Diplomarbeit

Thema der Diplomarbeit:

Erstellen eines Viewers für Motion Capture Daten auf Basis von OpenGL

Unternehmen, in dem die Diplomarbeit durchgeführt wurde:
metricminds GmbH & Co. KG in Frankfurt/Main

Diplomandin:	Simone Houdek
Referentin:	Prof. Dr.-Ing. Dipl.-Math. M. Lutz
Korreferent:	Dipl.-Math. C. Malerczyk
Betreuer der Firma metricminds:	P. Weiß

Wintersemester 2003/04

Fachhochschule Gießen-Friedberg
Bereich Friedberg
Fachbereich IEM
Fachrichtung Medieninformatik

Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig und nur unter Zuhilfenahme der angegebenen Quellen und Hilfsmittel angefertigt habe. Die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren wurden an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet.

……..….. ……..

Ort, Datum Simone Houdek

Inhaltsverzeichnis

1 Einleitung und Thema der Diplomarbeit

In der heutigen Zeit kommen in Kinofilmen, Videos, Computerspielen, usw. immer mehr 3D-Animationen zum Einsatz. Dabei sollen die Charaktere so schnell und so gut wie möglich animiert werden. Eine Möglichkeit, das zu realisieren, ist der Einsatz von Motion Capture-Aufnahmen. Hierbei werden die Bewegungen von Menschen oder Tieren aufgezeichnet, um sie anschließend auf im Computer erstellte Charaktere zu übertragen. Dies erspart einen großen Teil der Zeit, den man aufbringen müsste, um alles manuell zu animieren.

Bis die Motion Capture-Aufnahmen so weit bearbeitet sind, dass sie auf ein 3D-Modell angewendet werden können, sind jedoch noch weitere Arbeitsschritte durchzuführen. Die Bewegungen müssen beispielsweise bereinigt oder in ein, für das verwendete 3D-Animationsprogramm, passendes Format umgewandelt werden. Oft ist es nötig, dass Bewegungsdateien diese Arbeitsschritte mehrfach durchlaufen, bis der Kunde mit dem Resultat zufrieden ist.

Das Ziel dieser Arbeit bestand darin, einen Viewer zu erstellen, der auf einer OpenGL-Freeware-Engine basiert. In diesen Viewer kann der Kunde die Motion Capture-Daten laden, um sie auf seinem Charakter zu sehen. Weiterhin gibt es die Möglichkeit, die vorhandenen Fehler mit Hilfe einer Paint-Funktion zu markieren und mit Hilfe einer Textfunktion zu beschreiben.

Der Schwerpunkt dieser Arbeit liegt dabei darauf, die vorhandene Freeware-Engine so zu modifizieren, dass die entsprechenden 3D-Modelle geladen und die Bewegungsdaten direkt eingelesen werden können. Bei den 3D-Modellen beschränkt sich die Arbeit auf Modelle, die mit dem Animationsprogramm 3D Studio Max generiert wurden, da die verwendete Engine bisher nur mit diesem Programm

kompatibel ist. Weiterhin soll der Viewer so verändert werden, dass man darin malen und Text hinzufügen kann.

Mit diesem Viewer soll die Kommunikation zwischen Auftragnehmer und Auftraggeber vereinfacht werden, so dass besser auf Wünsche und Anmerkungen des Kunden eingegangen werden kann.

2 Grundlagen

2.1 Einleitung

Dieses Kapitel befasst sich mit den, in dieser Diplomarbeit verwendetet Hilfsmitteln. Dabei werden Hintergrundinformationen zu den Bereichen 3D-Animation und Motion Capture geliefert und auf die verwendeten Hilfsmittel, wie OpenGL, die C++-Bibliothek cal3d und Dateiformate (.c3d), eingegangen. Zum Abschluss wird noch kurz beschrieben, wie das 3D-Animationsprogram 3D Studio Max und das dazugehörige Plug-In Character Studio funktioniert.

2.2 3D-Animation und Motion Capture

Der Aufwand, der aufgebracht werden muss, um eine 3D-Animation zu erstellen, hängt von einigen äußeren Umständen ab. Zum einen ist es wichtig, wie lang die Animation dauern soll und was in der Animation vorkommt. Aber es ist auch wichtig, wie komplex das zu animierende Modell ist, wie viele Knochen es hat und wie viele Modelle in der Animation vorkommen sollen.

Selbst wenn es sich bei der Animation nur um eine einfache Bewegung eines einzelnen Charakters handelt, die nicht besonders lang werden soll, ist es ein großer Aufwand, das Modell per Hand zu animieren. Der Grund dafür ist, dass jeder Knochen des Charakters einzeln animiert werden muss und außerdem muss dabei darauf geachtet werden, dass die Animation real aussieht. Würde man die 3D-Modelle für z.B. ein Computerspiel alle per Hand animieren,

würde es sehr lange dauern, bis das Spiel fertig ist und verkauft werden kann.

Hier kommt die Technik des Motion Capture ins Spiel. Da bei Motion Capture-Aufnahmen die Bewegungen von Menschen oder Tieren aufgezeichnet werden, ist von vornherein gegeben, dass die Bewegungen realistisch sind. Auch ist der Zeitaufwand hier nicht so hoch, als würde der Charakter manuell animiert werden. Zwar müssen die Daten noch benannt (gelabelt), von Fehlern bereinigt (gecleant) und in das entsprechende Format des 3D-Animationsprogramms umgewandelt werden, der Prozess würde aber ohne Motion Capture wesentlich länger dauern.

Beim Motion Capture werden die Bewegungen von mehreren analogen Kameras (bei metricminds sind es 18) mit 120 Bildern pro Sekunde aufgezeichnet. Die Kameras haben einen Ring mit Dioden um die Linse (siehe Abb. 1).

Abb. 1: Eine Kamera von der Firma Vicon

Diese senden Licht aus, das von den Markern am Darsteller reflektiert wird und dann von den Kameras aufgezeichnet wird. Marker sind reflektierende Kugeln, die an leicht zu identifizierenden Stellen des Körpers angebracht werden (siehe Abb. 2 und 3).

Abb. 2: Darsteller mit Markern von vorne

Abb. 3: Darsteller mit Markern von hinten

Bevor aufgezeichnet werden kann, muss zum einen der Darsteller mit Markern versehen werden und zum anderen müssen die Kameras kalibriert werden. Das Kalibrieren ist dazu gut, die Position der Kameras im 3D-Raum und deren Orientierung zum Ursprung zu bestimmen, um später die Position der Marker genau bestimmen zu können. Nach dem Aufzeichnen der Bewegungen müssen die Bewegungsdaten rekonstruiert werden. Dies geschieht, indem die Position von jedem Marker in jedem Frame rekonstruiert wird und diese dann zu einer Bewegungskurve zusammengefügt werden. Das Rekonstruieren der Daten geschieht mit Hilfe der Software Workstation der Firma Vicon (siehe Abb. 4) [1].

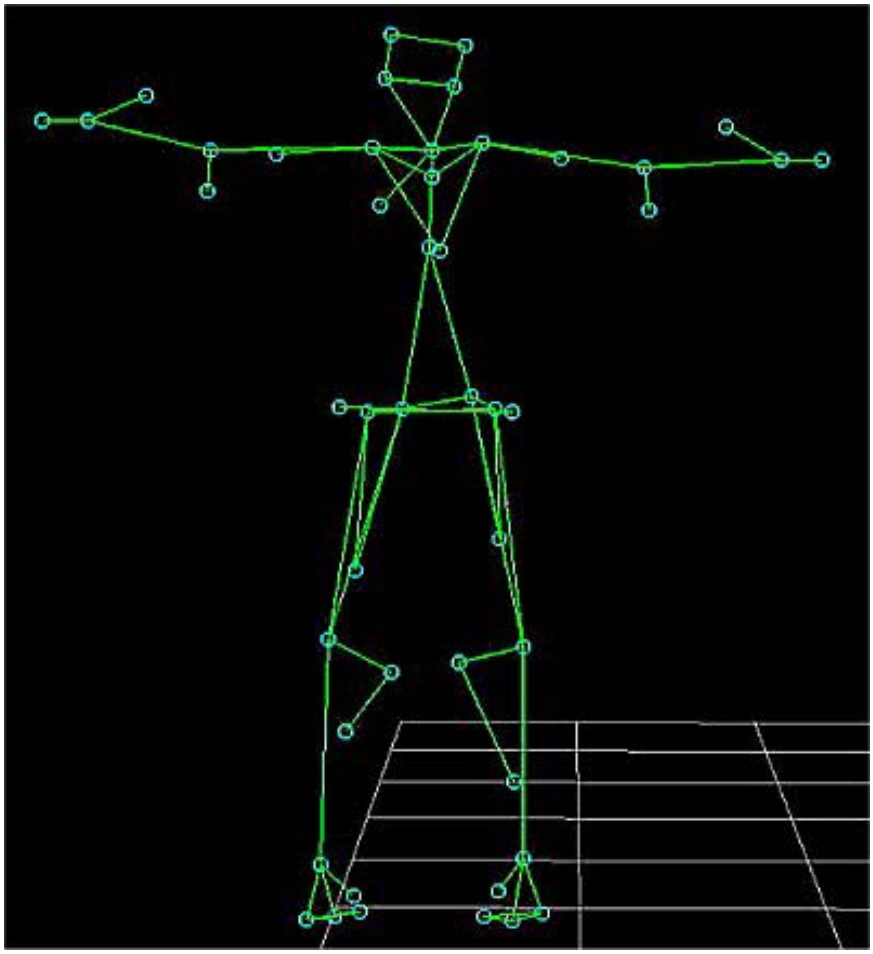

Abb. 4: Rekonstruierte und benannte Bewegungsdaten

Um nach dem Rekonstruieren die Daten weiterverarbeiten zu können, müssen die Marker benannt werden. Diesen Vorgang bezeichnet man als „labeln“. Wenn die Daten fertig benannt sind, werden sie im Format .c3d abgespeichert. Im Anschluss daran müssen sie noch von Fehlern bereinigt werden. Das wird bei metricminds mit dem Programm Diva von der Firma House of Moves gemacht.

Die Daten werden „gesäubert“, indem sie zuerst ein Skript durchlaufen, das vorhandene Fehler berichtigt und danach noch eventuell vorhandene Fehler manuell beseitigt werden. Fehler, die auftreten können, sind zum einen, dass die Bewegungskurve für einen gewissen Zeitraum keine Daten hat. Einen solchen Fehler bezeichnet man als Gap (siehe Abb. 5).

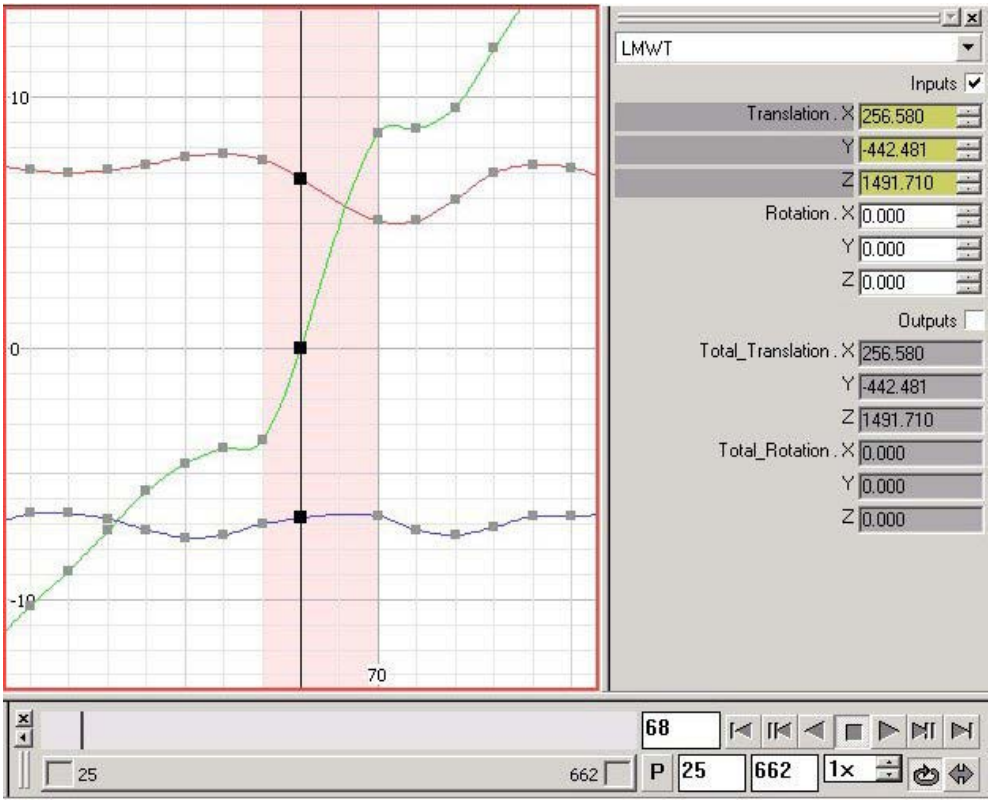

Abb. 5: Fehlende Daten in der Bewegungskurve (Gap)

Er kann auftreten, wenn ein Marker für eine gewisse Zeit für die Kameras nicht zu sehen war. So eine Lücke in der Bewegungskurve kann ausgebessert werden, indem man mit der Hilfe von mindestens zwei anderen Markern, die mit dem fehlerhaften Marker in Verbindung stehen, die Bewegung rekonstruiert.

Ein weiterer Fehler, der auftreten kann ist, wenn ein Marker eine Spitze (Spike) in der Bewegungskurve hat (siehe Abb. 6).

