FACHHOCHSCHULE GIESSEN FRIEDBERG
UNIVERSITY OF APPLIED SCIENCES

DIPLOMARBEIT

Fachgebiet der Diplomarbeit:
Graphische Datenverarbeitung

Thema der Diplomarbeit:
Entwicklung und Implementierung von Methoden und Konzepten zur automatisierten Generierung von 3D-Szenen und Geschichten

Unternehmen in dem die Diplomarbeit durchgeführt wurde:
Zentrum für Graphische Datenverarbeitung e.V. in Darmstadt,
Abteilung Digital Storytelling

Diplomand:	Alexander Hrotko
Referentin:	Prof. Dr.-Ing. Dipl.-Math. Monika Lutz
Korreferent:	Dr. Stefan Göbel
Betreuerin im ZGDV:	Dipl. Media System Designerin (FH) Anja Hoffmann

Fachhochschule Gießen-Friedberg, Bereich Friedberg
Fachbereiche Informationstechnik - Elektrotechnik - Mechatronik,
Mathematik, Naturwissenschaften und Informatik,
Mathematik, Naturwissenschaften und Datenverarbeitung

Fachrichtung Medieninformatik, Sommersemester 2005

Erklärung :

Hiermit erkläre ich, dass ich die Arbeit selbständig und nur unter Zuhilfenahme der aufgeführten Hilfsmittel sowie den Hinweisen meiner Betreuerin und Mitarbeitern des ZGDV e.V. angefertigt habe.

Friedberg, den 16.1.2006

Alexander Hrotko

Inhaltsverzeichnis

1 Einleitung

1.1 Zielsetzung der Arbeit

Ziel dieser Diplomarbeit ist die Entwicklung eines automatisierten interaktiven Geschichtengenerators, welcher durch die Einbindung neuester Technologien die Generierung eines 3D-Animation-Films unter Berücksichtigung der interaktiven Steuerung des Benutzers ermöglichen soll und diesen dann auf verschiedene Ausgabegeräte versendet.

Hierbei soll der Benutzer seine persönlichen Erlebnisse mittels Photos, Dialogen etc. mit in die Geschichte einbringen und somit eine personalisierte Version an andere als Kurzfilm verschicken können. Im Vordergrund dieses Projekts stand die Ermöglichung der Interaktivität des Benutzers, welcher jederzeit durch die Benutzung der von ihm gespeicherten Daten eine aktuelle Geschichte selbst anfertigen kann und somit über seine Erlebnisse während der Fußballweltmeisterschaft 2006 andere auf dem Laufenden halten kann. Natürlich soll auch eine spätere Anwendung für ähnliche Großereignisse oder andere Anwendungen durch Änderung der Gestaltung der Geschichte realisierbar sein.

Der StoryGenerator ist ein Baustein des Servingo-Projekts, welches zur Aufgabe hat, durch die Zusammenarbeit mehrerer Unternehmen (Zentrum für Graphische Datenverarbeitung e. V. (ZGDV), T-Systems International GmbH, Fraunhofer Institut für Graphische Datenverarbeitung (IGD), itCampus Software- und Systemhaus GmbH, infoRoad GmbH, ehotel AG, Fraunhofer Institut für Materialfluss und Logistik (IML), DAI-Labor der TU Berlin, CAS Software AG, Intergraph (Deutschland) GmbH, GIStec GmbH [1]) und die Förderung des Bundesministeriums für Wirtschaft und Arbeit ein komplexes Online-Portal (www.servingo.org) für die Fußball WM 2006 in Deutschland zu schaffen, welches dem Interessierten eine breite Vielfalt von Informations- und Unterhaltungsmöglichkeiten bieten soll.

Das Servingo-Projekt soll den effizienten und innovativen Ablauf der FIFA WM 2006 fördern und eine IT-gestützte Serviceplattform schaffen, die es erlaubt, bei (sportlichen) Großveranstaltungen eine integrierte Informations- und Logistikunterstützung über verschiedenste Zielgruppen hinweg anbieten zu können. Das Logo von Servingo ist in Abbildung 1.1 dargestellt. Die Services bzw. Inhalte werden über unterschiedliche und innovative Kanäle abrufbar sein: von Internet über Mobilfunk (GSM, GPRS, UMTS) bis hin zu breitbandigem DVB-Playout.

Diese Plattform integriert mobile Aspekte und berücksichtigt entsprechende Ansätze, wie z. B. Location-Based-Services, geht aber über die ausschließliche Unterstützung mobiler Systemkomponenten hinaus und ermöglicht auch den stationären Zugriff auf Daten, die durch den Benutzer in seinem „Tagebuch" abgelegt wurden.

Abbildung 1.1: Logo von Servingo [2]

Die Abteilung Digital Storytelling des ZGDV e. V. übernahm die Fertigstellung dieses Projekts, wobei die Aufgaben im Rahmen dieser Diplomarbeit im Folgenden bestanden:

- Design und Erstellung einer Plattform für automatisiertes Generieren von 3D-Szenen und Geschichten mittels der Software Blender.
- Schnittstellenprogrammierung, um eine Anbindung der XML-Inhalte über die Programmiersprache Python nach Blender zu ermöglichen.
- Generierung des Films unter Blender und die Ausgabe auf verschiedene Endgeräte (Notebook, PDA, Mobiltelefon).

Durch die Erstellung eines Demonstrators sollte die Funktionsweise des StoryGenerators bereits zum Confederation Cup 2005 einem breiten Fachpublikum vorgestellt werden. Hierbei handelte es sich um eine noch nicht automatisierte Version des Projekts, welche einen sehr positiven Anklang in den Medien fand.

Die Kombination der verschiedenen Technologien der graphischen Datenverarbeitung, der Informatik und des Mediendesigns, machten dieses Projekt zu einer praktischen Demonstration der Nutzung der neuen Medien. Einen Überblick darüber bietet die nachfolgende Abbildung 1.2 .

Abbildung 1.2: Servingo-Plakat StoryGenerator [1]

1.2 Gliederung der Arbeit

Im Nachfolgenden Kapitel 2 wird zunächst das Konzept des automatisierten Geschichtenerzählens vorgestellt und der Aufbau und die Funktionalität der einzelnen Komponenten des StoryGenerators erläutert. Außerdem werden die einzelnen Bestandteile des online Portals und deren Funktionsweise vorgestellt.

Kapitel 3 beschreibt die Design Anforderungen der Plattform und die Realisierung der 3D–Komponenten mittels der Software Blender, sowie die Verwirklichung der Animationen.

Das vierte Kapitel befasst sich mit der technischen Umsetzung der an das Projekt gestellten Aufgaben. Hier werden die Anbindungen der einzelnen Schnittstellen, sowie die programmiertechnische Bewerkstelligung der an das Projekt gestellten Anforderungen erklärt.

2 Der StoryGenerator

2.1 Automatisierte Geschichtengenerierung

Das personalisierte Informationsportal ist ein zentraler Service des Servingo-Systems. Es ist in besonderer Weise dazu gedacht, den sportbegeisterten Besucher zu begleiten und ihm ein durchgängiges Spielerlebnis zu ermöglichen. Der Fan soll die Möglichkeit erhalten, auf für ihn abgestimmte, Spiel begleitende und ortsbezogene Informationen zuzugreifen.

Der registrierte Nutzer soll dabei in jeder Situation optimal und ansprechend über Spiel bezogene Themen informiert werden und zugleich auch die Möglichkeit erhalten, seine Erlebnisse wie in einem persönlichen Tagebuch „festzuhalten". Nachdem der Nutzer sich mit seinem Profil im System registriert hat, wird ein persönliches WM-Logfile für ihn angelegt.

Dieses enthält außer den allgemeinen und öffentlichen Spielinformationen Berichte und Sequenzen, die seinem Interessenprofil und vor allem auch seiner Beteiligung an den Meisterschaften entsprechen. Bucht sich der Fan z. B. bei einem bestimmten Spiel ins System ein, kann er seine eigenen Informationen – wie mit dem Mobiltelefon/PDA aufgenommene Bilder oder SMS – in sein Event-Logbuch aufnehmen und zusammen mit den allgemeinen Spielinformationen zu einer „Geschichte" verarbeiten [3]. Einen kurzen Überblick über die Funktionalität bietet die Abbildung 2.1 .

Abbildung 2.1: Personalisiertes Portal [4]

Durch den verbreiteten Einsatz mobiler Endgeräte, wie z. B. Photohandys und PDAs, soll dem Benutzer durch die automatisierte Geschichtengenerierung eine Möglichkeit gegeben werden, seine Eindrücke aktuell, visuell und mittels des Einsatzes neuer Medien an andere weiterzugeben. Der Grundgedanke des StoryGenerators war eine voll automatisierte Plattform, die dem Benutzer erlauben sollte, durch eine Vielzahl von Einstellungsmöglichkeiten eine maßgeschneiderte Geschichte zu erzeugen und diese als Film an Notebooks oder Handys zu versenden. Dies wird in Abbildung 2.2 demonstriert.

Abbildung 2.2: Ausgabe auf Endgeräten [4]

Der Benutzer kann nach dem Login, durch den Zugriff auf sein persönliches Tagebuch, auch auf seine früher abgelegten Daten zugreifen und diese immer wieder in der neuen Geschichte verwenden. Außerdem hat er die Möglichkeit, durch die Anbindung an das Servingo-Portal, seine Points of Interest („seine Sehenswürdigkeiten") einzusehen und diese Orte in seine Geschichte mit einzubeziehen. Durch die Verwendung von 3D-Charakteren soll die Geschichte lebendiger und visuell attraktiver für andere Betrachter gemacht werden. Die Charaktere können vom Benutzer selbst zusammengestellt werden und es wird ihm die Möglichkeit gegeben, die Dialoge der Figuren zu beeinflussen. Somit ergibt sich ein individueller Film, welcher die Photos und die gespeicherten Tagebucheinträge des Benutzers auf eine attraktive Weise darstellt.

2.2 Aufbau der Komponenten

Der dem Benutzer sichtbare Bestandteil des Servingo-Portals ist die Servingo-Homepage, welche vom Portal aus nach erfolgtem Login erreicht werden kann. Vorher musste er sich bereits erfolgreich als Benutzer registriert haben. Sämtliche Eingaben, die er dann später in dem StoryGenerator tätigt, werden über HTML-Forms an das StoryGenerator-Servlet gesendet und dort weiterverarbeitet. Die Abbildung 2.3 verdeutlicht den Ablauf.

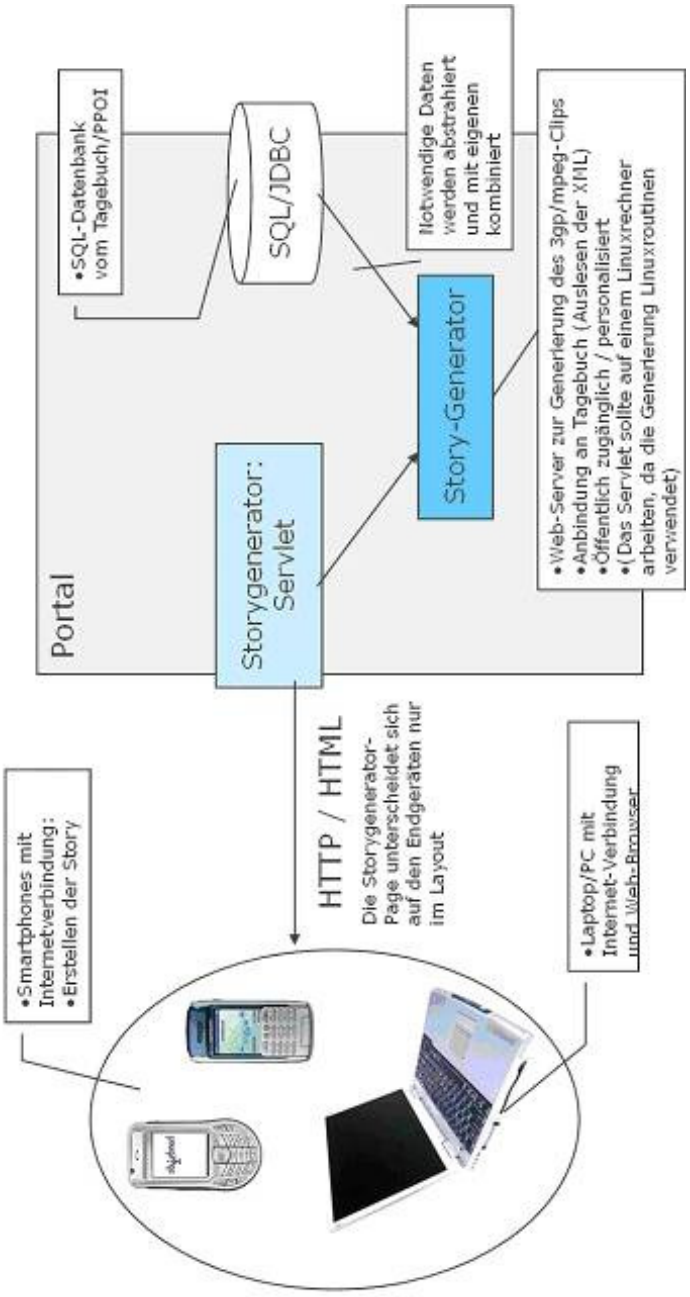


Abbildung 2.3: Funktionsweise des Portals [4]

Das Servlet verfügt seinerseits auch über eine Verbindung zur pPOI (personal Points of Interest)/Tagebuch-Schnittstelle, die durch ein XML-Formular realisiert wird. Dadurch erfolgt die Anbindung an die Datenbank und somit auf die Inhalte, die der Anwender früher in seinem Profil abgelegt hat.

Nach erfolgreicher Beendigung des benutzerseitigen Generierungsprozesses wird eine StoryGenerator-interne finale XML erstellt, wobei die von pPOI und Tagebuch gesammelten Daten veredelt und mit vorgefertigten Inhalten kombiniert werden. Diese XML dient für die serverseitige Generierung des finalen Films mittels Blender. Außerdem werden die vom Benutzer eingegebenen Dialoge in WAVE-Dateien umgewandelt und später in die fertige Geschichte eingebaut. Somit erhält man nicht nur einen visuell ansprechenden Film, sondern auch eine simultane sprachliche Begleitung, die durch den Dialog der zwei Charaktere und der Eingabe des Benutzers realisiert wird.

2.2.1 Erstellung und Archivierung

Der Autor einer Geschichte benutzt hierfür eine HTML-Seite, die er als registrierter Benutzer über das Servingo-Portal aufrufen kann. Diese wird von einer anderen Abteilung des ZGDV e.V. geliefert. Später sollen auch offline Editoren dazukommen, die allerdings nur auf ausgewählten Plattformen zur Verfügung gestellt werden (wie z. B. Stadion, Messe).

Um die Geschichte zu erstellen, wählt der Autor entweder Medien aus seiner lokalen Datenhaltung aus oder greift auf die im persönlichen Tagebuch archivierten Inhalte zu. Hierfür kann er eine Auswahlmenge („alle mit 5 Sternen", „alle aus der Kategorie *people*", etc.,…) angeben, die dann als Miniaturansicht oder mit Titeln angezeigt werden. Er kann nun ein spezielles Medienelement oder eine Auswahlmenge angeben.

Dieser Vorgang wird wiederholt, bis alle Szenen der Geschichte ausgefüllt sind. Der aktuelle Status wird jeweils auf dem Client zwischengespeichert. Der Benutzer kann auch jederzeit zu den einzelnen Auswahlmasken zurückkehren und seine Angaben korrigieren. Hat der Autor seine Geschichte finalisiert, schickt er diese per SOAP und HTTP auf den Server. Eine detaillierte Übersicht stellt die Abbildung 2.4 dar.

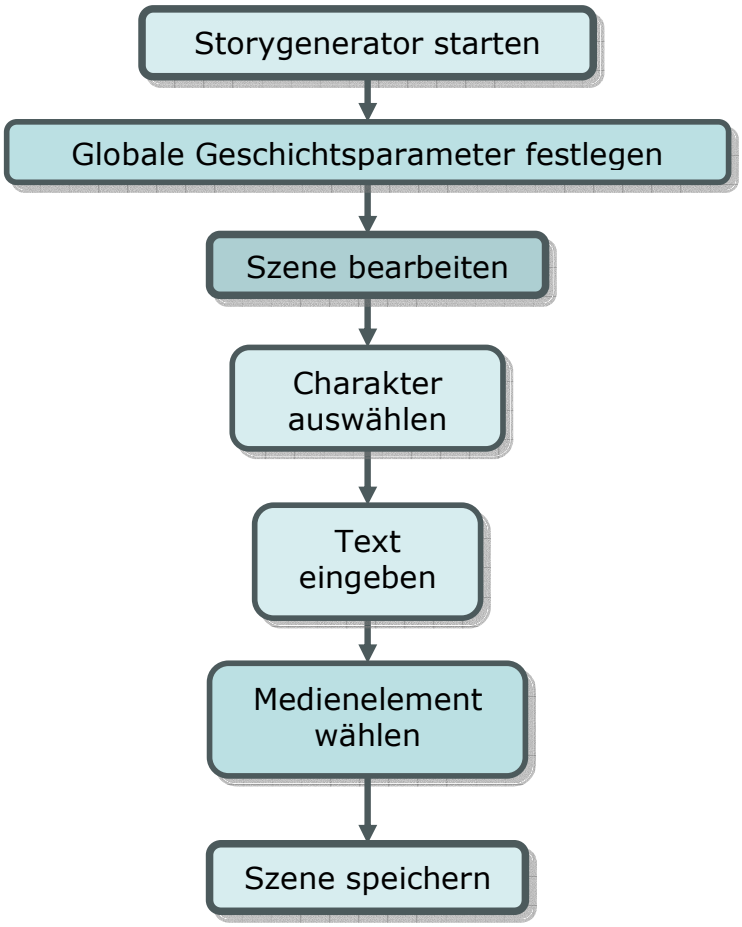


Abbildung 2.4: Diagramm der Erstellungsaktion

In dieser Auswahlmaske hat der Anwender die Möglichkeit, den Titel der Geschichte, das Geschlecht, die Frisur, diverse Accessoires und die Kleidung des Avatars zu wählen. Die fertige Version soll es auch möglich machen, die einzelnen Kleidungsstücke mit verschiedenen Texturen zu belegen (wie z. B. Länderfahnen der teilnehmenden Nationen als Aufdruck auf den T-Shirts). Dadurch soll es dem Fan ermöglicht werden, sein persönliches Aussehen über den 3D-Charakter seinen Freunden mitzuteilen, oder eben nach Lust und Laune den Charakter zu gestalten. Bei jeder Auswahl wird der dargestellte Avatar aktualisiert, wodurch der Benutzer zu jeder Zeit seine Entscheidung ändern kann. Beispiele dazu bieten die Abbildungen 2.5 und 2.6 .

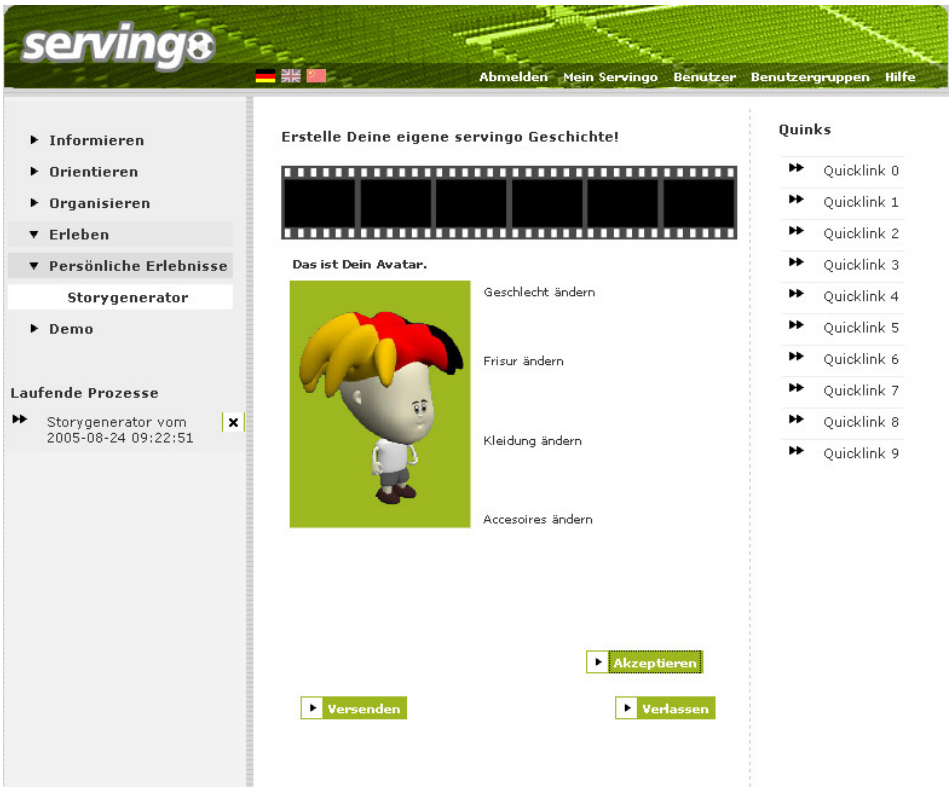


Abbildung 2.5: Auswahlmaske Servingo-Portal [4]

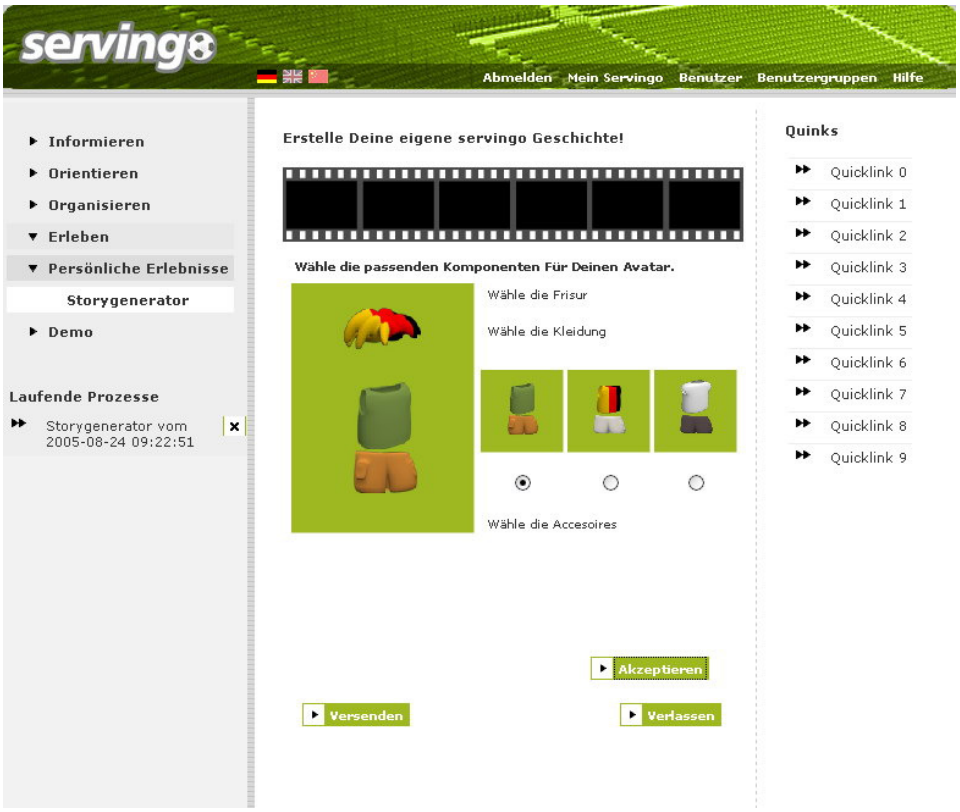


Abbildung 2.6: Auswahlmaske Servingo-Portal, Kleider [4]

Die Inhalte der Auswahlmaske können jederzeit modifiziert werden und somit für andere Großereignisse, oder ähnliche Events angepasst werden, d. h. durch die Modifikation der Charaktere und den Umbau der Szene und der Dialoge kann der StoryGenerator dem jeweiligen Anlass angeglichen werden. Dadurch ist ein multifunktionales Portal entstanden, welches auch in Zukunft für zahlreiche Anlässe nutzbar sein wird.

Nach der Gestaltung des Charakters kann der Anwender auf seine zuvor gespeicherten Mediendaten zugreifen und nun seine Geschichte zusammenstellen. Dazu hat er die Möglichkeit seine Bilder, die er zuvor in seinem Tagebuch abgelegt hat, einzusehen und diese dann in die Geschichte zu integrieren. Die Abbildung 2.7 zeigt ein Beispiel der Bilder Auswahlmaske.

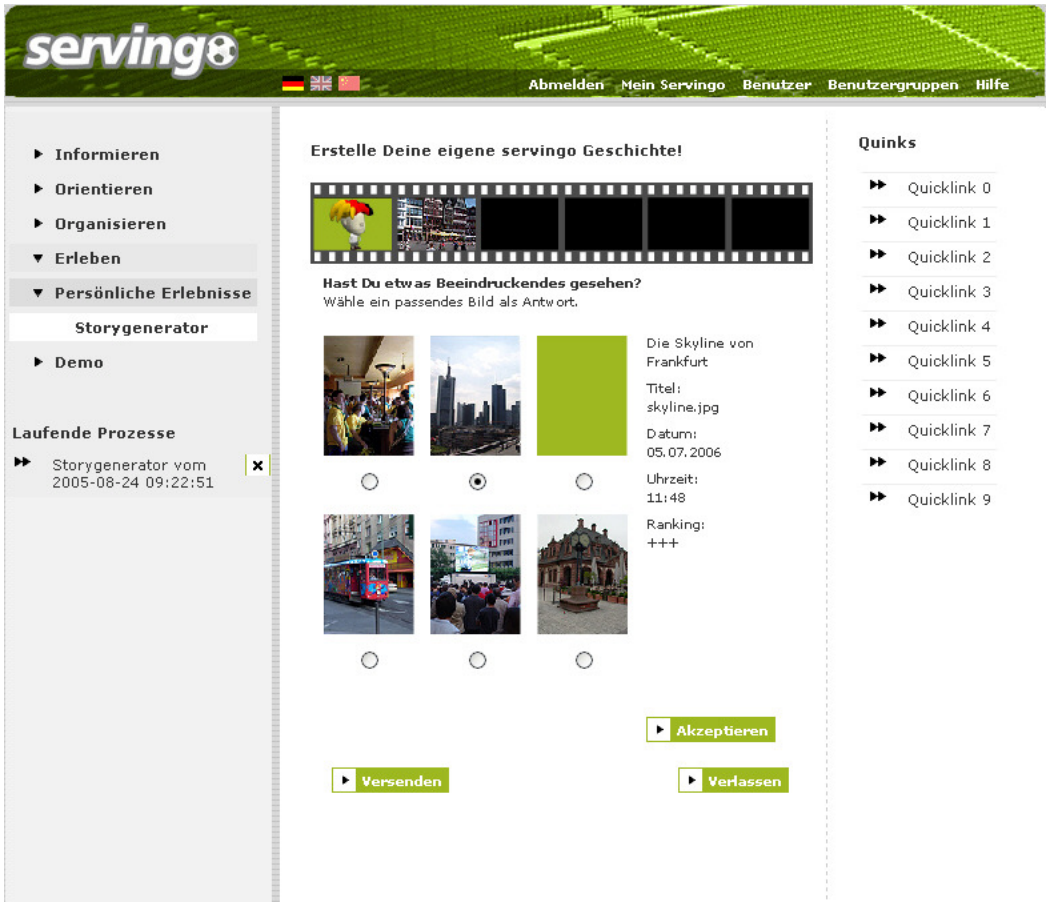


Abbildung 2.7: Auswahlmaske mit gespeicherten Bildern [4]

Hierbei wird er durch Aufforderungen, wie z. B. „Was war dein schönstes Erlebnis?" oder „Hast du etwas Spannendes erlebt?" durch die Erstellung der Geschichte begleitet. Diese Fragen dienen dazu, dass die spätere Geschichte in der die Charaktere einen Dialog über die Bilder führen, logisch stimmig ist. Der Benutzer hat dann die Möglichkeit während des Erstellungsvorgangs zu jedem Bild einen Namen und einen kurzen Dialog anzugeben und dieser wird dann zusammen mit der Darstellung des Bildes, z. B. wie folgt, im späteren fertigen Film von den Charakteren präsentiert (Bild: Elfmeterschießen (Abbildung 2.8), Benutzereingabe kursiv):

„A: *Das Elfmeterschießen in der 86. Minute*, das war vielleicht ein spannender Moment!

B: Da stimme ich Dir zu, so was sieht man nicht alle Tage!"

Abbildung 2.8: Screenshot fertiger Film

Nachdem die Bilder ausgesucht und die Dialoge ausgefüllt wurden, kann der Benutzer den Film generieren lassen. Hier hat er dann auch die Möglichkeit, die E-Mail Adresse oder Telefonnummer des Empfängers einzutragen, welcher den fertig generierten Film erhalten soll. Außerdem kann die Geschichte gespeichert werden, um sie später verändern zu können oder an andere zu verschicken. Die Abbildungen 2.9 und 2.10 stellen Beispiele dazu dar.

