ist.

[15]Patch (engl.) = Zusammenhängende Geometriedaten

4.3.2 Exporter

Für die Einbindung des verwendeten Modells war es notwendig, dieses von Alias Maya® ins OGRE-Format zu konvertieren. Leider war der Maya-Exporter noch nicht für Blendshapes ausgelegt, so dass diese Funktionalität eingebaut werden musste. Das Gesicht habe ich vom Hinterkopf getrennt, damit für die Animation mit Blendshapes (siehe Kapitel 2.7) weniger redundante Vertices gespeichert werden müssen.

4.4 Implementierung

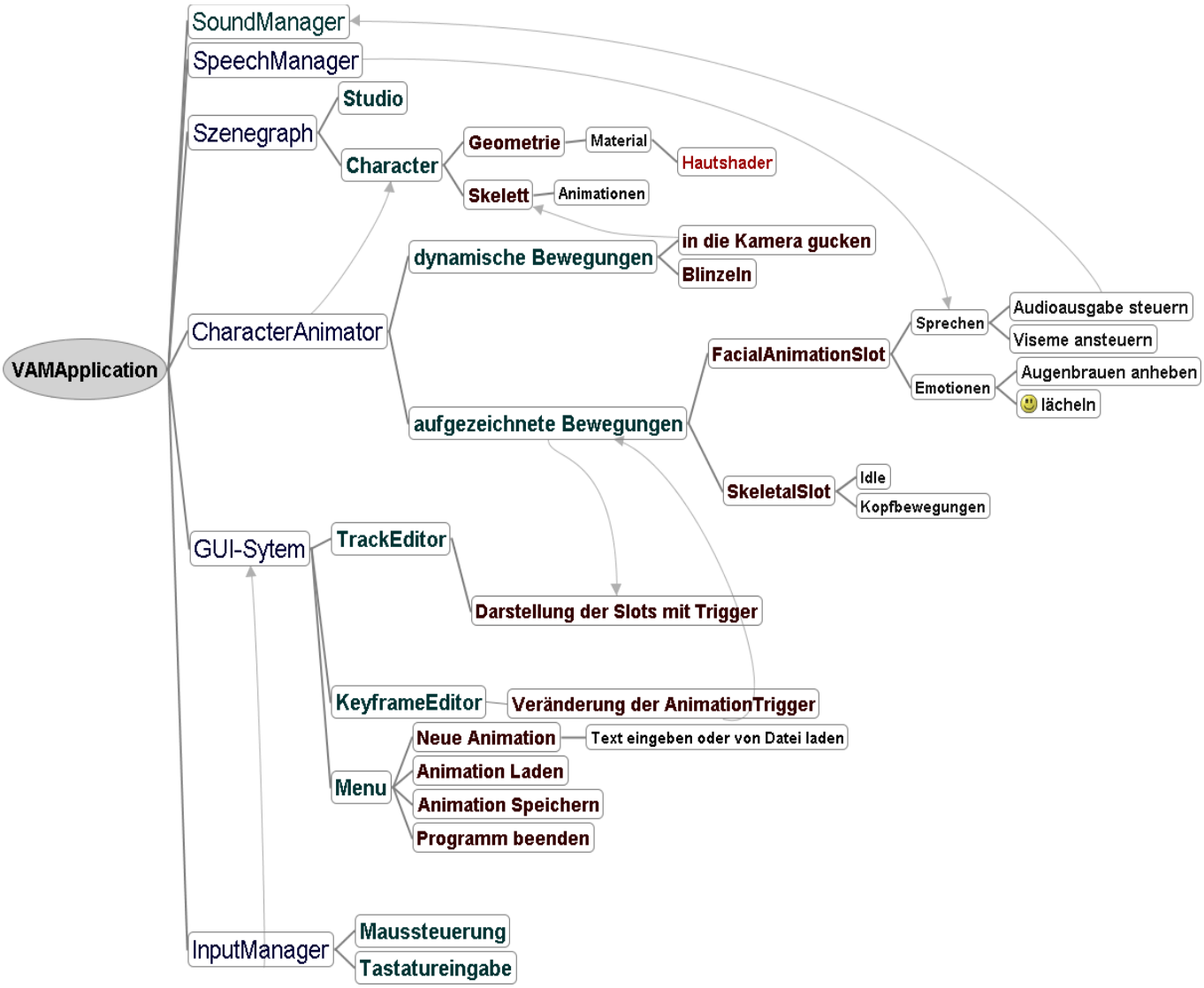


Abbildung 4.2. Struktur der Anwendung

Die logische Struktur von VAM wird in Abbildung 4.2 ersichtlich. Eintrittspunkt für die Anwendung ist die Klasse *VirtualAnchormanApplication*. Sie initialisiert alle Subsysteme, erstellt die Szene und startet die Rendering-Schleife. Die Szene besteht

aus dem statischen Studio, einer Kamera, einer Punktlichtquelle und dem Charakter, welcher vom *CharacterAnimator* gesteuert wird.

Der *InputManager* erfasst Benutzereingaben und leitet diese an das *GUI-System* weiter. Der *SpeechManager* verwaltet die Anbindung an das TTS-System und überträgt die nötigen Daten dem *CharacterAnimator.* Zur Soundausgabe wird der SoundManager verwendet, der alle Audio-Ressourcen verwaltet.

4.5 Animation

Die Animation des Charakters geschieht auf mehreren Ebenen gleichzeitig. Diese Ebenen müssen unabhängig voneinander angesprochen werden. Die einzelnen Ebenen heißen *AnimationSlots*. Es gibt im Wesentlichen zwei verschiedene Untertypen: den *SkeletalSlot*, der Animationen des ganzen oder eines Teils des Skelettes ansteuert, und den *FacialExpressionSlot*, der Gesichtsanimationen ansprechen kann. Jeder Slot kann beliebig viele *AnimationTrigger*[16] enthalten, um zu verschiedenen Zeiten eine Animation auslösen zu können.

4.5.1 Sprechen

Die Darstellung des Sprechaktes erfordert die Animation des Gesichtes durch Ansteuerung von Visemen. Die Blendshapes für die Viseme wurden zuvor den von Realspeak vorgeschlagenen Parametern entsprechend modelliert und sind nach dem Laden in OGRE im Modell gespeichert. Zur Erstellung einer Animation werden die Gewichtungen der Blendshapes zum entsprechenden Zeitpunkt in einen OGRE-Animation-Track eingetragen. Dem Animationsobjekt von OGRE muss dann beim Rendern nur noch der aktuelle Zeitpunkt übermittelt werden.

4.5.2 Skelettanimation (Idle, Kopfbewegungen)

Die Idle-Bewegung[17] wird verwendet, um einen Charakter lebendig wirken zu lassen, denn ein Mensch „wippt“ meistens ein wenig hin und her, auch wenn er einfach nur gerade steht, weil er sein Gleichgewicht ausbalancieren muss. Die Idle-Animation

[16] Trigger (engl.) = Auslöser
[17] to idle (engl.) = untätig sein

wurde schon für frühere Projekte mit Motion Capture aufgenommen. Sie kann natürlich durch eine beliebige Animation ersetzt werden, ist für einen stehenden Nachrichtensprecher aber ausreichend. Die Animation ist nur 7,3 Sekunden lang; damit sie nicht springt, wird sie am Ende rückwärts wieder abgespielt, was beliebig oft wiederholt werden kann.