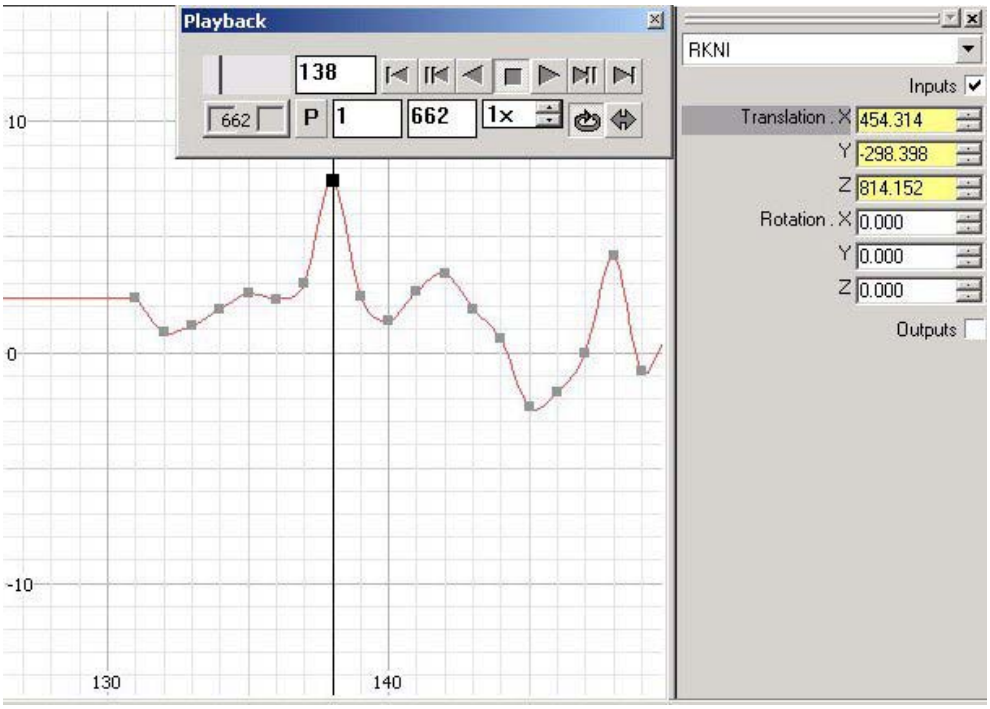

Abb. 6: Eine Spitze in der Bewegungskurve (Spike)

Das kann passieren, wenn ein Marker von den Kameras nicht richtig erfasst wurde und plötzlich von seiner Position an eine andere springt und wieder zurück. Das wird verbessert, indem für den entsprechenden Zeitraum die Bewegungsinformationen gelöscht und im Anschluss wie bei einem Gap rekonstruiert werden. Eine dritte Möglichkeit für Fehler ist das Zittern von Markern. Durch das Rekonstruieren der Bewegungsdaten kann es passieren, dass die Marker nicht immer an der gleichen Position sind, sondern leicht hin

und her springen. In einer Bewegung sieht das aus, als würde der Marker zittern. Im Allgemeinen ist dieser Fehler aber behoben, nachdem das bereits erwähnte Skript durchlaufen wurde. Sind die Daten fertig, können sie in des Format umgewandelt werden, mit dem weitergearbeitet werden soll, beispielsweise .csm oder .bvh für 3D Studio Max. Dieses Format wird dann auf den dafür vorgesehenen Charakter geladen und noch mal auf Fehler überprüft. Sind die Dateien in Ordnung, werden sie dem Kunden übergeben, der seinerseits überprüft, ob er mit den aufgezeichneten Bewegungen zufrieden ist oder ob eventuell noch Fehler übersehen wurden. Sollte er nicht zufrieden sein, teilt er das mit und die Daten müssen Teile des Prozesses, vielleicht sogar die gesamten Bearbeitungsschritte, noch einmal durchlaufen.

Der Einsatz von Motion Capture-Aufnahmen vereinfacht die 3D-Charakter-Animation im Vergleich zur manuellen Animation erheblich und verkürzt die aufzuwendende Zeit um einiges.

Die Diplomarbeit setzt an der Stelle an, nachdem die Bewegungsdaten benannt wurden und bevor sie gesäubert werden. Der Grund dafür ist, da das dort verwendete Dateiformat im Prinzip das erste Format ist, mit dem gearbeitet werden kann. Dieses Dateiformat soll dem Kunden schließlich über einen Viewer, der auf OpenGL basiert, zur Verfügung gestellt werden.

2.3 OpenGL

OpenGL ist ein Graphikstandard, der unabhängig vom verwendeten Betriebssystem eingesetzt werden kann. Diese Graphikschnittstelle nutzt unterschiedliche Befehle, um interaktive dreidimensionale Applikationen zu erstellen. Durch den Einsatz von einfachen Grafikelementen wie Linien, Flächen oder Punkten werden mit deren Kombination komplexe Körper erzeugt, die zusammen eine 3D-Szene bilden. Aufbauend darauf kann man mit OpenGL aber auch aus einfachen Grundkörpern wie Kugeln und Zylindern eine 3D-Szene generieren. Um OpenGL Programme zu erzeugen, kann man fast jede Entwicklungsumgebung verwenden. Dazu sind entsprechende Bibliotheken notwendig.

Zum einen gibt es die Standard-OpenGL-Graphikbibliothek (GL = Graphic Library). Diese umfasst ca. 150 Befehle und bildet die Basis von OpenGL. Mit Hilfe der Bibliothek GL werden im Allgemeinen einfache Elemente, wie Punkte, Linien und Polygone, erstellt. Des Weiteren wird die OpenGL-Utlity-Bibliothek (GLU) verwendet. Die GLU-Befehle werden dafür eingesetzt, um 3D-Körper, wie Kugeln, Zylinder, usw. zu generieren. Das geschieht mit Hilfe von Befehlen, die oft eine Zusammenfassung von GL-Basisbefehlen sind. Durch die GLU-Befehle kann der Anwender aber auch gekrümmte Flächen (sog. Nurbs) erzeugen, was mit GL noch nicht möglich ist. Die letzte der drei OpenGL-Bibliotheken ist die Graphic-Library-Toolkit-Bibliothek (GLUT). Diese gehört nicht zum standardmäßigen Bibliotheksumfang von OpenGL und ist eine Vereinfachung zur OpenGL-Nutzung [2, 3, 4].

Eine Windows-Applikation benutzt OpenGL genauso wie GDI oder andere APIs. OpenGL selbst arbeitet direkt mit dem Display-Driver,

der direkt mit der darstellenden Hardware interagiert (siehe Abb. 7) [4].

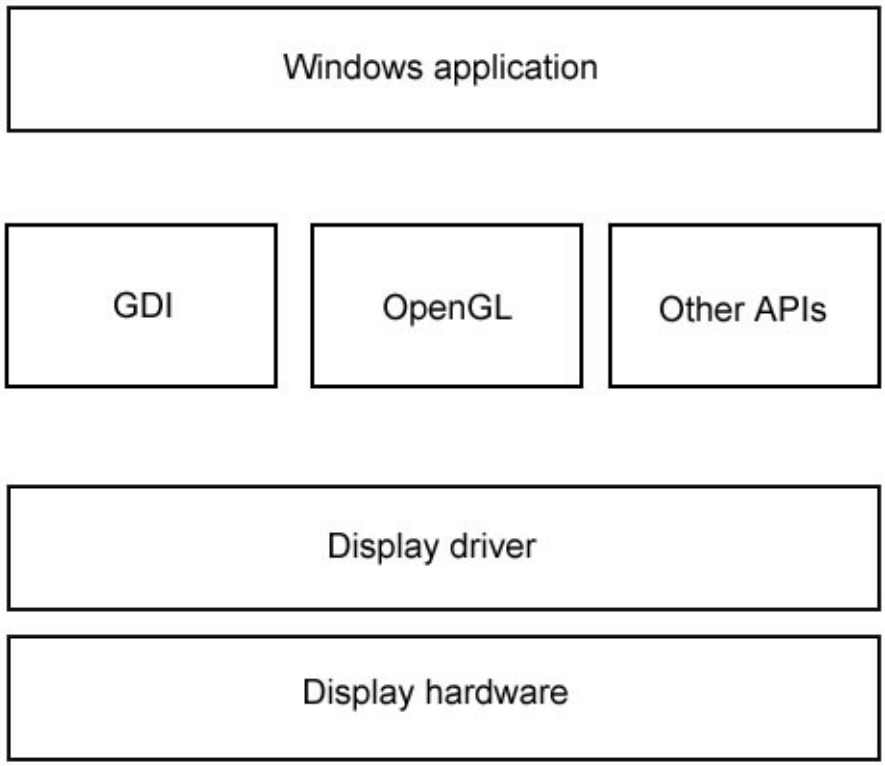

Abb. 7: OpenGL in der Windows Umgebung

Wie bereits erwähnt, ist OpenGL unabhängig von einem Betriebssystem, verwendet man es aber mit Windows, dann sind die Versionen Windows 2000 und Windows XP am geeignetsten. Der Grund dafür ist, dass bei diesen Betriebssystemen die OpenGL-Anwendung nicht durch laufende Hintergrundprogramme, Druckanweisungen, usw. beeinflusst wird. Die OpenGL-Anwendung läuft trotz solcher Nebenprozesse ohne Ruckeln weiter [2].

In dieser Diplomarbeit wurde die Anwendung mit Microsoft Visual C++ programmiert. Auch wenn die Programmierung über C++ geschieht, unterscheidet sich OpenGL trotzdem sehr von dieser

Sprache, da das in Visual C++ eingebettete OpenGL ablaufgesteuert ist und C++ eine ereignisgesteuerte Programmiersprache ist. Obwohl sich OpenGL und C++ so sehr voneinander unterscheiden, ist die Kombination der beiden gut geeignet, um interaktive, animierte Grafiksequenzen zu erzeugen.

2.4 Die OpenGL-Freeware-Engine cal3d

Bei der OpenGL-Freeware-Engine handelt es sich um den so genannten cal3d-miniviewer. Dieser basiert auf der skelettgestützten Animationsbibliothek cal3d, die in C++ programmiert und ursprünglich für WorldForge entworfen wurde, sich aber zu einer unabhängigen Bibliothek weiterentwickelt hat. Cal3d ist vom Betriebssystem unabhängig und kann in zwei Teile unterschieden werden, zum einen in die C++-Bibliothek und zum anderen in den Exporter.

Der Exporter ist im Prinzip ein Plug-in und dazu gut, dass man seine 3D-Modelle in das Format von cal3d, das die Bibliothek kennt und auch laden kann, umwandeln kann. Zur Zeit ist das aber nur für Modelle aus 3D Studio Max möglich, da der Exporter bisher nur dazu kompatibel ist. Da cal3d skelettgestützt arbeitet, ist es wichtig, dass zuerst das Skelett des Charakters exportiert wird. Im Anschluss kann dann das Exportieren des Modells an sich, der Texturen und der Animation durchgeführt werden [5].

Die C++-Bibliothek kann in eine selbst erstellte Applikation eingefügt werden. Das soll das Einbinden und Darstellen von 3D-Modellen in interaktiven Applikationen erleichtern.

Der cal3d-miniviewer ist eine Win32-Konsolenanwendung, bei der ein Fenster erzeugt wird, um animierte Modelle mit Hilfe der cal3d-Bibliothek und der OpenGL-Bibliothek GLUT anzuzeigen. Im cal3d-miniviewer ist ein fester Pfad und eine Datei vorgegeben, die geöffnet werden soll. Das geöffnete Modell befindet sich in einer Endlosschleife. Beendet wird der cal3d-miniviewer über die Tastatur-eingabe „Q“, „q“ oder „Esc“.

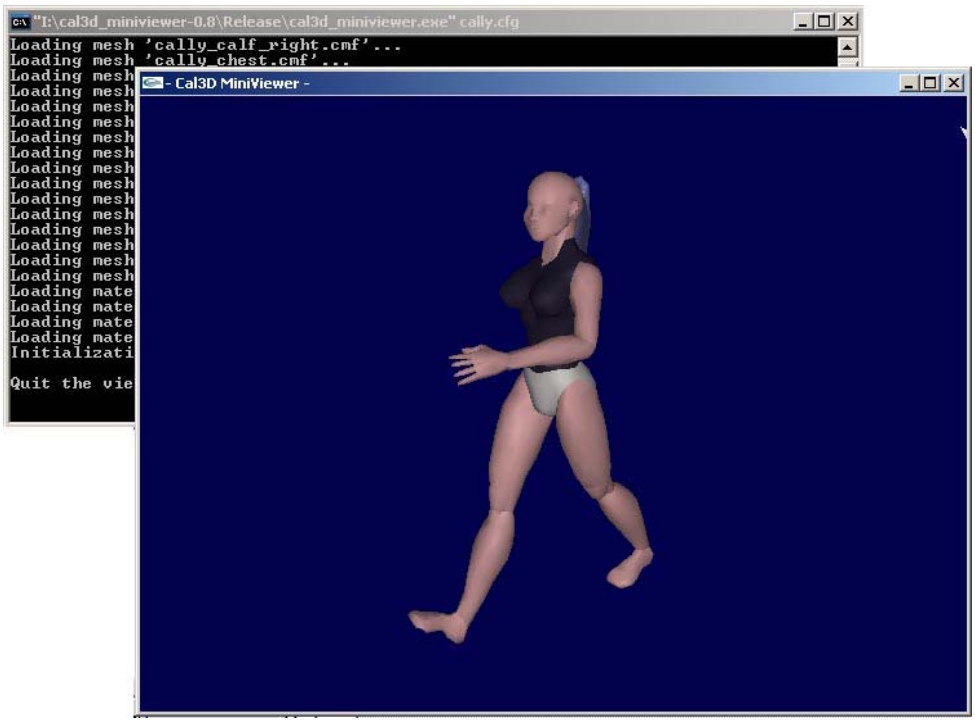

Abb. 8: Der cal3d-miniviewer

Der cal3d-miniviewer dient in dieser Diplomarbeit als Grundlage, da über cal3d bereits realisiert wurde, 3D-Modelle zu laden. So ist es möglich, Charaktere in den zu erstellenden Viewer zu laden, ohne dass auf jedes Format, beispielsweise texturemapping oder skinweights eingegangen werden muss. Es muss dann nur dafür gesorgt werden, dass das gewünschte Format (.c3d) der Bewegungsdateien geladen werden kann.

2.5 Das Dateiformat c3d

Firmen, die Motion Capture im medizinischen Bereich, in der Spieleproduktion, usw. benutzen, verwenden viele unterschiedliche Systeme. Bis vor ca. 15 Jahren war es üblich, dass die Hersteller von Motion Capture-Systemen ihre eigene Software entwickelten, die die Daten in einem eigenen, einmaligen Format abgespeichert hat. Aus diesem Grund war es nicht möglich, Bewegungsstudien, die mit einem bestimmten System erstellt wurden, mit Studien eines anderen Systems zu vergleichen. Auch war es für Nutzer nicht möglich, Daten innerhalb der gleichen Organisation oder des gleichen Netzwerks auszutauschen, wenn diese mit unterschiedlichen Motion Capture-Systemen erstellt wurden.

Deshalb kam die Überlegung auf, etwas Neues zu entwickeln. Dabei sollte die Möglichkeit bestehen, 3D-Daten und analoge Daten in einem Format zu kombinieren. Das Format sollte Informationen darüber enthalten, welche Marker (reflektierende Kugeln, die an bestimmten Punkten des Körpers angebracht sind) verwendet werden.

Außerdem wurde Flexibilität und Kompatibilität gewünscht, was bedeutet, dass neue Informationen gespeichert werden können, ohne dass ältere Daten überholt sind. Sehr wichtig war dabei auch, dass es ein öffentliches Format ist, so dass jeder Anwender Zugang auf die Daten hat, ohne vom Hersteller des Motion Capture-Systems abhängig zu sein.