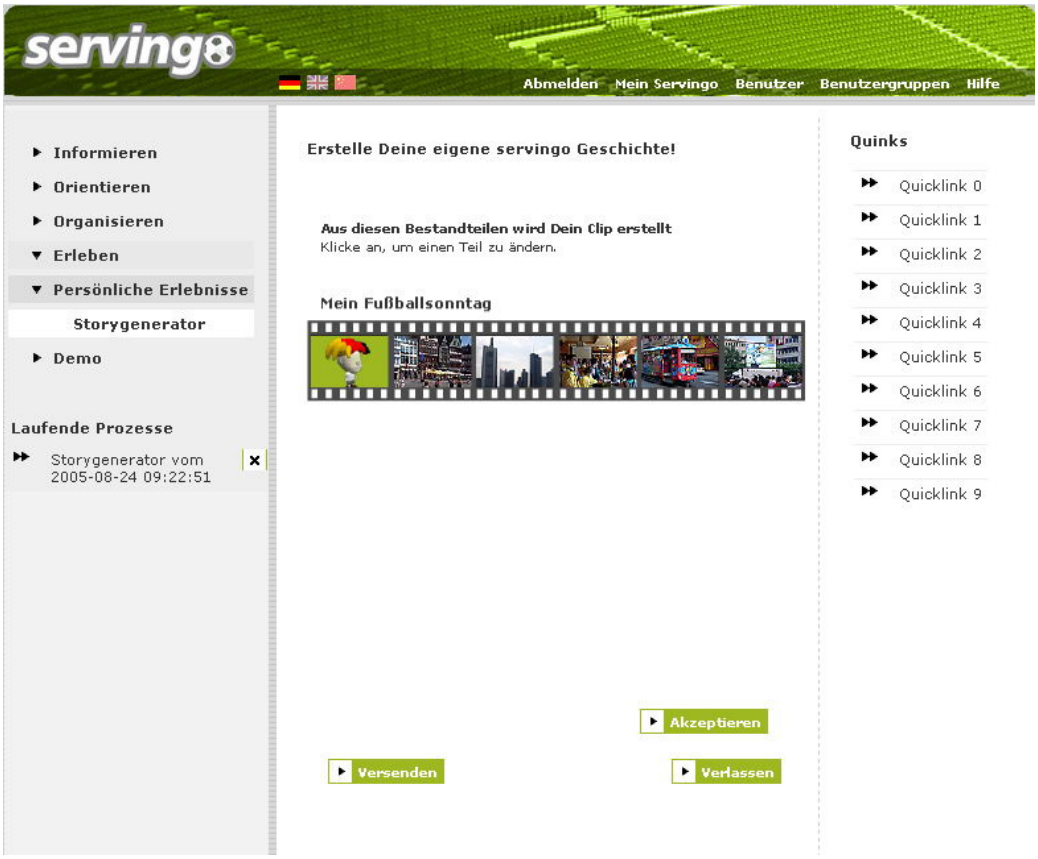


Abbildung 2.9: Fertige Filmzusammenstellung [4]

Zur Fußball-WM 2006 soll es auch noch möglich sein, verschiedene Sprachen für das Portal zu wählen und andere Szenarien und Hintergründe für den Dialog der Charaktere auszusuchen. Außerdem ist es später möglich, durch Änderungen der Dialoge, Charaktere und des Designs, die Servingo-Funktionalität in anderen Projekten zu verwenden.

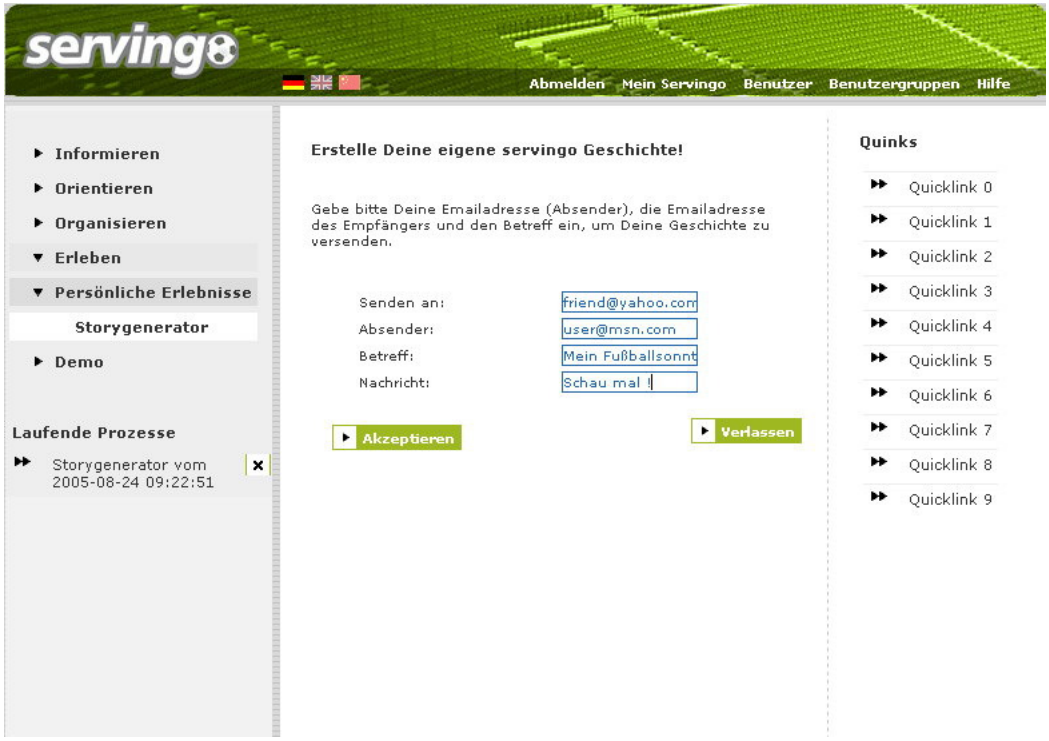


Abbildung 2.10: Eingabemaske für das Versenden [4]

2.2.2 Generierung und Publizierung

Serverseitig findet die Generierung der Story statt (Abbildung 2.11). Hierfür werden auf Basis eines StoryModels vorgefertigte Inhalte mit persönlichen Inhalten kombiniert und als Film in folgenden Formaten kodiert: .3gp für Smartphones, .mpg für Notebook/PC. Die Filme werden auf der persönlichen Website über das Portal zur Verfügung gestellt und gleichzeitig eine Meldung (wahlweise per SMS, MMS, E-Mail) an den Autor versandt. Diese Meldung beinhaltet die URL zum finalen Film und kann wiederum an Freunde per SMS/MMS/E-Mail versendet werden.

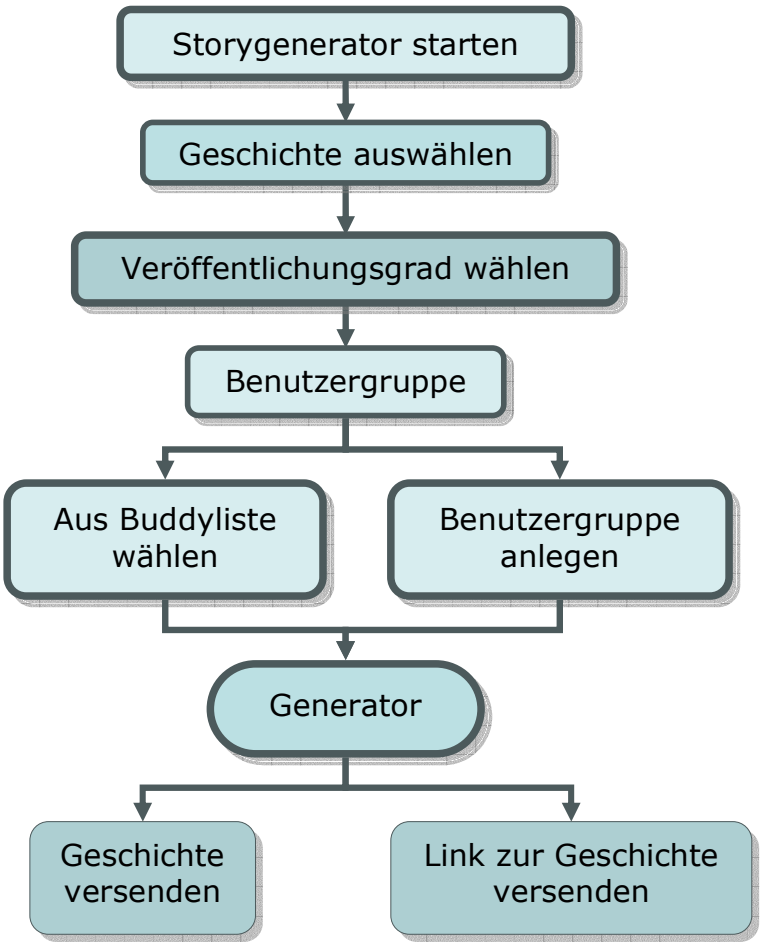


Abbildung 2.11: Diagramm des Generierungsprozesses

Technisch gesehen wird aus den Benutzereingaben eine XML-Datei gebildet, die alle benötigten Informationen an den Server sendet und dieser dann den Generierungsprozess durchführt. Die genauen technischen Einzelheiten werden im Kapitel 4 erläutert.

2.3 Zusammenfassung

In diesem Kapitel wurde der Umfang und die Zielsetzung des Servingo-Projekts und des StoryGenerators erklärt. Nachdem die theoretische und die zeitliche Planung abgeschlossen war und die Anforderungen an das Projekt festgelegt waren, wurde die Realisierung der einzelnen Komponenten in Angriff genommen. Dabei stand die Entwicklung und das Design der 3D-Objekte im Vordergrund, um diese schon vorläufig auf Messen und Events vorstellen zu können und sie gegebenenfalls anpassen zu können. Der Entstehung der 3D-Objekte widmet sich das folgende Kapitel.

3 Design der 3D-Komponenten

3.1 Einleitung

Eine der Hauptaufgaben bei dem Projekt StoryGenerator, die im Rahmen dieser Diplomarbeit durchgeführt wurde, war die graphische Umsetzung der Vorstellungen des Teams, welche die Gestaltung der Charaktere, die Animationen und der Bühne beinhaltete. Dabei war es wichtig, durch den steten Kontakt mit den Kollegen, die zwar für andere Teile des Projekts zuständig waren, gemeinsam unsere Vorstellungen über das endgültige Aussehen der einzelnen Komponenten zusammen zu diskutieren. Durch diesen effizienten Austauschprozess konnten schon während der Entwicklungsphase Probleme bei der Gestaltung und Programmierung verhindert oder behoben werden.

Außerdem wurden die Zwischenergebnisse äußerst erfolgreich auch auf Messen und vor allem zum Confederation Cup am 29.06.2005 in Frankfurt am Main in einem Demonstrator dem breiten Publikum und den Medien vorgestellt, wodurch die Wirkung getestet werden konnte und es möglich war, Verbesserungsvorschläge in das laufende Projekt zu integrieren.

Um die Visualisierung der generierten Geschichte zu ermöglichen, wurde eine 3D-Szene entwickelt, die die vom Benutzer eingegebenen Dialoge und Bilder auf unterhaltsame und optisch ansprechende Weise erzeugen sollte. Die nachfolgenden Kapitel bieten einen Überblick über die Entstehung der einzelnen 3D-Komponenten unter der Benutzung der 3D-Software Blender.

3.2 Die Software Blender

Bei Blender (Abbildung 3.1) handelt es sich um eine vollwertige Open Source 3D-Modellierungssoftware, d. h. sie ist kostenlos und ihr Quellcode ist frei zugänglich und wird deshalb ständig weiterentwickelt. In ihrer aktuellen Version 2.37 bietet diese Software vieles, was auch die bekannteren 3D-Programme, wie z. B. Maya und 3ds Max ermöglichen, mit dem Unterschied, dass die komplexeren Funktionalitäten nicht so leicht wie bei den Marktführern zugänglich sind. Das bedeutet, dass der Anwender zuerst viel Zeit investieren muss, um die einzelnen Funktionen und den Aufbau des Programms zu verstehen.

Abbildung 3.1: Startlogo von Blender 2.37

Dafür beherrscht er dann ein Programm, welches ständig in zahlreichen Foren und über Plug-ins weiterentwickelt wird und was vor allem kostenlos ist. Blender besitzt unter anderem eine eigene leistungsfähige Game Engine, hervorragende integrierte Renderer (wie z. B. Yafray), ein individuell einstellbares Benutzer-Layout, alle gängigen Modellierungstechniken, eine sehr leistungsfähige Python Schnittstelle (die im nächsten Kapitel näher erläutert wird), Animationswerkzeuge (Skelett Animation mit Kinematik, Vertex Key framing fürs Morphing, Character Animation Pose Editor etc.) und die Möglichkeit Blender-Dateien in zahlreichen Formaten (wie z. B. TGA, JPG, PNG, SGI Movie, IFF, AVI, GIF, TIFF) zu exportieren und auch zahlreiche Dateiformate zu importieren. Einen detaillierten Überblick über diese Software kann man auf der Homepage von Blender gewinnen [5].

Der Aspekt des nicht vorhandenen Kostenfaktors und die hervorragende Anbindungsfähigkeit an andere Programme und Programmiersprachen war auch der ausschlaggebende Punkt, welcher den Entschluss für die Nutzung von Blender für das Projekt Story-Generator brachte. Die Abbildungen 3.2 und 3.3 zeigen Beispieloberflächen von Blender.

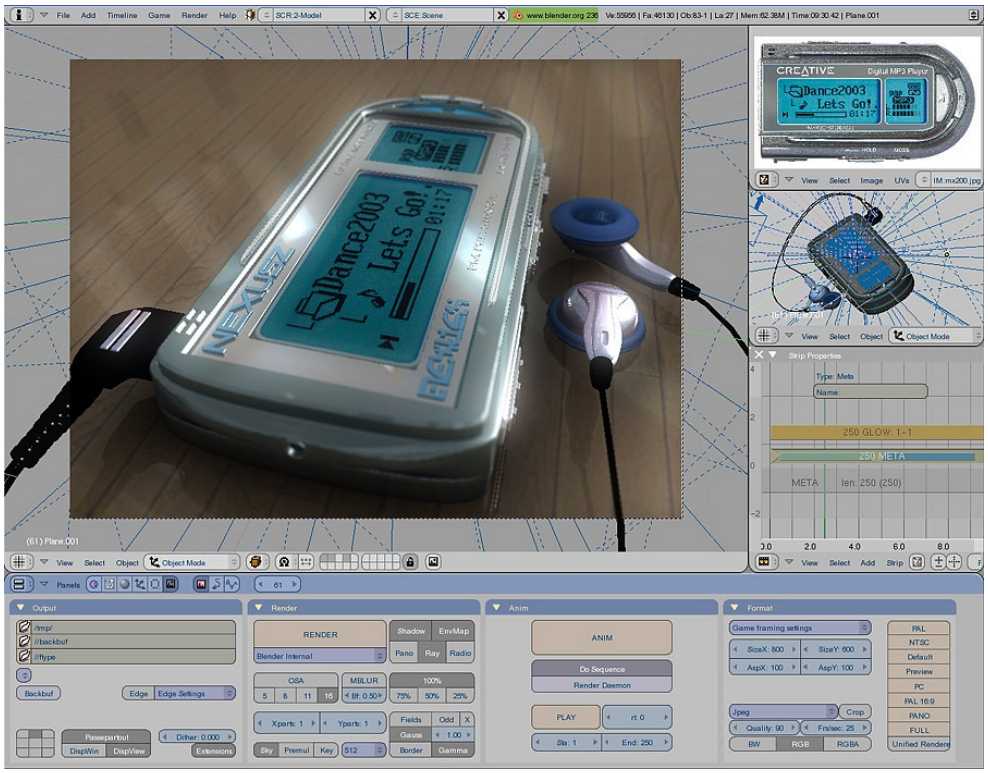


Abbildung 3.2: Beispieloberfläche Blender [5]

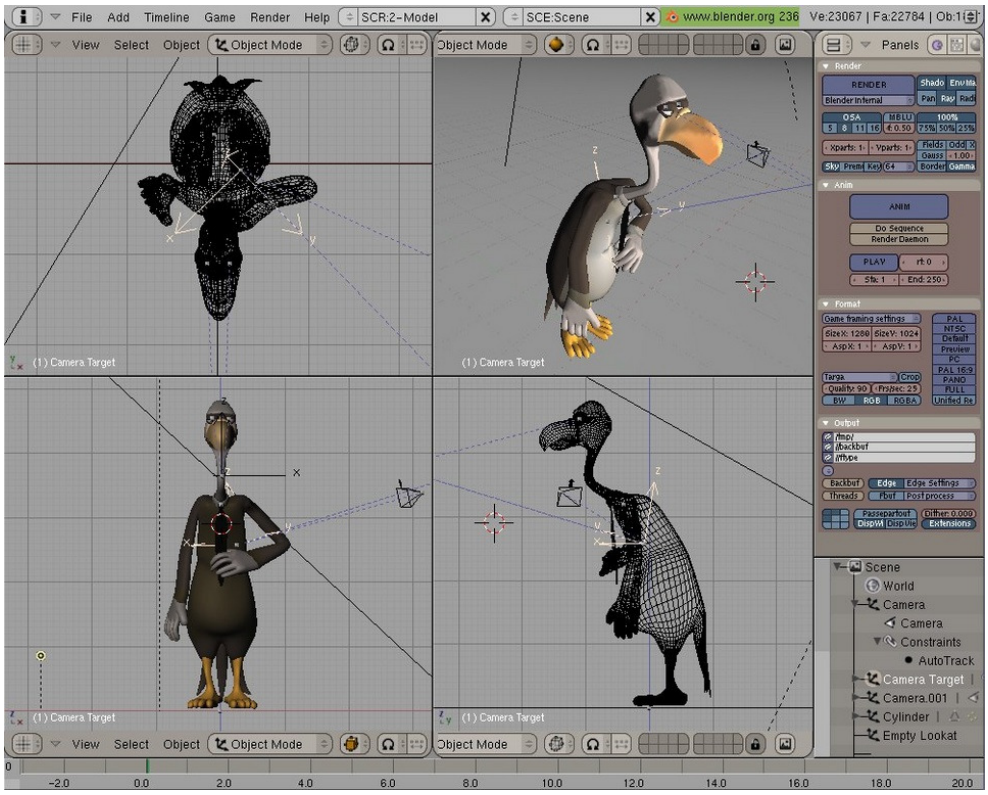


Abbildung 3.3: Beispieloberfläche Blender [5]

3.2.1 Charaktere

Den Anfang des Projekts StoryGenerator bildete der Entwurf und die Umsetzung der Charaktere, wobei es anfangs noch nicht klar war, wie viele verschiedene Figuren benötigt werden und in welchem Umfang sie animierbar und modifizierbar sein sollen. Dazu wurden im Team mehrere Skizzen besprochen und dann im Rahmen der Diplomarbeit in Blender umgesetzt (Abbildung 3.4).

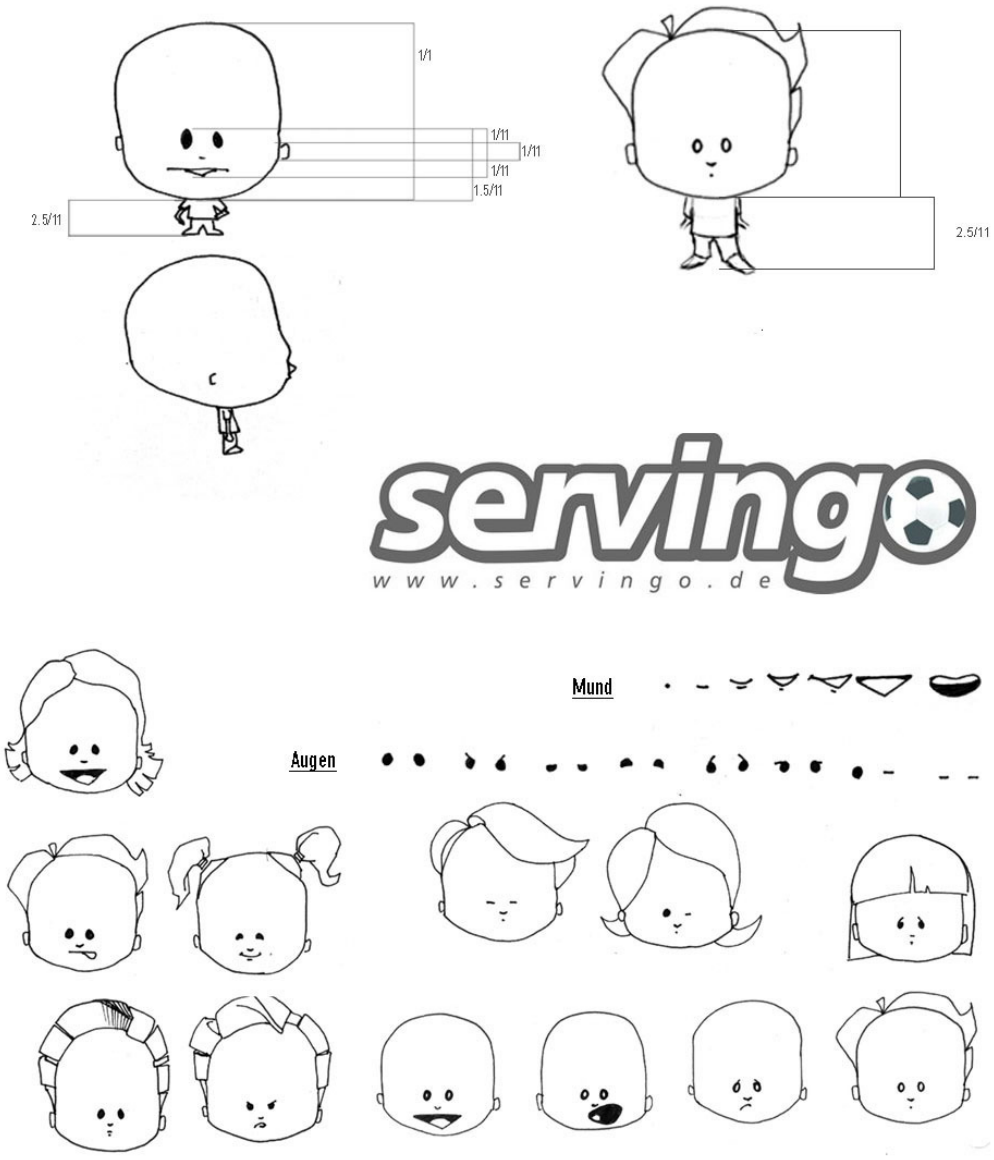


Abbildung 3.4: Skizze Charakterentwurf

Schließlich wurde die Entwicklung eines Reporters und eines männlichen und weiblichen Charakters beschlossen, welche dann durch verschiedene Frisuren und Kleidung geändert werden könnten. Außerdem sollten zahlreiche Accessoires und verschiedene Texturen erstellt werden, um eine spätere Individualisierung durch den Benutzer zu ermöglichen. Den Grundstein jeder Figur bildet der Kopf, der mit Hilfe von NURBS-Kurven (**N**on **U**niform **R**ational **B**-**S**plines) gestaltet wurde, die dann zu einem 3D-Gebilde kombiniert wurden (Abbildungen 3.5a-c).

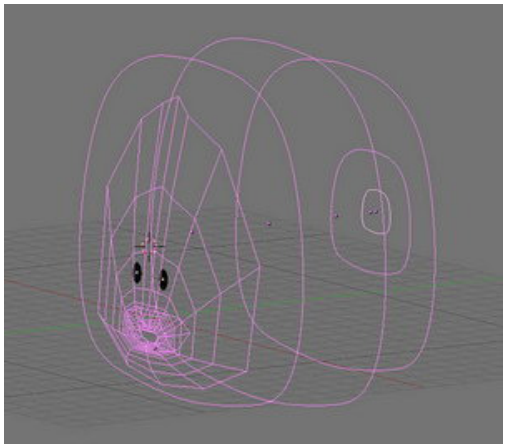


Abbildung 3.5a

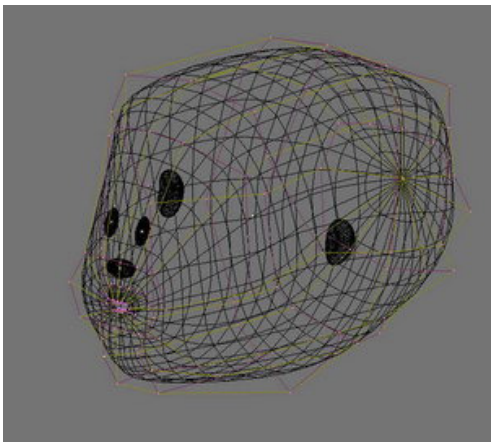


Abbildung 3.5b

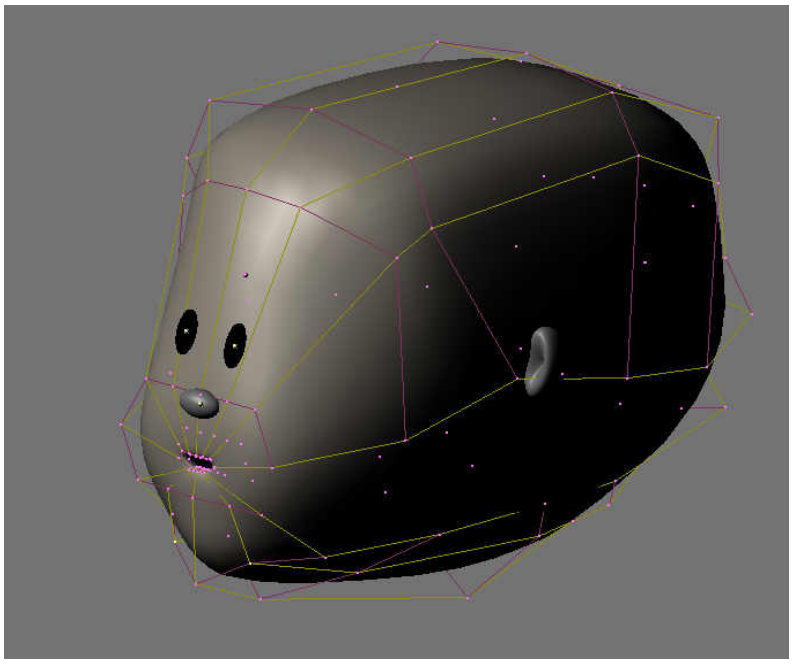


Abbildung 3.5c

3.5a NURBS Kurven, 3.5b Hülle aus NURBS Kurven, 3.5c Kopf (texturiert)

Rationale B-Spline-Kurven erhält man aus B-Spline-Kurven, die man für Punkte in homogenen Koordinaten aufstellt. Setzt man in homogenen Koordinaten die B-Spline-Darstellung an, dann erhält man formal:

$$\vec{p}^{\,h}(t) = \sum_{i=0}^{n} \vec{p}_i^{\,h} N_i^k(t) \text{ , wobei } \vec{p}^{\,h}(t),\ \vec{p}_i^{\,h}$$

Darstellungen der Kurvenpunkte und Kontrollpunkte in homogenen Koordinaten sind. Gilt für die Kontrollpunkte

$$\vec{p}_i^{\,h} = \begin{pmatrix} w_i \cdot x_i \\ w_i \cdot y_i \\ w_i \cdot z_i \\ w_i \end{pmatrix}, \quad w_i \neq 0 \in IR\,,$$

dann erhält man die zugehörigen Punkte $\vec{p}_i(t)$ in dreidimensionalen kartesischen Koordinaten durch Division mit ω_i und damit ergibt sich für die Kurve $\vec{p}(t)$:

$$\vec{p}^{\,h}(t) = \sum_{i=0}^{n} \begin{pmatrix} w_i \cdot x_i \\ w_i \cdot y_i \\ w_i \cdot z_i \\ w_i \end{pmatrix} N_i^k(t) = \begin{pmatrix} \sum w_i \cdot x_i \cdot N_i^k(t) \\ \sum w_i \cdot y_i \cdot N_i^k(t) \\ \sum w_i \cdot z_i \cdot N_i^k(t) \\ \sum w_i \cdot N_i^k(t) \end{pmatrix}$$

und im Dreidimensionalen der folgende rationale Ausdruck:

$$\vec{p}(t) = \frac{\sum_{i=0}^{n} w_i \cdot \vec{p}_i \cdot N_i^k(t)}{\sum_{i=0}^{n} w_i \cdot N_i^k(t)} \cdot\ \vec{p}(t)$$

wird bei nicht notwendigerweise uniformen Parametrisierung der Knotenwerte als Non Uniform Rational B-Spline-Kurve oder kurz NURBS-Kurve der Ordnung k bezeichnet. Die ω_i werden als Gewichte bezeichnet [6].

NURBS-Kurven besitzen folgende Eigenschaften:

- Sie können mit dem de Boor-Algorithmus ausgewertet werden, der dazu auf vier Koordinaten erweitert werden muss.
- Durch das Setzen aller Gewichte ω_i gleich Eins, bekommt man mittels der Eigenschaft der B-Spline-Basisfunktionen

$$\sum_{i=0}^{n-k} N_i^k(t) = 1$$

  die Darstellung einer nicht-rationalen B-Spline-Kurve. Somit enthalten die NURBS die B-Splines.
- NURBS besitzen dieselben Eigenschaften wie B-Splines: Convex-Hull-Property (ihr Bild verläuft vollständig innerhalb der konvexen Hülle ihrer Kontrollpunkte), variationsmindernde Eigenschaft, die Modelliereigenschaften und weitere Eigenschaften.

NURBS besitzen darüber hinaus folgende wesentliche Modelliereigenschaften:

- Mittels NURBS ist es möglich, eine größere Anzahl an analytischen Funktionen mathematisch genau abzubilden.
- Die Gewichte ω_i ermöglichen, zusätzlich zu den Kontrollpunkten, eine weitere Manipulation der Form einer Kurve. Eine Erhöhung des Gewichts ω_i führt zur Verformung des Kurvenverlaufs zum Kontrollpunkt $\vec{p}_i(t)$ hin, bei Verringerung bewegt sich der Kurvenverlauf vom Kontrollpunkt weg.