Zusätzlich zu dieser Grundbewegung schien es ratsam, den Kopf beim Sprechen zu bewegen, weil der Mensch beim Sprechen ganz automatisch minimal „nickt“, um dem Gesagten Ausdruck zu verleihen. Damit diese Bewegung realistisch wirkt, wurde sie der Originalsendung nachempfunden. Mit der „Facial Feature Extraction“-Software Nevenvision-SDK konnte die ungefähre Kopfbewegung extrahiert werden. Die Daten wurden anschließend geglättet, um weniger ruckhaft zu wirken.

4.5.3 Manuelle Kontrolle des Skeletts (In-die-Kamera-schauen)

Damit die Nachrichtensprecherin in die Kamera schauen kann, müssen die Augen unabhängig von allen anderen Animationen bewegt werden. Für die Augen wurden zwei *Pseudo-Bones* in das Skelett eingefügt, deren Rotation die Blickrichtung der Augen steuert. Man kann in OGRE die Transformation eines *Bones* auch unabhängig von allen Animationen manuell ändern. Es wird ein Zielpunkt angegeben, auf den der Charakter schauen soll, in diesem Fall die Kamera. Mit Hilfe von Quaternionen (siehe Anhang 7.1.5) werden beide Augen dann rotiert, so dass sie in Richtung dieses Punktes ausgerichtet sind.

4.5.4 Gesichtsausdrücke (Blinzeln, Augenbrauen anheben, Lächeln)

Zusätzlich zur Visemansteuerung, die gleich ganze Animationen von Gesichtsausdrücken ansteuert, schien es ratsam, einzelne Gesichtsausdrücke zu bestimmten Zeiten mit bestimmter Länge ansteuern zu können. Außerdem werden eine maximale Intensität sowie Ease-in/Ease-out-Parameter angegeben, mit denen weiches Überblenden gesteuert wird. Diese Werte lassen sich im GUI-System per Schieberegler einstellen.

In der Arbeit wurden „Anheben der Augenbrauen“ sowie ein leichtes Lächeln eingesetzt.

Der Blinzelalgorithmus ermittelt, wann der Charakter blinzeln soll, indem eine mittlere Zeit zwischen zweimal Blinzeln und eine Standardabweichung angegeben wird [Jörg (2005)].

4.6 Sprachausgabe und Synchronisation

Die Sprachausgabe wird mit dem RealSpeak SDK durchgeführt. Beim Live-Modus von Realspeak, in dem der Text nach dem Aufruf direkt vorgelesen wird, gibt es das praktische Problem, dass zu allen Zeitpunkten schon die nächste Visemgewichtung bekannt sein muss, da der aktuelle Ausdruck eine Interpolation zwischen der vorangegangenen und der zukünftigen Gewichtung ist. Es war leider nicht möglich, RealSpeak zu veranlassen, die Phoneminformationen schon vor ihrer aktuellen Soundausgabe zu übermitteln, zumindest nicht in einem konstanten Zeitfenster.

Daher wird im Offline-Modus gearbeitet: Beim Erstellen einer Animation wird zunächst der komplette Eingabetext satzweise zerlegt und die Synthese in Audiodateien und die entsprechenden Phoneminformationen in einer XML-Datei gespeichert. Beim Abspielen werden diese Dateien dann wieder eingelesen.

4.7 Hautshader

Auf die Eigenschaften der Haut und die mathematischen Hintergründe zur Simulation der Beleuchtung wurde in der Originalarbeit zum Hautshading von Jörg Unterberg (2004) bereits ausführlich eingegangen, weshalb hier nur eine Kurzfassung folgen soll:

Die Erscheinung der Haut hängt von mehreren Faktoren ab. Erstens ist die Oberfläche nicht glatt, sondern besteht aus vielen kleinen Strukturen. Das kann mit Normal-Mapping (siehe Kapitel 2.10.2) simuliert werden. Die zweite wichtige Eigenschaft ist die Lichtdurchlässigkeit: Der größte Teil des Lichts (93-96%) wird nicht direkt reflektiert, sondern dringt in die Haut ein und wird erst in unteren Schichten absorbiert oder reflektiert. Diese Materialeigenschaft, die mit *Subsurface-Scattering*[18] (SSS) bezeichnet wird, kann mit den herkömmlichen Beleuchtungsmodellen nicht berechnet werden, da diese typischerweise von einer BRDF (Bidirectional Reflectance Distribution Function[19])

[18]Subsurface-Scattering = Streuung unter der Oberfläche, auch Volumenstreuung genannt
[19]BRDF = bidirektionale Reflektanzverteilungsfunktion [Nicodemus et al. (1977)]

ausgehen. Für eine Simulation von SSS muss aber eine BSSRDF (Bidirectional Surface Scattering Reflectance Distribution Function)[20] angenähert werden.

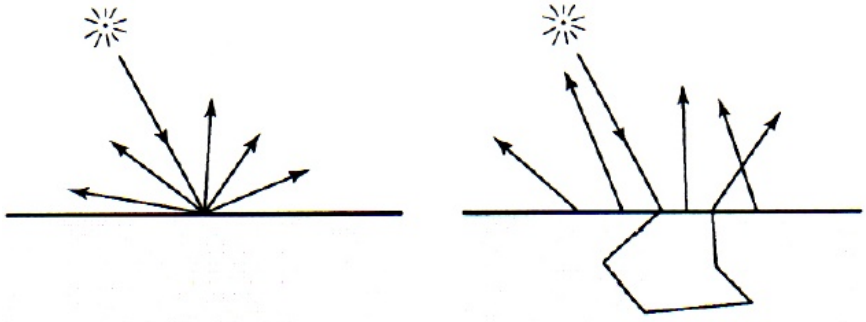


Abbildung 4.3.Vergleich von BRDF (links) und BSSRDF (rechts) [Unterberg (2004)]

Zur Annäherung an die tatsächliche Lichtstärkeverteilungsfunktion einer Haut genügt die Tatsache, dass das Licht an einer anderen Stelle herauskommt als es in die Haut eingedrungen ist. Aus technischer Sicht bedeutet das, dass die Helligkeit nicht in einem Schritt berechenbar ist. Daher wird die Berechnung in drei Phasen zerlegt:

4.7.1 Erste Phase: Helligkeitsverteilung

Im ersten Rechenschritt wird die Verteilung der diffusen Beleuchtung auf der Oberfläche mit dem Lambert-Beleuchtungsmodell berechnet. Die Geometrie wird unter Verwendung eines Vertex-Programms gerendert. Die X-und Y-Komponenten des Ausgabeparameters, der üblicherweise die transformierten 3D-Koordinaten der Eckpunkte beinhaltet, wird aber mit den Texturkoordinaten des Vertex gefüllt. Die Z-Komponente bleibt dabei null. Da die Projektionsmatrix so gewählt wird, dass die Bildfläche von 0,0 bis 1,1 definiert wird, führt das dazu, dass die Geometrie auf den ganzen Bildbereich „aufgeklappt“ wird. Dieses Bild wird in eine Textur gerendert, welche in Abbildung 4.4 zu sehen ist.