Das c3d-Dateiformat ist ein praktischer Mittelweg, um 3D-Daten und analoge Daten zusammen mit einigen allgemeinen Parametern, die die 3D-Daten beschreiben, in einem Format abzuspeichern. Es besteht aus den drei Komponenten Daten, Standardparameter und

kundenspezifische Parameter. Wenn man dabei von Daten spricht, heißt das so viel, dass eine c3d-Datei eine Binärdatei ist, die 3D-Informationen und analoge Daten beinhaltet. Unter Standard-parametern versteht man die Grundinformationen, die nötig sind, um Zugriff auf die analogen und die 3D-Daten zu erhalten. Dabei ist es nötig zu wissen, wie viele Marker verwendet wurden, wie viele Frames die Bewegung umfasst, und mit welcher Frequenz die Bewegung aufgezeichnet wurde. Die kundenspezifischen Parameter sind Informationen über eine bestimmte Applikation eines Systemherstellers, über eine Testperson, usw. Wenn es sich um Aufnahmen für Bewegungsstudien oder für medizinische Zwecke handelt, kann unter anderem der Name der Testperson, deren Alter zum Zeitpunkt der Aufnahme, ihr Gewicht und beispielsweise die Beinlänge mit in der .c3d-Datei gespeichert werden.

Ein großes Ziel der Entwicklung des c3d-Formats war es, den Zugriff auf die Daten für jeden Anwender zu vereinfachen, so dass der Dateiinhalt jederzeit gelesen oder verändert werden kann. Weiterhin kann jeder, der die Dateien öffnet, den Inhalt lesen und ihn benutzen, um die Daten zu analysieren oder zu interpretieren [6, 7].

Informationen, die das c3d-Format speichert, werden so behandelt, als würden sie einer von zwei Gruppen angehören:

- physikalische Abmessungen
- Parameterinformationen

Die physikalischen Abmessungen müssen entweder numerische Daten oder 3D-Koordinaten sein. Die 3D-Koordinaten werden hier mit Informationen über die X-, Y- und Z-Position und deren Genauigkeit gespeichert und mit welchen Daten die Kameras aufge-nommen wurden [6].

Die numerischen Daten beinhalten digitalisierte analoge Informationen, die an 3D-Informationen geknüpft sind, um die genauen Werte jeder 3D-Information innerhalb der gesamten Datei ermitteln zu können.

Zusätzlich zu den physikalischen Abmessungen enthält eine c3d-Datei auch Angaben über die verwendeten Parameter, wie z.B. den Namen des Subjekts, dessen Bewegungen aufgezeichnet wurden.

Viele c3d-Dateien beinhalten sowohl analoge als auch 3D-Daten, was ein großer Fortschritt ist gegenüber den Formaten, die analoge und Videodaten getrennt voneinander speichern.

Die c3d-Dateien sind im DEC-Format gespeichert. DEC ist eine Abkürzung für Digital Equipment Corporation. Diese Firma war in der Computerindustrie tätig und benutzte das c3d-Format als Dateiformat. Diese Firma erstellte eine Applikation, die komplette automatische Berechnungen von 3D-Bewegungen ermöglichte. Die c3d-Dateien beinhalten, wie viele Marker verwendet wurden, deren Namen und die Bewegungsinformationen jedes einzelnen Markers. Weiterhin ist in der c3d-Datei angegeben, wie viele Frames die Bewegung dauert und mit welcher Frequenz die Bewegungen aufgezeichnet wurden [6, 7].

In dieser Diplomarbeit sollte das c3d-Format verwendet werden, da es im Prinzip das erste Format ist, mit dem gearbeitet werden kann und somit weit am Anfang des Bearbeitungsprozesses steht.

Bei der Einbindung des c3d-Formates traten allerdings Probleme auf, die später genauer erläutert werden. Aus diesem Grund musste eine Alternative gefunden werden, mit der gearbeitet werden kann. Bei dieser Alternative wurde entschieden, einen Umweg zu gehen, damit

die Bewegungsdaten geladen werden können und dieser Umweg führt über 3D Studio Max.

2.6 3D Studio Max und Character Studio

Egal, in welchem Bereich der Medien man sich bewegt, man stößt immer mehr auf, vom Computer generierte, dreidimensionale Welten. Eine Möglichkeit, dreidimensionale Objekte zu erstellen, ist der Einsatz von 3D Studio Max.

Dieses Programm ist ein 3D-Animationsprogramm (siehe Abb. 9), in dem unter der Benutzung zahlreicher Hilfsmittel, wie beispielsweise einfache Grundkörper, Objekte erstellt und bearbeitet werden können. Wurden alle, für eine Szene notwendigen, Objekte erstellt, bearbeitet und an die gewünschte Position gesetzt, muss eine Kamera erstellt werden, durch die die Szene betrachtet werden kann. Damit man in der Szene auch etwas sehen kann, werden Lichtquellen eingesetzt, um die Objekte passend auszuleuchten. Hierbei kann beispielsweise auch ein Schattenwurf definiert werden, über den Tiefe und Realismus in die Szene gebracht werden können. Im Anschluss müssen noch Materialien den einzelnen Objekten zugewiesen werden. Die Materialien können im Materialeditor selbst kreiert werden oder man kann bereits vorhandene Materialien laden und verwenden. Im Allgemeinen sollte man die Materialien erst nach der Beleuchtung erstellen, da das Licht das Erscheinungsbild eines Materials beeinflussen kann [8].

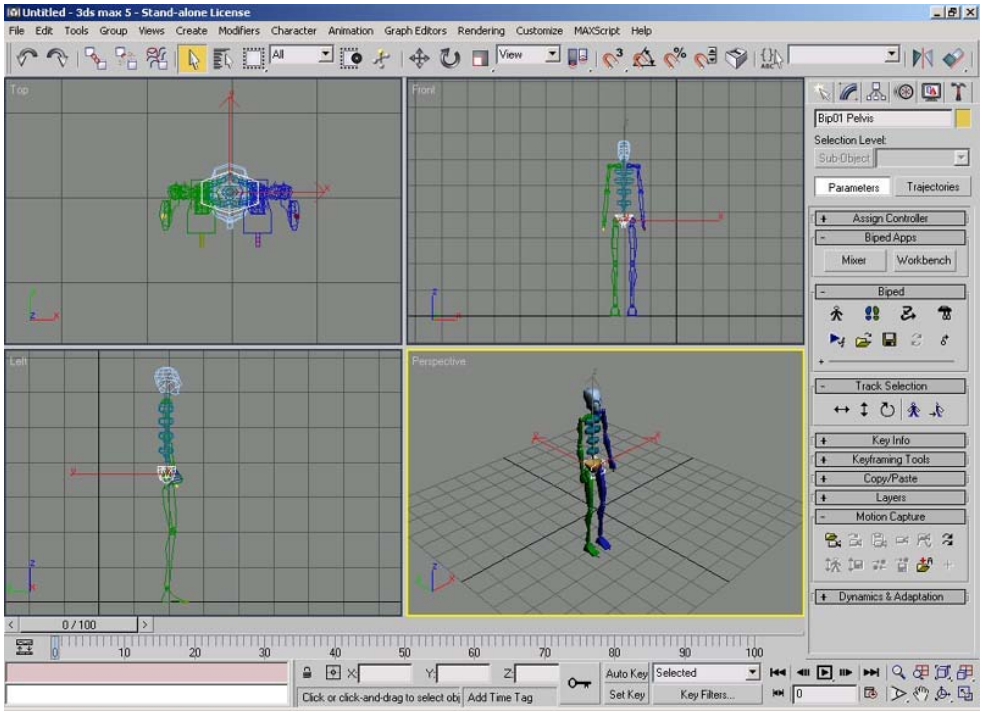

Abb. 9: Oberfläche von 3D Studio Max

Handelt es sich um eine statische Szene, ist die Bearbeitung der Szene an dieser Stelle abgeschlossen. Befinden sich in der Szene Charaktere, die Bewegungen ausführen sollen, müssen diese noch animiert werden.

An dieser Stelle kommt Character Studio zum Einsatz. Character Studio ist ein Programm, mit dem man zweibeinige Figuren erstellen und animieren kann. Die nötigen Funktionen werden über zwei Plug-In-Komponenten für 3D Studio Max zur Verfügung gestellt:

- Biped
- Physique

Die Funktion Biped automatisiert das Erstellen und Bearbeiten von zweibeinigen Charakteren. Weiterhin enthält die Funktion Hilfsmittel zum Animieren, die das Bearbeiten der Schritte, die der Charakter machen soll, das Importieren von Motion Capture-Daten und die Bewegungsüberblendung kombinieren [9].

Die Funktion Physique verbindet die Hautgeometrie mit der entsprechenden Biped- oder 3D Studio Max-Skelett-Hierarchie. Außerdem automatisiert diese Funktion das Hautverformungs-verhalten basierend auf Hauverformungsattributen und auf den animierten Knochenpositionen [9].

Für diese Diplomarbeit ist die Funktion Biped wichtig (siehe Abb. 10), da über sie Motion Capture-Daten geladen werden können.

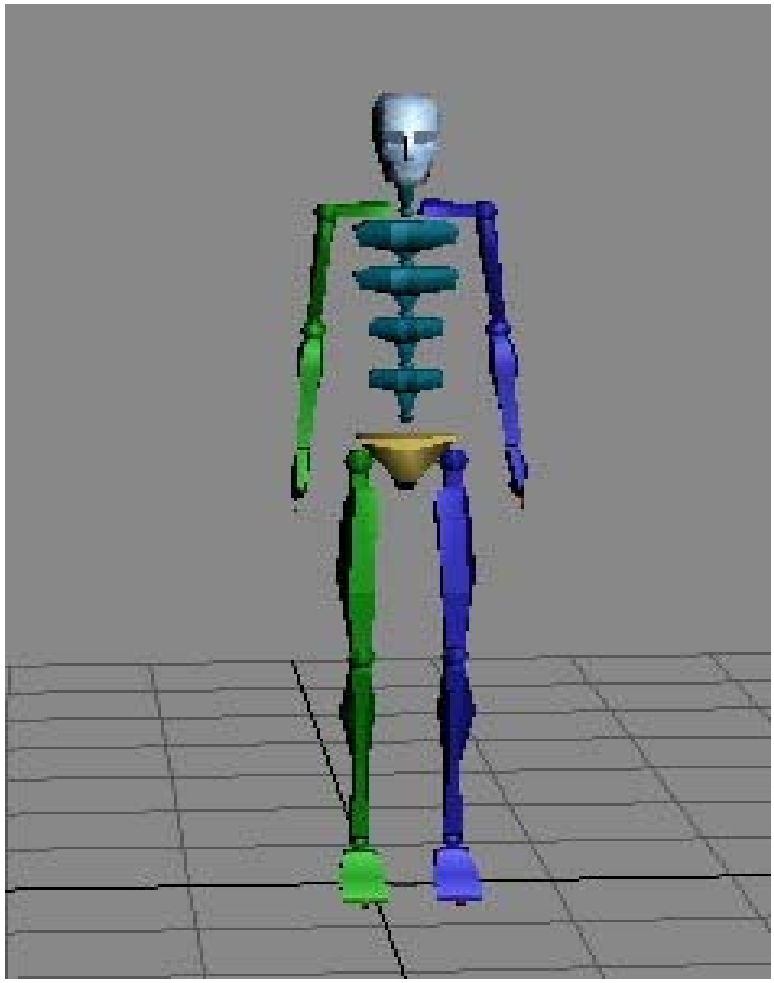

Abb. 10: Ein Biped-Skelett

Nach dem Installieren von Character Studio wird in der Befehls-palette „Erstellen“ unter dem Punkt „Systeme“ eine zusätzliche

Schaltfläche für die Erstellung von Bipeds angezeigt. Wird diese Schaltfläche aktiviert, kann mit der Maus ein Biped erstellt werden. Die Ausgangshöhe des Biped wird über die Länge der Verschiebung des Mauszeigers festgelegt. Das erstellte Biped-Skelett kann jetzt so verändert werden, dass es den Vorstellungen des Anwenders entspricht. Die ursprüngliche Breite und Tiefe bleibt dabei immer in Proportion zu der Höhe. Für die Erzeugung eines Biped können die Vorgabeeinstellungen, durch die seine Grundstruktur festgelegt ist über das Rollout „Biped erstellen“, beliebig verändert werden. Die Vorgabeeinstellungen sind für Figuren menschlicher Körper gedacht. Über diese Einstellungen kann beispielsweise festgelegt werden, ob ein Biped mit oder ohne Arme erstellt werden soll, auch die Anzahl der Finger kann festgelegt werden.

Durch das Erstellen eines Biped-Skeletts kann man schnell und einfach vollständige zweibeinige Figurenskelette erstellen. Unter der Verwendung von den parametrischen Steuerungselementen der Funktion Biped kann man die Struktur und Knochengröße des erstellten Skeletts verändern. Die angepassten Figuren können alle separat abgespeichert und erneut in anderen Szenen, auch mit anderen Bewegungen verwendet werden. So können schnell vollständig neue Figuranimationen erstellt werden.

Da es wie bereits erwähnt Probleme dabei gab, die Motion Capture-Daten direkt in den Viewer zu laden, wird ein Umweg über 3D Studio Max gegangen. Dieser Umweg ist, dass die Bewegungsdaten aus dem Format .c3d in das Character Studio Format .csm umgewandelt werden, um anschließend mit dem Exporter von cal3d exportiert zu werden, so dass sie in den Viewer geladen werden können.