Durch das Transformieren der Kontrollpunkte ist es in Blender sehr einfach, die 3D-Objekte dem eigenem Geschmack anzupassen. Dazu transformiert oder skaliert man die einzelnen Punkte und kann die Änderungen direkt in Echtzeit beobachten. Die anderen Teile des Charakters wurden mit Hilfe der so genannten Subdivision Surface Methode in Blender hergestellt, die zum Vorteil hat, dass man schnell und einfach schöne organische Formen erzeugen kann. Hierzu modelliert man zuerst die benötigten Teile mit Hilfe von ganz normalen Meshes (Netzen) und rundet diese dann einfach am Ende mit dieser Methode ab. Hierzu wird der Catmull-Clark Algorithmus verwendet, dessen Auswirkungen man in den Abbildung 3.6 und 3.7 sehen kann.

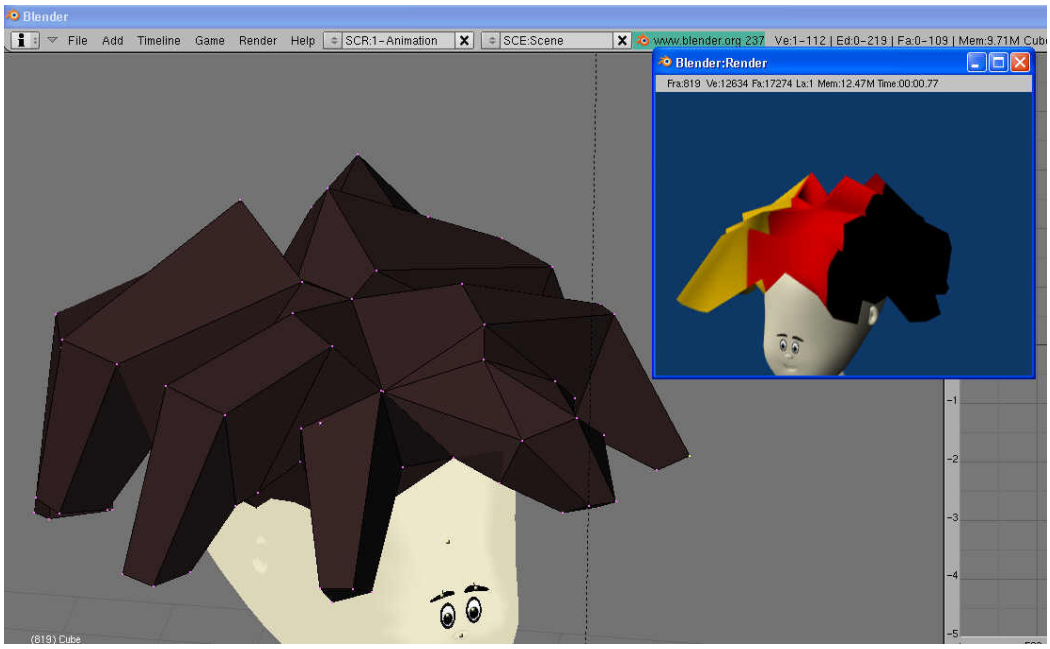


Abbildung 3.6: Frisur ohne Subdivision Surface

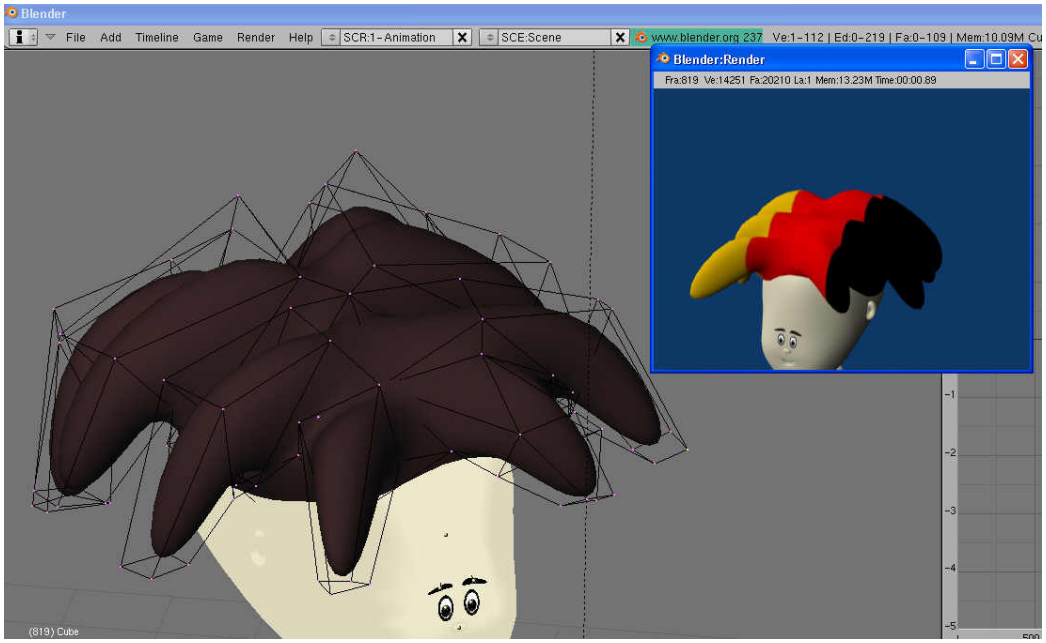


Abbildung 3.7: Frisur mit Subdivision Surface

Die Catmull-Clark-Unterteilung basiert auf bikubischen Tensorproduktflächen (Tensorproduktfläche = Geometrischer Ort der Punkte einer Kurve, die sich durch den Raum bewegt und dabei ihre Gestalt ändert). Das Catmull-Clark-Schema wird durch die Abbildung 3.8 verdeutlicht, wobei man hier ein Beispiel der Unterteilung von links (original) nach rechts (5-te Unterteilung) sehen kann:

Abbildung 3.8: Schematische Darstellung [7]

Der genaue Prozess läuft wie folgt ab:

• Das Ausgangsobjekt ist ein Viereckснetz (Abbildung 3.9)

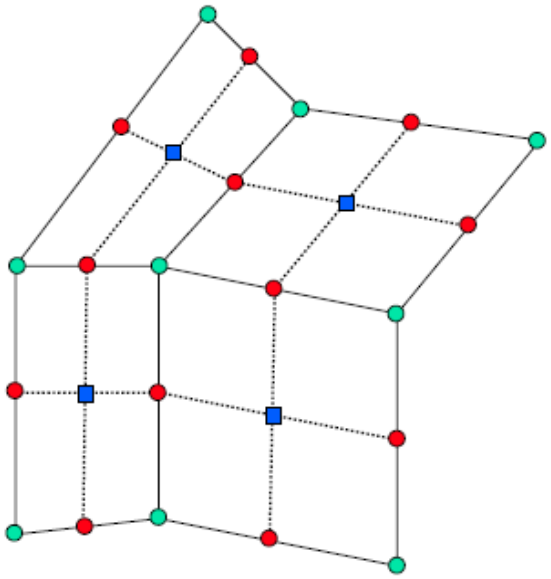


Abbildung 3.9: Viereckсmasche als Flächensplit [7]

• Das mehrfache Ausführen des Unterteilungsalgorithmus lässt eine glatte Fläche in der Grenze entstehen.

• Die Viereckmaschen werden jeweils in vier Maschen gespalten (Flächensplit).

• Bei dem Flächensplit wird folgende Vorgehensweise angewandt:

1. Einfügen eines Schwerpunkts in jede alte Viereckmasche.
2. Alle Kanten erhalten einen Mittelpunkt, der dann verlagert wird.
3. Verlagerung aller alten Ecken.

• Der Schwerpunkt s wird mittels der Koordinatenvektoren a,b,c,d der Eckpunkte einer Masche berechnet (Abbildung 3.10):

$$s = \frac{1}{4}(a+b+c+d)$$

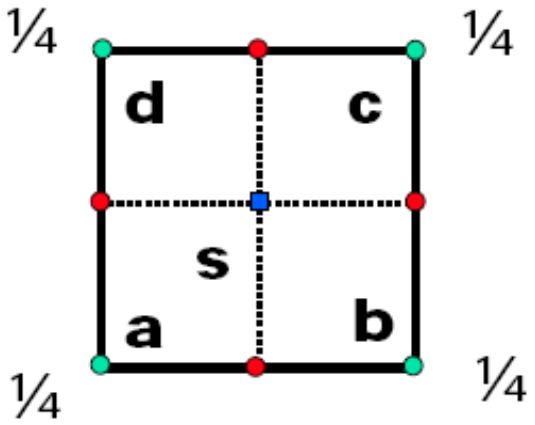


Abbildung 3.10: Berechnung des Schwerpunkts [7]

• Danach werden die eingefügten Mittelpunkte des alten Netzes verlagert, wobei sich die neue Lage gemäß der Abbildung 3.11 berechnen lässt:

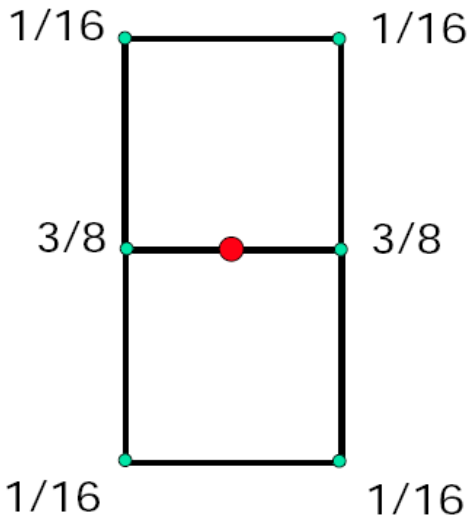


Abbildung 3.11: Schematische Darstellung [7]

• Das Gewichtsverhältnis ist 1:6, Summe der Gewichte ist 1.

• Berechnung der Verlagerung der alten Ecken, anhand der umliegenden acht Alten, entsprechend der Abbildung 3.12:

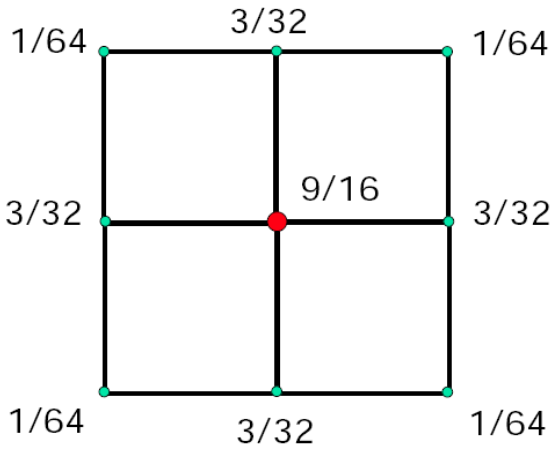


Abbildung 3.12: Schematische Darstellung [7]

• Gewichtsverhältnis ist 1:6:36, Summe der Gewichte ist 1.

• Verlagerung der irregulären alten Ecken gemäß des Schemas in Abbildung 3.13 (mit Valenz (Wertigkeit) n≠4):

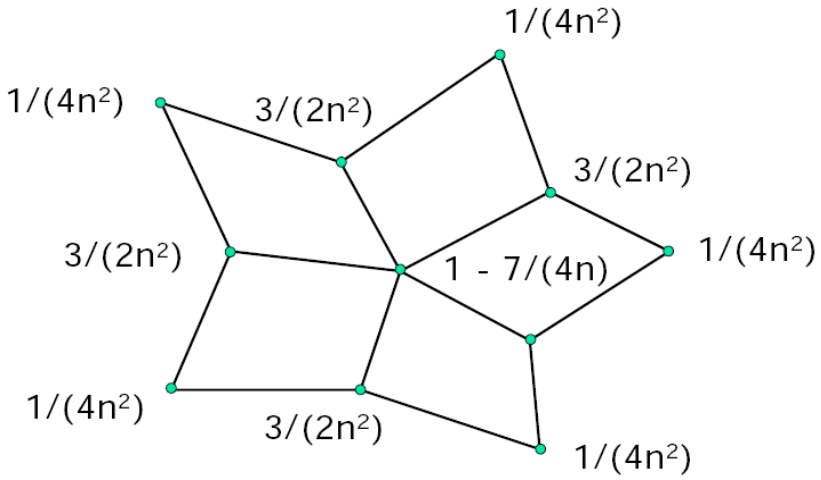


Abbildung 3.13: Schematische Darstellung [7]

• Somit kann jedes Netz im ersten Schritt durch das Einfügen der Kantenmitten und das dementsprechende Verbinden der Maschenschwerpunkte in ein Viereckснetz umgewandelt werden.

Die Abbildung 3.14 zeigt ein weiteres Beispiel einer fünffachen Catmull-Clark-Unterteilung.

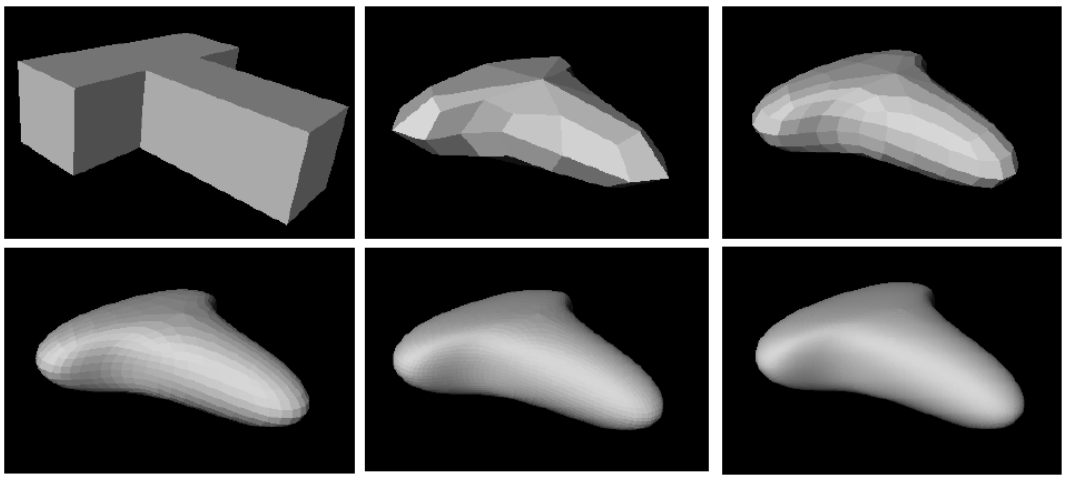


Abbildung 3.14: Beispiel Catmull-Clark Algorithmus [7]

Mit Hilfe dieser Technik und der Benutzung gängiger Mesh Modellierungswerkzeuge wurden die einzelnen Teile der Charaktere erstellt und zusammengefügt. Die Abbildung 3.15 zeigt einige Beispiele der Zusammenstellung von Charakteren und die Abbildung 3.16 verschiedene Accessoires und Kleidungsstücke.

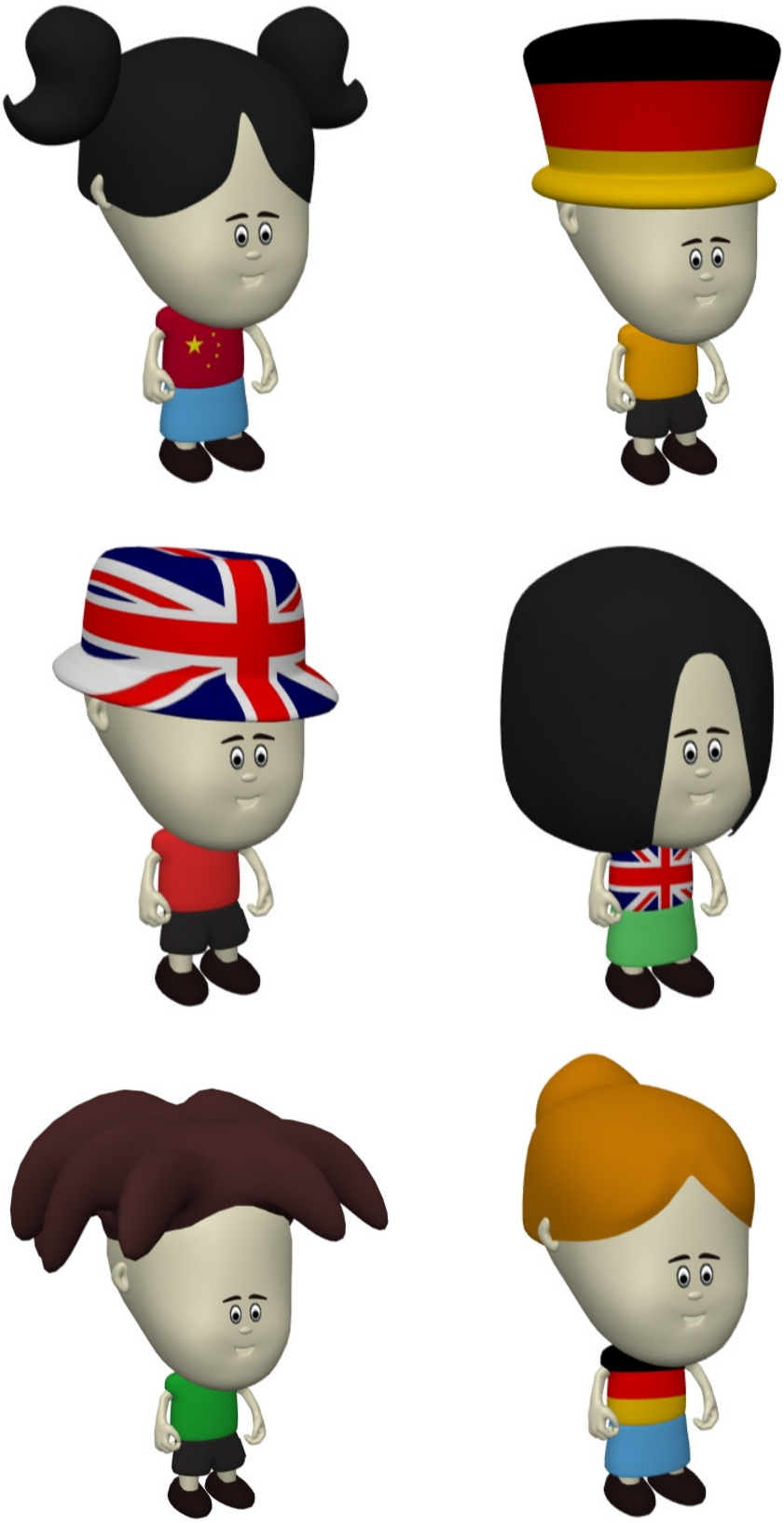
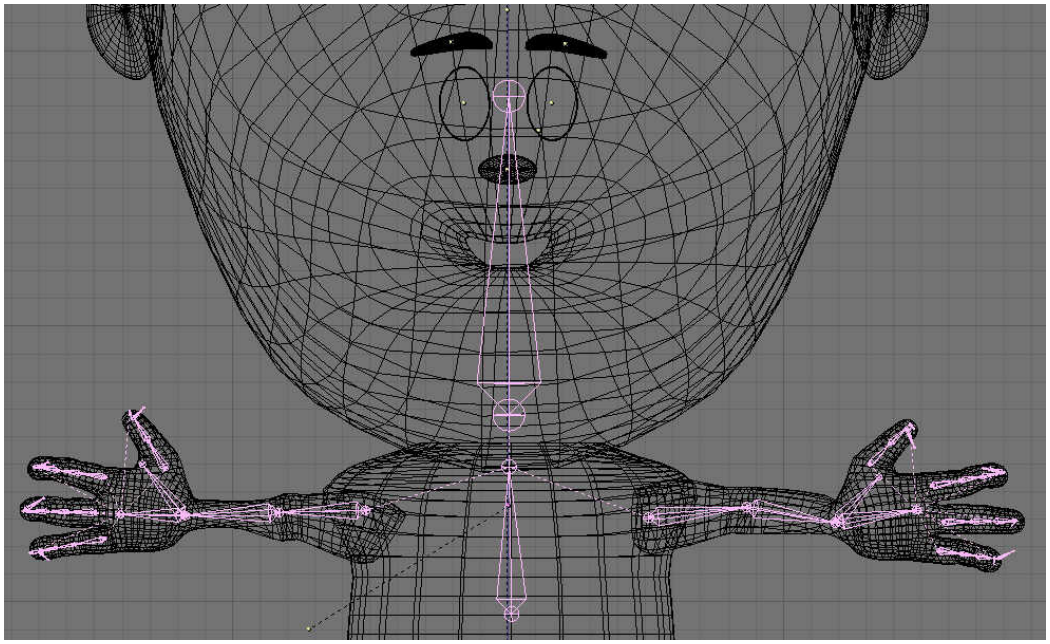


Abbildung 3.15: Beispiele fertiger Charaktere

Abbildung 3.16: Beispiele Accessoires und Kleidung

3.2.2 Armatur

Ein wichtiger Bestandteil des Charakters ist die Armatur, welche quasi das Skelett des Körpers darstellt. Diese wird in Blender, wie auch in anderen 3D-Programmen, durch die so genannten Bones (Knochen) gebildet, die dann zu einem komplexen Skelett zusammengefügt werden. Bei Servingo war es ausreichend, nur die Arme, Finger und den Kopf mit einer Armatur zu versehen, da nur diese Teile später animiert werden sollten. In Abbildung 3.17 sieht man die gesamte animierbare Armatur.



3.17: Armatur des Charakters

Hierbei war es äußerst wichtig, die einzelnen Bones von Anfang an richtig zu benennen, um die späteren kinematischen Einstellungen richtig vornehmen zu können. Diese werden dann durch die so genannten IK-Solver bewerkstelligt. IK-Solver sind Konstrukte, die an die vorhandenen Gerüste an Stellen, die als Gelenke funktionieren sollen, angehängt werden. So entstehen dann Haltepunkte, mit denen man zusammenhängende Bones auf einmal bewegen kann. Zuvor muss man diese Bones „parenten", d. h. zu Eltern machen, also einen Anfangsbone nehmen und den dann immer weiter an den nächsten Bone anhängen. Die Abbildung 3.18 veranschaulicht solche Vorgehensweise. Hier sieht man, dass der Bone ThumbNull.L ein Child of (also ein Kind) von ThumbB.L ist und das wiederum der IK-Solver IK_thumb.L ein Child of Hand.L ist, was bedeutet, dass mit diesem IK-Solver später der Finger (in diesem Fall der

Daumen) vom Handknochen aufwärts bewegt werden kann. Die Abbildung 3.19 zeigt einen Überblick der gesamten linken Hand.

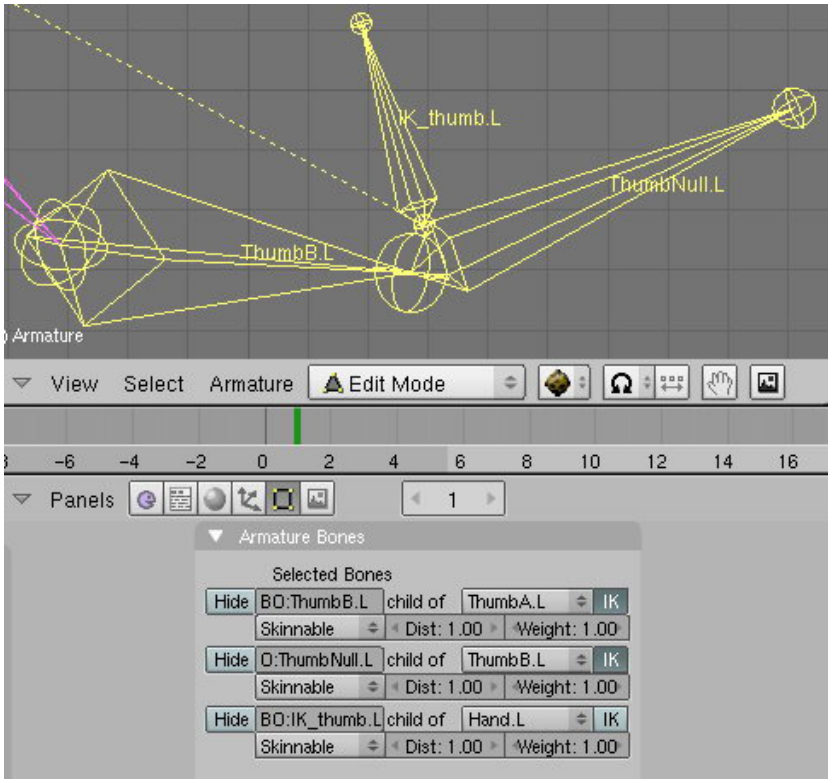


Abbildung 3.18: Zusammenhang der Bones

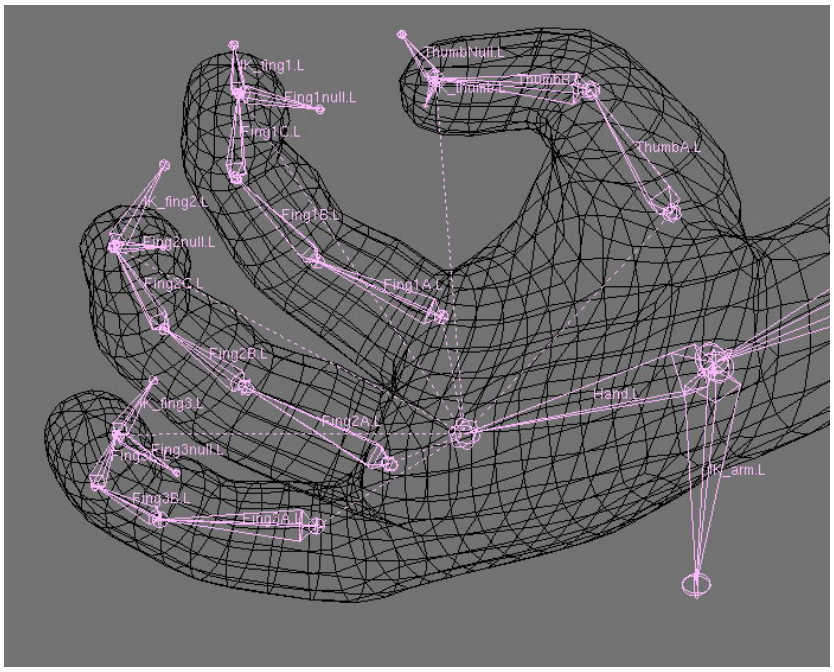


Abbildung 3.19: Bones der linken Hand

Nach der Verknüpfung der Bones müssen diese mit Constraints („Zwänge“) zu ihrer endgültigen Bewegung verbunden werden. Dazu wird der jeweilige IK-Solver an die Armatur gebunden und somit entsteht ein voll animierbarer Arm. In der Abbildung 3.20 ist die Einstellung für den Constraint zwischen IK_arm.L, Bone Hand.L und der Armatur zu sehen.

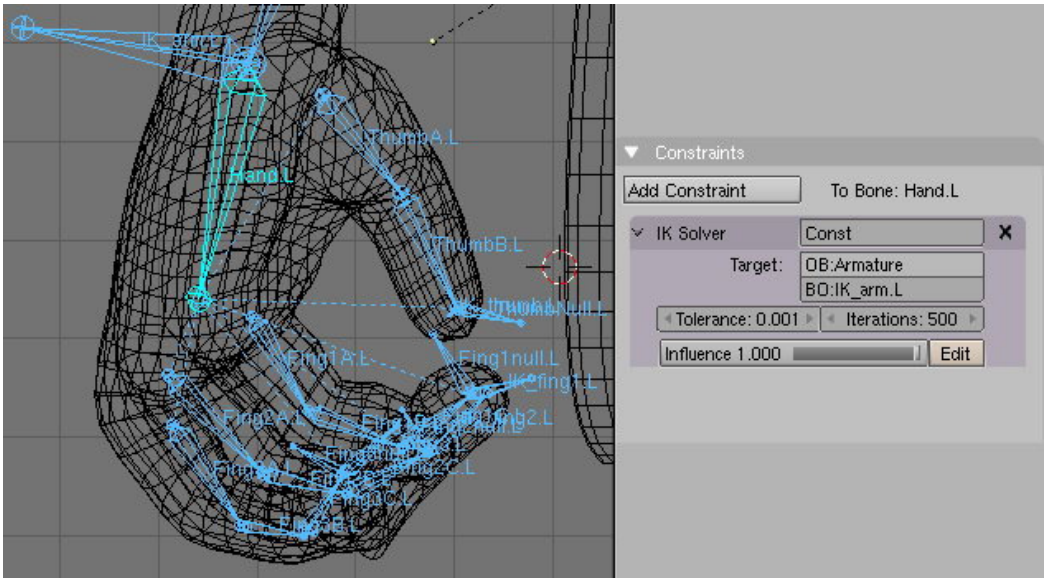


Abbildung 3.20: Beispiel Constraint IK_arm.L, Bone Hand.L und Armatur

Das so entstandene Gerüst wurde nun an den vorher erstellten Arm mittels Skinning angebunden. Skinning bedeutet, dass die modellierte Arm Mesh in Bereiche unterteilt wird in so genannte Vertex Groups, die sich dann entsprechend der Bewegung der Bones mit bewegen und mit deformieren sollen. Vertex Groups sind Punkte der Mesh, die zu den jeweiligen Objekten, die sie bewegen sollen (in diesem Fall Bones), zugeordnet werden. Diesen Vorgang kann man unter Blender auch automatisiert ablaufen lassen, wobei aber immer noch per Hand einzelne Punkte der Mesh, die sich dann in der Bewegung nicht korrekt mitbewegen, angepasst werden müssen. Die Abbildung 3.21 zeigt eine Übersicht aller Vertex Groups der gesamten Armatur und die Abbildung 3.22 und 3.23 ein Beispiel der Deformierung der Mesh bei Bildung einer Faust.