[20]BRSSDF = Bidirektionale Oberflächenstreuungs-Reflektanzverteilungsfunktion [Jensen et al. (2001)]

Abbildung 4.4: Das Ergebnis der ersten Phase ist eine Helligkeitsverteilung.

Abbildung 4.5: In der zweiten Phase wird die Textur mit Scattering gefiltert (hier: 4 Durchläufe)

4.7.2 Zweite Phase: Scattering

Im nächsten Berechnungsschritt wird die oben erzeugte Textur nichtlinear gefiltert. Das geschieht auch auf der Grafikkarte, indem ein Rechteck mit dieser Textur über den ganzen Bildschirm gelegt wird, das mit einem speziellen Fragment Programms gerendert wird. Dessen Ausgabe ist die Helligkeit jedes Fragments, welche als der Durchschnitt von acht Punkten in seiner Umgebung ermittelt wird. Welche acht Punkte das sind, wird relativ zum gegebenen Fragment festgelegt. Die relativen Koordinaten werden mit *Importance Sampling* vorab berechnet und bleiben für alle Pixel konstant [Unterberg (2004)]. Der Radius, in dem Helligkeitsinformationen eingesammelt werden, kann frei eingestellt werden. Die Beschränkung auf acht Punkte rührt daher, dass Fragment-Programme nicht beliebig lang sein dürfen. Das wird dadurch ausgeglichen, dass das Ergebnis wieder in die selbe Textur geschrieben wird und der Rendervorgang dann mit anderen relativen Koordinaten beliebig oft wiederholt wird.

In OGRE kann dieser Vorgang mit dem *CompositorManager* realisiert werden

Das Ergebnis der Scattering-Phase sieht man in Abbildung 4.5.

4.7.3 Dritte Phase: Finale Beleuchtung

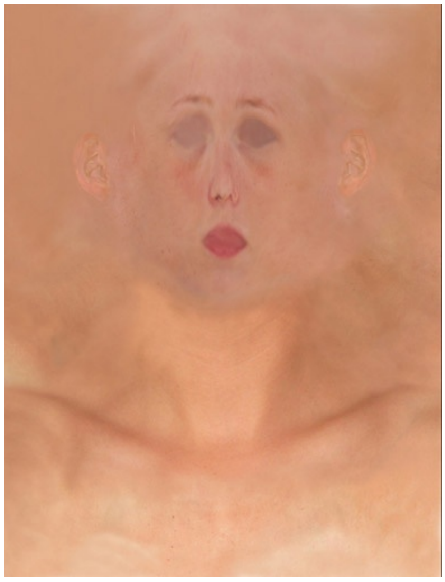


Abbildung 4.6.Diffuser Farbanteil

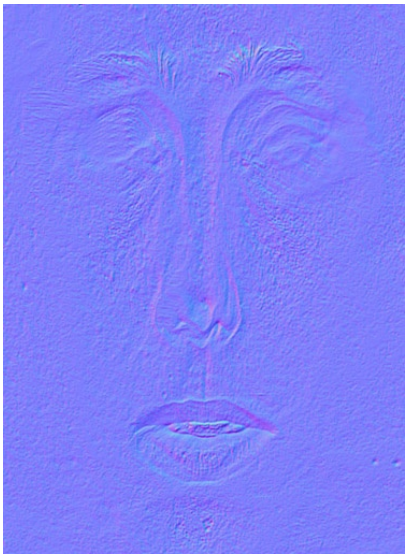


Abbildung 4.7. Ausschnitt der verwendeten Normalmap im Tangent-Space

Im letzten Schritt wird die endgültige Berechnung vorgenommen. Die diffusen Farbinformationen werden aus einer Textur gelesen (vgl. Abbildung 4.6) und mit der Helligkeitsverteilung aus der zuvor berechneten Textur multipliziert. Schließlich wird Normal-Mapping im Texture-Space durchgeführt (siehe Kapitel 2.10.2 und Abbildung 4.7) und nach dem Modell von Torrance-Sparrow [Torrance & Sparrow (1967)] eine Glanzschicht berechnet. Das finale Rendering zeigt Abbildung 4.8. Auf der linken Seite wurde zum Vergleich lediglich das Scattering weggelassen und die komplette Berechnung in nur einem Schritt erledigt. Man kann sehen, dass die Haut mit der Simulation von Tiefenstreuung wesentlich weicher und natürlicher wirkt. Allerdings erreicht man mit der aufwändigen Berechnung auch nur noch etwa die halbe Geschwindigkeit, gemessen in Bildern Pro Sekunde (fps).

Abbildung 4.8.Vergleich mit/ohne Hautshader

4.8 Benutzeroberfläche

Die Benutzeroberfläche dient als Autorenwerkzeug. Sie ermöglicht die Eingabe eines Textes sowie eine Visualisierung und Veränderung der Animationsdaten. Sie wurde mit CEGUI erstellt und zum Nachstellen der Testszene verwendet. In Abbildung 4.9 sieht man eine Aufnahme davon. Im unteren Bereich des Bildes sieht man den Animationseditor, der mehrere Tracks anzeigt. Im Fenster oben rechts lassen sich die Parameter für den gerade ausgewählten *AnimationTrigger* verändern.

Abbildung 4.9. Benutzeroberfläche

5 Evaluation

5.1 Vorgehensweise

Um die Qualität der Software zu bewerten, wurden Umfragen durchgeführt. Als Grundlage diente ein 23-sekundenlanger Ausschnitt einer Tagesthemen-Sendung. Dieser wurde von der virtuellen Nachrichtensprecherin nachgesprochen während sie mit den in Kapitel 2.7.2 besprochenen Verfahren animiert wurde. Die Anhebung der Augenbrauen sowie gelegentliches Lächeln wurde analog zu den entsprechenden Bewegungen im Originalvideo manuell eingefügt .(vgl. Anhang 7.2)

Abbildung 5.1.Begrüßung im Originalvideo

Zunächst wurde ein Video mit allen Features erstellt, um die Qualität zu bewerten. Dabei sollte ermittelt werden, wie hoch die Akzeptanz der Betrachter gegenüber der virtuellen Nachrichtensprecherin ist. Es wurden fünf Fragen gestellt, die mit einer Zahl zwischen null und zehn zu bewerten waren. Null bedeutet dabei Negativ/Ablehnung und zehn Positiv/Akzeptanz.