Mit Character Studio können Motion Capture-Daten importiert werden. Die Dateien, um die es sich dabei handelt, sind Character

Studio-Markierungsdateien (.csm). Diese Dateien enthalten Daten für Markierungspositionen, die mit einem Motion Capture-Gerät erzeugt wurden. Die Datei beinhaltet, wann die Datei erstellt wurde, wie lang sie ist und mit welcher Frequenz sie aufgezeichnet wurde. Weiterhin sind die Angaben gespeichert, welche Marker vorhanden sind und zu jedem Marker ist für jeden Frame die Position angegeben, an der sich der Marker befindet (siehe Abb. 11).

```
$Date Wed Jul 09 14:49:09 2003

$Filename w:\desperados2\csm\M004-T03.csm
$Actor HOMie

$Comments
Motion capture produced by House of Moves LLC

$FirstFrame 1
$LastFrame 1829
$Rate 120.000000

$Order
LSHO LFIN RSHO RFIN LBWT RFWT RBWT LFWT C7 T10 CLAV ST

$Points
1   -42.325 19.824 1577.107   -710.294 63.733 1560.614
2   -42.380 19.770 1577.107   -710.348 63.624 1560.669
3   -42.434 19.715 1577.107   -710.348 63.515 1560.723
4   -42.543 19.661 1577.107   -710.348 63.405 1560.778
5   -42.598 19.606 1577.107   -710.348 63.296 1560.833
6   -42.653 19.551 1577.107   -710.348 63.242 1560.887
7   -42.707 19.497 1577.107   -710.348 63.242 1560.942
8   -42.762 19.497 1577.107   -710.403 63.296 1560.996
9   -42.871 19.442 1577.107   -710.403 63.351 1561.051
10  -42.926 19.388 1577.107   -710.403 63.460 1561.106
11  -42.980 19.333 1577.053   -710.403 63.515 1561.160
12  -43.035 19.278 1577.053   -710.403 63.624 1561.215
13  -43.089 19.224 1577.053   -710.457 63.788 1561.270
14  -43.199 19.169 1577.053   -710.457 63.897 1561.324
15  -43.253 19.114 1577.053   -710.457 64.061 1561.379
16  -43.308 19.060 1577.053   -710.457 64.170 1561.433
17  -43.363 19.005 1577.053   -710.457 64.334 1561.488
18  -43.417 18.951 1577.053   -710.512 64.498 1561.543
```

Abb. 11: Ausschnitt aus einer .csm-Datei

Die erste Zeile gibt an, wann die Datei erstellt wurde, in diesem Beispiel war das am Mittwoch, den 9. Juli 2003 um 14:49:09. Die nächste Zeile gibt den Pfad und den Dateinamen an. Darunter steht die Bezeichnung des Darstellers und in der darauf folgenden Zeile

steht ein Kommentar, mit welchem Programm die .csm-Datei erstellt wurde. Dabei wird allerdings nur der Name des Softwareherstellers angegeben und nicht der Name des Programms. In den nächsten zwei Zeilen wird die Nummer des ersten und des letzten aufgezeichneten Frames angegeben. Danach steht, dass die Rate 120 beträgt. Das bedeutet, dass die Bewegungen mit einer Frequenz von 120 Hz aufgenommen wurde. In der nächsten Zeile stehen die Namen der verwendeten Marker. Die Reihenfolge ist wichtig, denn unter den Markern stehen in der gleichen Reihenfolge die x-, y- und z-Positionen der Marker im 3D-Raum. Die Zahlen, die in der ersten Spalte von jeder Zeile stehen, geben an, in welchem Frame sich die Positionen befinden. Diese Informationen werden von Character Studio zu einer Bewegungskurve zusammengefügt, in der in jedem Frame ein Key gesetzt ist.

Wurde ein Biped erstellt, können über die Befehlspalette „Bewegung“ und das Rollout „Motion Capture“ die Motion Capture-Dateien importiert werden. Beim Laden wird in jedem Frame ein Key gesetzt.

Das Biped-Skelett passt sich automatisch an die Bewegungsdaten an. Nach dem Laden der .csm-Dateien muss festgestellt werden, ob der Biped angepasst werden muss, indem sich der Anwender die Markierungen anzeigen lässt. Muss der Biped sowohl skaliert als auch die Glieder neu positioniert werden, wird zuerst die Skalierung der einzelnen Glieder durchgeführt und diese dann an die Markierungen angepasst. Das Skalieren geschieht im sogenannten „Schauspieler-Figurmodus“. Wird dieser Modus beendet, passt Character Studio die Biped-Keys automatisch an. Im Anschluss werden die Biped-Glieder relativ zu den angezeigten Markierungen ausgerichtet. Danach muss über den Button „Schauspielerstellung anpassen“ die Änderung für die gesamte Animation übernommen werden, damit die Animation richtig abläuft.

2.7 Zusammenfassung

In diesem Kapitel wurde auf die verwendeten Hilfsmittel eingegangen und Hintergrundinformationen zu den einzelnen Bereichen geliefert. Im Kapitel 3D-Animation und Motion Capture wurde der Zusammenhang zwischen der Animation von Charakteren und Motion Capture beschrieben und geschildert, wie der Arbeitsablauf bei Motion Capture aussieht. Das Kapitel über OpenGL stellt dar, was OpenGL ist und an welcher Stelle es bei der Erstellung von Applikationen steht.

Das nächste Kapitel befasst sich mit cal3d und beschreibt die Komponenten C++-Bibliothek und Exporter, aus denen cal3d besteht. Danach wird auf das Dateiformat c3d eingegangen, bei dem es sich um das Animationsformat handelt, mit dem gearbeitet werden soll. Zum Schluss werden noch Informationen zu dem Animationsprogramm 3D Studio Max und Character Studio geliefert, da auf Grund eines Problems mit dem Dateiformat c3d, das später genauer beschrieben wird, über 3D Studio Max gegangen werden muss.

3 Entwicklung eines Konzepts

3.1 Einleitung

In diesem Kapitel wird nochmal eine Beschreibung des, in der Diplomarbeit durchzuführenden, Projekts und das Projektziel beschrieben. Weiterhin wird eine Problemanalyse durchgeführt und Lösungswege aufgeführt. Abschließend wird noch eine Entwurfsbeschreibung des zu erstellenden Programms angeführt.

3.2 Projektbeschreibung und Projektziel

Die Diplomarbeit wurde in einer Firma durchgeführt, die Motion Capture-Aufnahmen für unterschiedliche Projekte erstellt. Bei der Diplomarbeit ging es darum, dem Kunden so schnell wie möglich die aufgezeichneten Bewegungen auf seinem 3D-Modell zeigen zu können und ihm die Möglichkeit zu geben, Fehler zu markieren.

Der Arbeitsablauf in der Firma setzt sich wie folgt zusammen (siehe Abb. 12):

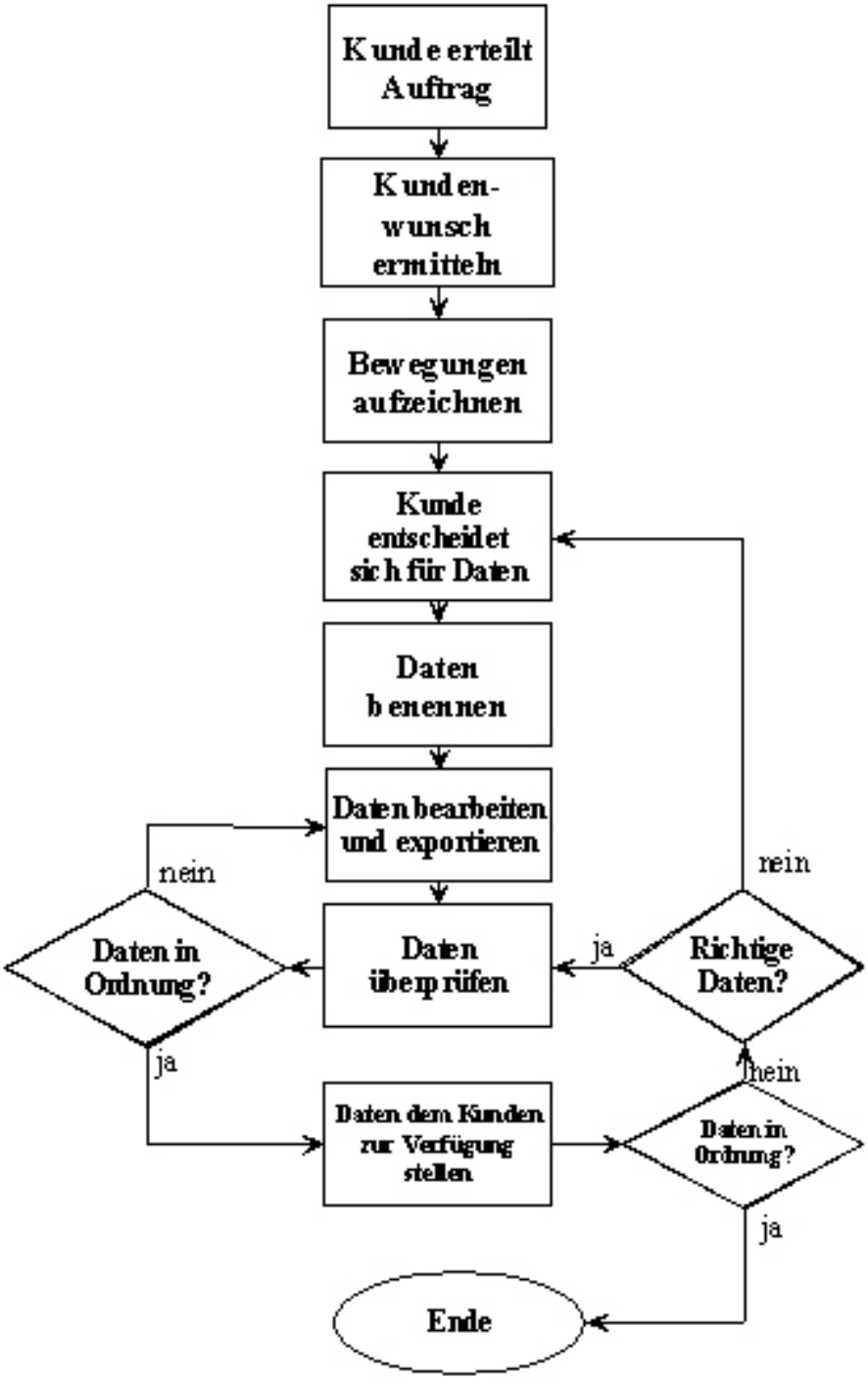

Abb. 12: Istzustand des Bearbeitungsvorgangs der Bewegungsdaten

Die Firma, die ein Projekt, beispielsweise im Bereich Spiele-programmierung, durchführt und dafür Motion Capture-Aufnahmen benötigt, erteilt einen Auftrag. Darauf hin muss von der Firma, die die Aufnahmen durchführt festgestellt werden, was genau für

Bewegungen aufgezeichnet werden sollen und was dafür benötigt wird.

Bei jedem Dreh ist einer der den Auftrag erteilenden Firma anwesend, um sicherzustellen, dass die ausgeführten Bewegungen den Vorstellungen entsprechen. Alle Bewegungen werden mehrfach durchgeführt und der Vertreter der Firma sagt, welche Dateien zur Verfügung gestellt werden sollen.

Diese Daten werden dann benannt und von Fehlern gesäubert und weiter bearbeitet. Im Anschluss wird überprüft, ob sich eventuell noch Fehler in der Bewegung befinden. Ist das der Fall, durchlaufen sie den Bearbeitungsprozess ein weiteres Mal. Sind die Daten in Ordnung, werden sie dem Kunden zur Verfügung gestellt.

Dieser sieht die Bewegungen dann zum ersten Mal auf seinem Charakter und kann noch auftretende Fehler oder sonstige Anmerkungen mitteilen. Das ging bisher aber nur, indem umschrieben wurde, um was für einen Fehler es sich handelte und die ungefähre Position angegeben wurde. Da der Kunde die Bewegungsdaten zum erstenmal auf seinem Charakter sieht, kann er jetzt erst feststellen, ob er sich für die richtige Bewegung entschieden hat. Treten in der Animation noch Fehler auf oder handelt es sich um die falschen Daten, kommen die Dateien wieder zur Motion Capture-Firma zurück, mit einer Umschreibung der Kundenwünsche. Sind es die richtigen Daten, werden sie noch mal bearbeitet, exportiert und im Anschluss nochmal auf Fehler überprüft. Sind sie jetzt in Ordnung, werden sie wieder dem Kunden zur Verfügung gestellt. Handelt es sich um völlig falsche Daten, sagt der Kunde, welche Daten er haben möchte und diese durchlaufen dann den gesamten Arbeitsablauf, bis der Kunde zufrieden ist.

Die Diplomarbeit setzt an der Stelle ein, an der die Daten aufgezeichnet und benannt wurden (siehe Abb. 13).

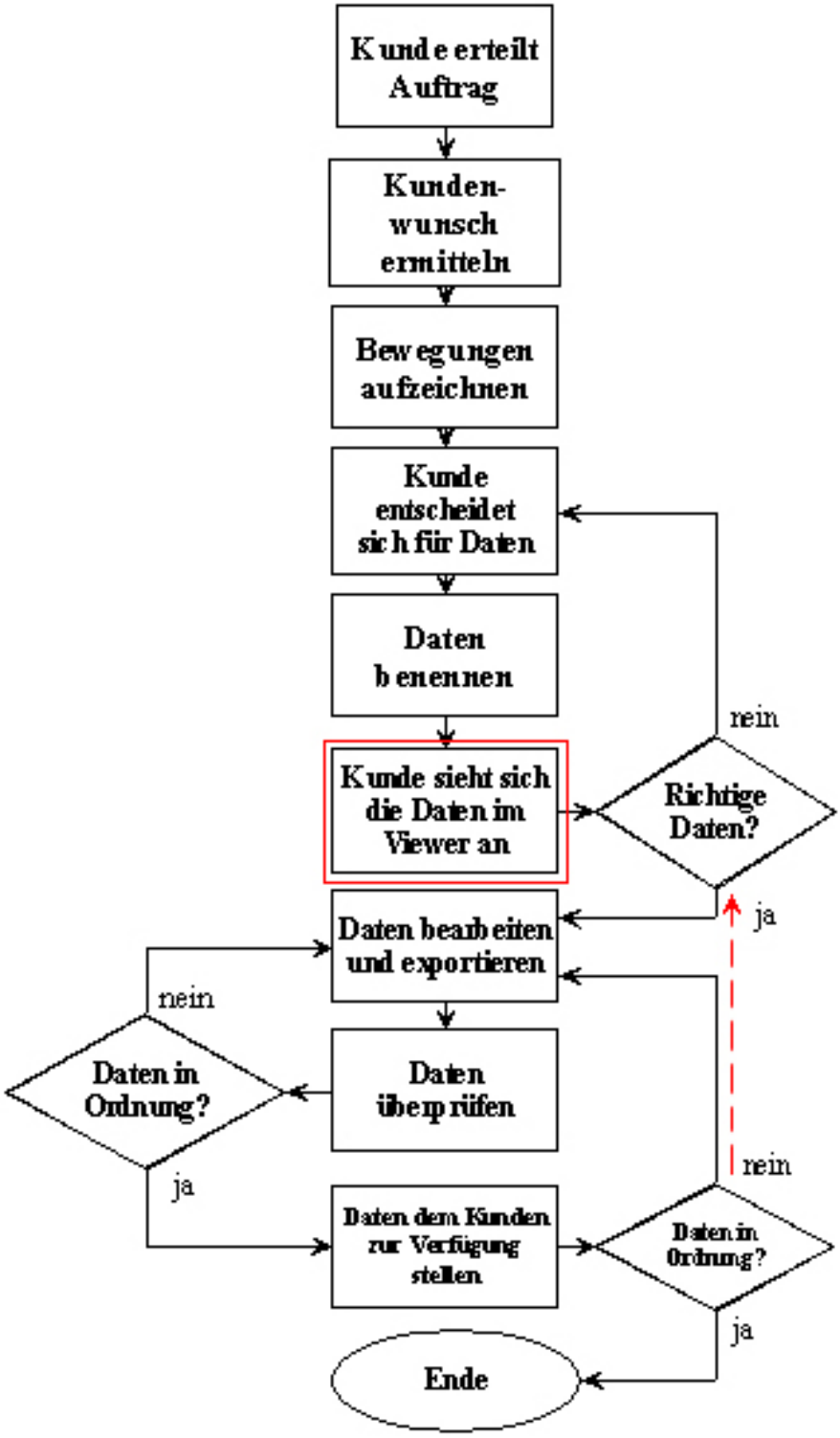

Abb. 13: Sollzustand des Bearbeitungsvorgangs der Bewegungsdaten

Nach dem Erstellen des Viewers soll der Bearbeitungsprozess wie folgt aussehen:

Was hauptsächlich verändert wurde, ist in Abbildung 13 rot dargestellt. Der Viewer, in dem sich der Kunde die ausgewählten Daten ansehen kann, wurde nach dem Benennen und vor dem Bearbeiten der Daten eingefügt. Hierdurch ergibt sich eine weitere Veränderung im Bearbeitungsprozess. Es gibt keine direkte Verbindung mehr zwischen der Abfrage ob die Daten in Ordnung sind und ob es sich um die richtigen Daten handelt. Da der Kunde die Bewegungen schon früher sehen kann, noch bevor sie fertig bearbeitet sind, kann er schon früher erkennen, ob er sich für die richtigen Daten entschieden hat, was die Verbindung der beiden Abfragen überflüssig macht.