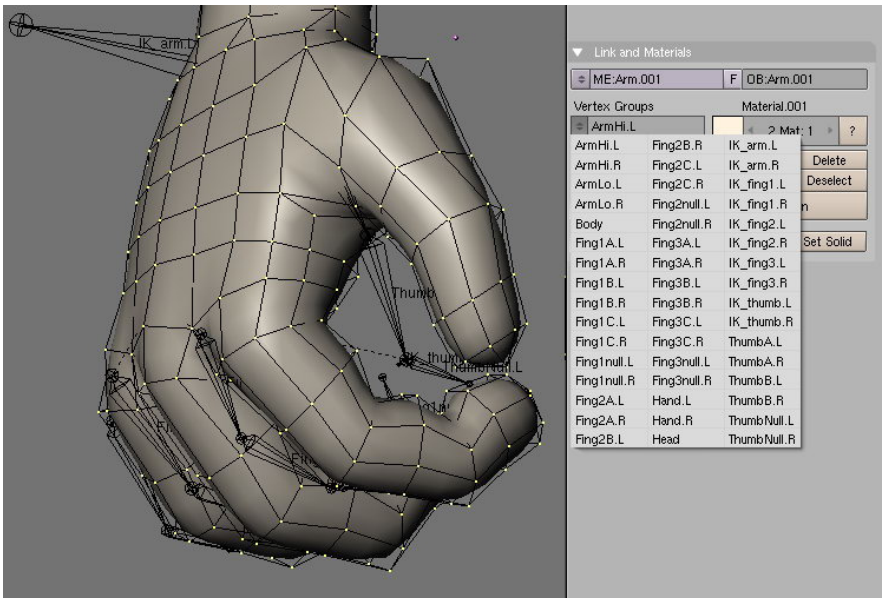


Abbildung 3.21: Übersicht Vertex Groups

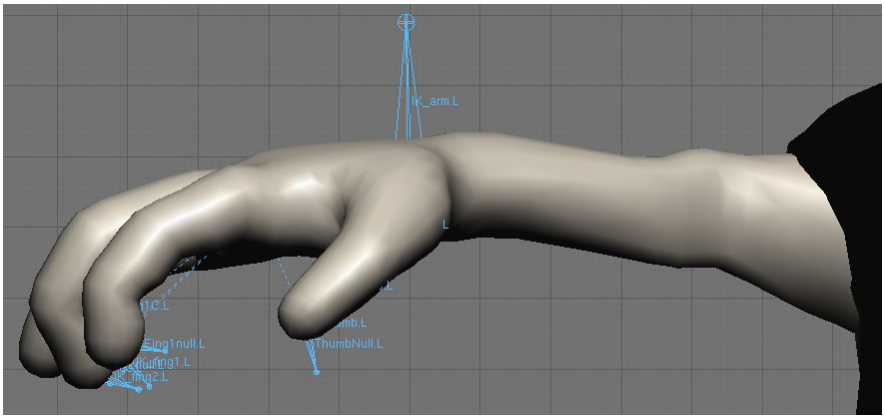


Abbildung 3.22: Beispiel Mesh Deformierung

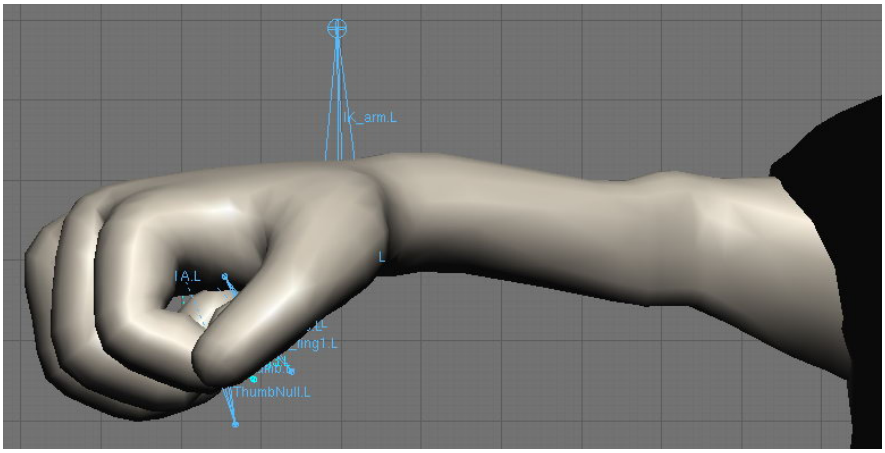
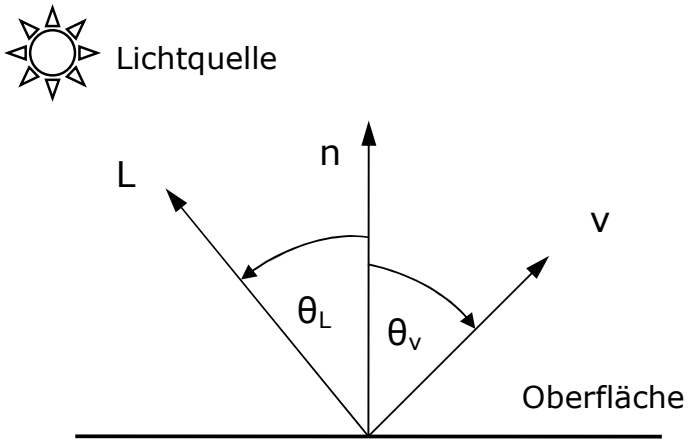


Abbildung 3.23: Beispiel Mesh Deformierung Faustbildung

3.2.3 Materialien und Texturen

Die Gestaltung der Figuren und der zugehörigen Materialien wurde anfangs auf Comic Art festgelegt, d. h. die Charaktere sollten wie Comic Figuren aussehen und auch die gesamten Materialien den Eindruck erwecken, dass es sich bei dem fertigen Film um einen Zeichentrickfilm handelt. Zu diesem Zweck wurde die Mehrheit der Materialien mit dem in Blender verfügbaren Toon Shader realisiert, wessen Funktionsweise später in diesem Kapitel erklärt wird.

In Blender werden Materialien und Licht, wie auch in den meisten 3D-Anwendungen, durch die Render Engine nach den gängigen Beleuchtungs- und Schattierungsmodellen erzeugt. Diese basieren auf den physikalischen Gesetzen der Strahlenoptik, die in der Abbildung 3.24 kurz vorgestellt werden. In dieser ist n der nach außen orientierte Normalenvektor, der die Lage des lokalen Koordinatensystems bestimmt. Der Lichtvektor L zeigt die Richtung des einfallenden Lichtstrahls an und der Sichtvektor v die Betrachtungsrichtung, in der ein Beobachter bzw. die Kamera auf den Oberflächenpunkt schaut. Die Beschaffung der Oberfläche und die Winkel der Vektoren sind dann für das Zustandekommen des sichtbaren Eindrucks verantwortlich.

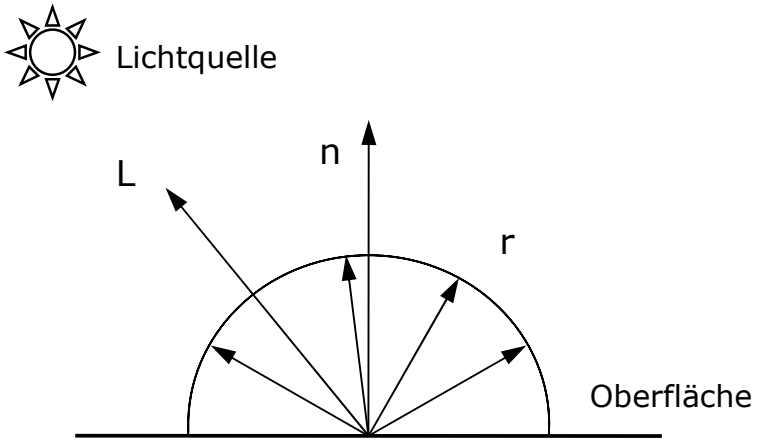


v = Betrachtungsrichtung
L = Lichtvektor
n = Normalenvektor
θ_l = der Winkel zwischen dem Normalenvektor und dem einfallenden Lichtvektor
θ_v = der Winkel zwischen dem Normalenvektor und dem Sichtvektor

Abbildung 3.24: Veranschaulichung der 3D-Darstellung

Beim Auftreffen eines Lichtstrahls auf die Oberfläche treten nun in jedem Punkt zwei physikalische Phänomene auf, die in der graphischen Datenverarbeitung eine bedeutende Rolle spielen und zwar eine diffuse sowie eine spiegelnde Reflexion.

Die ideale diffuse Reflexion entsteht bei der Streuung von Licht an matten Oberflächen, siehe Abbildung 3.25, wobei der reflektierte Anteil des Lichts in einem Oberflächenpunkt in alle Richtungen gleich ist. Diese physikalische Grundlage ist im Lambertschen Gesetz für Reflektion an matten Oberflächen definiert.



r = Reflexionsrichtung
L = Lichtvektor
n = Normalenvektor

Abbildung 3.25: Veranschaulichung der idealen diffusen Reflexion

Dieses Gesetz besagt, dass die von einer Lichtquelle auf eine Oberfläche A einfallende Helligkeit vom Winkel der Oberfläche zur Lichtquelle abhängig ist und die Helligkeit auf der Fläche A proportional zum Inhalt der projizierten Fläche Ap ist. Die Abbildung 3.26 bietet einen genauen Überblick über dieses Gesetz.

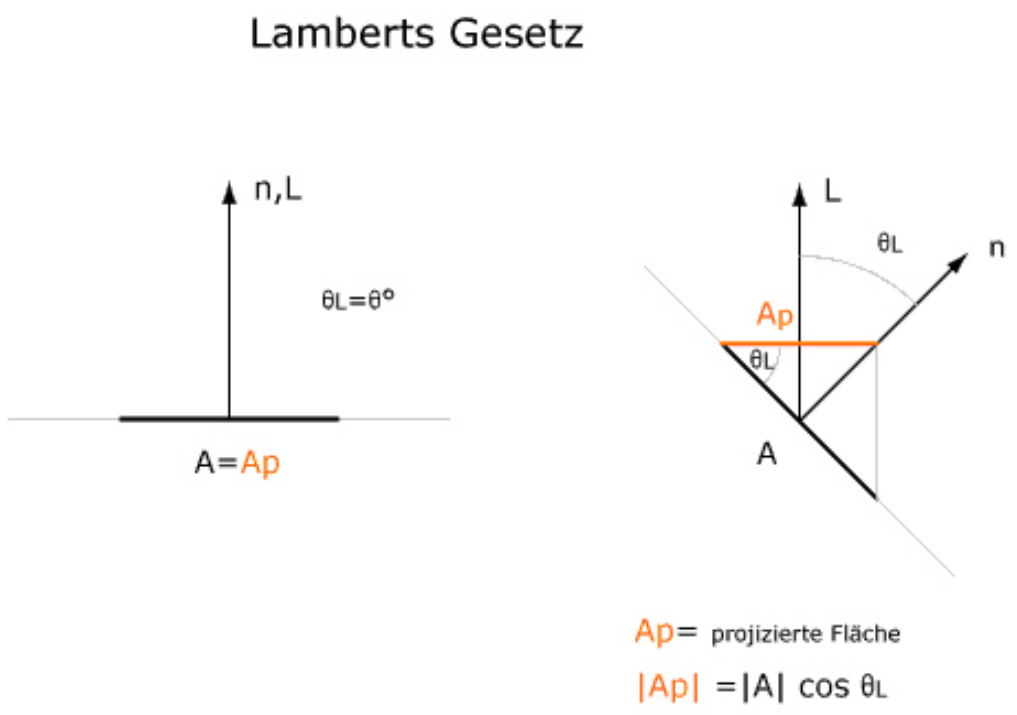


Abbildung 3.26: Veranschaulichung des Lamberts Gesetz [8]

Aus den Erkenntnissen dieses Gesetzes wurde das Lambert-Beleuchtungsmodell definiert, welches die Beleuchtung von ideal diffus reflektierenden Oberflächen simuliert. Wie bereits erwähnt, ist hierbei der reflektierte Anteil des Lichts in einem Oberflächenpunkt in alle Richtungen gleich und außerdem spielt die Orientierung der Oberfläche gegenüber der Betrachtungsrichtung v keine Rolle. Bedeutend ist nur die existierende Abhängigkeit vom Winkel θ_L zwischen dem Lichtvektor L, dem Normalenvektor und der Intensität des diffus reflektierten Lichts Id.

Die reflektierte Intensität Id einer Fläche ist proportional zur projizierten Fläche Ap, wobei eine maximale diffuse Reflexion bei einem Einfallswinkel von 0° und keine Reflexion bei einer Winkelstellung über 90° stattfindet. Bei genau 90° leuchtet die Lichtquelle an der Oberfläche entlang. Daraus resultiert die folgende Formel für die reflektierte Intensität Id nach dem Lambertschen Beleuchtungsmodell (Abbildung 3.27):

$$Id = IL * rd * \max(0, <L, n>)$$

Hierbei beschreibt Id die reflektierte Intensität, IL die Intensität des Lichtvektors und rd den diffusen Reflexionskoeffizienten, mit welchem sich der Anteil des zu reflektierenden Lichts und somit die Objektfarbe einstellen lassen kann. Der Parameter <L,n> ist das Skalarprodukt des Lichtvektors (L) mit dem Normalvektor (n), wobei, da mit Einheitsvektoren gearbeitet wird, gilt : cos θL = <L,n> und für max (0, <L,n>), da nur Winkeleinstellungen zwischen -90° bis 90° eine Intensität liefern, muss ein positives Skalarprodukt verwendet werden. Einen detaillierten Überblick bietet die Abbildung 3.27.

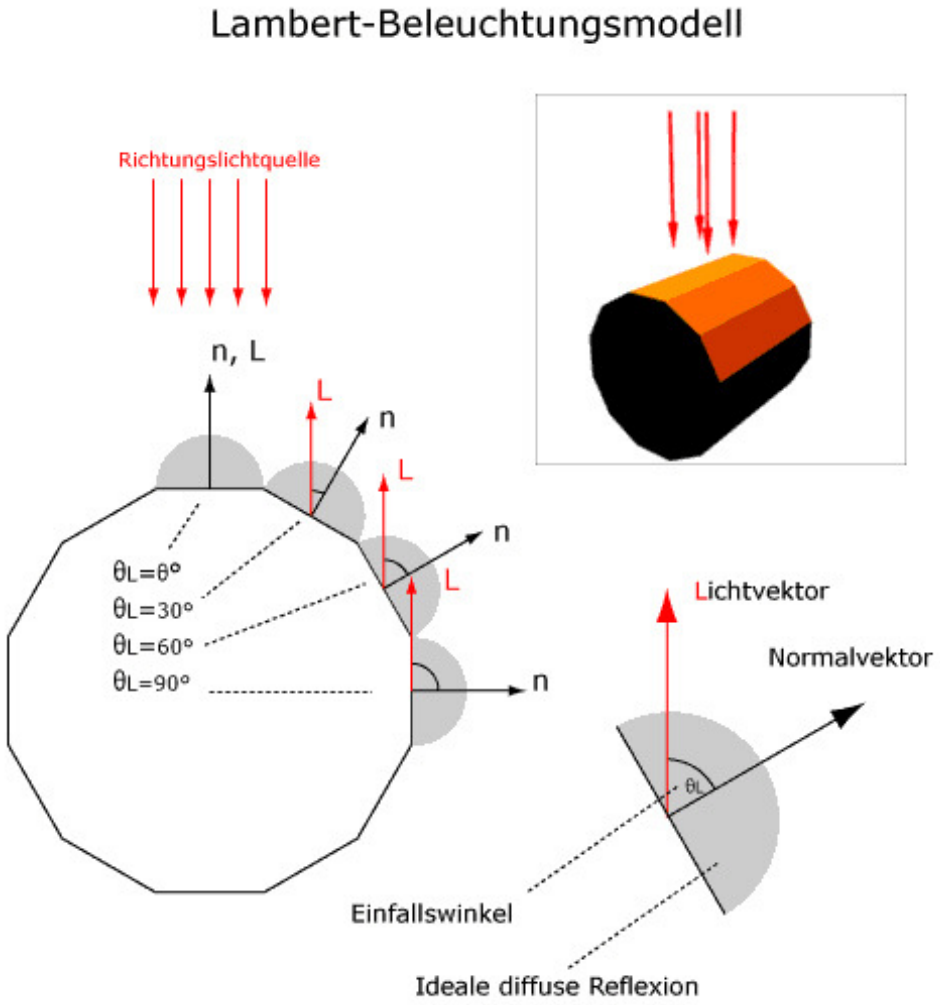


Abbildung 3.27: Veranschaulichung des Lambert Beleuchtungsmodells [8]

Seit der Version 2.28 verfügt Blender über drei verschiedene mathematische Formeln, um Diffusion berechnen zu können. Außerdem wurden die Einstellungen für die diffuse und spiegelnde Reflektion, die normalerweise in einer Materialdefinition zusammengebunden sind, aufgespalten, um die jeweiligen Parameter getrennt einstellen zu können. Die drei Implementierungen der Diffusionsberechnung (oder auch Shader genannt) benutzen jeweils zwei oder mehr Parameter, wobei die ersten zwei von allen diffusen Shadern benutzt werden und der erste die Farbe des Materials und der zweite die Menge des vorkommenden diffundierten Lichts angibt.

Die implementierten Shader sind (Abbildung 3.28):

• Lambert: Dieser war der Standard diffuser Shader von Blender bis Version 2.27 und besitzt nur einen Reflektionsparameter.

• Oren-Nayar: Dieser Shader, den es seit der Version 2.28 gibt, ermöglicht eine physikalischere Darstellung des Phänomens der Diffusion, da er noch einen zusätzlichen Parameter für die mikroskopische Rauheit der Oberfläche bietet.

• Toon: Dieser Shader kam auch zuerst in der Version 2.28 als ein sehr unphysikalischer Shader dazu und ermöglicht das Rendern comicartiger Oberflächen mit klaren Licht-Schatten Grenzen und einheitlich beleuchteten und schattierten Flächen. Desto trotz ist er relativ einfach und hat zusätzlich zwei Parameter, welche die Größe der beleuchteten Fläche und die Schärfe der Schattengrenzen festlegt.

• Minnaert: Dieser Shader ist erst ab der Version 2.37 verfügbar und dient vor allem zur Simulation von Kleidung, wie z. B. Samt. Er verfügt über einen „Darken" Parameter, mit welchem man Oberflächen aus bestimmter Blickrichtung verdunkeln kann.

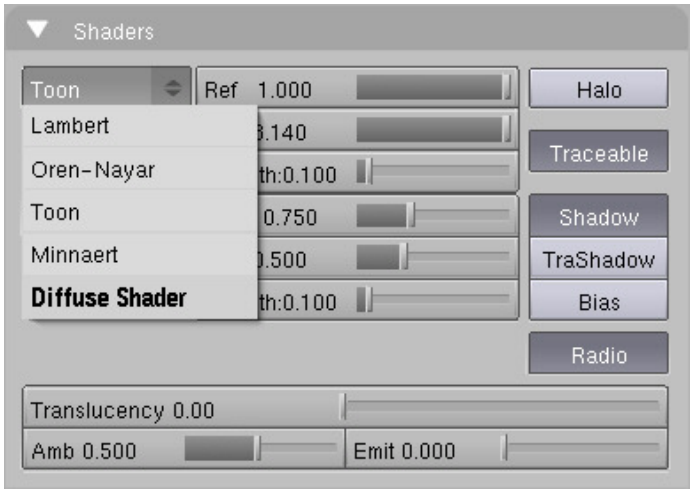
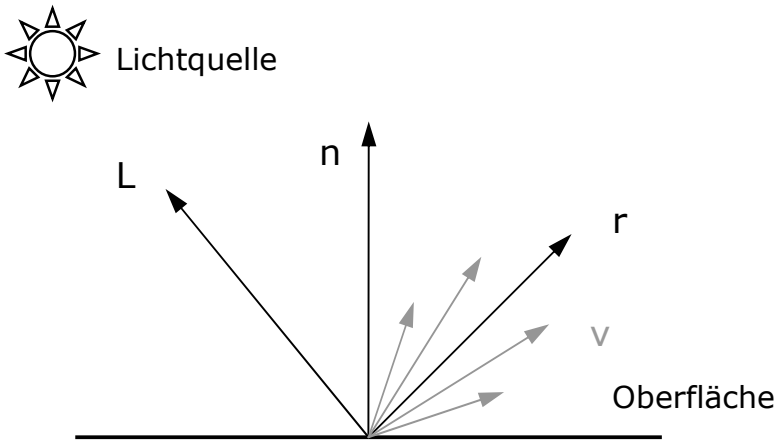


Abbildung 3.28: Blender Auswahl diffuser Shader

Die spiegelnde Reflexion (Abbildung 3.29) ist im Unterschied zur diffusen, vom Betrachterstandpunkt abhängig, da glänzende Flächen das Licht nicht gleichmäßig reflektieren und sie erzeugt so genannte Highlights (Glanzpunkte), die in Abbildung 3.30 demonstriert werden.



v = Betrachtungsrichtung
r = Reflexionsrichtung
L = Lichtvektor
n = Normalenvektor

Abbildung 3.29: Veranschaulichung der realen spiegelnden Reflektion

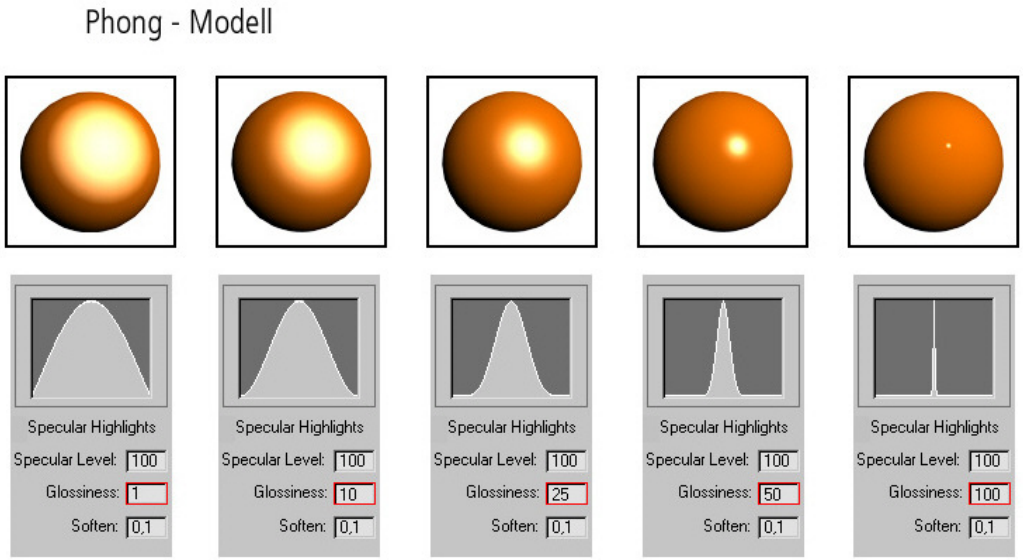
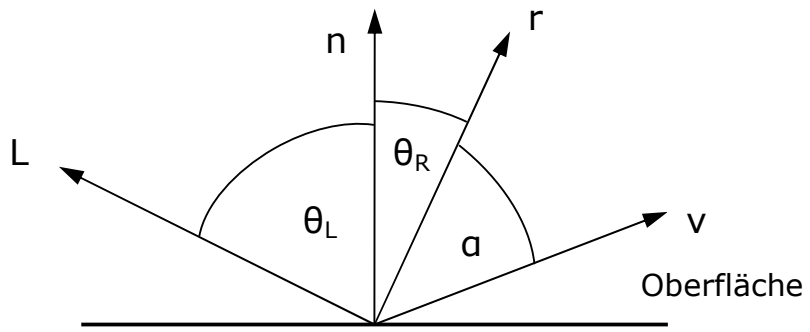


Abbildung 3.30: Beispiel Glanzlichter [8]

Um eine realistische Beleuchtung und Schattierung in der Computergrafik zu ermöglichen, wird eine Mischung aus diffusen und spiegelnden Shadern benutzt, welche in dem so genannten Phong-Beleuchtungsmodell beschrieben sind.

Phong Bui Tuong entwickelte 1975 das Phong-Beleuchtungsmodell, wodurch das Lambertsche Beleuchtungsmodell um eine reale spekulare, also spiegelnde Reflexion ergänzt wurde. Beim Phong-Modell ist die Blickrichtung r sehr bedeutsam, da die Größe des Winkels α, der zwischen der Betrachtungsrichtung v und der idealen Reflexionsrichtung r liegt, über die Lichtintensität des spiegelnden Lichts in Blickrichtung entscheidet. Dabei gilt, je größer α ist, desto kleiner ist der Anteil des spekularen Lichts.

Beim Phong-Beleuchtungsmodell wird die Abhängigkeit des Abfalls der reflektierten Intensität von der Abweichung α der Betrachtungsrichtung von der idealen Reflexionsrichtung durch eine Potenz von cos α dargestellt. Hierzu wird der Cosinus des Winkels mit Hilfe des Skalarprodukts zwischen r und v gebildet, wobei Winkelstellungen über 90° ignoriert werden. Einen Überblick bietet die Abbildung 3.31.



v = Betrachtungsrichtung
r = Reflexionsrichtung
L = Lichtvektor
n = Normalenvektor
θ_L = der Winkel zwischen dem Normalenvektor und dem einfallenden Lichtvektor
θ_R = der Winkel zwischen dem Normalenvektor und der Reflexionsrichtung
α = Winkel zwischen der Betrachtungsrichtung und der Reflexionsrichtung

Abbildung 3.31: Phong-Beleuchtungsmodell

Die Formel für die reflektierte Intensität Is nach dem Phong-Beleuchtungsmodell lautet:

$$Is = IL * rs * \max(0, <r, v>^{n}),$$

hierbei ist Is die Intensität des spiegelnd reflektierten Lichts in Richtung v, IL die Intensität des von der Lichtquelle einfallenden Lichts, n der Spiegelungsexponent und rs der spiegelnde Reflexionskoeffizient.

Einen Vergleich zwischen dem Lambert-Beleuchtungsmodell und dem Phong-Beleuchtungsmodell, welchen die Abbildung 3.32 darstellt, zeigt die Auswirkung der Blickrichtung bei Phong.

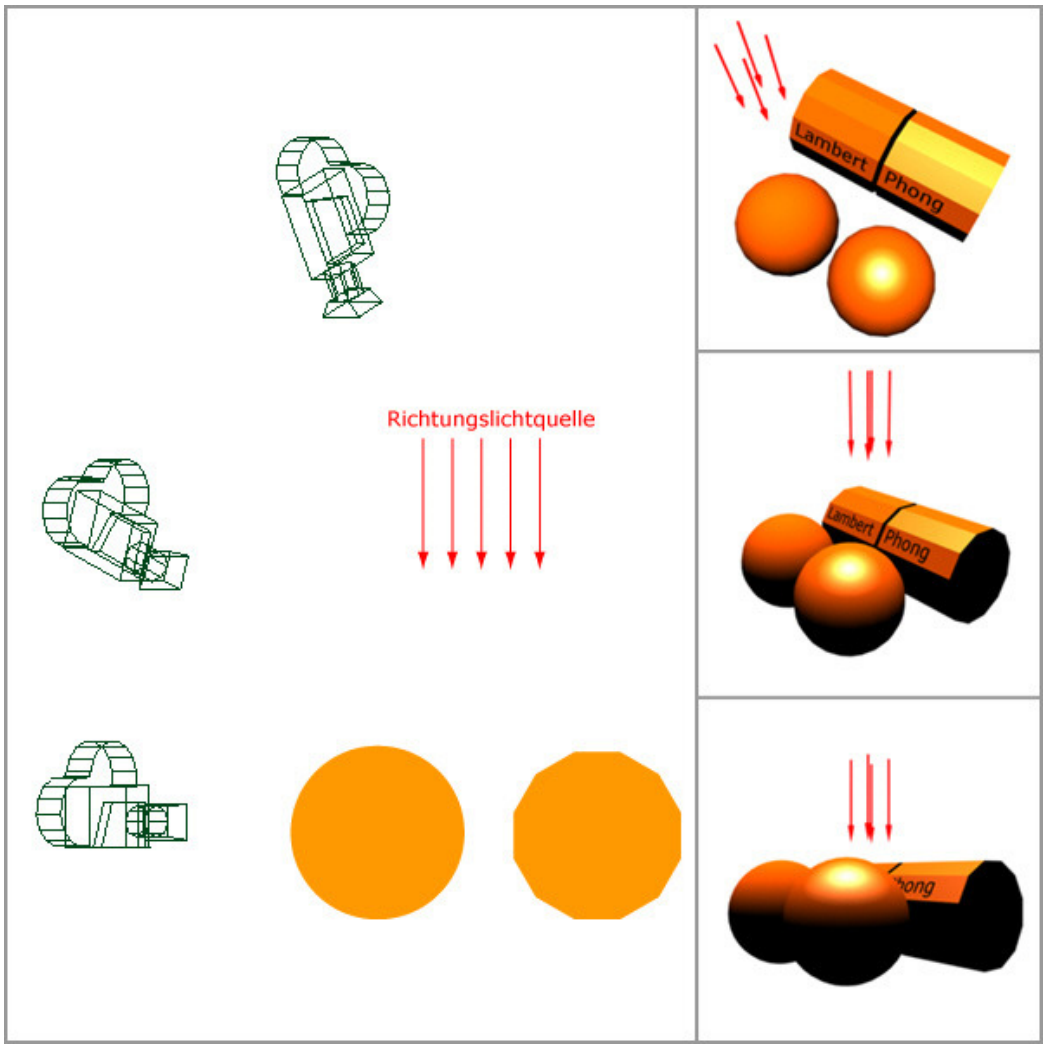


Abbildung 3.32: Vergleich zwischen Lambert und Phong [8]

In Blender wird die spiegelnde Reflektion durch eine Reihe von spiegelnden Shadern realisiert, die alle, wie auch die diffusen, zwei gleiche Parameter besitzen und zwar die Specular Color (Farbe des spiegelnden Lichts) und Specularity (Spiegelung), die die Stärke der Spiegelung angibt.