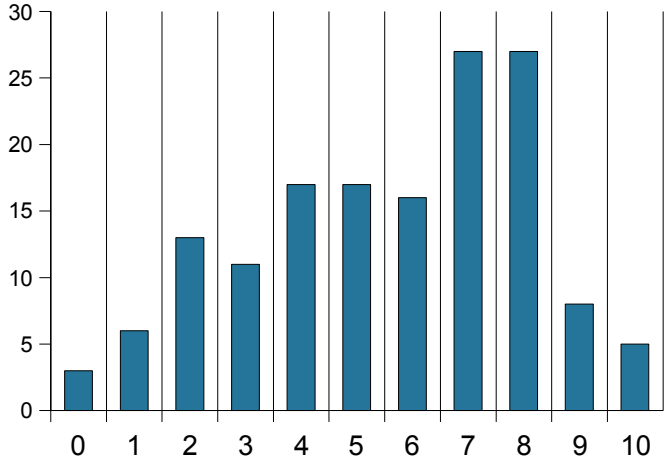
Anschließend wurden sieben weitere Videos gezeigt, in denen jeweils ein Feature des Programms abgeschaltet war. Nach Betrachten dieser Videos sollte angeben werden, wie wichtig das fehlende Feature für den subjektiven Gesamteindruck ist.

Die Umfrage wurde mit einer Internetseite realisiert, auf der man die einzelnen Videos sowie einen Vergleich eines Standbildes einsehen (vgl. Anhang 7.2) und anschließend mit einem Formular seine Bewertung auswählen konnte. Zusätzlich wurde gefragt, was besonders gut und was besonders schlecht gefallen habe sowie was man noch verbessern könne. Es haben 150 Personen teilgenommen.

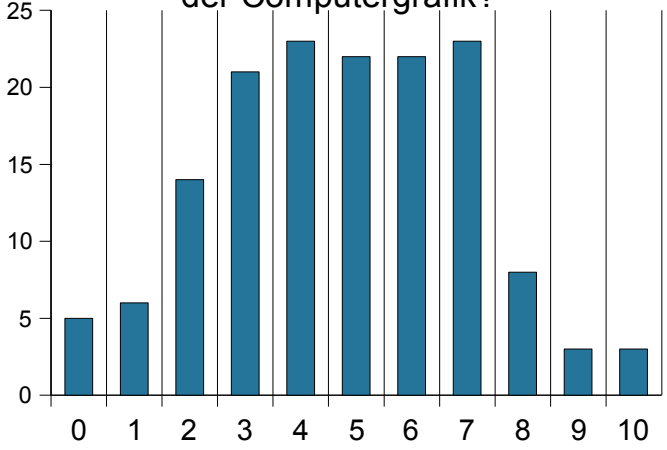
5.2 Umfrageergebnisse

5.2.1 Verteilung der Ergebnisse (Video mit allen Features)

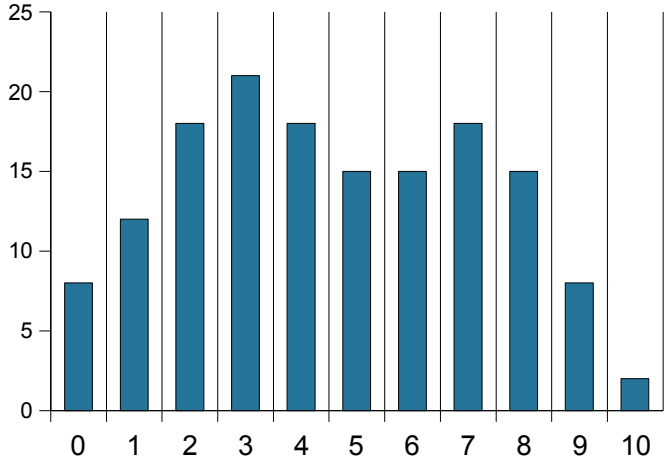
Wirkt die Bewegung auf Sie überzeugend?



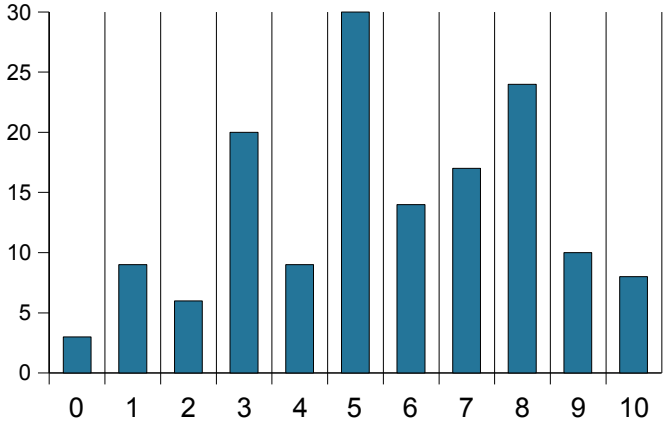
Wie bewerten Sie den Realismusgrad der Computergrafik?



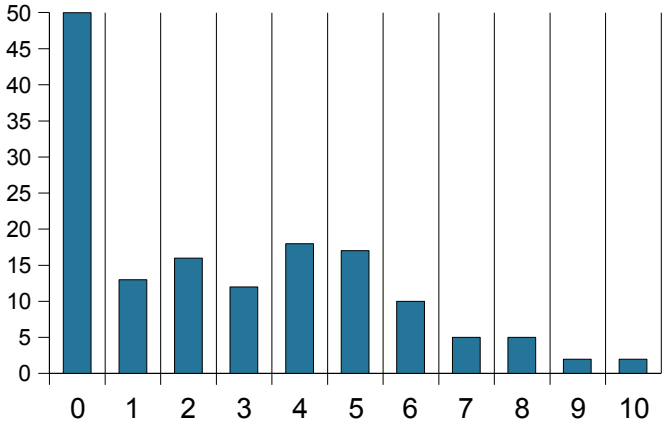
Kann die Sprachsynthese Sie überzeugen?



Passen die Lippenbewegungen zum Gesprochenen?

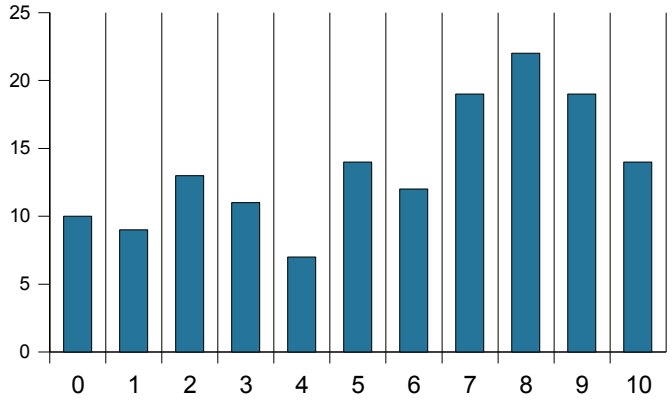


Würden Sie dieser Dame einen Job beim Fernsehen geben?