Der gesamte Bearbeitungsprozess nach Einführung des Viewers sieht wie folgt aus:

Die Firma, die die Motion Capture Aufnahmen benötigt erteilt den Auftrag dazu und der Kundenwunsch wird ermittelt, welche Bewegungen mit welchen Hilfsmitteln aufgezeichnet werden sollen. Wurden die Bewegungen aufgezeichnet, entscheidet sich der Kunde für die Daten, die seiner Meinung nach am Besten den Vorstellungen entsprechen. Diese Bewegungsdateien werden benannt und dem Kunden zur Verfügung gestellt, so dass er sich die Bewegungen in dem Viewer ansehen kann. Jetzt hat der Kunde die Möglichkeit zu sagen, ob es sich um die richtigen Daten handelt oder ob er lieber eine andere Datei haben möchte. Ist es die Bewegung, die sich der Kunde vorgestellt hat, werden die Dateien bearbeitet und in das vom Kunden gewünschte Dateiformat exportiert. Im Anschluss werden die Bewegungen auf Fehler überprüft. Sind sie in Ordnung, werden sie

dem Kunden zur Verfügung gestellt, sonst werden sie noch mal bearbeitet und exportiert.

Ist der Kunde mit den Bewegungsdateien nicht zufrieden, kommen diese noch mal zurück zur Motion Capture-Firma um dort überarbeitet zu werden. Sind die Daten in Ordnung, kann der Kunde sie, beispielsweise in seinem Computerspiel, verwenden.

Ursprünglich sollten diese Daten direkt in den, in der Diplomarbeit erstellten Viewer, geladen werden, dabei traten aber Probleme auf, auf die in Kapitel 3.3 genauer eingegangen wird.

Mit dem erstellten Viewer soll der Kunde das Modell, für das die Bewegungen aufgezeichnet wurden laden können und die entsprechenden Bewegungen ausführen lassen. Weiterhin soll die Möglichkeit bestehen, über Lesezeichen die Stelle zu markieren, an der der Fehler zuerst auftrat und diesen zu kennzeichnen. Außerdem muss berücksichtigt werden, dass der Kunde Anmerkungen und Änderungswünsche anführen kann.

Damit der Viewer erstellt werden konnte, musste festgestellt werden, was alles zu tun und wie dies zu realisieren war.

3.3 Analyse des Problems und Lösungswege

Durch die Firma war die Vorgabe gegeben, dass als Basis der Diplomarbeit OpenGL und die OpenGL-Freeware-Engine cal3d verwendet werden soll.

Wie in Kapitel 2.4 bereits beschrieben, handelt es sich bei cal3d um eine skelettgestützte Animationsbibliothek, auf deren Grundlage der cal3d-miniviewer programmiert wurde. Dieser bietet die Möglichkeit, ein bestimmtes 3D-Modell und eine bestimmte Animation zu laden und sie sich anzuschauen.

Wird der miniviewer gestartet, wird ein Konsolenfenster geöffnet und ein Fenster erzeugt, in dem das 3D-Modell und die Animation dargestellt wird.

Der miniviewer ist eine gute Grundlage für die Diplomarbeit, weil er das Öffnen und Darstellen von 3D-Charakteren ermöglicht. Da die zu erstellende Anwendung aber ohne Konsolenfenster ablaufen soll, muss ein völlig neues Hauptprogramm erzeugt werden, um ein Fenster darzustellen.

Das Fenster sollte eine Menüleiste besitzen, über die Modelle und Animationen geladen werden können und über die andere Funktionen gesteuert werden können.

Der Viewer sollte außerdem eine Zeichnen- und eine Kommentar-funktion erhalten, über die der Benutzer Fehler oder Wünsche markieren und beschreiben kann. Auch diese Funktionen mussten neu geschrieben werden, da sie für den miniviewer nicht vorgesehen waren. Außerdem sollten die gekennzeichneten Fehler auch für spätere Anwender einsehbar sein, so dass eingefügte Lesezeichen, in denen die entsprechenden Informationen enthalten sind, abgespeichert werden können.

Weiterhin sollte der Viewer einen Slider mit Steuerbuttons haben, worüber die Animation angesprochen werden kann. Sobald der Slider bewegt wird, soll sich die Animation in der entsprechenden

Richtung und Geschwindigkeit bewegen oder über die Buttons beispielsweise gestartet und gestoppt werden.

Da mit Hilfe von cal3d bereits Modelle geladen werden können, muss noch realisiert werden, dass die Bewegungsdateien des Formats c3d geladen und abgespielt werden können.

Da ein völlig neues Hauptprogramm geschrieben werden muss, sollte als erstes das OpenGL-Fenster erstellt werden, in dem dann das Modell und die Animation dargestellt wird.

Im Anschluss sollte die Menüleiste eingefügt werden, damit die Menüsteuerung bei der weiteren Programmierung gleich berücksichtigt werden kann.

Die Menüleiste lässt sich einfach über die Ressourcen von Microsoft Visual C++ erstellen. Dort können den einzelnen Menüpunkten IDs zugewiesen werden, über die das Programm sie ansprechen kann, um Aktionen auszuführen.

Bisher war es bei dem cal3d-miniviewer so, dass ein fester Pfad für die Modell-Datei angegeben werden musste. In den Viewer sollen aber unterschiedliche Modelle geladen werden und für diese soll nicht immer wieder der Viewer über Microsoft Visual C++ neu erstellt werden müssen, nachdem ein neuer Pfad angegeben wurde. Die Lösung dafür ist es, einen Öffnen-Dialog zu erstellen. Über das Menü besteht dann die Möglichkeit, diesen Dialog aufzurufen und die gewünschte Datei auszuwählen.

Um die Animationsdateien in dem c3d-Format zu laden, muss auch ein Öffnen-Dialog erzeugt werden. Weiterhin müssen die Bewegungsdateien mit den Modelldateien kombiniert werden, damit

die Modelle die Animationen ausführen können. Dazu muss festgestellt werden, welche Informationen die c3d-Dateien enthalten. Da die Dateien nicht im ASCII-Format gespeichert werden, kann man den Inhalt allerdings nicht ohne weiteres lesen.

Mit Hilfe einer Applikation, die über die Homepage www.c3d.org zugänglich ist, lässt sich erkennen, welche Informationen die c3d-Dateien beinhalten.

Bei diesen Informationen handelt es sich um die Anzahl und die Namen der beim Aufzeichnen der Bewegungen verwendeten Marker. Weiterhin ist der Name des Darstellers, die Länge der Animation in Frames und eine Bewegungskurve in x-, y- und z-Richtung für jeden Marker abgespeichert. Es sind also nur die reinen Positionsdaten der Marker für jeden Frame abgespeichert. Außerdem ist noch angegeben, mit welcher Frequenz die Bewegungen aufgezeichnet wurden.

Beim Verbinden der Modelle mit den Animationsdaten traten allerdings ein paar Probleme auf. Da die c3d-Dateien Informationen über die verwendeten Marker und deren x-, y- und z-Position beinhalten, aber keine Rotationsinformationen besitzen, konnten diese Informationen nicht auf das 3D-Modell übertragen werden.

Um einen Charakter zu animieren, sind sowohl Translations- als auch Rotationsinformantinen nötig. Diese müssten extra berechnet werde, aber weder das c3d-Format, noch das cal3d-Format bieten dafür Möglichkeiten.

Ein zusätzliches Problem, das bei dem Verbinden des cal3d-Modells mit den c3d-Daten auftrat war, dass cal3d zeitbasiert arbeitet und

c3d auf Frames basiert. Diese Unterschiede ließen sich nicht miteinander vereinbaren.

Damit trotzdem ein Viewer erstellt werden konnte, in dem der Kunde die Möglichkeit hat, die aufgezeichneten Bewegungen auf seinem Charakter anzusehen, musste eine Alternative gefunden werden.

Dabei kam die Überlegung auf, dass es einen, mit cal3d erstellten, Exporter für 3D Studio Max gibt und dass in 3D Studio Max über Character Studio Motion Capture-Daten geladen werden können. Daraus ergab sich dann die Alternative, die c3d-Dateien in das Character Studio-Format .csm zu konvertieren. Dieses Format kann dann über die Biped-Funktion in 3D Studio Max geladen und anschließend in das cal3d-Format exportiert werden.

Dieses Format kann dann über den erstellten Öffnen-Dialog dargestellt werden und die Animation kann ablaufen.

Als nächstes muss für den Viewer realisiert werden, dass der Kunde sein Modell rotieren und skalieren kann, um sicherzugehen, das die Bewegung aus jeder Perspektive in Ordnung ist. Dabei ist es am sinnvollsten, das über die Maussteuerung zu regeln. Da aber auch eine Zeichnen- und eine Kommentarfunktion erstellt werden sollen, und es auch bei diesen sinnvoll wäre, wenn sie mit Hilfe der Maussteuerung realisiert werden, muss sichergestellt werden, dass das Modell beispielsweise nicht gleichzeitig rotiert wird, wenn gezeichnet werden soll.

Dieses Problem lässt sich einfach über die Menüsteuerung lösen, indem festgelegt wird, dass immer nur ein Menüpunkt aktiv sein kann und dass bei der Abfrage von Mausereignissen überprüft wird, welcher Menüpunkt aktiv ist.

Ein weiteres Problem ist, das die Perspektive des Modells an der Stelle, an der ein Fehler auftritt, auch für spätere Anwender einsehbar sein soll. Genauso muss später ersichtlich sein, an welcher Stelle der Slider steht. Wenn gezeichnet wurde oder ein Kommentar eingegeben wurde, muss auch das einem anderen Anwender zugänglich sein.

Die Lösung dafür ist es, an den entsprechenden Stellen Lesezeichen einzufügen. In diesen wird der Name des Lesezeichens, die Kameraposition, die Position und der Inhalt eines Kommentars, die Position der eingezeichneten Markierung und die Position des Sliders abgespeichert. Über einen Speichern-Dialog können diese Lesezeichen über die Menüsteuerung abgespeichert werden und mit Hilfe eines Öffnen-Dialogs über die Menüsteuerung wieder dargestellt werden.

Damit eine Datei aber nicht für jede Animationsdatei geladen werden kann, muss in der abgespeicherten Datei auch der Name der Animationsdatei angegeben werden, für die die Lesezeichen gelten. Beim Öffnen der Lesezeichen-Datei wird also eine Abfrage gestartet, für welche Animationsdatei sie erstellt wurde und mit der aktuellen Bewegungsdatei verglichen. Handelt es sich um den richtigen Namen, wird die Datei mit den Lesezeichen geöffnet, sonst wird eine Fehlermeldung angezeigt.

3.4 Entwurf

Der bereits existierende miniviewer ist eine Konsolenanwendung. Der zu erstellende Viewer soll eine Win32-Anwendung werden, deshalb muss ein neues Projekt erstellt werden. Die bereits vorhandenen Dateien des miniviewers können allerdings in die neue Anwendung mit eingebunden werden.

Da es sich um eine Win32-Anwendung handelt, arbeitet diese mit der Funktion „int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)“. Über diese Funktion wird die Funktion zum Erzeugen des OpenGL-Fenster aufgerufen und der Viewer initialisiert.

Der Viewer soll auf der Basis von OpenGL programmiert werden. Aus diesem Grund wird die Funktion „BOOL CreateGLWindow (char* title, int width, int height, int bits, bool fullscreenflag)“ dafür verwendet, das Fenster mit der gewünschten Menüleiste und den Buttons zu erzeugen.

Wurde der Viewer erzeugt, wartet das Programm auf Aktionen, die vom Anwender ausgeführt werden (siehe Abb. 14).

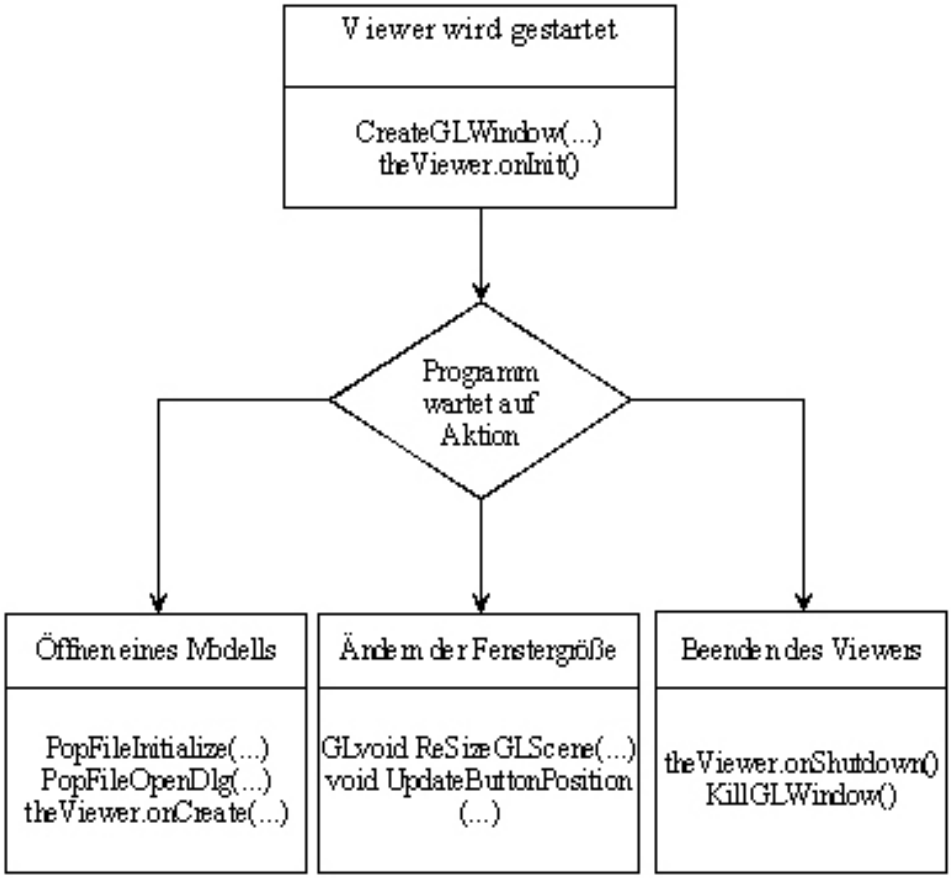

Abb. 14: Nach dem Start des Viewers wird auf Aktionen gewartet

Wenn der Viewer gestartet wurde, sollen noch nicht alle Menüpunkte und der Slider aktiv sein. Direkt nach dem starten des Viewers wartet das Programm auf Aktionen des Anwenders. Dieser hat die Möglichkeit, ein Modell zu öffnen, die Fenstergröße des Viewers zu ändern oder das Programm wieder zu schließen.