Die implementierten Shader sind:

• CookTorr: Dieser Shader implementiert das so genannte Cook-Torrance Lighting Model, welches eine sehr gute, physikalisch richtige Darstellung von Oberflächen ermöglicht. Er hat einen dritten Parameter für Hardness (Härte), der die Breite der Glanzlichter festlegt, wobei je geringer die Härte ist, desto breiter die Glanzlichter dargestellt werden.

• Phong: Dies ist ein anderer mathematischer Algorithmus, der zur Berechnung von Glanzlichtern benutzt wird, aber dem Cook-Torrance sehr ähnlich ist.

• Blinn: Dies ist ein mehr physikalischer spiegelnder Shader, der dem diffusen Oren-Nayar entsprechen soll. Er besitzt zusätzlich einen vierten Parameter und zwar den Index of Refraction (Index der Lichtbrechung), der zur korrekten Berechnung der Reflexionsintensität und Ausdehnung dient.

• Toon: Dieser spiegelnde Shader entspricht dem diffusen und wird benutzt, um scharfe gleichmäßige Glanzlichter bei Zeichentrick Oberflächen zu erzeugen. Er besitzt keinen Hardness Parameter, dafür aber einen Size (Größe) und Smooth (Glätte), welche die Ausdehnung und Schärfe der Glanzlichter bestimmen.

• WardIso: Dieser Shader ist erst ab der Version 2.37 implementiert und ermöglicht die physikalisch korrekte Darstellung anisotroper (richtungsabhängiger) Oberflächen, wie z. B. gebürstetes Metall oder Haare.

Dank der flexiblen Implementierung, welche die diffuse und spiegelnde Reflektion auseinander hält, erlaubt es Blender die Menge an diffudiertem, gespiegeltem und absorbiertem Licht an einem Punkt der Oberfläche genau einzustellen. Eine kleine Auswahl an möglichen Mischungen der einzelnen Shader Gruppen sieht man in der Abbildung 3.33.

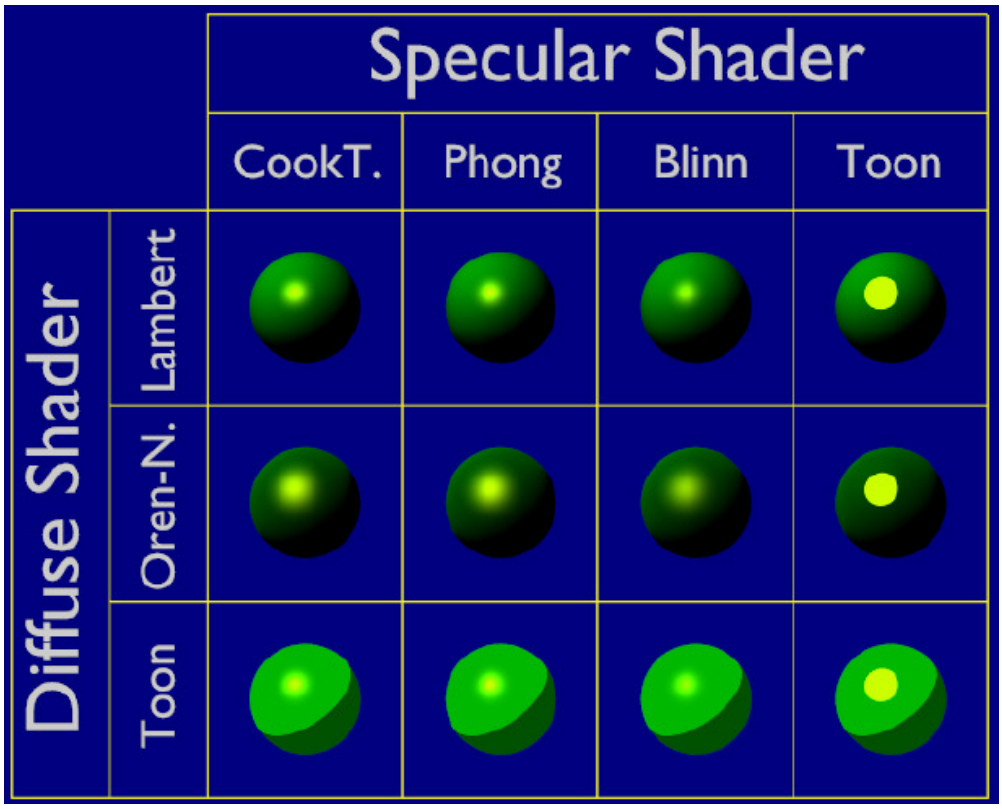


Abbildung 3.33: Überblick Shader Kombinationen [9]

Wie schon eingangs erwähnt, wurden die meisten Oberflächen des Servingo-Projekts mit Hilfe der Toon Shader gestaltet, um ein Zeichentrickfilm ähnliches Aussehen der Objekte und vor allem der Charaktere zu erhalten. Hierbei wurden mehrere Methoden und verschiedenste Einstellungen zur Herstellung des erwünschten Effekts ausprobiert, wobei sich zum Schluss die in Abbildung 3.34 (hier am Beispiel der Gesichtsfarbe) eingestellte Parameter Kombination als die passende heraus kristallisierte.

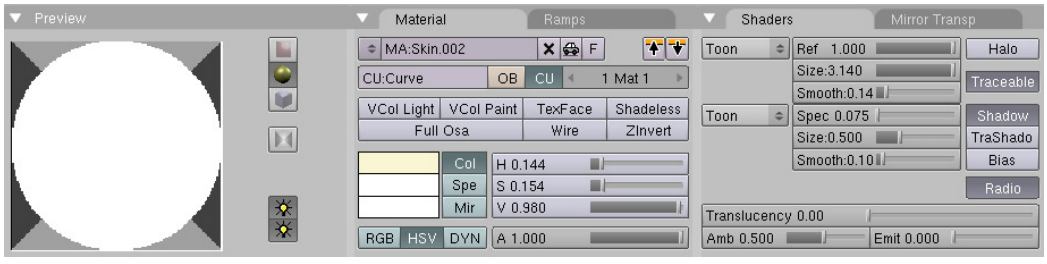


Abbildung 3.34: Überblick Shader Kombinationen

Diese Einstellung erzeugt ein Comic Material mit einem breiten Schatten, welcher durch eine Lichtquelle realisiert wird. Es besitzt keinen diffusen Schatten, anstatt dessen fungiert der spiegelnde Shader als der diffuse und der diffuse an sich dient als Schatten. Hierzu wird der diffuse Anteil durch die Einstellung des Shaders auf den maximalen Wert von 3.14 eingestellt und somit eliminiert. Außerdem wird der Smooth Wert auf Null gesetzt, um die Oberfläche nicht zu glatt zu erscheinen lassen und die Größe des spiegelnden Shaders auf 0.5 eingestellt, um den gewünschten Effekt zu erhalten. Eine genaue Übersicht über die Werte stellt die Abbildung 3.35 dar.

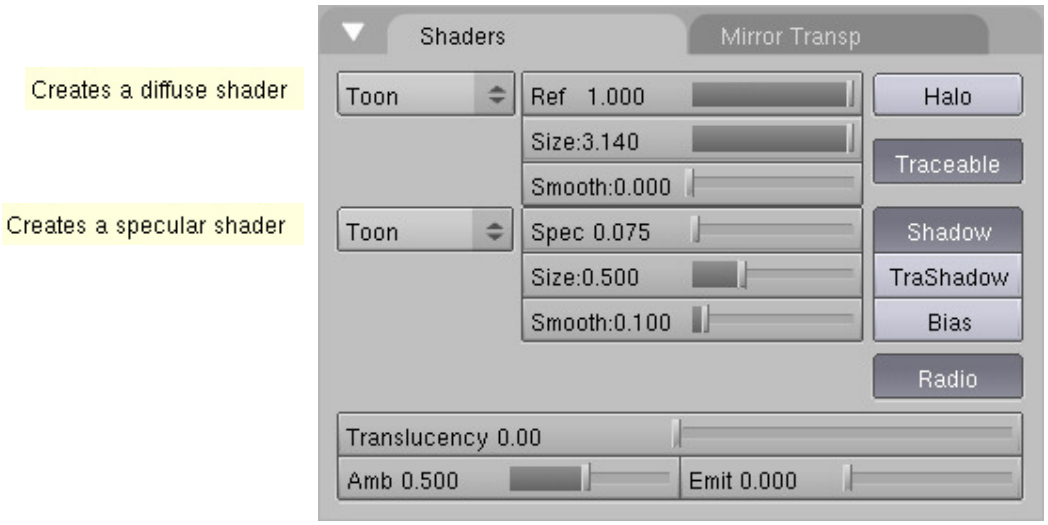


Abbildung 3.35: Überblick Toon Shader Einstellungen

Bei dieser Einstellung werden nun die Schatten durch die Wahl der Farbe des Materials kontrolliert und die Farbe des spiegelnden Lichts beeinflusst nun die diffusen Shader. Hierbei können durch die Herabsetzung der Helligkeit des Materials, die Schatten dunkler und durch Verminderung der Größe der spiegelnden Komponente, die Oberfläche glänzender gemacht werden. In der Abbildung 3.36 sieht man einen fertig gerenderten Charakter unter Benutzung dieser beschriebenen Einstellungen.

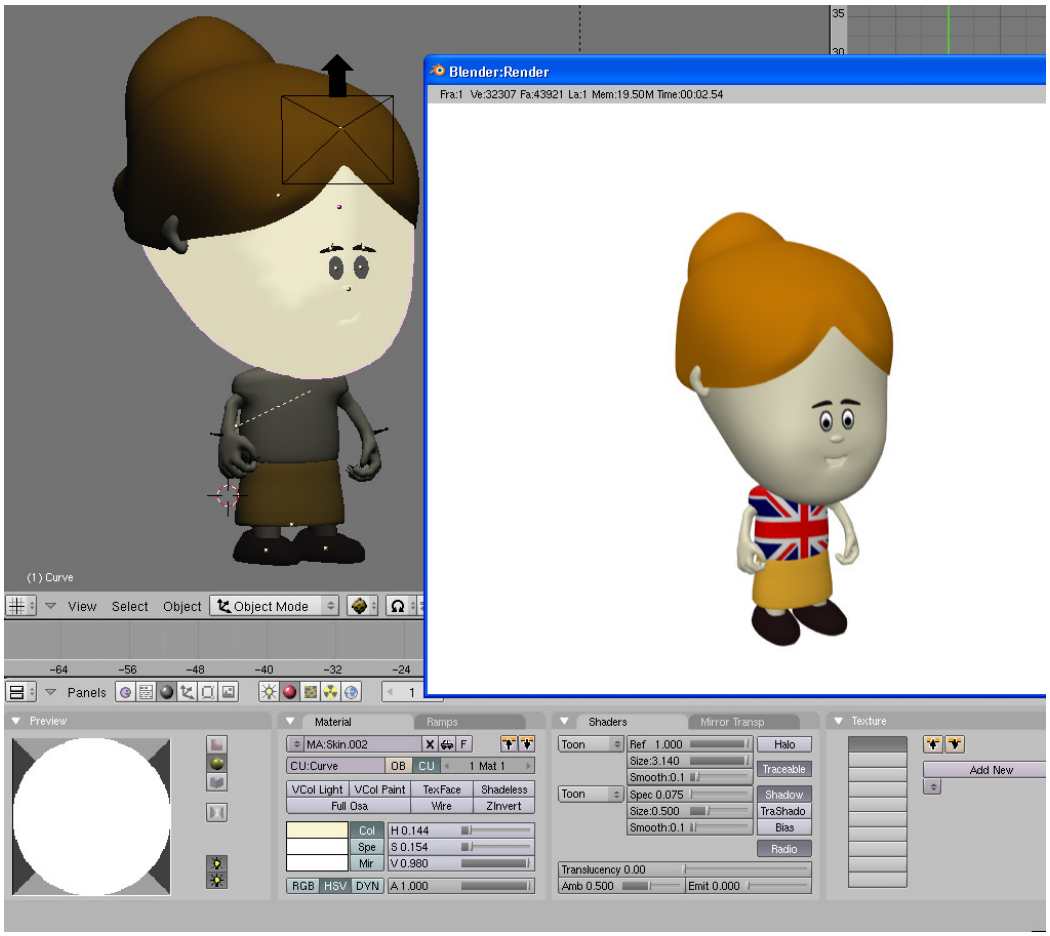


Abbildung 3.36: gerenderter Charakter

Das Servingo-Projekt soll für die Fußball-WM 2006 in drei Sprachen verfügbar sein und zwar Deutsch, Englisch und Chinesisch. Um einen Bezug zu diesen Ländern herzustellen und dem Benutzer eine weitere Möglichkeit der individuellen Gestaltung des Charakters zu ermöglichen, wurden die Texturen der Nationalflaggen dieser Länder als Auswahlmöglichkeit realisiert. In Blender werden Texturen, ähnlich wie in anderen 3D-Programmen, über das Material eines Objekts gelegt und mittels der „Texture Buttons" (Textur Schaltflächen) angepasst. Hierbei war es besonders wichtig, die Bilddateien der Flaggen mittels des so genannten Map Inputs in Ihrer Größe und Lage an die Oberkörper Armatur oder die einzelnen Hüte anzupassen. Dies geschieht durch das Verändern der ofsX und ofsY Parameter für die Lage der Textur und der sizeX und sizeY Werte für die jeweilige Größe der Textur auf dem Objekt. Die Abbildungen 3.37 und 3.38 zeigen die Auswirkungen dieser Anpassung und die Abbildung 3.39 gibt einen Überblick über die Texture Buttons.

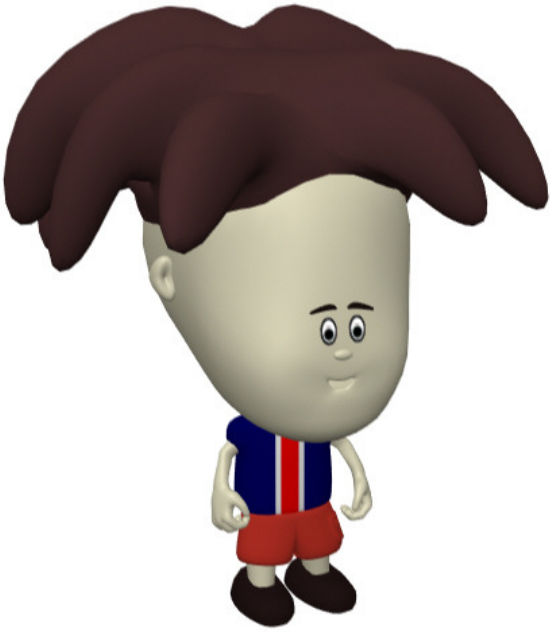


Abbildung 3.37: Textur ohne Anpassung

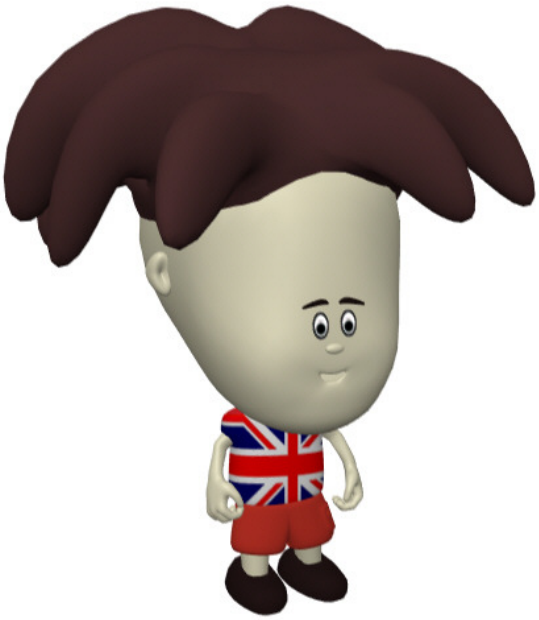


Abbildung 3.38: angepasste Textur

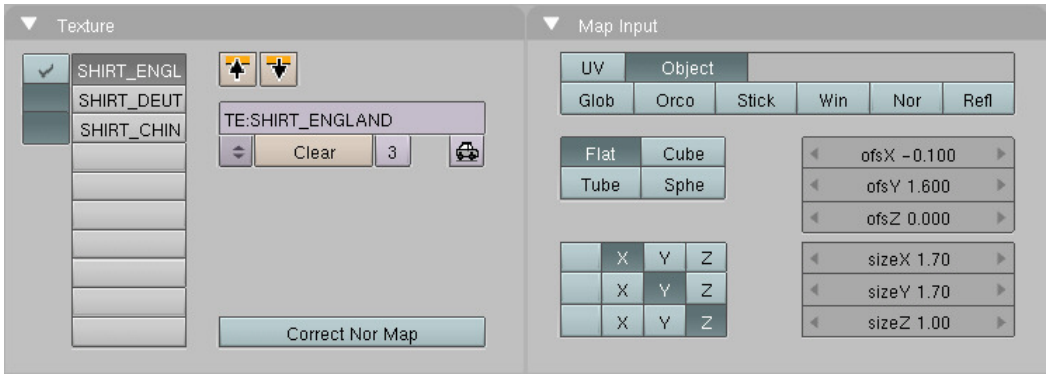


Abbildung 3.39: Texture Buttons mit gesetzten Parametern

Die drei so angepassten Texturen wurden dann mit der Armatur verbunden, um sie später automatisch über das Python Skript setzen zu können. Mit Hilfe dieser Technik ist es später relativ einfach möglich, neue Texturen, wie z. B. aller teilnehmenden Nationen der Fußball-WM, hinzuzufügen und somit Servingo beliebig zu erweitern (Abbildung 3.40).

Abbildung 3.40: Beispiele von Haartexturen

3.2.4 Aufbau der Szene

Die eigentliche Szene wurde mit einer Hemi-Lichtquelle realisiert, um die gewünschten Materialeffekte zu erhalten. Diese Lichtquelle simuliert eine Hintergrundbeleuchtung, ähnlich dem Himmel und liefert eine gleichmäßige Ausleuchtung von einer die Szene umgebenden Halbkugel. Dabei ist die Position der Hemi-Lichtquelle unwichtig, wichtig ist nur ihre Orientierung, wobei die gestrichelte Linie die Richtung, in der die maximale Energie abgegeben wird, darstellt. Die Abbildung 3.41 zeigt die Lichtsituation in der Szene.

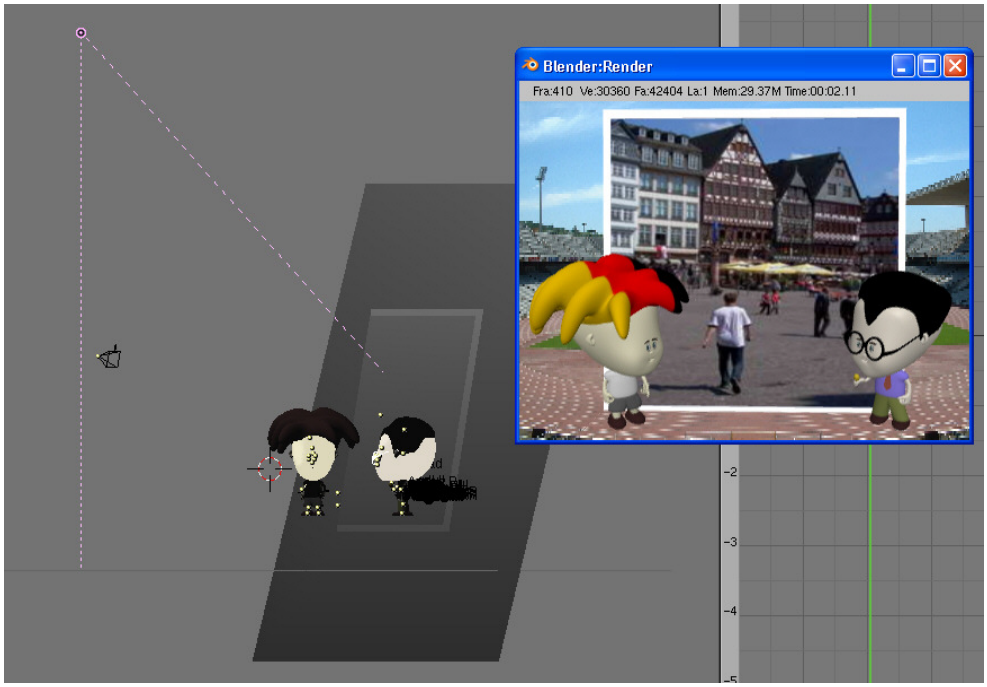


Abbildung 3.41: Lichtsituation

Um die spätere individuelle Gestaltung des Charakters mittels der Ansteuerung über die XML und des Python Skripts zu ermöglichen, wurden die einzelnen Frisuren und Kleidungsstücke auf verschiedene Layer (Schichten) gelegt. Blender bietet mit seinem Layer-System eine sehr effektive Methode an, Objekte im Verlauf der Animation sichtbar oder unsichtbar zu machen. Außerdem lassen sich die einzelnen Layer, sehr bequem mit Hilfe von Python setzen, womit es sehr schnell möglich ist, das Geschehen in einer 3D-Szene auch aus externen Programmen oder Prozessen zu steuern. Die Hauptszene wurde auf den Layer 1 gesetzt und die Frisuren anhand der Vorgaben des Python Skripts auf die Layer 2 bis 7 und die Kleidungsstücke auf 8 und 9 und die Wand mit der wechselnden Bildtextur auf den Layer 10 gelegt.

Somit entstand eine dynamisch veränderbare Szene, die durch das Verändern weniger Parameter, eine völlig andere Gestalt annehmen kann. Die Abbildungen 3.42 (Layer 1,3,8,10) und 3.43 (Layer 1,6,9) zeigen das Ergebnis des Setzens der verschiedenen Layer.

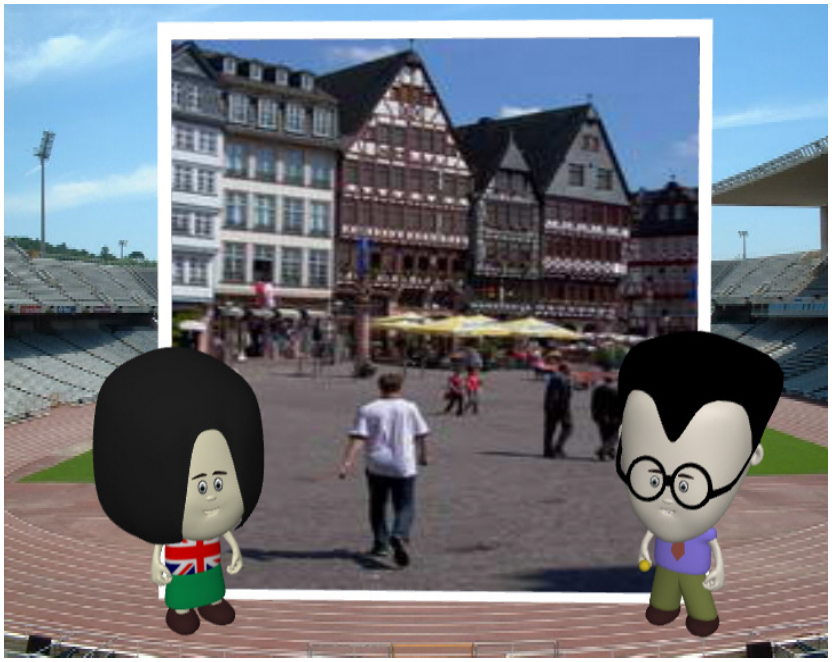


Abbildung 3.42: Layer (1,3,8,10) gesetzt

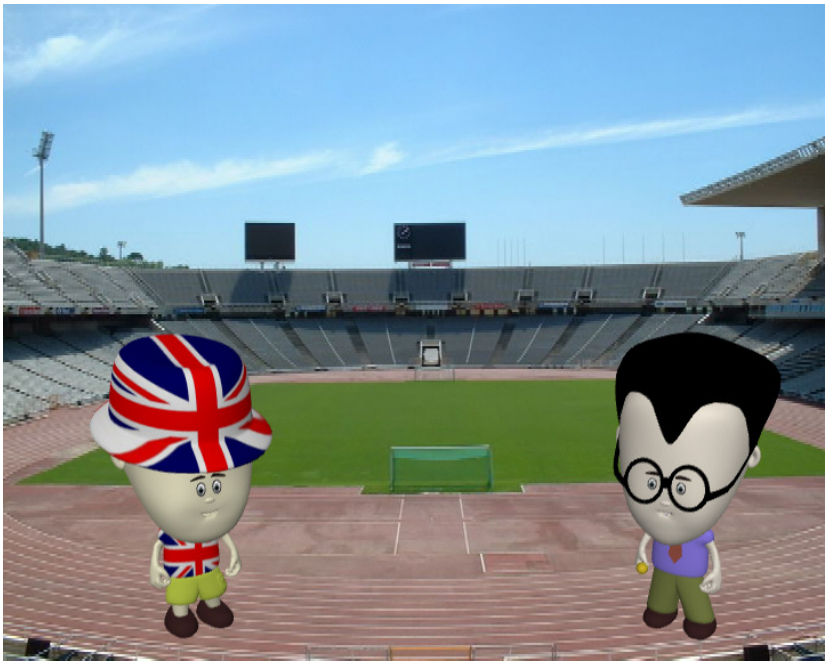


Abbildung 3.43: Layer (1,6,9) gesetzt

Die Gestaltung der Farben und der Texturen wird alleine durch die Vorgaben im Python Skript gesteuert, welches in einem späteren Kapitel näher erläutert wird. Als ein weiterer Vorteil des Layer-Systems, erwies sich die Möglichkeit, die einzelnen Layer auch mittels Keyframes zu animieren und somit eine Steuerung während der Animation zu ermöglichen. Die genaue Beschreibung des Animationsprinzips in Blender wird im Kapitel 3.2.5 erklärt. In der Servingo Szene wurde hiermit das Erscheinen der Bildwand erst ab einem bestimmten Zeitpunkt realisiert. Dazu wurde bei Frame 1 die Bildwand auf dem Layer 10 mit einem Key versehen (somit lag sie auf Layer 10) und erst später (ab ca. Frame 230) auf den Layer 1 gelegt und wieder mit einem Key versehen, wodurch sie erst ab Frame 230 in der Szene sichtbar wird.

Ein weiterer wichtiger Aspekt von Blender, der beim Servingo-Projekt eine tragende Rolle gespielt hat, ist die Fähigkeit in einer 3D-Szene mehrere Szenen anzulegen. Dadurch ist es möglich, später verschiedene Szenen miteinander zu mischen und beliebig zusammen zu setzen und dies, ohne neue Dateien anzulegen oder öffnen zu müssen. Außerdem können so verschiedene Ereignisse in neue Szenen ausgelagert werden, um eine Beeinflussung von Objekten untereinander auszuschließen.

Die Vorgabe beim Servingo-Projekt war es, eine je nach Benutzer-Eingabe dynamisch veränderbare Applikation zu erstellen. Dazu war es nötig, eine Möglichkeit zu realisieren, die Animation an die vom Benutzer eingegebenen Texte und die später daraus erzeugten Audio-Dateien je nach ihrer Dauer automatisch anpassen zu können. Um dies realisieren zu können, wurde die Hauptszene (Scene) kopiert und als Scene2 neu angelegt, wodurch man 2 gleiche Szenen mit denselben Objekten und Materialien erhielt. Anschließend wurden die, in der Scene2 nicht benötigten, verlinkten (verbundenen) Einstellungen, wie z. B. die Mundbewegungen oder Animationskurven der Armatur gelöscht, um diese neu anlegen zu können und die Beeinflussung der einzelnen Szenen untereinander auszuschließen.

Nun wurde der, für den Benutzer unsichtbare Dialog, der beiden Hauptfiguren in statische und dynamische, d. h. vom Benutzer veränderbare Sequenzen zerlegt. Die statischen Teile und die Dauer der Audio-Dateien in Frames wurden in der Hauptszene realisiert und die dynamischen in der Scene2. Dazu wurde die Position der Figuren am Ende der jeweiligen Sequenz kopiert und diese dann in die Szene2 übertragen. In dieser Szene verbleiben die Charaktere solange in der Position, wie es die Dauer der anhand der Benutzer

Eingaben generierten Audio-Datei vorgibt und bewegen den Mund dazu. Die Dauer wird mittels des Python Skripts ermittelt und an den laufenden Renderprozess weitergegeben. Dieser Vorgang wird im Kapitel 4 genau erläutert.