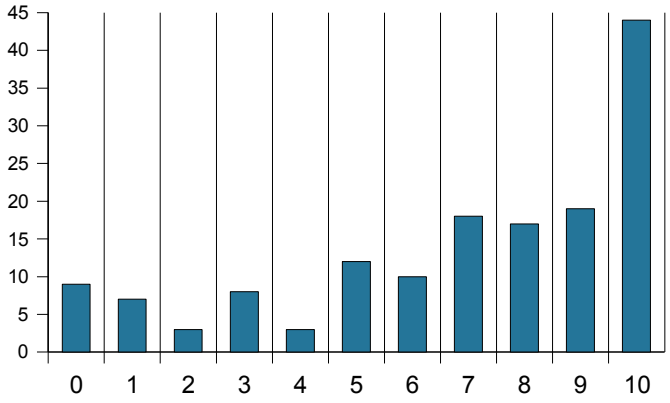


5.2.2 Verteilung der Ergebnisse (Wichtigkeit der einzelnen Features)

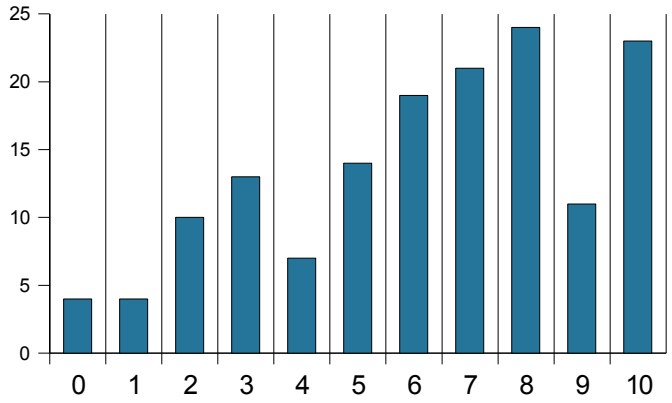
Hautshading



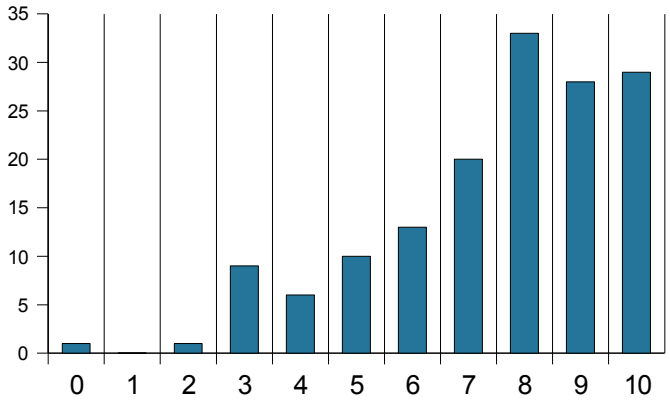
Haare



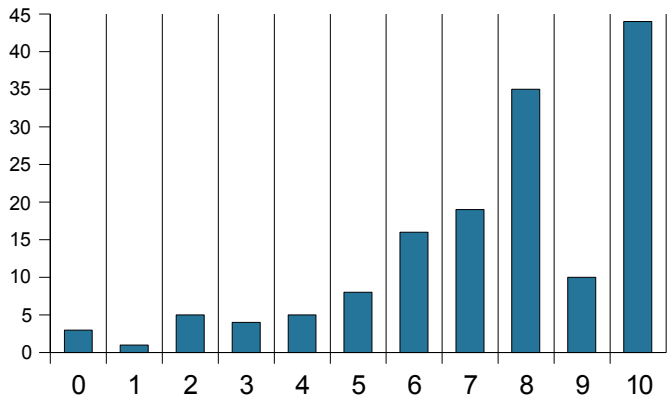
Bewegungen des Körpers



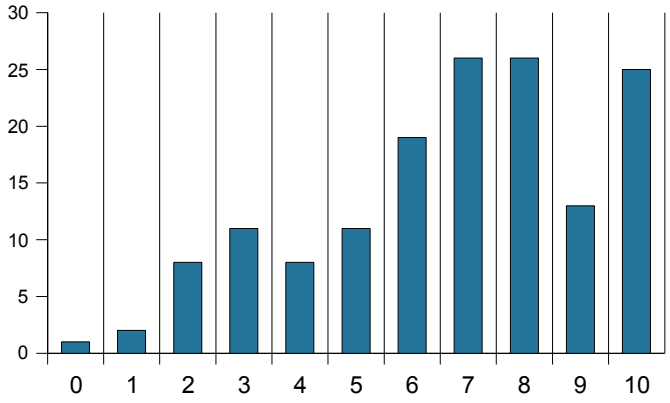
Bewegungen des Kopfes



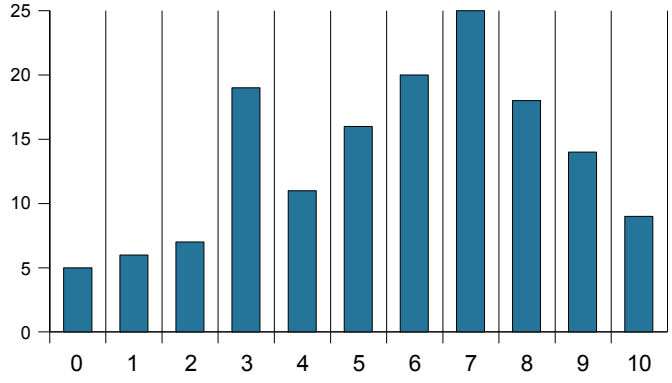
In die Kamera gucken



Blinzeln



Anheben der Augenbrauen



5.2.3 Rangliste nach Mittelwerten

Die Mittelwerte der Antworten zu den Fragen sind in folgender Tabelle ersichtlich:

Rang	Frage	Mittelwert:
1.	Wirkt die **Bewegung** auf Sie überzeugend?	5,61
2.	Passen die **Lippenbewegungen** zum Gesprochenen?	5,55
3.	Wie bewerten Sie den **Realismusgrad der Computergrafik**?	4,76
4.	Kann die **Sprachsynthese** Sie überzeugen?	4,57
5.	Würden Sie dieser Dame einen **Job beim Fernsehen** geben?	2,74

Aus den Beurteilungen zu den einzelnen Features ergeben sich folgende Mittelwerte:

Rang	Feature	Mittelwert:
1.	Bewegungen des Kopfes beim Sprechen	7,51
2.	In die Kamera schauen	7,48
3.	Haare	6,95
4.	Blinzeln	6,73
5.	Bewegungen des ganzen Körpers (idle)	6,29
6.	Hautshader	5,72
7.	Anheben der Augenbrauen	5.71

5.2.4 Was fanden die Testpersonen besonders gut? (Auswahl)

„Blinzeln und Augenbrauen, sowie Kopfbewegung sehen sehr gut aus.“

„Die Echtzeitdarstellung und das Subsurface Scattering“

„Der gesamte Bewegungsablauf wirkt schon recht realistisch.“

„Die kleinen Bewegung, die man beim zweiten oder dritten mal erst bemerkt, wirken sehr gelungen „