Wird die Aktion „Öffnen eines Modells“ ausgewählt, wird über die Funktion „BOOL PopFileOpenDlg (HWND hWnd, PTSTR pstrFileName, PTSTR pstrTitleName)“ ein Öffnen-Dialog erzeugt, über den der Anwender das gewünschte Modell auswählen kann. Dieses wird dann im Viewer dargestellt und mit Hilfe der Funktion „void UpdateMenu()“ werden weitere Menüpunkte aktiviert.

Wählt der Anwender die Aktion „Ändern der Fenstergröße“ aus, dann wird über die Funktion „GLvoid ReSizeGLScene (GLsizei width, GLsizei height)“ die Fenstergröße angepasst und die Position der Buttons und die Länge des Slider muss angepasst werden. Das geschieht mit Hilfe der Funktion „void UpdateButtonPosition(int width, int height)“, über die die Position der Buttons und die Länge des Sliders neu berechnet werden. Nach dem Verändern der Fenstergröße stehen dem Anwender die gleichen Auswahlmöglichkeiten zur Verfügung, wie vorher.

Wird die Aktion „Beenden des Viewers“ ausgewählt, wird der Viewer geschlossen, indem „theViewer.onShutdown()“ und „KillGLWindow()“ aufgerufen wird.

Nach dem Öffnen eines Modells kann der Anwender eine Animation laden. Diese kann über den Slider und die Buttons gesteuert werden und mit der Maus rotiert oder skaliert werden (siehe Abb. 15).

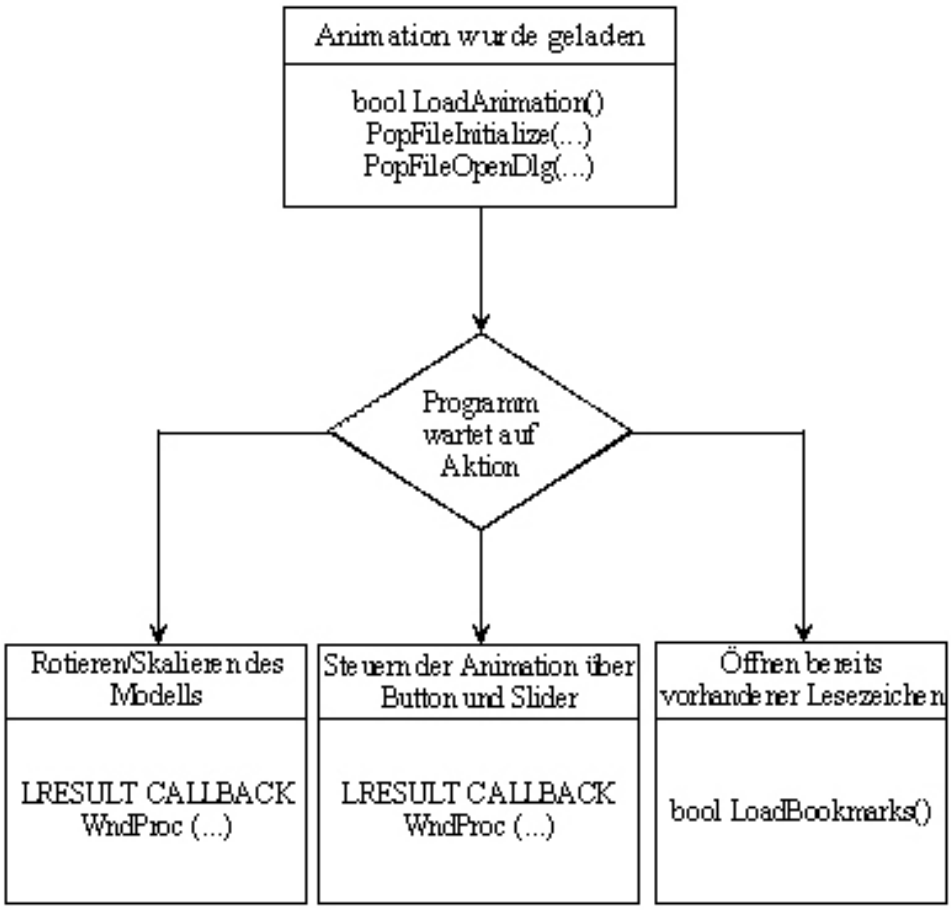

Abb. 15: Nach dem Laden einer Animation wartet das Programm auf Aktionen

Wenn eine Animation über „bool LoadAnimation()“ geladen wurde, befindet sich diese in einer Endlosschleife und das Programm wartet auf eine Aktion des Anwenders.

Der Anwender hat jetzt die Möglichkeit, das Modell im 3D-Raum zu skalieren und zu rotieren, da dieser Menüpunkt mit Hilfe der Funktion „void UpdateMenu()“ aktiviert wurde. Das Programm achtet auf Mausereignisse und rotiert das Modell, wenn die linke Maustaste gedrückt wurde und skaliert das Modell, wenn die rechte Maustaste gedrückt wurde.

Über die, in der Funktion „BOOL CreateGLWindow (…)“, erzeugten Buttons und den Slider kann die Animation gesteuert werden. Mit Hilfe von switch-Anweisungen werden die Buttonereignisse überprüft und ausgeführt. Wird der Stop-Button gedrückt, werden über „void UpdateMenu()“ weitere Menüpunkte freigeschaltet.

Sobald eine Animation geladen wurde, hat der Anwender unter dem Menüpunkt „Bookmarks -> Öffnen“ die Möglichkeit, bereits angelegte Lesezeichen, die der Anwender selbst oder ein anderer Benutzer zuvor erzeugt hat, zu öffnen.

Wurde die Animation angehalten, hat der Anwender die Gelegenheit, Lesezeichen einzufügen und darin Fehler und Wünsche anzumerken (siehe Abb. 16).

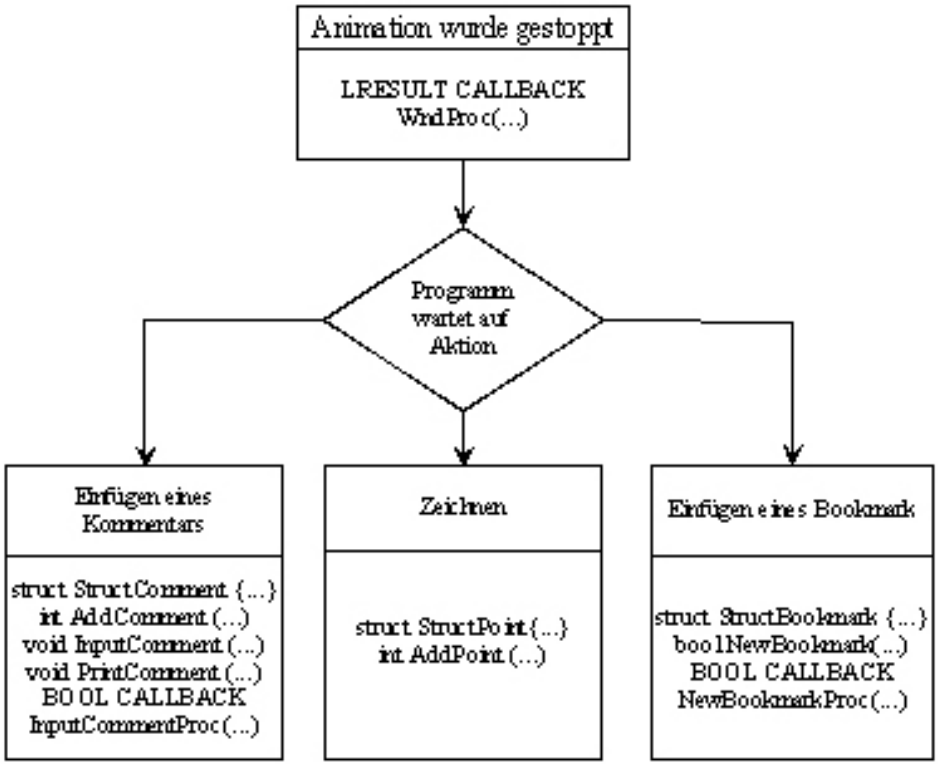

Abb. 16: Auswahl der möglichen Aktionen, nachdem die Animation gestoppt wurde

Nachdem die Animation mit Hilfe des Stop-Buttons angehalten wurde, werden weitere Menüpunkte aktiviert. Die Menüpunkte geben dem Anwender die Möglichkeit, einen Kommentar einzufügen, zu zeichnen oder ein neues Lesezeichen zu erstellen.

Das Zeichnen und das Einfügen von Kommentaren wurde über verkettete Listen realisiert. Dazu wurden die Strukturen „struct StructPoint {…}“ und „struct StructComment {…}“ festgelegt.

Entscheidet sich der Anwender dafür, einen Kommentar einzufügen, wartet das Programm auf ein Mausereignis. An der Stelle, an der ein Anwender mit der linken Maustaste in den Viewer klickt, wird der Kommentar mit Hilfe der Funktion „void InputComment (…)“ eingefügt.

Möchte der Anwender etwas im Viewer zeichnen, wird auch hier auf ein Mausereignis gewartet. Über die Funktion „int AddPoint (…)“ wird an der Stelle, an der die linke Maustaste gedrückt wird gezeichnet. Sobald die Maustaste losgelassen wird, wird mit dem Zeichnen aufgehört und der Anwender kann an einer anderen Stelle weitermalen.

Sobald gezeichnet wird oder ein Kommentar eingefügt werden soll, wird automatisch ein Lesezeichen eingefügt. Das geht aber auch manuell, indem über „Bookmarks -> Neuer Bookmark“ ein neuer Bookmark erstellt wird. Das geschieht über die Funktionen „BOOL CALLBACK NewBookmarkProc (…)“ und „bool NewBookmark (…)“.

Schließlich muss noch sichergestellt sein, dass der Anwender bereits erstellte Bookmarks wieder löschen kann und dass er seine Bookmarks in einer Datei abspeichern kann, um sie anderen

Benutzern zugänglich zu machen oder sie selbst zu einem späteren Zeitpunkt zu öffnen (siehe Abb. 17).

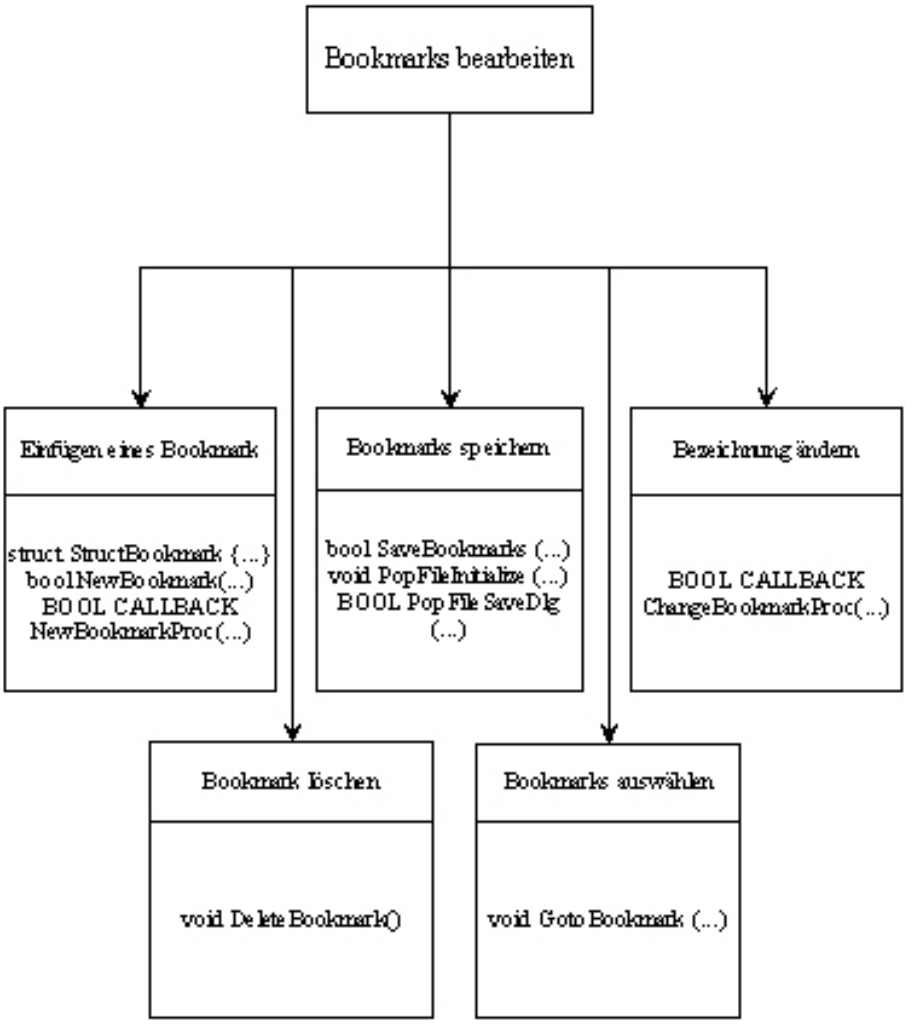

Abb. 17: Möglichkeiten, Bookmarks zu bearbeiten

Wie bereits beschrieben kann über die Funktionen „BOOL CALLBACK NewBookmarkProc (…)“ und „bool NewBookmark (…)“ ein neuer Bookmark eingefügt werden. Der Anwender hat jetzt die Möglichkeit, unterschiedliche Aktionen bezüglich der Bookmarks durchzuführen.

Stellt der Benutzer fest, dass er ein Lesezeichen an der falschen Stelle eingefügt hat oder dass es nicht mehr gebraucht wird, kann er das entsprechende Lesezeichen löschen. Die geschieht, indem der Menüpunkt „Bookmark löschen“ im Menü „Bookmarks“ ausgewählt wird und durch Aufruf der Funktion „void DeleteBookmark()“ gelöscht wird.