Nach dem Ende der dynamischen Benutzersequenz läuft die Animation wieder in der Hauptszene weiter und zwar genau ab dem Frame, bei welchem die erste Sequenz aufgehört hat. Durch das Wiederholen dieser Technik entstehen so mehrere Videosequenzen, die Frame genau zueinander passen und später zu einem kompletten Film zusammen gesetzt werden. Der nachfolgende Dialog vom Anfang der Animation verdeutlicht diese Vorgehensweise und zeigt die jeweilige Dauer der statischen Sätze:

Hallo und herzlich willkommen zu [statisch, Frame 1-48, Scene1]
„___Titel___" [dynamisch, Länge variabel, Scene2]

Heute zu Gast bei uns ist [statisch, Frame 49-80, Scene1]
„___Name____" [dynamisch, Länge variabel, Scene2]

Hallo. [statisch, Frame 81-100, Scene1]
„__Name__", [dynamisch, Länge variabel, Scene2]

du hast uns paar persönliche Eindrücke mitgebracht. Vielleicht erzählst du etwas dazu. [statisch, Frame 101-224, Scene1]

3.2.5 Animationen

Um die Szene lebendiger und somit attraktiver für den Benutzer zu machen und um die Dialoge der Charaktere visuell zu unterstützen, wurden die Charaktere mit Hilfe gängiger Animationswerkzeuge in Blender animiert. Dazu wurde die so genannte Keyframe Technik verwendet, welche nach folgendem Prinzip funktioniert:

Das zu animierende Objekt wird im Ausgangsframe mit einem Key versehen, d. h. seine Position wird zu diesem bestimmten Zeitpunkt gespeichert, wobei man verschiedene Parameter, wie z. B. Lage, Rotation, Größe etc. speichern kann. Danach wird die Animation zum gewünschten Endpunkt bewegt und das Objekt, je nachdem wie es zu diesem Zeitpunkt aussehen soll, verändert und wiederum mit einem Key versehen. Anhand dieser Parameter berechnet Blender dann die dazwischen liegenden Frames und erstellt so-

mit eine flüssige Animation. Der Schwerpunkt beim Animieren lag bei der Bewegung der Arme und des Mundes, um die Charaktere möglichst lebendig erscheinen zu lassen. Da die später generierte Video-Datei laut Vorgabe nicht zu groß werden sollte (bis ca. 1,5 MB), wurden die Animationen eher schlichter, aber dafür ausdrucksvoller gehalten.

Eine besonders komfortable Methode, um Charaktere in Blender zu animieren, ist der so genannte Pose Mode. Dieser Modus erlaubt es die IK-Solver oder die einzelnen Bones der Armatur in einem Art Vorschaumodus zu bewegen, in welchem man die Bewegungen und deren Auswirkungen auf andere Bones oder Objekte in Echtzeit sehen kann. In den Abbildungen 3.44 und 3.45 sieht man ein Beispiel für den Pose Mode, wobei die Abbildung 3.44 die Ausgangsposition in Frame 1 und die Animationskurve für den Key des IK-Solvers der linken Hand zeigt. In der Abbildung 3.45 sieht man die Pose im Frame 82 und die veränderten Animationskurven für die verschiedenen Richtungen des Bones. Die schwarzen Punkte auf den Animationskurven stellen die Keys dar und die verschieden farbigen Kurven sind bei diesem Bone, die Location (Lage) Parameter für X,Y und Z.

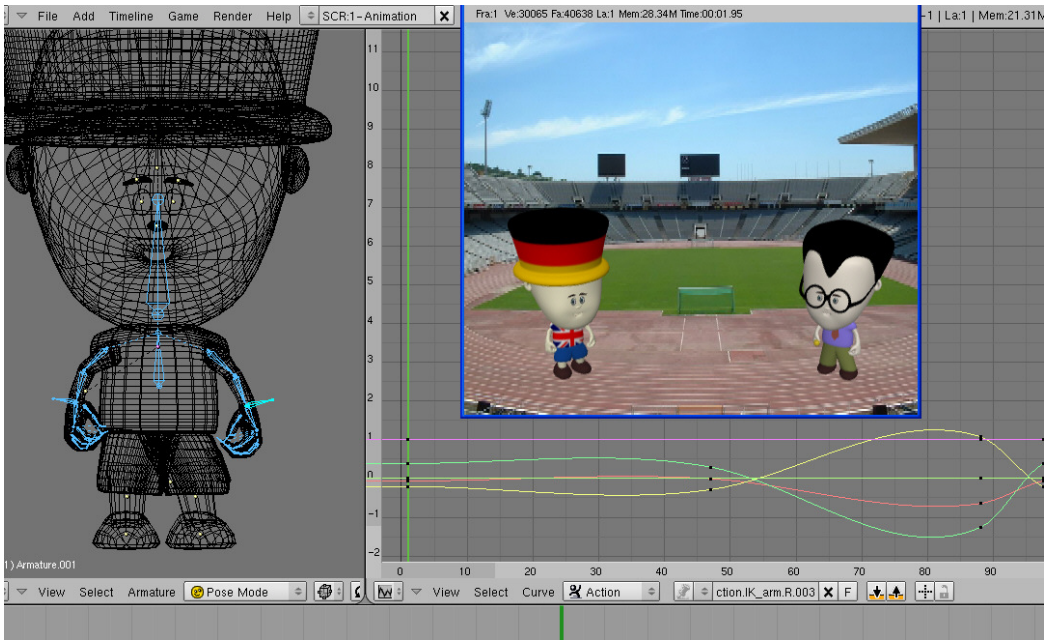


Abbildung 3.44: Pose im Frame 1

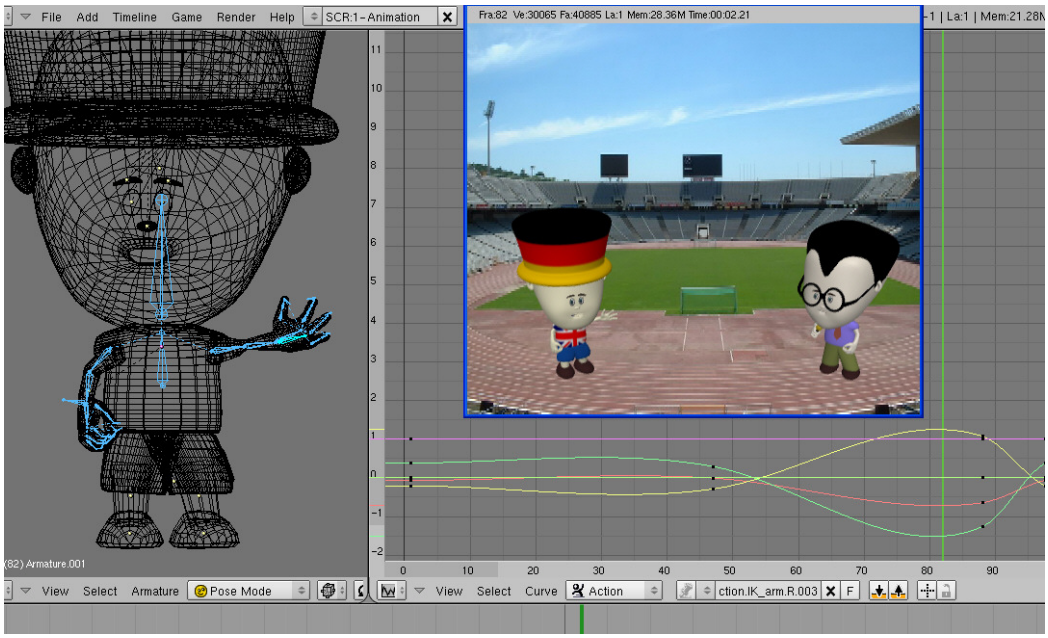


Abbildung 3.45: Pose im Frame 82

Eine weitere wichtige Animation war die Mundbewegung der Charaktere, welche durch die Vertex Keys Animation bewältigt wurde, die sich besonders gut für Gesichtsbewegungen eignet. Hierbei handelt es sich um die Möglichkeit, einzelne Punkte eines Objekts zu animieren und somit dessen Deformation zu ermöglichen. Außerdem können die einzelnen Veränderung des Objekts in Keys gespeichert werden und sind dann als Parameter einsetzbar. Sie müssen dann nicht mehr für jede Mundbewegung nachmodelliert werden, sondern werden durch das Setzen des jeweiligen Keys in die Animation eingefügt und die dazwischen liegenden Bewegungen von Blender automatisch interpoliert. Durch die Interpolation der einzelnen Mundbewegungen entsteht der Eindruck, der Charakter würde zu dem zu hörendem Text sprechen. Hierzu sind vorher die Sprechphasen der beiden Charaktere dem Storyboard (Drehbuch [Anhang A]) entsprechend, in der Blenderszene voreingestellt und gespeichert worden. Die Abbildungen 3.46 und 3.47 zeigen zwei verschiedene Keys und ihre Auswirkungen auf den Mund des Charakters, wobei rechts im Bild die Keys und ihre Maxima zu sehen sind.

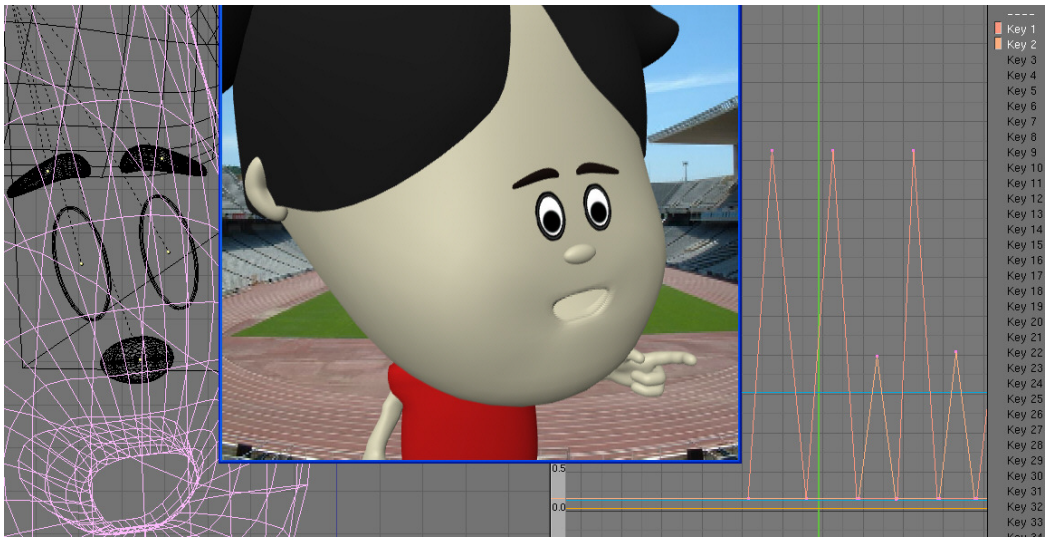


Abbildung 3.46: Mundbewegung Key 1

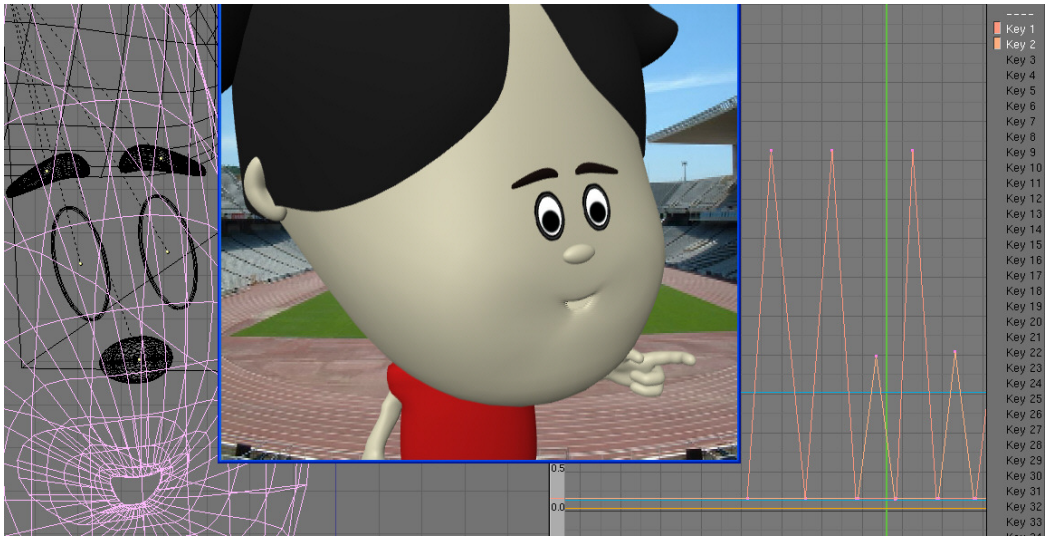


Abbildung 3.47: Mundbewegung Key 2

3.2.6 Gestaltung des Servingo-Logos

Um das Servingo-Logo kreativ in die Szene einbinden zu können, wurde dieses mittels Blender aus einer Bildvorlage in ein 3D-Objekt umgewandelt (Abbildungen 3.49 bis 3.53). Dadurch war es möglich, das Logo auch zu repräsentativen Zwecken zu nutzen und es als animiertes Logo auf den Endgeräten darzustellen. Die Animation des Logos war ein zusätzliches Projekt, welches zum Ziel haben sollte, den Abspann im fertigen Film, in welchem sich die Figuren verabschieden, lebendiger zu machen.

Zuerst wurden die Umrisse des Originallogos in Blender mittels Bézierkurven nachgezeichnet. Die Abbildung 3.48 zeigt eine Bézierkurve mit gestricheltem Kontrollpolygon.

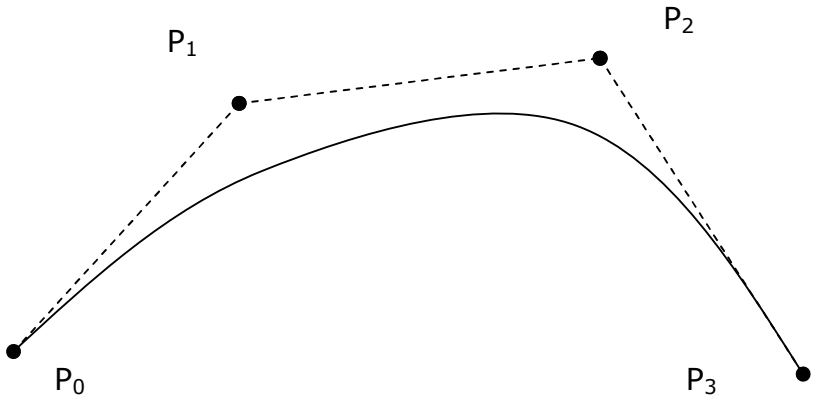


Abbildung 3.48: Schema einer Bézierkurve

Eine *n*-dimensionale Bézierkurve ist eine Kurve der Form

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

mit den Kontrollpunkten P_i und den Bernsteinpolynomen $B_{i,n}(t)$ und $0 \leq t \leq 1$. Durch die Annäherung der Bézierkurve an das Polygon der Kontrollpunkte ist diese Form besonders gut für den interaktiven Entwurf und die Bearbeitung am Bildschirm geeignet. Außerdem besitzt sie folgende wichtige Eigenschaften, die sich aus den Eigenschaften der Bernsteinpolynome ergeben:

- Die Kurve verläuft genau durch den Startpunkt P_0 und den Endpunkt P_3.
- Sie befindet sich innerhalb der konvexen Hülle des Kontrollpolygons.

- Von einer Linie wird sie maximal so oft geschnitten, wie oft diese ihr Kontrollpolygon schneidet.
- Die affine Transformation (Verschiebung, Skalierung, Rotation, Scherung), kann durch die Transformation des Kontrollpolygons erreicht werden.
- Befinden sich alle Kontrollpunkte auf einer Linie, so wird die Bézierkurve zu einer Strecke.

Die Abbildungen 3.49 - 3.52 zeigen die schrittweise Modellierung des Logos in Blender mittels Bézierkurven.

Abbildung 3.49: Originalvorlage Servingo-Logo [2]

Abbildung 3.50: Hintergrundbild mit Bézierkurven

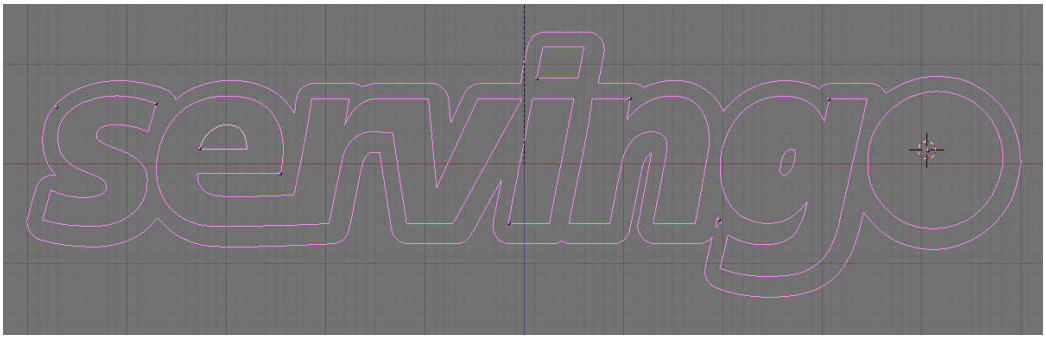


Abbildung 3.51: Bézierkurven

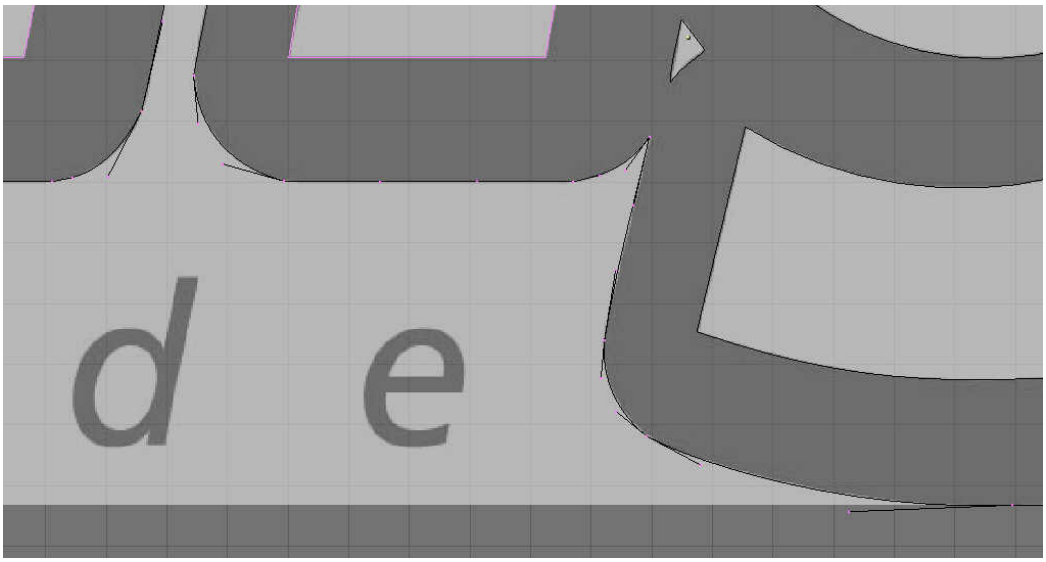


Abbildung 3.52: Detail Bézierkurven

Abbildung 3.53: Extrudierte Kurvenfläche mit gerendertem Bild

Die so entstandenen Kurven wurden dann extrudiert, d. h. in die Breite gezogen, um ein dreidimensionales Abbild des Logos zu erhalten (Abbildung 3.53). Anschließend wurde der Fußball durch das LSCM-Unwrap-Tool (LSCM = Least Square Conforming Map) jeweils in seine Bestandteile, also Fünf- und Sechsecke zerlegt. Dieses Werkzeug erlaubt es in Blender, über mathematische Methoden, Objekte, die aus mehreren Teilflächen bestehen, in Einzelteile zu zerlegen, um sie dann leichter bearbeiten zu können.

Dadurch erhält man ein ebenes Abbild der gesamten Flächen und kann sie nun z. B. einfärben. Um das Aussehen eines Fußballs zu realisieren, wurde nun die Flächenstruktur auf ein weißes Hintergrundbild gelegt und die benötigten Fünfecke mittels des UV-Paint-Tools schwarz eingefärbt. Das UV-Paint-Tool erlaubt es, wie in einem Grafikprogramm über Werkzeuge wie Pinsel, Sprühdose etc., 3D-Objekte zu bemalen. Einen Überblick über den gesamten Vorgang bietet die Abbildung 3.54. Hier sieht man die schwarz eingefärbten Fünfecke und das fertig gerenderte Logo.

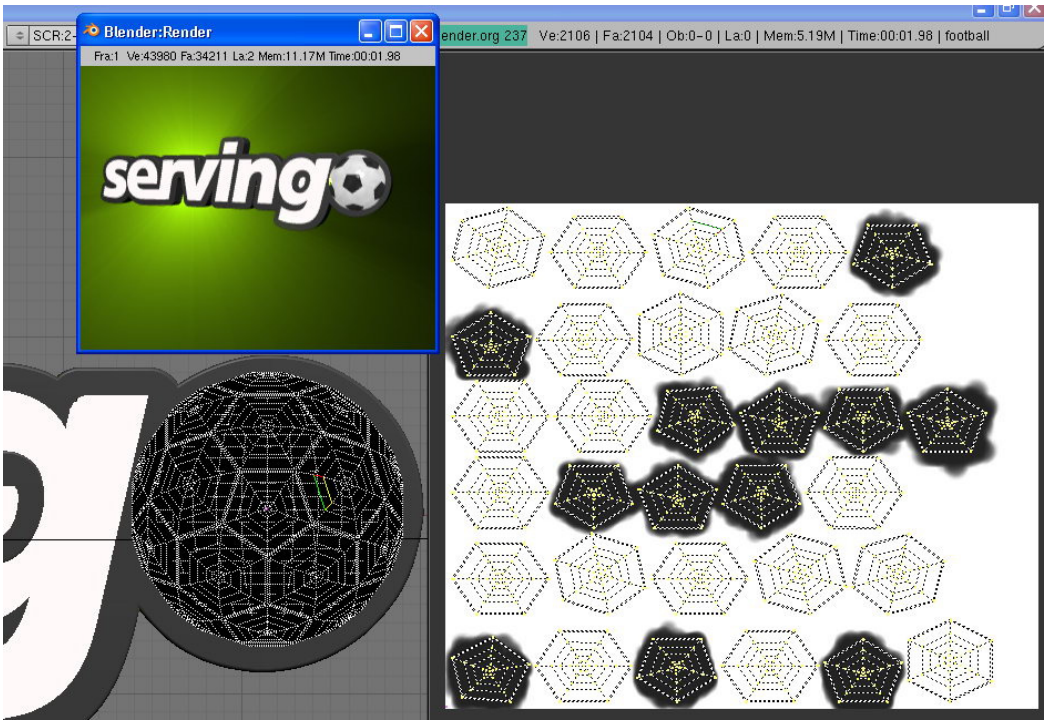


Abbildung 3.54: Übersicht LSCM Unwrap Tool

Um das Logo visuell ansprechender zu machen, wurde zuerst der Fußball zum Rotieren gebracht. Dies geschah mittels linearer Animation, über welche man hier die Rotation um die eigene Achse festlegt und diese dann linear als Endlosschleife wiederholen lässt. Animationen werden über die so genannten Keys bewerkstelligt, mittels welcher man einem Objekt eine Ausgangslage zuweist und diese dann zu einem späteren Zeitpunkt ändert und wieder mit einem Key versieht. Die Zwischensequenzen werden dann je nach Vorgaben automatisch von dem 3D-Programm erzeugt und angezeigt.

Die Abbildung 3.55 zeigt den Fußball bei Frame 1 und mit einem gesetzten Key für die Rotation um die Z-Achse (blaue Linie) und die Abbildung 3.56 zeigt beispielsweise Frame 110 und die dementsprechende Bewegung des Fußballs.

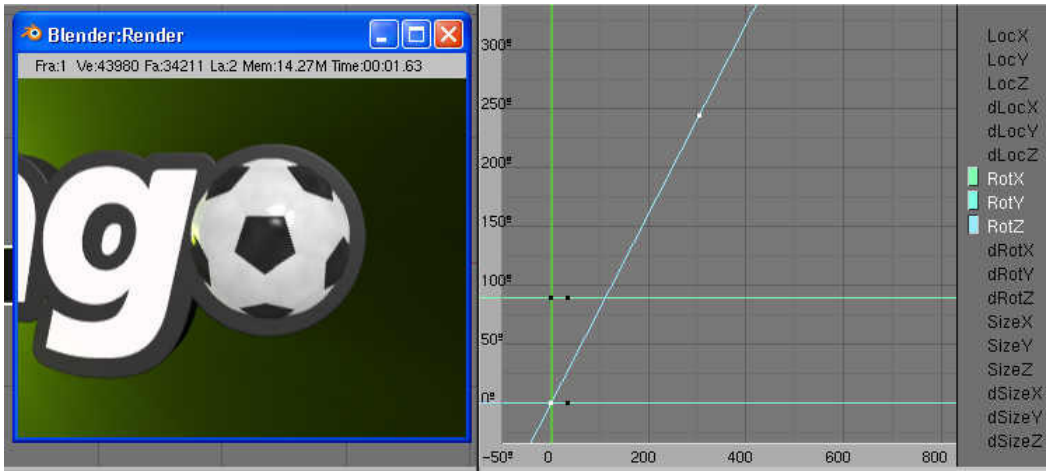


Abbildung 3.55: Beispiel lineare Animation Frame 1

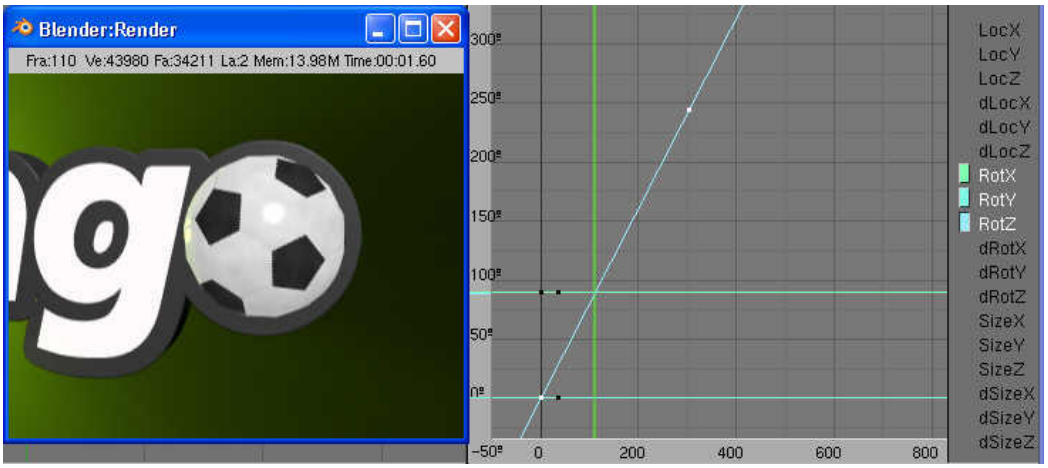


Abbildung 3.56: Beispiel lineare Animation Frame 110

Ein weiterer Aspekt des Designs des Logos war die Lichtsetzung. Hierbei sollte das Licht das Logo interessanter machen und es voluminöser erscheinen lassen. Um diesen Effekt zu erreichen, wurde ein Spotlicht von hinten auf die Buchstaben gerichtet und die Kamera vor das Logo postiert. Das Licht wurde mit der Funktion Buffered Shadows eingestellt, welche es in Blender erlaubt, einen Puffer für die zu entstehenden Schatten anzulegen und somit den Vorgang der Schattenerzeugung beim Rendern zu beschleunigen und außerdem weichere Schatten zu erzeugen.

Durch eine lineare Animation der Bewegung der Lichtquelle und eine dazugehörige Kamerafahrt entstand das im späteren Servingo-Film sichtbare Endergebnis. Die Abbildung 3.57 zeigt die Lichtsetzung in der Logo Szene und die 3.58 ein fertig gerendertes hoch aufgelöstes Bild.

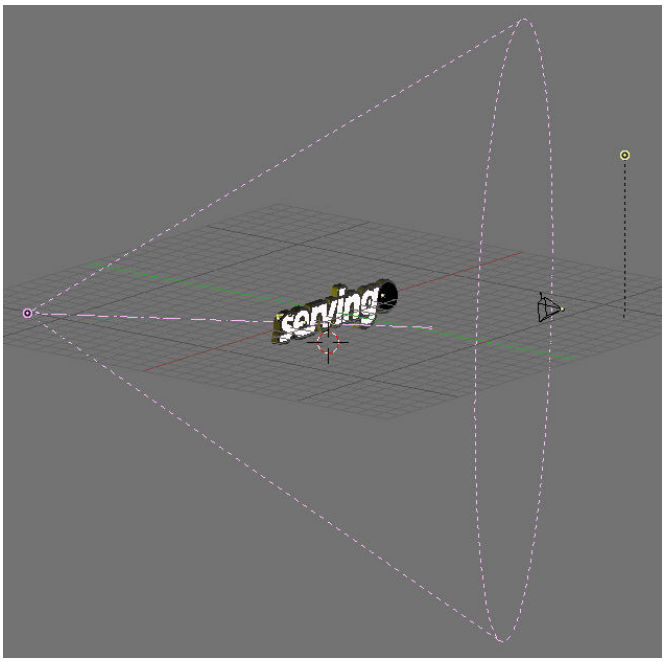


Abbildung 3.57: Lichtsetzung

Abbildung 3.58: Gerendertes Logo

3.3 Zusammenfassung

In diesem Kapitel wurde die Gestaltung und die Umsetzung der 3D-Komponenten, sowie die Realisierung der Animationen und der Aufbau der Szene näher erläutert. Das Interessante am Servingo-Projekt war, dass die Gestaltung der Charaktere stets veränderbar bleiben sollte, um die sich ändernden Anforderungen zu erfüllen. So entstand eine 3D-Szene, die später über einfache Änderungen beliebig erweiterbar ist und somit an beliebige Veranstaltungen oder andere Verwendungen angepasst werden kann. Nachdem die gestalterischen Aufgaben bewerkstelligt waren, wurde die Umsetzung der automatisierten Generierung und die Anbindung an die vom Server gelieferten Inhalte in Angriff genommen.