„Die Augen wirken lebendig und ansprechend, die Halspartie ist gelungen.“

5.2.5 Was fanden die Testpersonen besonders schlecht? (Auswahl)

„Sie sieht insgesamt ein wenig unfreundlich aus.“

„Die Haut und die Frisur, der Kragen ist zu groß.“

„Anne Will sieht doch sehr viel besser aus.“

„Durch die abgehackte und leicht verzerrte Stimme, wirkt es um einiges unrealistischer. Die Figur sieht ohne Haare realistischer aus als mit Haaren“

„Die Szene ist unreal ausgeleuchtet, zumindest nicht wie ein Fernsehstudio.„

„Die Sprache hört sich unecht an. Die Betonung ist auf manchen Silben falsch.“

5.2.6 Was könnte man ihrer Meinung nach verbessern? (Auswahl)

„Angenehmere Beleuchtung da zu kühle Stimmung“

„Kopfbewegungen beibehalten aber verringern“

„Natürliche Sprache, statt Synthese verwenden“

„Beim Sprechen sollte sich die Wangenmuskulatur deutlicher bewegen.„

„Längere, bewegliche Haare könnten zum Realismus beitragen, da sie sich zusätzlich mitbewegen würden.“

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

Es wurde eine Anwendung erstellt, die gemäß Aufgabenstellung einen virtuellen Charakter in Echtzeit simuliert:

- Ansteuerung von mehreren, sich teilweise überlagernden Skelettanimationen

  hier: Eine Idle-Animation mit Kopfbewegungen während des Sprechens
- künstliche Sprachausgabe mit lippensynchroner Viseme-Ansteuerung.
- Zusätzliche Gesichtsausdrücke, um Emotionen ausdrücken zu können.
- Blinzeln in unregelmäßigen Abständen
- Automatisches In-die-Kamera-schauen.
- Realistische Darstellung der Haut
- Echtzeit: 200 Bilder in der Sekunde bei moderner Grafikhardware. (Geforce6)
- Eine Benutzeroberfläche, um Animationen erstellen, laden, bearbeiten und speichern zu können.

Die Benutzerumfrage lieferte für die qualitativen Fragen normalverteilte Ergebnisse. Signifikant war jedoch, das 50 Personen, also ein Drittel aller Teilnehmer, die Frage „Würden Sie dieser Dame einen Job beim Fernsehen geben?“ mit 0 Punkten bewerteten. Die Kommentare einiger Teilnehmer lassen darauf schließen, dass dies mit einer allgemeinen Ablehnung gegenüber virtuellen Charakteren zusammenhängt.

So schreibt eine der Testpersonen: „Die heutigen Möglichkeiten Menschen etwas unechtes als echt zu präsentieren finde ich beängstigend!“ und eine andere: „Ich bin mir nicht sicher, ob ich eine solche Tagesschausprecherin akzeptieren würde. Vermutlich würde ich das Radio dem Fernsehen vorziehen, wenn ich dort keine Menschen mehr sehen könnte.“

Interessant ist auch, dass die Bewegung des Kopfes als am Wichtigsten empfunden wurde. Eine realistische Darstellung der Haut schien weniger wichtig als die der Haare,

obwohl letztere den Kommentaren nach zu urteilen, insgesamt schlechter bewertet wurden.

6.2 Ausblick

Erweiterungen zu der bestehenden Software wären in vielerlei Hinsicht möglich.

Was die Implementierung des Programmes betrifft, wäre es sicherlich interessant, eine neuartige Technik zur Skelettanimation in den Open-Source Engine einzubauen. Die Haare könnten dynamisch simuliert werden, ebenso wie die Kleidung. Das System könnte interaktiv erweitert werden, etwa mit Spracherkennung, Benutzerlokalisation und einem Dialog-System.

Was die Daten angeht, könnten mehr Gesichtsausdrücke generiert werden um auch Wut, Trauer etc. ausdrücken zu können. Ebenso müssten wesentlich mehr Ganzkörperanimationen aufgenommen werden, damit die Bewegungen mehr Abwechslung zeigen. Darüber hinaus wäre eine Einbringung anderer Charaktere und Studios wünschenswert.

Die Sprachsynthese ließe sich noch verbessern, indem der zu sprechende Text mit Annotierungen versehen und dem System noch unbekannte Wörter definiert würden. Diese Funktion könnte wiederum von der grafischen Benutzeroberfläche unterstützt werden.

Damit hätte man ein umfangreiches Autorentool, mit dem z.B. Fernsehsendungen produziert werden könnten.

Ob diese jedoch von den Zuschauern akzeptiert würden, bleibt fraglich.

7 Anhang

7.1 Mathematische Grundlagen

7.1.1 Transformation mit Matrizen

Die Koordinaten eines Punktes werden in Vektoren festgehalten. Für den zweidimensionalen Fall, die euklidische Ebene R^2, besteht ein Vektor aus den zwei Koordinaten x und y, also p=(x,y).

Eine Drehung um den Ursprung um den Winkel α entgegen dem Uhrzeigersinn kann mit einer Matrix realisiert werden:

$$R=\begin{pmatrix}\cos(\alpha) & \sin(\alpha)\\ -\sin(\alpha) & \cos(\alpha)\end{pmatrix}$$

Formel 7.1.1.1.:Rotationsmatrix im R^2

Den gedrehten Punkt p' kann man dann mit p' = R◦p errechnen (Multiplikation von rechts).

Damit eine Matrix als eine Rotationsmatrix bezeichnet werden kann, müssen die folgenden beiden Bedingungen erfüllt sein:

- Die Spalten bilden Einheitsvektoren einer Othonormalbasis, d.h. die Spaltenvektoren stehen alle senkrecht aufeinander.
- Die Determinante der Matrix ist 1.

Im dreidimensionalen Raum R^3 ist das Konstruieren einer Rotationsmatrix etwas komplizierter, weil eine Drehachse benötigt wird:

$$R_x=\begin{pmatrix}1 & 0 & 0\\ 0 & \cos(\alpha) & -\sin(\alpha)\\ 0 & \sin(\alpha) & \cos(\alpha)\end{pmatrix}$$

Formel 7.1.1.2.:Drehung im R^3 um die x-Achse

$$R_y=\begin{pmatrix}\cos(\alpha) & 0 & \sin(\alpha)\\ 0 & 1 & 0\\ -\sin(\alpha) & 0 & \cos(\alpha)\end{pmatrix}$$

Formel 7.1.1.3.:Drehung im R^3 um die y-Achse

$$R_z=\begin{pmatrix}\cos(\alpha) & -\sin(\alpha) & 0\\ \sin(\alpha) & \cos(\alpha) & 0\\ 0 & 0 & 1\end{pmatrix}$$

Formel 7.1.1.4.:Drehung im R^3 um die z-Achse

7.1.2 Eulersche Winkel

Man kann Rotationen akkumulieren, indem man mehrere solcher Matrizen hintereinander multipliziert, da die Matrixmultiplikation assoziativ ist. Als Parameter für eine Drehung reichen dann die Drehwinkel für jede Achse. Diese werden *Eulersche Winkel* oder einfach *Eulerwinkel* genannt.