Weiterhin kann der Anwender über den Aufruf der Funktion „BOOL CALLBACK ChangeBookmarkProc (…)“ die Bezeichnung eines gerade aktiven Bookmarks ändern.

Ein Lesezeichen kann aktiviert werden, indem der Anwender auf den Namen des gewünschten Bookmarks geht. Die Namen werden unter dem Menü „Bookmarks“ aufgelistet und sind durch anklicken anwählbar. Über die Funktion „void GotoBookmark (…)“ springt die Animation an die Stelle, an der das Lesezeichen eingefügt wurde und zeigt das Modell aus der Perspektive, die in dem entsprechenden Bookmark steht. Auch der Slider springt an die Position, an der das Lesezeichen eingefügt wurde.

Schließlich kann der Benutzer die von ihm erstellten Bookmarks in einer Datei abspeichern, um sie für spätere Anwendungen verfügbar zu machen. Soll ein Bookmark abgespeichert werden, wird die Funktion „bool SaveBookmarks (…)“ aufgerufen, in der die nötigen Parameter in eine Datei geschrieben werden und über die Funktion „BOOL PopFileSaveDlg (…)“ wird ein „Speichern unter“-Dialog erzeugt, in dem der Benutzer den gewünschten Namen eingeben kann, um die Datei abzuspeichern.

Damit der Anwender aber nicht versehentlich den Viewer schließen kann, ohne vorher die Bookmarks abgespeichert zu haben wurde

noch die Funktion „void SaveReminder()“ eingefügt. Über diese Funktion wird eine Abfrage durchgeführt, ob die erstellten Bookmarks bereits abgespeichert wurden. Ist dies nicht der Fall, wird über diese Funktion ein Dialog angezeigt, der nachfragt, ob die Änderungen an der Bookmark-Datei abgespeichert werden sollen.

3.5 Zusammenfassung

In diesem Kapitel wurde das Projekt, das durchgeführt werden soll und das Projektziel beschrieben. Danach wurde eine Problemanalyse durchgeführt und dazu wurden Lösungswege ermittelt und aufgezeigt.

Schließlich wurde noch ein Entwurfsplan des zu erstellenden Programmes aufgeführt, in dem auf die einzelnen, im Viewer zur Verfügung stehenden Funktionen eingegangen wurde.

4 Realisierung und Funktionsweise

4.1 Einleitung

Dieses Kapitel befasst sich mit dem Viewer, der im Rahmen dieser Diplomarbeit erstellt werden sollte. Dabei wird darauf eingegangen, wie der Viewer erstellt wurde. Außerdem wird beschrieben wie die Oberfläche des Viewers aussieht und wie er funktioniert.

4.2 Die Erstellung des Viewers

Der Viewer wurde mit dem Programm Microsoft Visual C++ programmiert. Zum Erstellen des Viewers für die Motion Capture Daten wurde die Freeware-Bibliothek Cal3d und der damit erstellte cal3d-miniviewer verwendet. Bei dem miniviewer handelt es sich um eine Win32-Konsolenanwendung. Dabei wird ein Windows-Fenster erzeugt, in den ein animiertes Modell geladen wird. Das Arbeitsverzeichnis und der Name der zu öffnenden Datei sind fest vorgegeben. Im Fenster wird die Animation automatisch abgespielt und man kann das Modell mit Hilfe der Maus rotieren und skalieren.

Für das Laden der Motion Capture Daten musste der Viewer so verändert werden, dass unterschiedliche Modelle und Animationen vom Anwender geöffnet werden können, ohne vorher einen festen Pfad anzugeben. Weiterhin musste eine Menüsteuerung und ein Slider mit Buttons eingefügt werden.

Bei dem endgültigen Viewer handelt es sich um eine Win32-Anwendung. Da der miniviewer eine Konsolenanwendung ist, musste ein neues Projekt angelegt werden (siehe Abb. 18).

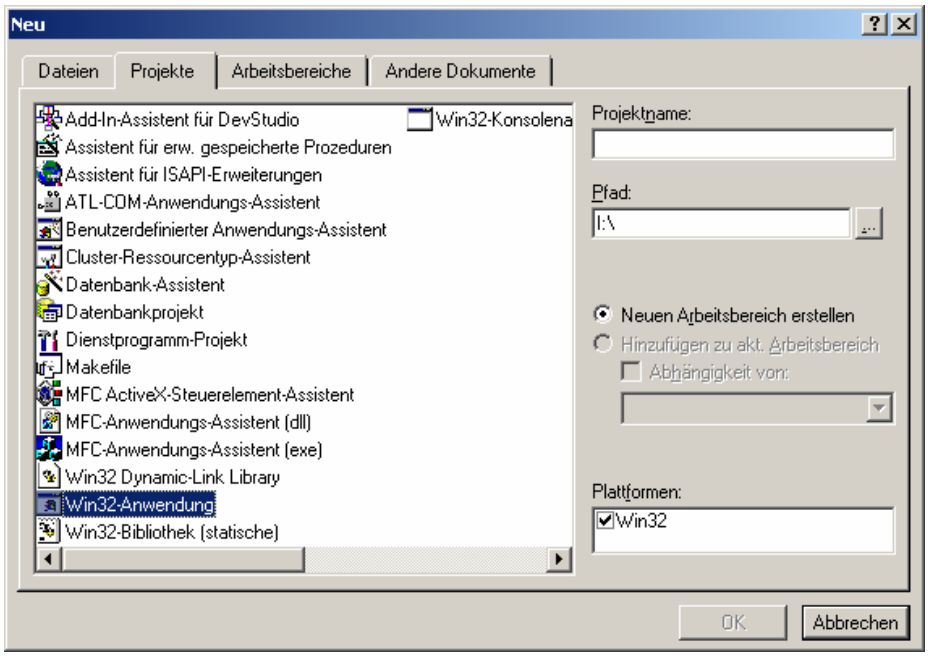

Abb. 18: Dialog zum Erstellen eines neuen Projekts

In das neue Projekt wurden dann die Dateien aus dem miniviewer eingefügt, modifiziert und ein neues Hauptprogramm geschrieben. Mit Hilfe der Ressourcen wurde eine Menüleiste mit drei Menüpunkten und Untermenüs erstellt, über die man Dateien laden kann. Weiterhin funktionieren die Text-, Paint- und Skalieren/Rotieren-Funktionen mit Hilfe der Menüsteuerung und über den dritten Menüpunkt können Lesezeichen eingefügt werden.

Die meisten Menüpunkte und der Slider mit den Buttons sind deaktiviert, wenn das Programm gestartet wird. Sobald eine Szene oder eine Animation geöffnet wird, wird die Funktion „UpdateMenu()“ aufgerufen:

Diese Funktion wird auch aufgerufen, sobald ein Menüpunkt ausgewählt wird. Das geschieht, um sicherzugehen, dass das Menü immer aktuell ist. Als erstes wird eine Abfrage gestartet, ob die Animation angehalten wurde und ob es sich um die erste Benutzung des Viewers handelt. Sind beide Aussagen falsch, das heißt, dass die Animation läuft und der Viewer schon etwas geladen hat, wird eine Nachricht des Typs WM_COMMAND an das Fenster gesendet und der Menüpunkt „Drehen/Skalieren“ wird aktiviert. Als nächstes wird eine if-Schleife durchlaufen, bei der als erstes die Abfrage läuft, ob der Viewer zum ersten Mal gestartet wurde. Ist das der Fall, werden alle Menüpunkte, bis auf „Szene laden“ und „Beenden“ deaktiviert. Wurde der Viewer bereits gestartet, werden die drei Menüpunkte „Drehen/Skalieren“, „Animation laden“ und unter dem Menü Bookmarks der Punkt „Öffnen“ aktiviert. Im Anschluß werden weitere Abfragen durchlaufen, um sicherzustellen, welche Menüpunkte aktiviert bzw. deaktiviert werden sollen. Zunächst wird abgefragt, ob noch keine Bookmarks vorhanden sind. Sind noch keine vorhanden, ist der Menüpunkt „Speichern unter“ deaktiviert, sonst wird er aktiviert. Danach wird eine if-Abfrage durchlaufen, ob bereits eine Bookmarkdatei vorhanden ist, ob diese verändert wurde und ob überhaupt schon Bookmarks bestehen. Treffen diese Bedingungen zu, wird der Menüpunkt „Speichern“ im Menü Bookmarks aktiviert, ist eine davon falsch, ist der Menüpunkt nicht aktiv. Die nächste Abfrage entscheidet, ob die Menüpunkte „Kommentar“ und „Zeichnen“ anwählbar sind. Es wird geprüft, ob die Animation angehalten wurde und nur wenn das der Fall ist, werden die beiden Menüpunkte schwarz dargestellt.

Zum Schluss wird noch überprüft, ob man sich gerade in einem aktiven Bookmark befindet. Befindet man sich in einem, kann man zu diesem Zeitpunkt kein neues Lesezeichen einfügen, man hat dann die Möglichkeit den aktiven Bookmark umzubenennen oder zu

löschen. Wird kein Bookmark angezeigt, wird überprüft, ob die Animation gestoppt ist. Ist das so, dann kann man einen neuen Bookmark einfügen, läuft die Animation ist das nicht möglich. Außerdem werden die Menüpunkte „Bezeichnung ändern“ und „Bookmark löschen“ deaktiviert. Das Aktivieren (schwarz darstellen) und Deaktivieren (grau darstellen) der Menüpunkte geschieht über die Funktion „EnableMenuItem(…)“. Diese Funktion übergibt an das Menü, ob der jeweilige Menüpunkt aktiviert (MF_ENABLED) oder deaktiviert und grau dargestellt (MF_GRAYED) sein soll.

Um den Slider und die Buttons zu aktivieren, wird beim Öffnen einer Szene die Funktion „EnableButtons(…)“ aufgerufen:

```
void EnableButtons(bool state)
{
        EnableWindow(button1, state);
        EnableWindow(button2, state);
        EnableWindow(button3, state);
        EnableWindow(button4, state);
        EnableWindow(button5, state);
        EnableWindow(button6, state);
        EnableWindow(hWndSlider,state);
}
```

Die Variable „state“ ist am Anfang auf „false“ gesetzt. Sobald die Funktion „OpenScene()“ aufgerufen und ein Modell geladen wird, wird die Variable auf „true“ gesetzt. Auf den Aufruf „EnableWindow(…)“ reagiert Windows, indem beim Status „false“ die Beschriftung der Buttons grau dargestellt wird und eine Reaktion auf Mausereignisse unterdrückt wird. Ist der Status der Variablen „true“, werden die Buttons schwarz dargestellt und reagieren auf Mausereignisse. Über „EnableWindow(…)“ wird auf die gleiche Art wie bei den Buttons der Slider aktiviert und deaktiviert.

Damit im Programm kein fester Pfad angegeben sein muss, sondern der Anwender ein beliebiges Modell bzw. Animationsdatei laden kann, wurde ein Öffnen-Dialog erstellt.

```
BOOL PopFileOpenDlg (HWND hWnd, PTSTR pstrFileName,
PTSTR pstrTitleName)
{
        ofn.hwndOwner = hWnd;
        ofn.lpstrFile = pstrFileName;
        ofn.lpstrFileTitle = pstrTitleName;
        ofn.Flags = OFN_HIDEREADONLY|OFN_CREATEPROMPT;

        return GetOpenFileName (&ofn);
}
```

Wird ein Öffnen-Menüpunkt ausgewählt, wird die Funktion “PopFileOpenDlg(...)” aufgerufen. Diese Funktion bekommt den Handle zum Hauptfenster und zwei Zeiger auf Textpuffer übergeben. Die zwei Zeiger verweisen auf den Fenstertitel und den vollständigen Dateinamen. Mit diesen Daten werden hwndOwner, lpstrFile und lpstrFileTitle gesetzt. Der Aufruf von „GetOpenFileName“ hat den Öffnen-Dialog zur Folge. Wird dieser Dialog beendet, endet auch die Funktion „GetOpenFileName“ und in lpstrFile und lpstrFileTitle sollte sich jetzt der ausgewählte Dateiname befinden.

Über den Aufruf der Funktion wird die OPENFILENAME Struktur initialisiert. Dabei wird die Länge der Struktur in Bytes angegeben und das Handle zu dem Fenster, zu dem der initialisierte Dialog gehört. Mit Hilfe eines Filters wird an die Funktion „PopFileInitialize(…)“ übergeben, welche Datei geladen werden soll. Dieser Filter wird in „ofn.lpstrFilter“ gespeichert und dann wird dem Dialog übergeben, welche Dateien geöffnet bzw. gespeichert werden sollen.

Wenn ein Lesezeichen abgespeichert werden soll, wird das auch mittels Filter an diese Funktion übergeben. Der „Speichern unter“-Dialog wird auf eine ähnliche Weise erzeugt, wie der „Öffnen“-Dialog. Der Unterschied ist, dass die aufgerufene Funktion, den Handle und die zwei Zeiger übergeben bekommt „PopFileSaveDlg(…)“ heißt und die Funktion, die den „Speichern unter“-Dialog zur Folge hat, heißt „GetSaveFileName“.

Die Modell-Dateien, die in den Viewer geladen werden, sind ursprünglich Modelle, die mit 3D Studio Max erstellt wurden. Sie werden aus dem Animationsprogramm mit Hilfe eines Plug-ins in ein Format exportiert, das cal3d versteht.

Die Animationsdateien, die in den Viewer geladen werden sollten, sind im Format c3d. Dabei traten allerdings ein paar Probleme auf, die in Kapitel bereits beschrieben wurden. Die c3d-Dateien sollten direkt in den Viewer geladen und an das Modell gebunden werden.

Die gefundene Alternative dazu, die c3d-Dateien direkt zu laden, war es einen kleinen Umweg zu gehen. Bei diesem Umweg müssen die Dateien erst in ein Format von Character Studio umgewandelt werden, um dann für den Viewer exportiert zu werden. Das ist deshalb möglich, da Character Studio ein Feature von 3D Studio Max ist und cald3d dazu kompatibel ist.