4 Realisierung der Generierung

4.1 Einleitung

Der wichtigste Teil des Servingo-Projekts war die automatisierte Generierung der in Blender vorgefertigten Animationen anhand der vom Server gelieferten XML-Inhalte. Um die Aufgaben der eigentlichen Programmierung zu bewerkstelligen und das Zusammenspiel der einzelnen Komponenten, Module und Programme aufeinander abzustimmen, war eine umfangreiche Recherche im Internet, zahlreichen Foren und Büchern notwendig. Die für das Servingo-Projekt benötigten Komponenten werden nun nachfolgend, näher beschrieben.

4.2 Beschreibung der Komponenten

Das Ergebnis dieser Diplomarbeit sollte eine Art Modul realisieren, welches autark funktionieren kann und von externen Programmen oder Serverbefehlen ansprechbar ist und die durch eine XML-Datei übermittelten Daten automatisch verarbeiten kann. Um diese gewünschte Funktionalität zu erreichen, wurden mehrere Programme und Module zusammen verbunden und deren Funktionalität aufeinander abgestimmt.

Das Besondere an der Blender-Python Schnittstelle ist es unter anderem, dass Blender über eine so genannte Script Linking Funktion verfügt, mit der es möglich ist, Python Skripte mit einer Blender-Szene zu verbinden. Dadurch entsteht eine Datei, die einfach per Befehl von Programmen geöffnet werden kann und die dann ihre Code-Funktionalität der nachfolgenden Komponenten automatisch ausführt.

Somit ist es nicht notwendig, Parameter oder Skripte an Blender zu übergeben, sondern es reicht ein einziger „Öffnen"-Aufruf, um das Servingo-Modul zu starten. Die nächsten Kapitel erläutern die einzelnen Komponenten, die beim Starten der Blender-Datei automatisch zusammen funktionieren.

4.2.1 Sprachgenerierung

Um die vom Benutzer eingegebenen Texte wie Name, Titel der Geschichte und Bilduntertitel in den Film einbinden zu können, müssen diese anhand der übermittelten Daten automatisch erzeugt werden. Diese Aufgabe wird beim Servingo-Projekt mittels der Text-to-speech (Sprachsynthese) Software AT&T Natural Voices bewältigt, deren Logo in Abbildung 4.1 zu sehen ist.

Abbildung 4.1: Logo AT&T Natural Voices [10]

Diese Software erzeugt aus einem Text eine sehr hochwertige synthetische Sprache und kann, was für die Automatisierung besonders wichtig war, über DOS-Befehle oder eine Batch-Datei gesteuert werden. Text-to-speech Systeme bestehen aus zwei Komponenten und funktionieren im Allgemeinem nach folgendem Prinzip:

- NLP (Natural Language Processing): Konversion von Text in Lautschrift und Prosodie(Silbenbetonung)-Beschreibung.
- DSP (Digital Speech Processing): Ausgehend vom Output der NLP-Komponente erfolgt die Synthese des Sprachsignals.

Bei AT&T Natural Voices arbeitet die DSP-Komponente, oder auch Sprach-engine genannt, nach dem so genannten Non-uniform unit selection Verfahren, welches aus einer sehr großen Datenbasis, die am besten passenden Sprachteile (Units), mit einer variablen Länge (Non-Uniform), miteinander verkettet, wobei die Datenbasis bis zu 500 MB pro Stimme betragen kann. Dabei wird eine doppelte Kostenfunktion minimiert, d. h. die Stücke sollen gut aneinander passen (Verkettungs-Kosten) und die Vorgaben der Ziel-Prosodie erfüllen (Ziel-Kosten). Die verwendete Version 1.4 kann die Sprachen: US Englisch, Spanisch, UK Englisch, Deutsch und Französisch mit mehreren männlichen und weiblichen Stimmen generieren. Der genaue Ablauf und die Ausgabe der Sprache wird im Kapitel 4.3 beschrieben.

4.2.2 Die Programmiersprache Python

Diese Programmiersprache spielte bei dem StoryGenerator eine bedeutende Rolle, da Blender über eine implementierte Python-Schnittstelle verfügt und wird deshalb nachstehend kurz vorgestellt.

Abbildung 4.2: Logo Python [11]

Bei Python (Abbildung 4.2) handelt es sich um eine objektorientierte Programmiersprache, die Anfang der 90er Jahre von Guido van Rossum am Centrum voor Wiskunde en Informatica in Amsterdam entwickelt wurde. Das Ziel bei der Entwicklung war es, eine möglichst einfache und übersichtlich strukturierte Sprache zu schaffen, was vor allem durch einen sehr mächtigen Funktionsumfang mittels nur weniger Schlüsselwörter und eine auf Übersichtlichkeit optimierte grammatikalische Syntax realisiert wird.

Somit entstand eine Sprache, die leicht zu erlernen und anzuwenden ist und wegen ihrer Übersichtlichkeit zunehmend geschätzt wird. Python bietet auch die Möglichkeit an, andere Sprachen als Module einzubinden oder als Skriptsprache für Programme zu dienen. Bekannte Beispiele dafür sind OpenOffice, Blender und Gimp (Abbildung 4.3).

Abbildung 4.3: Logos OpenOffice, Blender, Gimp

Da Python sowohl objektorientierte und strukturierte als auch funktionale Programmierung unterstützt, gibt sie damit dem Programmierer mehr Freiheiten als vergleichbare Sprachen und obwohl bei Python stets die Einfachheit als Skriptsprache im Vordergrund stand, sind damit eine Reihe großer Softwareprojekte bewältigt worden. Beispiele dafür sind der Application-Server Zope sowie das

Dateisystem Mojo Nation oder auch die Tatsache, dass Teile von Google mit Python programmiert wurden (Abbildung 4.4).

Abbildung 4.4: Logos Zope, Mojo Nation, Google

Anders als in anderen Programmiersprachen benutzt Python Einrückungen als Strukturierungselement, was vor allem für Programmierneulinge vom Vorteil ist, weil dadurch die Struktur der Programme besser sichtbar wird.

Am besten wird dies an einem kurzen Beispiel zur Berechnung der Fakultät einer Zahl in C und Python ersichtlich:

Fakultätsfunktion in C (ohne Einrückung, unleserlich):

```
int factorial(int x){ if(x == 0){ return 1; } else{ return x *
factorial(x - 1); } }
```

Fakultätsfunktion in C (mit Einrückung):

```
int factorial(int x){
    if(x == 0){
        return 1;
    }
    else{
        return x * factorial(x - 1);
    }
}
```

Jetzt die gleiche Funktion in Python:

```
def factorial(x):
    if x == 0:
        return 1
    else:
        return x * factorial(x – 1)
```

Neben der Einfachheit und der leichten Anwendung besitzt Python eine große Standardbibliothek, die sich für viele Anwendungen gut eignet und jederzeit durch eigens geschriebene Module in C oder Python erweitert werden kann. Die umfangreiche Standardbibliothek ist eine der größten Stärken von Python, weil das meiste hiervon plattformunabhängig ist, was zur Folge hat, dass auch größere Python-Programme oft auf Unix, Windows, Macintosh und anderen Plattformen ohne Änderungen lauffähig sind. Python ist für die meisten gängigen Betriebssysteme frei erhältlich. Einen guten Überblick über diese Sprache bietet das Buch „Einführung in Python“ [12].

4.2.3 Python Module

Wie schon im vorhergehenden Kapitel beschrieben, verfügt Python über zahlreiche Bibliotheken und Module, die auch stets auf Basis des Open Source Gedanken weiterentwickelt werden. Damit ist es jedem Benutzer möglich, sich seine Arbeitsumgebung und die benötigten Eigenschaften von Python selbst zusammenzustellen. Diese Module und Zusatz Plug-ins sind alle kostenlos und stehen für alle gängigen Betriebssysteme und Plattformen auf zahlreichen mit Python verwandten Internet Seiten zur Verfügung. Die für das Servingo-Projekt relevanten Module werden nun nachfolgend näher beschrieben.

Das Modul wxPython ist eine GUI (Graphic User Interface, d. h. Benutzeroberfläche) für die Programmiersprache Python, welche es dem Python Programmierer erlaubt, Programme leicht und effizient mit Hilfe einer robusten und hoch funktionellen, graphischen Benutzeroberfläche zu schreiben. Außerdem ist wxPython plattformübergreifend, was bedeutet, dass die damit erstellten Programme ohne Modifikationen auf verschiedenen Plattformen laufen können. Zurzeit werden das 32-bit Microsoft Windows, Macintosh OS X und die meisten Unix Systeme unterstützt.

Sie wird als ein externes Modul in Python implementiert und ist auch an die Blender und Python Version anpassbar. Für das Servingo-Projekt wurde die Version (wxPython2.6-win32-ansi-2.6.1.0-py23.exe [13]) in Kombination mit der Python Version 2.3.5 verwendet.

Ein weiteres wichtiges Modul ist das PyXML Paket (Version PyXML-0.8.4.win32-py2.3.exe [14]), welches zahlreiche Werkzeuge und Bibliotheken für die Arbeit mit XML und Python bereitstellt. Grundlegend besteht es unter anderem aus folgenden Bestandteilen:

- xmlproc: Ein XML-Parser mit Gültigkeitsprüfung.
- Expat: Ein schneller nicht prüfender Parser.
- PySAX: SAX1 and SAX2 Bibliotheken mit Treibern für die meisten Parser.
- 4DOM: Eine voll unterstützende DOM Level 2 Implementation.
- javadom: Ein Adapter von Java DOM Implementationen zur standardisierten Python DOM Bindung.
- pulldom: Eine DOM Implementation, die eine langsame Instanzierung von Knoten ermöglicht.
- marshal: Ein Modul mit zahlreichen Möglichkeiten zur Serialisierung von Python Objekten nach XML, sowie WDDX und XML-RPC.

Zur Arbeit mit Python und Blender wurde der Stani's Python Editor (Version: SPE-0.7.3.a-wx2.5.4.1.-bl2.35.exe [15]) verwendet, welcher ein Ergebnis eines Open Source Projekts unter der Obhut von SourceForge.net steht. Er wird direkt in Blender gestartet und ist eine Art Modul, welches die Kommunikation zwischen Blender und Python Code veranschaulicht. Dadurch ist es möglich, programmierten Python-Code direkt in Blender auszuführen und dessen Auswirkungen zu beobachten. Die Abbildung 4.5 zeigt beispielhaft eine Arbeitsoberfläche des Stani's Python Editors, in der man im Hintergrund die Blender-Szene und im Vordergrund den Python-Code und dessen Ergebnis im Render-Fenster sieht.

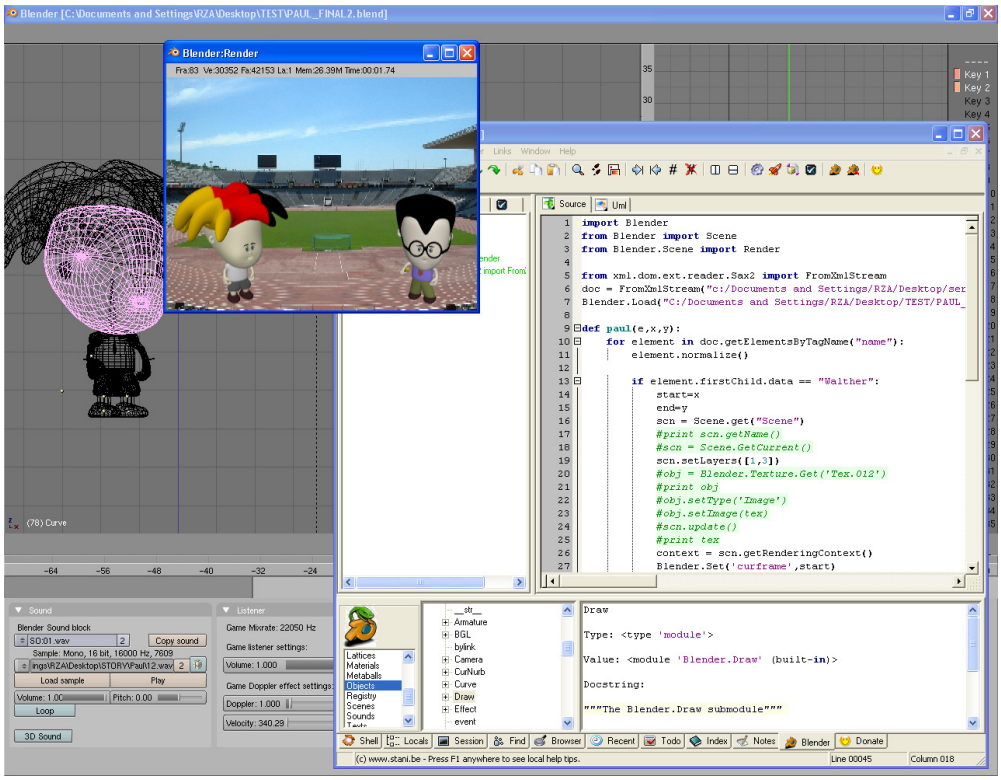


Abbildung 4.5: SPE integriert in Blender

Dieser Editor beinhaltet wiederum zahlreiche Plug-ins, die das Arbeiten mit Blender und Python erleichtern.

Er hat selbst wiederum folgende Module implementiert:

- Kiki: Eine reguläre Ausdruckskonsole.
- PyChecker: Ein Quellcode „Doktor".
- Python Debugger: Ein Fehler Entferner für Python.
- wxGlade: GUI Designer (graphische Benutzeroberfläche).

Er ist nur 2 MB groß, kostenlos und besitzt folgende Eigenschaften:

- Intelligenz: Er verfügt über eine der besten automatischen Vervollständigungen von Python Code.

- UML-Diagramme: Ein UML Betrachter ist gleichzeitig zum Code Editor aktiv, so dass es möglich ist, simultan UML-Diagramme zu erstellen und auszudrucken.

- Elegante Übersicht: Automatische Aktualisierung der Seitenleisten, alphabetischer Index aller Funktionen und Klassen und Code Analysierung.

- Design: Eine sehr bequeme Methode zur Erstellung von wxPython Oberflächen, durch die Verwendung von wxGlade.

- Debug: Integrierter RPDB2 Debugger [16].

Außerdem verfügt er über eine Übersicht der Blender-Python API [17], wodurch man alle für Blender relevanten Einstellungen und Python Befehle bereits in diesem Editor erfahren kann.

4.2.4 XML Integration

Die Servingo Anwendung und deren Ausgabe wird durch den Inhalt der vom Server an sie übermittelten XML-Datei gesteuert und deshalb wird diese Sprache im Nachstehenden kurz vorgestellt. Als ein besonders geeignetes Nachschlagewerk zu dieser Thematik erwies sich das Buch „Python & XML" [18].

Die Sprache XML (Extensible Markup Language) ist ein Standard zur Erstellung maschinen- und menschenlesbarer Dokumente in Form einer Baumstruktur. Einer Ihrer Vorteile ist, dass sie selbst die Regeln für den Aufbau von XML-Dokumenten festlegen kann und diese dann in Form der Dokumenttypdefinition (DTD) bei dem konkreten Datenaustausch weitergibt. Durch diese Wohlgeformtheit erlangt das jeweilige Dokument seine Gültigkeit, womit es von allen XML-konformen Programmen verarbeitet werden kann. Außerdem sind die Namen der Strukturelemente frei wählbar, wodurch es eine unbegrenzte Anzahl an Möglichkeiten gibt, eine XML-Datei in ihrer Form und ihrem Inhalt zu gestalten. Ein XML-Dokument muss genau ein Element in der obersten Ebene enthalten. Unterhalb von diesem Dokumentelement können weitere Elemente verschachtelt werden.

Beispiel einer XML-Datei:

```
<?xml version="1.0" standalone="yes" encoding="UTF-8"?>
<character>
     <titel>Servingo</titel>
     <parameter>
          <haare>rasta</haare>
          <kleidung>hose</kleidung>
     </parameter>
     <parameter>
          <farbe_1>grün</farbe_1>
          <textur>de.jpg</textur>
     </parameter>
</character>
```

Für das Projekt Servingo waren vor allem die Kerntechnologien zur Verarbeitung von XML-Dokumenten, die so genannten APIs (Application Programming Interfaces), die die Schnittstellen zur Anwendungsprogrammierung zwischen Betriebssystemen oder anderen Softwaresystemen und Programmen zur Verfügung stellen, von Interesse. Hierzu gehört das DOM Modell, das im folgenden Kapitel näher erläutert wird.

4.2.5 Das DOM Modell

Das DOM (Document Object Model) entstand, um den Zugriff auf strukturierte Daten in HTML (Hyper Text Markup Language)- und XML-Dokumente zu ermöglichen. Die ersten DOM-Standards bildeten den Versuch, die entstandenen JavaScript- und DHTML-Techniken (Dynamic HTML) zusammenzuführen und zu standardisieren, mit dem Ziel, sie abzulösen. Das DOM ist seit 1998 ein Standard des W3C [19] und wurde seitdem mehrfach aktualisiert und erweitert, wobei es mittlerweile verschiedene Versionen, so genannte Levels gibt und zwar DOM Level 0 bis Level 3. Die für den Umgang mit Python und XML relevanten Features werden im Level 2 definiert.

Die DOM-Struktur besteht aus einer Hierarchie von Knoten-Objekten (sog. Nodes), die aus verschiedenen Typen bestehen können, wobei jeder Knoten eine Liste von Referenzen auf Kindknoten besitzt. Die Darstellung eines Dokuments mit DOM lässt sich am besten am folgenden Beispiel demonstrieren.

Der nachstehende HTML-Code definiert eine Tabelle mit dem Element *table* und verschiedenen Unterelementen:

```
<table>
   <thead>
     <tr>
       <th>Name</th>
       <th>Vorname</th>
     </tr>
   </thead>
   <tbody>
     <tr>
       <td>Donald</td>
       <td>Duck</td>
     </tr>
   </tbody>
 </table>
```

Das DOM repräsentiert das *table* Element und dessen Unterelemente in der folgenden Baumstruktur (Abbildung 4.6):

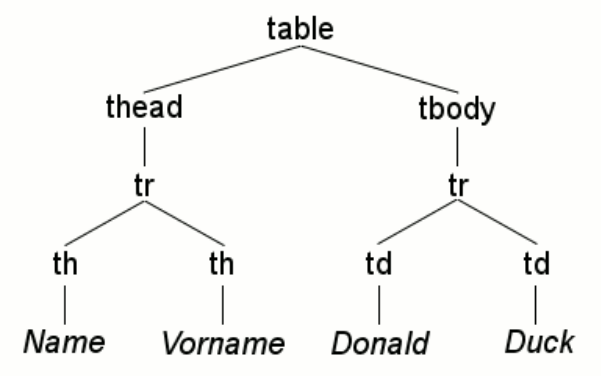


Abbildung 4.6: DOM Baumstruktur [20]

Dadurch stehen die Knoten-Objekte in folgenden Beziehungen zueinander [20]:

- Der Wurzelknoten *table* hat als Kind (Child) den Elementknoten *tbody*.
- Der *table*-Elementknoten ist umgekehrt ein Vater (Parent) von *thead* und *tbody*.
- Knoten mit gemeinsamem Vater (z. B. die beiden *th*-Elementknoten) werden Geschwister (Siblings) genannt.

Ausgehend vom Wurzelknoten ist jeder andere Knoten über diese Verwandschaftsbeziehungen erreichbar.

Die wichtigsten Knotentypen im DOM sind [20]:

- Ein Dokumentknoten stellt die gesamte Baumstruktur dar.
- Ein Dokumentfragmentknoten stellt einen Teil der Baumstruktur dar.
- Ein Elementknoten entspricht exakt einem Element in HTML oder XML.
- Ein Attributknoten entspricht exakt einem Attribut in HTML oder XML.
- Ein Textknoten stellt den textuellen Inhalt eines Elements oder Attributs dar.

Eine besondere Art von Knoten sind die Attributknoten, da sie nicht Knoten in der Baumstruktur, sondern Eigenschaften von Elementknoten darstellen. Die weitere Verarbeitung eines Dokuments geschieht durch das so genannte parsen, welches folgendermaßen realisiert wird:

Parser funktionieren als eine Art Übersetzer, die Texte in neue Strukturen umwandeln und dadurch die Hierarchiebeziehungen der Elemente darstellen können. Die Analyse des Texts erfolgt meist mit Hilfe eines lexikalischen Scanners (auch Lexer genannt), der das Dokument in Token (dem Parser verständliche Blöcke) zerlegt, die dann als atomare Eingabezeichen des Parsers fungieren. Außerdem wird der Parser zur Grammatikprüfung, syntaktischen Kontrolle der Eingangsdaten und zur Erstellung eines Ableitungsbaums verwendet, der dann zur weiteren Verarbeitung der Daten herangezogen wird.

4.3 Ablauf der automatisierten Generierung

Bei der automatisierten Generierung findet ein Zusammenspiel mehrere Programme statt, welches in der folgenden Abbildung 4.7 kurz vorgestellt und nachfolgend näher erläutert wird.

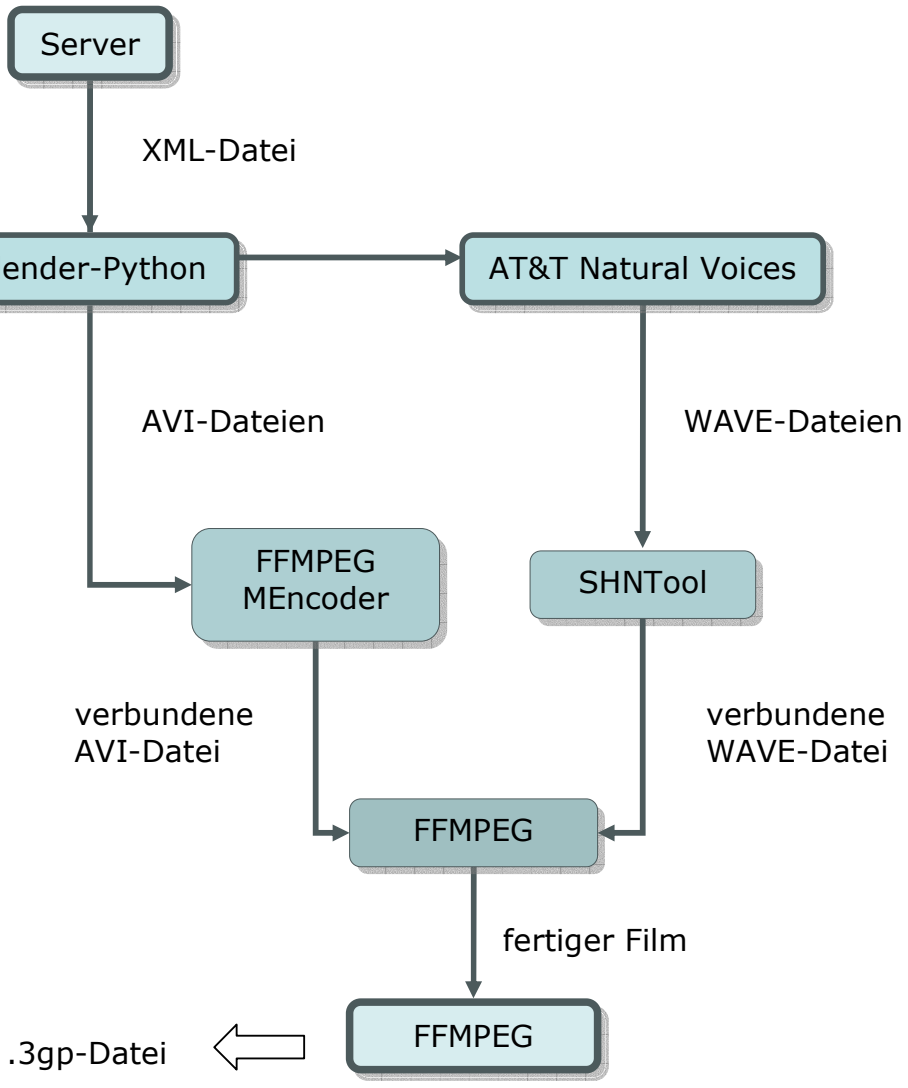


Abbildung 4.7: Überblicksskizze Servingo-Modul

Am Anfang der Generierung steht das Parsen, welches in dem Python Skript mittels des SAX2 Parsers geschieht, der im 4DOM implementiert und ein Bestandteil des PyXml Moduls ist.

```
from xml.dom.ext.reader.Sax2 import FromXmlStream
doc = FromXmlStream("C:/tmp/Servingo.xml")
Blender.Load("C:/tmp/Servingo.blend")
```

Durch diesen Aufruf wird die Datei Servingo.xml vom SAX2 Parser gelesen und ein Dokumentbaum erstellt, der, wie im vorherigen Kapitel beschrieben, eine Hierarchie der enthaltenen Knoten-Objekte und deren Beziehungen darstellt. Somit sind nun die Knoten und deren Attribute über die Node-Schnittstelle ansprechbar. Anschließend wird die Datei Servingo.blend in Blender geladen.

Die vom Server gelieferte XML-Datei enthält die vom Benutzer im Online Portal eingegebenen Parameter, die vorher anhand einer Objekt-Identifikationsliste bestimmt wurden. In dieser sind die einzelnen Einstellungen für das spätere Aussehen des Charakters durch Zahlen festgelegt. Außerdem enthält Sie die Texte, die der Benutzer im Portal eingegeben hat und Verweise zu den ausgewählten persönlichen Bilddateien, die später in dem Film passend zum Dialog der Charaktere erscheinen sollen. Im folgenden Kapitel wird die zuerst ablaufende Generierung der Sprache erklärt.

4.3.1 Sprachgenerierung

Die zu Beginn dieses Kapitel beschriebene Sprachsynthese Software AT&T Natural Voices übernimmt bei Servingo die Aufgabe der automatisierten Spracherzeugung. Dazu werden die benötigten Stimmen und Sprachen vorinstalliert und die weitere Steuerung vom Python Skript übernommen. Dieses realisiert die Sprachgenerierung auf folgende Weise:

Am Anfang werden die aus der XML-Datei, mit Hilfe des Parsers ermittelten Texte in einem temporären Ordner als Text-Datei zwischengespeichert. Nun werden mittels Python die einzelnen Text-Dateien per DOS-Befehl an AT&T Natural Voices übergeben und von dieser Software in Audio-Dateien im WAVE-Format umgewandelt. Hierzu benutzt die Software eine sehr große Datenbasis, die anhand der im Text enthaltenen Silben eine möglichst gute Betonung der Sprache zu generieren versucht.

Die Auswahl der Stimme und Sprache erfolgt ebenfalls über Python und die DOS-Konsole, womit ein einfacher DOS-Befehl zur Generierung der gewünschten Sprache und des Inhalts ausreichend ist. Die erzeugten WAVE-Dateien werden nun ebenfalls temporär zwischengespeichert und mittels Python ausgelesen, um ihre genaue Länge zu bestimmen. Dieser Parameter ist für die spätere Anpassung der dynamischen Szene besonders wichtig und wird nun von Python ermittelt. Hierzu benutzt man in Python das Time- und das WAVE-Modul, welche in das Skript importiert werden müssen. Mit deren Hilfe wird die Dauer der WAVE-Datei ausgelesen und deren Länge in Frames umgerechnet, da die spätere Anpassung der Szenen über den Frame Parameter erfolgt. Dies geschieht auf folgende Weise:

```
w = wave.open(liste_servingo_wav_end[a], "r")
time = (1.0 * w.getnframes ()) / w.getframerate ()
frames=time*25
liste_wav.append(round(frames,0))
```

Der Aufruf wave.open öffnet die WAVE-Datei und setzt sie über den Parameter „r" als Read, also als lesbar. Nun werden über die Methode *getnframes()* und *getframerate()*, die Audio-Frames und die Abtastrate ermittelt und über eine einfache mathematische Funktion dividiert. Diese Funktion liefert einen sekundengenauen Wert der Länge der erzeugten Audio-Datei und wird jetzt mit dem Wert 25 multipliziert, um auf die Dauer in Frames zu kommen, da die Bildfrequenz in der Animation 25 Frames pro Sekunde beträgt. Durch eine Schleifenfunktion erhält man so für jede Audio-Datei die genaue Länge in Frames und speichert diese gerundet in einer Liste ab, um sie im späteren Programmablauf benutzen zu können.