Da die Matrixmultiplikation aber nicht kommutativ ist, ist die Reihenfolge der Drehachsen nicht beliebig. Wenn man Rotationen mit *Eulerschen Winkeln* beschreiben will, ist es daher notwendig, zu definieren, in welcher Reihenfolge die Drehungen angewendet werden sollen.

7.1.3 Gimbal Lock

Die Angabe von Rotationen mit Eulerwinkel hat ein gravierendes Problem: Wenn der erste Drehwinkel ±90 Grad beträgt, fallen die anderen beiden Achsen aufeinander. Dann addieren sich die beiden anderen Winkel und ein Freiheitsgrad geht somit verloren. Diese mit Eulerwinkeln nicht vermeidbare Singularität nennt man *Gimbal Lock*.

7.1.4 Homogene Koordinaten

Damit sich zusätzlich zu Rotationen auch Translationen mit Hilfe von Matrixmultiplikation ausdrücken lassen, wird dieser Vektor um die homogene Koordinate w erweitert:

$$p = (x,y,z,w).$$

Für Vektoren, die einen Punkt im Raum beschreiben, setzt man w=1. Möchte man mit dem Vektor jedoch eine Richtung ausdrücken, die ja nicht verschoben werden soll, so setzt man w=0. Eine 4x4 Transformationsmatrix hat diese Struktur:

$$M = \begin{pmatrix} m_{00} & m_{10} & m_{20} & t_x \\ m_{01} & m_{11} & m_{21} & t_y \\ m_{02} & m_{12} & m_{22} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Formel 7.1.4.1.: Transformationsmatrix

Die ursprüngliche 3x3 Rotationsmatrix ist nun in den Werten m_{00} bis m_{22} enthalten.

Die Werte t_x,t_y,t_z entsprechen der Translation. Diese werden nach den Regeln der Matrixmultiplikation immer zu den Werten des Vektors addiert. Eine Skalierung in x-,y- oder z-Richtung kommt zustande, wenn man die jeweiligen diagonalen Werte m_{00}/ m_{11}/m_{22} mit einem Skalierungsfaktor multipliziert.

7.1.5 Quaternionen

Mit diesen 4x4 Matrizen lässt sich jeder Punkt im R^3 beliebig transformieren. Zum platzsparenden Speichern der Rotation eines Objektes im Raum könnte man Eulerwinkel verwenden. Eulerwinkel und Rotationsmatrizen lassen sich nämlich ineinander überführen. Die Eulerwinkel sind aber, wie die Rotationsmatrizen auch, nicht immer intuitiv bestimmbar. Hinzu kommen praktische Probleme mit dem *Gimbal Lock*. Deshalb wird noch eine andere Darstellung für Rotationen benötigt.
Eine elegante Art, um Rotationen auszudrücken, sind Quaternionen. Ein Quaternion beinhaltet einen Drehwinkel (w) und eine Drehachse v=(x,y,z), also vier reelle Komponenten.

Man kann Quaternionen als eine Erweiterung der imaginären Zahlen begreifen, wobei die Komponenten der Drehachse mit den 3 imaginären Anteilen *i*, *j* und *k* multipliziert werden.

Ein Quaternion ist also

$$q = (w, v) = (w + xi + yj + zk).$$

Für die imaginären Komponenten gelten folgende Zusammenhänge:

$$i^2 = j^2 = k^2 = -1$$

$$ij = -ji = k$$

$$jk = -kj = i$$

$$ki = -ik = j$$

Auf Quaternionen kann eine Norm definiert werden:

$$|a+bi+cj+dk|^2 = a^2+b^2+c^2+d^2$$

Das konjungierte Quaternion hat folgende Form:

$$\bar{q} = \overline{a+bi+cj+dk} = a-bi-cj-dk$$

Wenn die Norm eines Quaternion |q|=1 ist, spricht man von einem *Einheitsquaternion*.

Dabei ist die Inverse q^{-1} gleich der Konjungierten.

Eine Multiplikation zwischen zwei Quaternionen (q_1, q_2) wird definiert als:

$$q_q \cdot q_2 = (w_1, v_1) \cdot (w_2, v_2) = (w_1 w_2 - v_1 \cdot v_2, w_2 v_1 + w_1 v_2 + v_1 \times v_2)$$

Formel 7.1.5.1.:Multiplikation von Quaternionen

Damit kann man nun jeden beliebigen Punkt im R^3 drehen. Sei r der Punkt, der um eine Achse v mit dem Winkel α gedreht werden soll, konstruiert man aus p ein rein imaginäres Quaternion p = *(0, r)*.

Aus der normalisierten Achse n = ||v|| und dem Winkel α wird das Quaternion q konstruiert, das immer ein Einheitsquaternion ist:

$$q = (\cos(\frac{\alpha}{2}), \sin(\frac{\alpha}{2}) \cdot n)$$

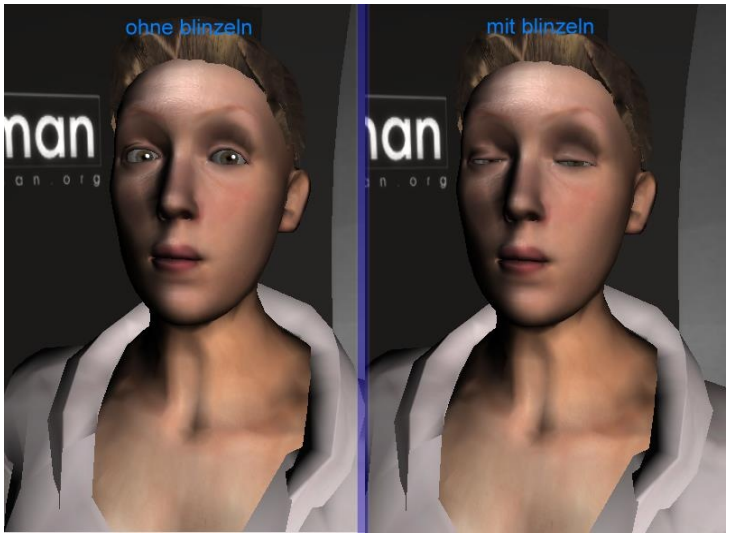
Eine Rotation erreicht man nun mit:

$$p' = q \cdot p \cdot q^{-1}$$

p' ist jetzt wiederum ein rein imaginäres Quaternion, das den gedrehten Vektor enthält.

Durch mehrmaliges multiplizieren lassen sich, wie auch bei Rotationsmatrizen, Rotationen akkumulieren.

7.2 Zur Umfrage verwendete Bilder



Die Idle-Animation sowie die Kopfbewegung lassen sich mit einem Einzelbild nicht adäquat erfassen.