4.3.2 Python-Blender Schnittstelle

Das Hauptskript übernimmt diese Frames-Liste und fängt an, die restlichen Informationen aus der XML-Datei zu verwerten. So enthält die XML-Datei z. B. folgende Parameter:

`<hair>113</hair>`: Kopfbedeckung

`<wear_color>139</wear_color>`: Farbe der Kleidung

`<shirt_color>321</shirt_color>`: Farbe des T-Shirts

`<pic>C:/tmp/userpic1.jpg</pic>`: Verweis zur Bilddatei

`<wav>Meine Fußballgeschichte</wav>`: Text

Diese Werte werden nun mittels einer for-Schleife ausgelesen und als Parameter gesetzt.

```
for hair in doc.getElementsByTagName("hair"):
        if hair.firstChild.data == "011":
            haare = 2
            gen = 8
```

Das Auslesen geschieht über die Methode *getElementsByTagName()*, die das vorhandene Dokument nach einem bestimmten Namen durchsucht. Anschließend wird der gefundene Parameter mittels einer if-Abfrage über das Node-Attribut *firstChild.data* verglichen und die gesetzten Werte für Haare und Geschlecht zwischengespeichert. Ähnlich läuft auch das Bestimmen der Farben der Kleidungsstücke des Charakters, wobei hier bereits aktiv Parameter in der Blender Szene gesetzt werden. Dazu wird über die Blender-Python Schnittstelle das zu verändernde Objekt (hier am Beispiel des Objekts Arm, welches für den Oberkörper steht) durch den Aufruf *Blender.Object.Get()* aktiviert und das in der Szene voreingestellte Material (hier am Beispiel einer Textur), mittels der Methode *setTexture()* festgelegt.

```
for shirt_color in doc.getElementsByTagName("shirt_color"):
        if shirt_color.firstChild.data == "323":
            obj = Blender.Object.Get('Arm')
            mat = Material.Get('SHIRT')
            mat.clearTexture(2)
            footex = Texture.Get('SHIRT_ENGLAND')
            mat.setTexture(0, footex,Texture.TexCo['OBJECT'])
```

Um eine Veränderung der Farbe zu erreichen, verwendet man RGB-Werte, die durch das Attribut *rgbCol* eines Objekts modifiziert werden können. Im folgenden Beispiel ergibt der Wert von Col die Farbe Orange, die dann über die Zuweisung *mat.rgbCol = Col* dem Material des Rockes zugewiesen wird.

```
for wear_color in doc.getElementsByTagName("wear_color"):
        if wear_color.firstChild.data == "033":
            Col = [1,0.748,0.203]
           mat = Material.Get('ROCK')
          mat.rgbCol = Col
```

Die vom Benutzer im Portal ausgewählten persönlichen Bilddateien, die in der Animation passend zum Dialog erscheinen, werden am Anfang als Verweise ausgelesen und in einer Liste angelegt. Dazu verwendet man in Python die Listenmethode *append()*, mit der man Elemente zu Listen hinzufügen kann. Zuvor werden sie über die Konvertierungsmethode *str()* zu Strings umgewandelt, um sie später als Verweis in Blender verwenden zu können.

```
for pic in doc.getElementsByTagName("pic"):
    liste_pics.append(str(pic.firstChild.data))
```

Im laufenden Renderprozess werden die JPEG-Dateien dann über eine Schleife passend zur Animation, als Textur der Leinwand geladen. Dies geschieht in Python über den Blender Aufruf *Blender.Image.Load()*. Davor wird die zu modifizierende Textur mit *Blender.Texture.Get()* aktiviert und durch *setType()* ihr Typ, in diesem Falle Image, also eine Bilddatei bestimmt und anschließend durch die Methode *setImage()* geändert.

```
tex = Blender.Texture.Get('TEXTUR')
    tex.setType('Image')
    im = Blender.Image.Load(liste_pics[0])
    tex.setImage(im)
```

Mit der Methode *Scene.Get()* wird nun die Szene in Blender geladen und mittels der Methode *setLayers()*, die zuvor ermittelten Werte für die Kopfbedeckung (haare) und das Geschlecht (gen) als Layer in der Szene gesetzt. Durch das vorher in der Szene angelegte Baukastenprinzip mit Hilfe der Layer sind nun nur noch zwei Parameter notwendig, um das Aussehen des Charakters zu gestalten.

```
    scn = Scene.get("Scene")
    scn.setLayers([1,haare,gen])
```

Die so eingestellte Szene wird nun über die Blender interne Rendering Engine gerendert, wobei der jeweilige Anfang und das Ende der Szene vom Skript aus einer vordefinierten Liste benutzt werden. Diese Werte der Liste wurden in Blender anhand der Animation festgestellt und im Skript voreingestellt. Bei dieser Szene handelt es sich um die statische, also gleich bleibende, die immer zuerst gerendert wird und deren Einstellungsparameter dann an die dynamische Szene weitergereicht werden. Das Rendern der dynamischen Szene geschieht ähnlich der statischen, bis auf das, dass der Endwert, anhand der bei der Sprachgenerierung erzeugten Liste der Längen der WAVE-Dateien, gesetzt wird. Dadurch entsteht eine jeweils zur Länge der Audio-Datei passende dynamische Rendersequenz, die genau wie die statische in einem temporären Ordner in Form eines AVI-Films gespeichert wird.

Der benutzte Video-Codec Xvid MPEG-4 , also das Komprimierungsverfahren für die zu erstellende Video-Datei, wurde zuvor in der Blender-Datei eingestellt und gespeichert, wobei die Einstellungen auch zur Laufzeit über das Python Skript geändert werden können.

Durch das wiederholte Aufrufen dieses Render-Mechanismus werden nacheinander alle statischen und alle an die jeweilige Länge der Audio-Datei angepassten dynamischen Szenen heraus gerendert und gespeichert. Somit erhält man am Ende des Skriptdurchlaufes die einzelnen Video- und die vorher erzeugten Audio-Dateien, die nun im nächsten Prozess zusammengefügt werden.

4.3.3 Synthese der Ausgabeformate

Das Zusammensetzen der einzelnen Audio- und Video-Dateien wurde mit Hilfe des frei erhältlichen Converters (Umsetzers), ffmpeg bewältigt. Dieser gehört zum FFmpeg-Projekt [21], welches eine Reihe von Open Source Programmen, die zum konvertieren, aufnehmen und senden von Video- und Audiomaterial, in einer Art Plattform miteinander verbindet. Ursprünglich für Unix-artige Systeme entwickelt, stellt es mittlerweile auf den meisten Betriebssystemen eine Vielzahl von Möglichkeiten, Video- und Audio-Dateien zu bearbeiten, zur Verfügung.

Dieses Projekt besteht aus folgenden Komponenten:

- ffmpeg: Ein Kommandozeilenprogramm, um Video-, Audio- und Bild-Formate zu konvertieren.
- ffserver: Ein HTTP Multimedia Streaming Server für Live-Übertragungen über das Internet.
- libavcodec: Eine Bibliothek, die alle ffmpeg Audio- und Video-Encoder und –Decoder beinhaltet.
- libavformat: Eine Bibliothek, die alle Container-Parser und Container-Ersteller (Container = Dateiformat, dessen Inhalt mehrere andere Dateiformate erlaubt), für alle herkömmlichen Audio- und Video-Containerformate (z. B. AVI, MKV, Ogg Media, etc.) enthält.

Ein weiteres Programm, welches Verwendung fand, war der ebenfalls dem FFMpeg-Projekt zugehörende MPlayer [22], der auf den meisten Betriebssystemen lauffähig ist. Dieser frei verfügbare, kommandozeilenorientierte Media Player bietet herausragende Format- und Plattform-Kompatibilität und unterstützt ungefähr 192 Video- und 85 Audiocodecs. Optional verfügbar sind eine graphische Bedienungsoberfläche, sowie „MEncoder", ein Programm, dass sämtliche abspielbaren Dateien und Datenströme in andere Formate wandeln kann.

Das eigentliche Zusammenfügen der einzelnen Sequenzen, geschieht auf folgende Weise:

Im ersten Schritt werden die generierten, mit den vorgefertigten Audio-Dateien, anhand des Storyboards zusammengesetzt. Hierzu sind im Vorfeld, die zu den statischen Szenen passenden WAVE-Dateien in den verschiedenen Sprachen vorgefertigt worden und werden nun in den mit den aktuell erzeugten Sprach-Dateien temporären Ordner hinein kopiert. Somit liegen alle benötigten WAVE-Dateien in einem Ordner und können per Kommandozeilen-Aufruf kombiniert werden. Diese Aufgabe übernimmt das frei verfügbare Programm shntool [23], welches sich besonders gut zum Bearbeiten von WAVE-Dateien eignet und das über die Kommandozeile steuerbar ist. In Python ist es relativ einfach möglich, Kommandozeilen-Aufrufe auszuführen. Dazu wird anfangs das os-Modul (os = operating system) importiert, welches dann Python alle Möglichkeiten eröffnet, Funktionen des Betriebssystems mittels Code auszuführen.

Somit reicht nun ein einfacher Kommandozeilen-Aufruf aus, um die WAVE-Dateien mit Hilfe des shntools und des Befehles *join* zu verbinden. Der Aufruf *os.system(cmd)* übergibt den Befehl anschließend an die Kommandozeile und lässt ihn vom Betriebssystem ausführen.

```
cmd="D:\shntool-2.0.3\shntool.exe join D:/tmp/Servingo1.wav
  D:/tmp/titel.wav   D:/tmp/Servingo2.wav   D:/tmp/name.wav
  D:/tmp/Servingo3.wav D:/tmp/name.wav ... etc. „
os.system(cmd)
```

Die Reihenfolge der Kombination der einzelnen WAVE-Dateien wird an dieser Stelle festgelegt, was zum Vorteil hat, dass Dateien, wie z. B. der Name, die mehrmals im Dialog vorkommen, nicht jedes mal neu generiert werden müssen, sondern an die benötigte Stelle kopiert werden. Außerdem haben Kommandozeilen-Aufrufe den Vorteil, dass sie vom Betriebssystem besonders schnell ausgeführt werden und dass andere Programmierer anhand ihrer Parameter ihre Funktion und Steuerung relativ einfach deuten können. Den Ablauf der WAVE-Kombination in der DOS-Konsole zeigt die Abbildung 4.8.

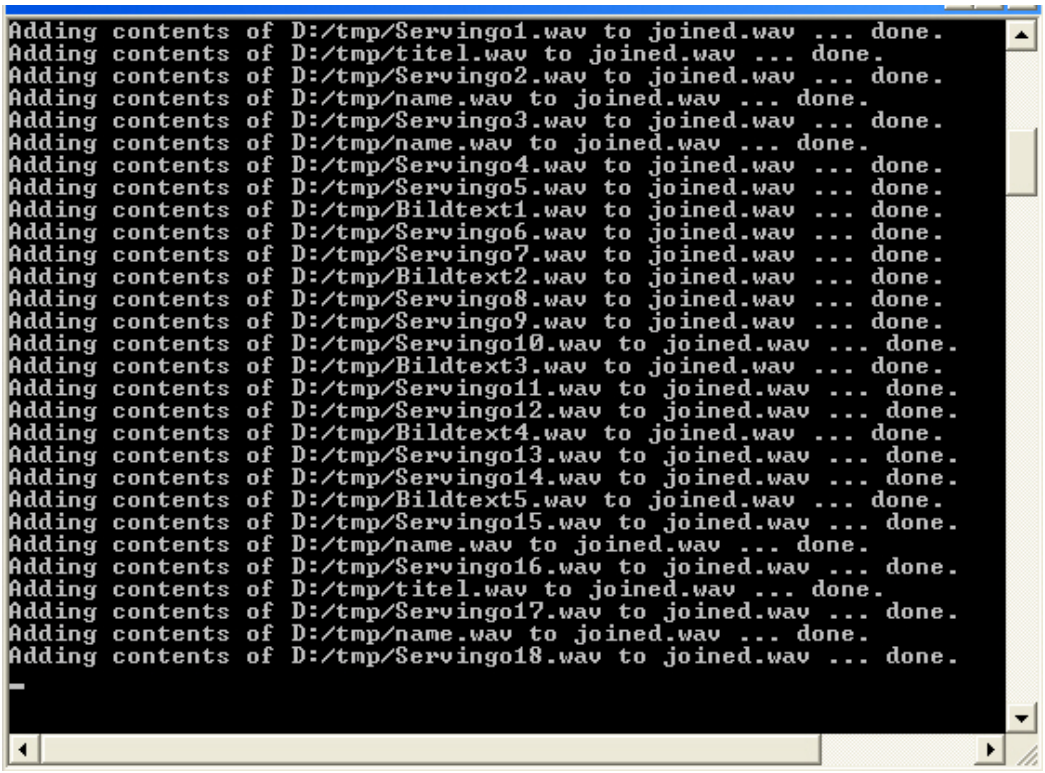


Abbildung 4.8: Beispiel WAVE-Kombination

Das Ergebnis ist nun eine WAVE-Datei, die den gesamten Dialog einschließlich der generierten WAVE-Dateien darstellt. Durch das Frame-genaue Generieren und das logische Zusammenfügen entspricht sie exakt der Länge der nachfolgend zu kombinierenden Video-Datei und ist außerdem zu den Mundbewegungen der Charaktere im Dialog passend.

Als nächstes werden die einzelnen AVI-Dateien zu einem ganzen Film verbunden. Da es in Blender nicht möglich ist, die heraus geschriebenen Dateien direkt über Blender mit bestimmten Namen zu versehen, sind diese mit Hilfe von Python in alphabetischer Reihenfolge vorher umbenannt worden, um diese nun an dieser Stelle korrekt verbinden zu können. Zum Verbinden wird der anfangs erwähnte MEncoder benutzt, der nicht nur zum Umwandeln von Daten in andere Formate genutzt wird, sondern auch zum Bearbeiten von Dateien.

```
cmd="D:\mencoder.exe -nosound -ovc copy C:/tmp/*.avi -o
     D:/tmp/OUTPUT.avi"
os.system(cmd)
```

Durch diesen Befehl werden alle AVI-Dateien, die sich in dem entsprechenden Ordner befinden, zu der Film-Datei OUTPUT.avi verbunden. Außerdem ist es auch möglich, bereits in diesem Befehl eine Konvertierung zu einem bestimmten Format durchführen zu lassen, da MEncoder auch die Codec-Bibliothek von FFmpeg zur Verfügung steht. Dies wird später am Beispiel der .3gp-Konvertierung demonstriert.

Die nun vorliegende Video- und Audio-Datei wird mittels FFmpeg zu einem kompletten Film kombiniert.

```
cmd="D:\ffmpeg.exe -i D:/tmp/OUTPUT.avi -i C:/tmp/joined.wav
     D:/SERVINGO.avi "
        os.system(cmd)
```

Das Werkzeug FFmpeg bietet wiederum selbst eine Fülle an Einstellungsmöglichkeiten und Konvertierungsmethoden, die am besten aus der Dokumentation auf der FFmpeg Homepage ersichtlich sind [24]. Nach diesem Schritt liegt nun der fertige Film mit der passenden Audio-Spur vor und kann von Server-Programmen weitergeleitet oder verwaltet werden. In der Abbildung 4.9 sieht man eine typische Ausgabe des Ablaufes einer solchen Kombinierung.

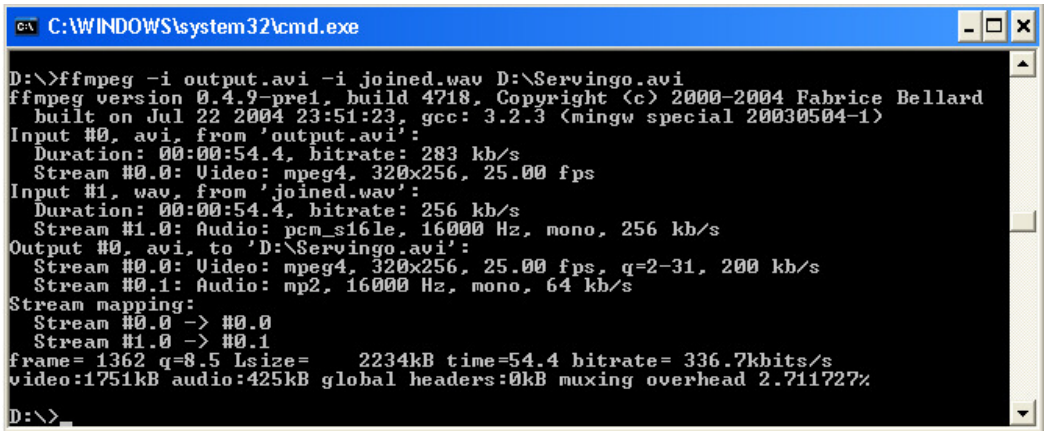


Abbildung 4.9: Beispiel Video- und Audio-Kombination

Bei der beschriebenen Kodierung mit Xvid MPEG-4 Codec und WAVE-Dateien ergab sich am Ende eine Gesamtgröße des Films zwischen 1,5 und 2 MB, die den Vorgaben an das Projekt gerecht wurde. Da FFmpeg eine große Anzahl an Kodierungsverfahren von Audio- und Video-Dateien bietet, ist es den zukünftigen Projekt-

Betreuern überlassen, die für Ihre Bedürfnisse am besten geeignete Kombination zu verwenden.

4.3.4 Ausgabe auf mobilen Endgeräten

Eine wichtige Anforderung an das Servingo-Projekt war es, die Benutzung des Portals, sowie die Ausgabe des fertigen Films auf mobilen Endgeräten, also Mobiltelefonen oder PDAs, zu ermöglichen. Hierzu müssen diese MMS-fähig (Multimedia Message Service) sein und das für solche Anwendungen geläufige .3gp-Format wiedergeben können.

Das .3gp-Format ist ein von der 3rd Generation Partnership Project (3GPP) [25] spezifiziertes Format, welches es ermöglicht, insbesondere Video-Dateien, die besonders stark komprimiert sind, auf mobilen Geräten abzuspielen. Es ist dem MPEG-4-Format sehr ähnlich und unterstützt als Video-Codec MPEG4 [26] und H.263 [26] und als Audio-Codec AMR [27] und AAC [28]. Als Bildgröße sind die Formate 176x144 und 128x96 Pixel zugelassen, obwohl aktuelle Mobiltelefon-Modelle bereits eine Bildgröße von bis zu 320x240 Pixel unterstützen.

Um einen Film in das gewünschte .3gp-Format umzuwandeln, wird ebenfalls die FFmpeg-Plattform verwendet, da diese alle dazu benötigten Werkzeuge zur Verfügung stellt. Im ersten Schritt wird die AVI-Datei mit Hilfe von MEncoder auf die QCIF (Quarter CIF (Common Intermediate Format)) Auflösung (176x144 Pixel) herunter skaliert und die Tonspur entfernt. Als Video-Codec wird der MPEG4-Codec benutzt und über den Befehl *-ops* die Bildfrequenz auf 12 Frames pro Sekunde reduziert, was für die Darstellung auf einem Mobiltelefon ausreichend ist. Die so umkodierte Video-Information wird in der Datei movie.avi zwischen gespeichert.

```
cmd="D:\mencoder.exe SERVINGO.avi –nosound –ovc lavc –lavcopts
     vcodec=mpeg4 –vop expand=176:144,scale=176:-2 –o mo
     vie.avi –ofps 12"
os.system(cmd)
```

Mittels des MPlayers wird nun die Tonspur aus dem ursprünglichen Film extrahiert und zwischen gespeichert. Hierbei gibt es eine Vielzahl von Einstellungsmöglichkeiten, um die später gewünschte Qualität zu erreichen. Der nachfolgende Befehl z. B. extrahiert die Audio-Informationen und wandelt (*resample*) die Frequenz auf 8000 Hz, um den Vorgaben des AMR Audio-Codecs zu entsprechen. Anschließend wird die Lautstärke (*volume*) um 4 dB erhöht (Lautstärkeanhebung für eine bessere Wiedergabe auf Mobiltelefonen) und die so gewandelte Audio-Information automatisch in der Datei audiodump.wav gespeichert.

```
cmd="C:\tmp\mplayer.exe -vo null -ao pcm -af resam
     ple=8000,volume=+4db:sc SERVINGO.avi"
os.system(cmd)
```

Nun folgt die eigentliche Umwandlung zu .3gp, die automatisch über FFmpeg erfolgt. Der nachfolgende Aufruf verbindet die Video-Datei movie.avi und die Audio-Datei audiodump.wav automatisch zu der Servingo.3gp Datei, wobei die Bitrate für die Video-Spur (*-b*) auf 48 KBit/s und für die Audio-Spur (*-ab*) auf 12 KBit/s eingestellt wurde.

```
cmd="D:\ffmpeg\ffmpeg -i movie.avi -i audiodump.wav -b 48 -ac
     1 -ab 12 -map 0.0 -map 1.0 SERVINGO.3gp"
os.system(cmd)
```

Diese Einstellungen haben eine ca. 300 KB große .3gp-Datei zur Folge, die sich dadurch sehr gut zum Versenden an mobile Endgeräte eignet. Da die Steuerung dieser Prozesse über die Kommandozeile erfolgt, ist es später sehr einfach möglich, neue Formate oder Kodierungsmethoden mittels Python-Befehlen zu ergänzen oder zu verändern.

4.4 Zusammenfassung

In diesem Kapitel sind die für die automatisierte Generierung benötigten Komponenten, die Spracherzeugung und der Ablauf der automatisierten Generierung beschrieben worden. Durch das behutsame Abstimmen der einzelnen Module entstand eine autark funktionierende Plattform, die es dem Anwender ermöglicht, eine automatisierte 3D-Generierung anhand von XML-Daten durchzuführen. Die FFmpeg Software erlaubt es außerdem, über ihre Vielzahl an Einstellungsmöglichkeiten, die Ausgabeformate an die jeweils benötigte Anwendung anzupassen.

5 Fazit und Ausblick

Im Rahmen dieser Diplomarbeit wurde ein System zur automatisierten Generierung von 3D-Szenen entwickelt, welches es dem Benutzer erlaubt, seine persönlichen Daten in 3D-Geschichten umzuwandeln und diese in Form eines animierten Filmes weiter zu verwenden. Dieses Kapitel bietet einen kurzen Überblick über die verschiedenen Teilbereiche der Diplomarbeit und fasst sie abschließend zusammen.

Nach der Beschreibung der Zielsetzung und der Vorstellung des Projekts wurde der Aufbau des zu erstellenden Moduls StoryGenerator näher vorgestellt. Dieser beinhaltete anfangs die Erstellung der 3D-Komponenten, die, wie das gesamte Projekt, im Bezug zur kommenden Fußballweltmeisterschaft 2006 stehen. Als eine besondere Herausforderung hierbei erwies sich die Realisierung der dynamisch anpassbaren Szenen, die erst die Anpassung an die vom Benutzer eingegebenen Texte ermöglichen.

Die nächste Phase des Projekts war die Umsetzung der automatisierten Generierung der vom Server gelieferten Daten, die den Hauptteil dieser Diplomarbeit bildete. Hierzu wurden zuerst die für diesen Abschnitt benötigten Komponenten vorgestellt und ihre Anbindung an das Modul erläutert. Eine besondere Rolle hierbei spielte die Programmiersprache Python und ihre frei verfügbaren Module, durch die es erst möglich wurde, die Blender-Szene anzusteuern und somit zu automatisieren. Die Server-Anbindung mittels XML stellt dem StoryGenerator eine der gängigsten Formen der Parameterübergabe zur Verfügung und macht ihn somit auch für andere Projekte interessant. Die automatisiert ablaufende Spracherzeugung über die Software AT&T Natural Voices mit ihrer Vielzahl an Stimmen und Sprachen erlaubt es zukünftig neue Sprachen bequem zu integrieren. Durch das Verwenden der FFmpeg Plattform zur Bearbeitung und Konvertierung der Ausgabeformate, stehen eine Vielzahl von Codierungsmöglichkeiten zur Verfügung, die es erlauben, die Ausgabeformate an die jeweilige Anwendung oder Ausgabegerät anzupassen. Bei den verwendeten Modulen handelt es sich bis auf die Software AT&T Natural Voices, um frei verfügbare und kostenlose Software, wodurch es möglich ist eine solche Anwendung ohne große finanzielle Mittel zu betreiben.

Der Abschluss dieser Diplomarbeit stellt ein funktionsfähiges System zur Verfügung, welches in der Zukunft weiter entwickelt und erweitert werden kann. Denkbar sind z. B. Implementierungen von weiteren Länderfahnen und anderen Frisuren oder neue Sprachen und Stimmen, um Fans weiterer, an der Fußballweltmeisterschaft teilnehmender Länder einzubinden. Durch die Einbindung von anderen Benutzerdaten wie z. B. Video-Dateien von Mobiltelefonen oder Audioaufzeichnungen während der Fußballspiele kann die Interaktivität weiter gesteigert werden. Die Verknüpfung mit anderen Anwendungen des Servingo-Portals ist ein weiteres denkbares Projekt für die zukünftigen Programmierer.

Außerdem ist es in der Zukunft möglich, den StoryGenerator durch das Modifizieren der Blender-Szene und der XML-Daten an andere Veranstaltungen und Anwendungen anzupassen. Beispielsweise könnte der StoryGenerator in Museen eingesetzt werden, um den Besuch eines Benutzers zu dokumentieren. Dieser könnte dann Photos der schönsten Ausstellungsstücke auswählen und daraus eine Geschichte generieren lassen. Außerdem könnten z. B. Lehrer aus einem Datensatz an Lehrmaterialien, die zu ihrem Lehrplan passenden Filme generieren lassen und diese an ihre Schüler versenden. Somit ist es in der Zukunft möglich, den StoryGenerator auf vielfältige Art und Weise zu nutzen.

6 Literaturverzeichnis

[1] Konsortialpartner - servingo Projekt
http://www.servingo.org/partner.html (10. Januar 2006)

[2] Servingo Homepage
http://www.servingo.org (8. Januar 2006)

[3] Personalisiertes Portal - servingo Projekt
http://www.servingo.org/projekt/portal.html (8. Januar 2006)

[4] Zentrum für Graphische Datenverarbeitung e.V. in Darmstadt (ZGDV)
http://www.zgdv.de (10. Januar 2006)

[5] blender3d.org :: Features
http://www.blender3d.org/cms/Features.155.0.html
(10. Januar 2006)

[6] Lutz, M.; Malerczyk, C.: Geometrische Modellierung mit Kurven und Flächen, FB MND, Fachhochschule Gießen - Friedberg SS 2004 (nicht veröffentlichtes Skript)

[7] Pottmann, H.: CAGD SS 2005,
http://www.dmg.tuwien.ac.at/nbkdir/leopoldseder/ppt/cagdss05.pdf (10. Januar 2006)

[8] DMA - Digital Media for Artists - Kunstuniversität Linz
http://www.dma.ufg.ac.at/dma/dma/pane/206.346/module/188077 (8. Januar 2006)

[9] blender3d.org :: Documentation
http://download.blender.org/documentation/BlenderManualIen.23.html.tar.gz
BlenderManualIen.23.pdf, S.169 (8. Januar 2006)

[10] AT&T Natural Voices - Homepage
http://www.naturalvoices.att.com (10. Januar 2006)

[11] Python Programming Language
http://www.python.org (10. Januar 2006)

[12] Lutz, M.; Ascher, D.: Einführung in Python.
Köln, O'Reilly 2000

[13] wxPython Homepage
http://www.wxpython.org (8. Januar 2006)

[14] SourceForge.net: Python/XML
http://sourceforge.net/projects/pyxml (10. Januar 2006)

[15] SPE - Stani´s Python Editor
http://www.stani.be/python/spe (10. Januar 2006)

[16] Winpdb - An Advanced Python Debugger
http://www.digitalpeers.com/pythondebugger
(8. Januar 2006)

[17] blender3d.org :: Documentation
http://www.blender3d.org/documentation/240PythonDoc/index.html (10. Januar 2006)

[18] Christopher A. Jones & Fred L. Drake, Jr.: Python & XML.
Köln, O'Reilly 2002

[19] W3C Document Object Model
http://www.w3.org/DOM (10. Januar 2006)

[20] Document Object Model - Wikipedia
http://de.wikipedia.org/wiki/Document_Object_Model
(8. Januar 2006)

[21] FFmpeg Homepage
http://ffmpeg.sourceforge.net/index.php (10. Januar 2006)

[22] MPlayer - The Movie Player
http:/www.mplayerhq.hu (10. Januar 2006)

[23] shntool Homepage
http://www.etree.org/shnutils/shntool (10. Januar 2006)

[24] FFmpeg Documentation
http://ffmpeg.sourceforge.net/ffmpeg-doc.html#SEC16
(10. Januar 2006)

[25] 3GPP Homepage
http://www.3gpp.org (10. Januar 2006)

[26] MPEG-4 description
http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm (8. Januar 2006)

[27] Global System for Mobile Communications - Wikipedia
http://de.wikipedia.org/wiki/Global_System_for_Mobile_Communications#Adaptive_Multirate_Codec_.28AMR.29
(8. Januar 2006)

[28] Advanced Audio Coding - Wikipedia
http://de.wikipedia.org/wiki/Advanced_Audio_Coding
(8. Januar 2006)

Anhang A: Storyboard

In welcher Stadt warst du?	Welches Fußballspiel hast du dort erlebt?	Was war das lustigste Erlebnis?	Was wird dir am meisten in Erinnerung bleiben?	Was hast du beim Sightseeing gesehen?
				
Hallo und herzlich willkommen zu „___Titel___“ Heute zu Gast bei uns ist „___Name____“	*Hier habe ich „__Bildtitel (Spiel)__“ gesehen.*	*Das hier war echt lustig: „__Bildtitel (Ereignis)__“*	*Etwas ganz Besonderes war „__Bildtitel (Ereignis)_____“*	*Wir haben uns auch ein bisschen die Stadt angeguckt.* *Das hier ist „__Bildtitel (Ort)___“*
Hallo. „__Name__“, du hast uns ein paar persönliche Eindrücke mitgebracht. Vielleicht erzählst du etwas dazu. *Also hier war ich in „__Bildtitel (Stadt)_____* Ah, welches Fußballspiel Hast du denn dort erlebt?	Oh, das war sicher aufregend.	Was für ein Spaß.	Das würde ich auch nicht vergessen.	Sehr imposant! Vielen Dank, „__Name_“, für Dein Kommen. Sie sahen „_Titel___“ von „__Name__“. Tschüss, bis zum nächsten Mal!

Anhang B: Inhalt der CD-Rom

Die CD-Rom enthält eine digitale Version dieser Diplomarbeit und Beispielfilme der Ausgabe des StoryGenerators, sowie einen Film des animierten Logos. Ebenfalls enthalten sind die Blender-Szenen und die Python Skripte, die für den StoryGenerator verwendet wurden.

Verzeichnisstruktur der CD-Rom:

- Ordner Blender:
  Blender Szenen vom StoryGenerator und dem Logo

- Ordner Diplomarbeit:
  Digitale Version der Diplomarbeit

- Ordner Film:
  StoryGenerator Filme in verschiedenen Codierungen

- Ordner Skript:
  Verwendete Python Skripte