7.3 Literaturverzeichnis

7.3.1 Wissenschaftliche Artikel und Bücher

Armstrong & Green (1985) - W. Armstrong, M. Green: The dynamics of articulated rigid bodies for purposes of animation. Proceedings of Graphics Interface '85 (1985)

Blanz & Vetter (1999) - V. Blanz and T. Vetter: A Morphable Model for the Synthesis of 3-D Faces. ACM SIGGRAPH 99 (1999)

Blinn (1978) - Blinn, James F.: Simulation of Wrinkled Surfaces, Computer Graphics, Vol 12(3), pp. 286-292 (1978)

Bußmann (2002) - H. Bußmann: Lexikon der Sprachwissenschaft. 3. akt. u. erw. Auflage (2002)

deGraf et al. (1988) - deGraf, Wahrmann et al.: Mike The Talking Head, Computer Graphics World (1988)

Ekman (1978) - Ekman, P. , Friesen, W.: The Facial Action Coding System. Consulting Psychology Press Inc. (1978)

Fua (2000) - P. Fua: Regularized Bundle-Adjustment to Model Heads from Image Sequences without Calibration Data. International Journal of Computer Vision, vol. 38(2) (2000)

Gamma et. al (1994) - : Design Patterns - Elements of Reusable Object-Oriented Software

Guenter et al. (1998) - B. Guenther, C. Grimm, D. Wood, H. Malvar, F. Pighin: Making Faces. SIGGRAPH '98 (1989)

Hunt & Black (1996) - Hunt, A. J. Black, A. W.: Unit selection in a concatenative speech synthesis system using a large speech database, Proceedings ICASSP '96, Volume 1, S. 373–376 (1996)

Lee et al. (2000) - W. Lee, N. Magnenat-Thalmann: Fast Head Modeling for Animation. Journal of Image and Vision Computing, vol. 18(4)

Lewis et. al (2000) - J.P. Lewis, Matt Cordner, Nickson Fong: Pose space deformation (2000)

Jensen et al. (2001) - Jensen, H.W., Marschner S.R., Levoy, M., Hanrahan, P.: A practical Model for Subsurface Light Transport. In: Proceedings of SIGGRAPH (2001)

Jokisch et al. (1998) - Jokisch, O., Hirschfeld, D., Eichner, M., Hoffmann, R.: Multi-level rhythm control for speech synthesis using hybrid data driven and rule-based approaches. Proc. ICSLP 98, Sydney, vol. 3 (1998)

Jörg (2005) - Jörg, Sophie: Echtzeitsteuerung eines virtuellen Charakters (2005)

Klatt (1979) - Klatt, D.H.: Synthesis by rule of segmental durations in English sentences. In Lindblom, B.,Öhman,S. (Editor): Frontiers of Speech Communication Research, London (1979)

Merry et al. (2006) - Merry, B., Marais, P. and Gain, J: Animation space: a truly linear framework for character animation (2006)

Mixdorff & Fujisaki (1995) - Mixdorff, H, Fujisaki, H.: A scheme for a model-based synthesis by rule of F0 contours of German utterances.In Proceedings of the '95 Eurospeech, Madrid, vol. 3, pp 1823-1826 (1995)

Müller & Grosch (2003) - Müller, S., Grosch, T.: Handout zur Vorlesung Computergrafik II im Wintersemester 03/04, Universität Koblenz. (2003)

Muybridge(1887) – Muybridge, E.: Human and Animal Locomotion (1887)

Nicodemus et al. (1977) - Nicodemus, F.E., Richmond, J.C., Hsia, J.J., Ginsber, I.W., Limperis, T. : Geometrical Considerations and Nomenclature for Reflectance (1977)

Osipa (2003) - Osipa, Jason: Stop Staring - Facial Modeling and Animation Done Right (2003)

Orthony (1990) - Andrew Ortony, Gerald L. Clore, Allan Collins: The Cognitive Structure of Emotions (1990)

Parke & Waters (1996) - F. I. Parke, K. Waters: Computer Facial Animation (1996)

Pandzic & Forchheimer (2002) - I. Pandzic, R. Forchheimer: MPEG-4 Facial Animation: The Standard, Implementation and Applications (2002)

Pixar (1998) - Pixar Animation Studios: Meet Geri: The New Face of Animation. Computer Graphics World, vol. 21(2) (1998)

Rabiner (1989) - Rabiner, L.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, Proceedings of the IEEE, vol. 77, no. 2, pp. 257-286 (1989)

Rost (2004) - Rost, R.: OpenGL Shading Language. (2004)

Sonntag (1999) - Sonntag, G.: Evaluation von Prosodie (1999)

Stöcker (2004) - Stöcker, M.: Erstellen eines Frameworks, das Emotionen in generische Gesichtsmodelle einbindet. (2004)

Thalmann et al.(2004) - N.M. Thalmann, D. Thalmann: Handbook of Virtual Humans (2004)

Torrance & Sparrow (1967) - Torrance, K.E., Sparrow, E.M.: Theory for Off-Specular Reflection from Roughened Surfaces. In J. Opt. Soc. (1967)

Unterberg (2004) - Jörg Unterberg: Dynamisches echtzeitfähiges Hautshading (2004)

Walters (1989) - Graham Walters: The story of Waldo C. Graphics, 3D Character Animation by Computer, ACM SIGGRAPH (1989)

Wang et al. (2002) - Xiaohuan Corina Wang & Cary Phillips: Multi-Weight Enveloping (2002)

Watt & Watt (1992) – Watt, A., Watt, M.: Advanced Animation and Rendering Techniques. Theory and Practice (1992)

7.3.2 Internetseiten:

www:3DLabs - http://www.3dlabs.com/

www:Ananova - http://www.ananova.com (Ananova Ltd., 2000)

www:Borelli - http://www.deutsches-museum.de/bibliothek/unsere-schaetze/biologie/borelli/ (A. Borelli, 1680)

www:Eyetronics - Facesnatcher (Eyetronics Inc., 2006)

www:gamedev - http://www.gamedev.net/columns/hardcore/cgbumpmapping/

www:InterSense - http://www.isense.com/ (Intersense Inc., 2006)

www:Lander - http://www.gamasutra.com/features/20000327/lander_pfv.htm (Jeff Lander, 2000)

www:lighthouse - http://www.lighthouse3d.com/opengl/glsl/

www:Marey - http://www.acmi.net.au/AIC/MAREY_BIO.html (Russell Naughton, 1999)

www:nvidia - http://www.nvidia.de

www:Polhemus - http://www.polhemus.com/ (Polhemus Inc., 2006)

www:Santos - http://www.digital-humans.org/main.htm (University of Iowa, 2006)

www:scanalyze - http://graphics.stanford.edu/software/scanalyze/ (Stanford University, 2006)

www:Seungwoo - http://vr.kaist.ac.kr/~redmong/research.htm (Seungwoo Oh, 2004)

www:ttssamples - http://ttssamples.syntheticspeech.de/deutsch/ (F. Burkhardt, 2006)

www:Vicon - http://www.vicon.com/ (Oxford Metrics, 2006)

www:volfill - http://graphics.stanford.edu/software/volfill/ (Stanford University, 2006)

www:vrdepot - http://www.vrdepot.com/vrteclg.htm (VR Depot, 2006)

www:wikipedia - http://en.wikipedia.org/wiki/Bump_mapping

www:Xsens - http://www.xsens.com/ (Xsens Technologies, 2006)