Die Animationsdateien werden über einen Slider mit den passenden Buttons („First“, „<<“, „Stop“, „Play“, „>>“ und „Last“) gesteuert. Für den Slider wird am Anfang ein Handle festgelegt, über den Befehl

```
hWndSlider = CreateWindowEx(WS_EX_CLIENTEDGE,
             TRACKBAR_CLASS, TEXT(""), TBS_TOP |
             WS_TABSTOP | WS_CHILD | WS_VISIBLE, 0, 655,
             750, BTN_HEIGHT, hWnd, (HMENU)0x1234,
             hInstance, NULL);
```

wird er erstellt und unter anderem mit Hilfe des Befehls „Send Message" gesteuert. Mit „CreateWindowEx" können sich überlappende Fenster, Pop-ups oder Child-Fenster in einem erweiterten Stil erzeugt werden. Der erste Parameter gibt an, in welchem Stil das zu erzeugende Fenster dargestellt werden soll. In diesem Fall sorgt „WS_EX_CLIENTEDGE" dafür, dass das Fenster einen Rahmen mit einer versunkenen Kante hat. „TRACKBAR_CLASS" gibt an, dass das erzeugte Fenster der Klasse Trackbar angehört und erzeugt den Slider. Der nächste Parameter ist für den Titel oder die Beschriftung des Fensters verantwortlich. Da es sich um einen Slider handelt, wird kein Titel bzw. keine Beschriftung benötigt. Als nächstes wird die Art des zu erzeugenden Fenster festgelegt. Das kann mit Hilfe einer Kombination möglicher Stilrichtungen passieren. In diesem Fall wird „TBS_TOP | WS_TABSTOP | WS_CHILD | WS_VISIBLE" angegeben. "TBS_TOP" sorgt dafür, dass die Markierung der Abschnitte oberhalb des Schiebereglers des Sliders liegt. Mit „WS_TABSTOP" wird festgelegt, dass der Schieberegler ein Stück weiter- oder zurückspringt, wenn mit der Maus an irgendeine Stelle des Sliders geklickt wird. Über den Parameter „WS_CHILD" wird angegeben, dass ein Child-Fenster erzeugt werden soll. Damit wird auch festgelegt, dass das Fenster keine Menüleiste haben kann. Über den letzten Parameter „WS_VISIBLE" wird geregelt, dass der Slider von Programmstart an sichtbar ist. Die nächsten drei Angaben des Befehls „CreateWindowEx" „0, 655, 750" gibt die x- und die y-Position und die Breite des Sliders an. „BTN_HEIGHT" wurde am Anfang des Programms definiert und legt fest, wie hoch der Slider sein soll. Danach wird das Handle des Eltern-Fensters angegeben.
Mit „(HMENU)0x1234" wird dem Slider eine Identifizierung gegeben, über die dem Haupftfenster mitgeteilt wird, wenn irgendein Ereignis stattfindet. „hInstance" wird bei Windows NT/2000/XP ignoriert, bei

Windows 95/98/ME ist „hInstance“ ein Handle zu einer Instanz des Fensters. Das letzte angegebene Element ist ein Zeiger, dieser zeigt auf „NULL“, da er hier nicht benötigt wird [10, 11].

Die Buttons werden mit Hilfe der gleichen Funktion erzeugt.

```
button1 = CreateWindowEx(0, "BUTTON", "First", WS_CHILD |
          WS_VISIBLE | BS_PUSHBUTTON, 750, 655,
          BTN_WIDTH, BTN_HEIGHT, hWnd, (HMENU)
          ID_FIRST, hInstance, NULL);
```

Da für die Buttons kein erweiterter Stil benötigt wird, ist der erste Parameter „0”. Der zweite Parameter sagt dem Programm, dass das zu erzeugende Element der Klasse „BUTTON“ angehört. An dritter Stelle steht der Text, der auf dem Button stehen soll, in diesem Fall ist das der Text „First“. Als nächstes wird, wie beim Slider der Stil der Buttons über „WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON“ festgelegt. „WS_CHILD“ gibt wie bereits gesagt an, dass ein Child-Fenster angelegt werden soll und „WS_VISIBLE“ gibt an, dass die Buttons von Anfang an sichtbar sind. Über „BS_PUSHBUTTON“ wird ein Button erzeugt, der reagiert, wenn er gedrückt wird und sendet dann eine „WM_COMMAND“-Nachricht an das Hauptfenster. Dann wird angegeben, an welcher Position die Buttons erzeugt und welche Maße sie haben sollen. Die Parameter „hWnd“, „hInstance“ und „NULL“ haben die gleiche Funktion wie beim Slider. „(HMENU) ID_FIRST“ wird zur Identifikation des erzeugten Buttons festgelegt [10, 11].

Damit der Anwender die Fenstergröße des Viewers auch verändern kann, müssen die Breite des Sliders und die Positionen der Buttons an die jeweilige Fenstergröße angepasst werden. Dazu wird die Funktion „void UpdateButtonPosition(int width, int height)“ aufgerufen. Über diese Funktion wird die Position bei einer Veränderung der Fenstergröße neu berechnet.

Die in der Funktion „UpdateButtonPosition(…)“ aufgerufene Funktion „MoveWindow(…)“ sorgt dafür, dass die Position des angesprochenen Elements verschoben und die Größe angepasst wird. Der erste Parameter ist das Handle des angesprochenen Parameters, in diesem Fall die Buttons und der Slider. Für z.B. Button 1 werden mit „width - 6*BTN_WIDTH” und „height - BTN_HEIGHT” die neuen x- und y-Positionen berechnet. Beim Slider ändert sich die x-Position nicht. Hier muss nur die y-Position mit „height - BTN_HEIGHT” neu errechnet werden. Als Nächstes werden die Höhe und Breite der Buttons und des Sliders angegeben. Bei den Buttons ändern sich die Höhe und die Breite nicht, nur die Breite des Sliders muss neu bestimmt werden. Der letzte Parameter gibt an, ob die Child-Fenster neu gezeichnet werden sollen. Da der Wert auf „true” gesetzt ist, sendet die Funktion die Nachricht, dass das Neuzeichnen durchgeführt werden soll [10, 11].

Wird ein Button gedrückt, werden mit Hilfe einer switch-Anweisung Befehle ausgeführt, über die sowohl Slider als auch Animation gesteuert werden. Wird beispielsweise der Stop-Button gedrückt, wird eine Abfrage gestartet, ob die Animation läuft oder schon gestoppt ist. Läuft die Animation, wird der entsprechende Wert auf „true“ gesetzt und die Animation wird angehalten. Ist die Animation bereits gestoppt, passiert nichts.

```
switch (LOWORD(wParam))
{
        …
        case ID_STOP:
                        //Button STOP wurde gedrückt.
                        if (theViewer.getPause() == false)
                        {
                                SetPause(true);
                                SendMessage(hWnd, WM_PAINT, 0,0);
                        }
                        break;
```

```
        …
}
```

Die Animation kann außerdem gesteuert werden, indem der Slider verschoben wird.

```
case WM_HSCROLL:
                    theViewer.setPosition(((float)SendMessage
                    (hWndSlider, TBM_GETPOS, 0, 0)/100000.0));
                    SetPause(true);
                    SendMessage(hWnd, WM_PAINT, 0,0);
```

Unabhängig davon, ob die Animation gestartet oder angehalten wurde, kann man das 3D-Modell über die Maussteuerung drehen und skalieren. Das wird über so genannte Mausereignisse gesteuert. Sobald eine Maustaste gedrückt wurde, sendet Windows eine Nachricht an die entsprechende Prozedur. Über die linke Maustaste lässt sich das Modell rotieren und über die rechte Taste skalieren. Soll das Modell z.B. rotiert werden, führt das Programm die folgenden Anweisungen aus:

Damit der Anwender Fehler in der Animation markieren und beschreiben kann, wurden eine Zeichen- und eine Textfunktion in den Viewer eingebaut. Beide Funktionen befinden sich unter dem Menüpunkt „Bearbeiten“ und erhalten genau wie „Drehen/Skalieren“, einen Haken, wenn einer von ihnen ausgewählt wird. Von den drei Punkten kann jeweils nur einer aktiv sein. Wählt man einen der Menüpunkte aus, werden über eine switch-Anweisung die entsprechenden Befehle ausgeführt. Sobald dann ein Mausereignis stattfindet, wird die entsprechende Funktion ausgeführt.

Jeder Punkt, der gezeichnet wird und jeder Kommentar, der eingefügt wird, wird mit Hilfe von verketteten Listen gespeichert.

Damit man beim Zeichnen der Punkte und beim Einfügen von Kommentaren unabhängig von der Fenstergröße ist, wurde ein Bezugskoordinatensystem eingeführt. Mit Hilfe der Funktion werden die aktuellen Koordinaten in Koordinaten des Bezugs-koordinatensystems umgewandelt. Auf diese Art und Weise befinden sich die gezeichneten Punkte und die Kommentare immer an der gleichen Stelle. Ohne die Einführung des Bezugskoordinatensystem würde sich der Punkt, an dem beispielsweise ein Kommentar eingefügt wurde beim Vergrößern bzw. Verkleinern des Fensters verschieben.

Um mit verketteten Listen arbeiten zu können, wird eine Struktur für die Punkte und eine für die Kommentare angelegt. Beim Erstellen eines Kommentars wird ein Eingabedialog geöffnet, der über die Ressourcen erzeugt wurde und beim Einfügen von mehreren Punkten werden diese und auch alle Kommentare mit Hilfe von den jeweiligen Zeigern aneinandergehängt.

Über die Funktionen int AddComment(int x, int y, char strText[255]) und int AddPoint(int x, int y) werden neue Elemente in die Listen eingetragen. Innerhalb dieser Funktionen wird auch der Sonderfall durchlaufen, dass noch nichts in den Listen steht. In diesem Fall wird ein Zeiger auf das erste Element der Liste erstellt. Die hierfür vorgesehenen Zeiger sind in der Struktur für die Bookmarks festgelegt.

```
struct StructBookmark
{
        float position;
        float camDistance;
        float camTwistAngle;
        float camTiltAngle;
        StructPoint *firstPoint;
        StructComment *firstComment;
        char strText[51];
```

} bookmark[MAX_BOOKMARKS+1];

Wird ein neuer Bookmark erstellt, erscheint ein Dialog, der über die Ressourcen angefertigt wurde und über den der Name der Bookmarks eingegeben werden kann. Will man den Namen eines Bookmarks ändern, erfolgt dies ebenfalls über einen Dialog, der über die Ressourcen erzeugt wurde. Möchte man einen erstellten Bookmark löschen, geschieht dies über die Funktion „void DeleteBookmark()“.

Für jeden Bookmark wird über die Funktion void DrawBookmarkBorder() ein Rahmen erzeugt und die zeitliche Position des Bookmarks, der Kameraabstand zum Ursprung, der Neigungswinkel und der Drehwinkel der Kamera werden ermittelt. Weiterhin läuft ein Zähler mit, der mit jedem erstellten Bookmark um eins erhöht wird.

Jedes erstellte Bookmark wird über AppendMenu(…) dem Menüpunkt Bookmarks hinzugefügt und ist darüber anwählbar. Möchte sich der Anwender einen bestimmten Bookmark ansehen, dann kann er diesen anwählen und gelangt an die entsprechende Stelle, indem die Funktion

```
void GotoBookmark(int nr)
{
        theViewer.setPosition(bookmark[nr].position);
        theViewer.setDistance(bookmark[nr].camDistance);
        theViewer.setTwistAngle(bookmark[nr].camTwistAngle);
        theViewer.setTiltAngle(bookmark[nr].camTiltAngle);
        SetPause(true);
        bookmarkActive = nr;
        SendMessage(hWnd, WM_PAINT, 0,0);
}
```

aufgerufen wird.

Damit sich ein anderer Anwender des Viewers die Bookmarks auch ansehen kann, wurde eine Speichern-Funktion erstellt. Diese schreibt eine Dateiversion und eine Identifizierung des Dateiformats in die Datei, damit das Programm beim Öffnen auch erkennt, ob es sich um eine Bookmark-Datei handelt. Als nächstes wird der Name der aktuellen Animationsdatei in die zu speichernde Datei geschrieben und schließlich werden die Kommentare und die gezeichneten Punkte in die Datei geschrieben.

Für den Fall, dass ein neuer Bookmark erstellt, dieser aber noch nicht abgespeichert wurde, ist die Variable „bool saveReminder = false;“ da. Diese Variable steht dann auf „false“, wenn die Bookmark-Datei abgespeichert wurde. Sobald der Anwender die Datei irgendwie verändert hat, wird „saveReminder“ auf „true“ gesetzt. Soll der Viewer jetzt geschlossen oder eine neue Datei geladen werden, wird der Wert von „saveReminder“ abgefragt und für den Fall, dass der Wert auf „true“ steht, wird die Funktion „bool SaveBookmarks(bool dialog)“ aufgerufen.

```
void SaveReminder()
{
	int result;
	if ( (saveReminder) && (bmCount > 0) )
	{
		result = MessageBox(hWnd, "Möchten Sie die Änderungen an Ihrer Bookmarkdatei vor dem Beenden speichern?", APPNAME, MB_YESNO | MB_ICONEXCLAMATION | MB_DEFBUTTON1);
		if (result == IDYES)
		{
			if (strcmp(strCurrentBookmarkFile,"") != 0)
				SaveBookmarks(false);
			else
				SaveBookmarks(true);
		}
	}
}
```

Bestehen für eine Animationsdatei bereits Bookmarks, können diese über den Menüpunkt “Öffnen“ unter dem Menü „Bookmarks“ geladen werden. Wählt man diesen Menüpunkt aus, wird die Funktion „bool LoadBookmarks()“ aufgerufen und der Inhalt der Datei wird ausgelesen. Dazu wird die Datei zum binären lesen geöffnet und alle Daten werden aus der Datei geladen:

Möchte der Anwender das Programm schließlich verlassen, wird eine Nachricht an die entsprechende Prozedur gesendet.

```
case WM_CLOSE:
                    SaveReminder();
                    theViewer.onShutdown();
                    KillGLWindow();
                    KillTimer(hWnd, ID_TIMER);
                    DeleteObject(hPen1);
                    DeleteObject(hPen2);
                    DeleteObject(hBrush);
                    return 0;
```

Vor dem Schließen wird noch mal die Funktion „saveReminder“ aufgerufen, die nötigen Objekte werden gelöscht und das Fenster wird zerstört.

4.3 Die Funktionsweise und Oberfläche des Viewers

Durch Öffnen der ausführbaren Datei metricminds.exe wird ein Fenster erzeugt, das über die bekannten Icons minimiert, maximiert und geschlossen werden kann. Am oberen Rand des Viewers befindet sich eine Menüleiste (siehe Abb. 19)

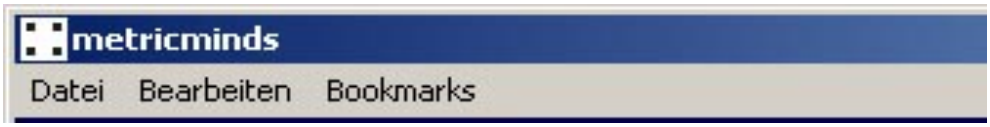

Abb. 19: Menüleiste des Viewers

und am unteren Rand befindet sich ein Slider mit Steuerbuttons (siehe Abb. 20).

Abb. 20: Slider des Viewers

Die Menüleiste hat drei Menüpunkte: Datei, Bearbeiten und Bookmarks (siehe Abb. 21, 22, 23).

Abb. 21: Menüpunkt Datei
mit Untermenü

Abb. 22: Menüpunkt Bearbeiten mit
Untermenü

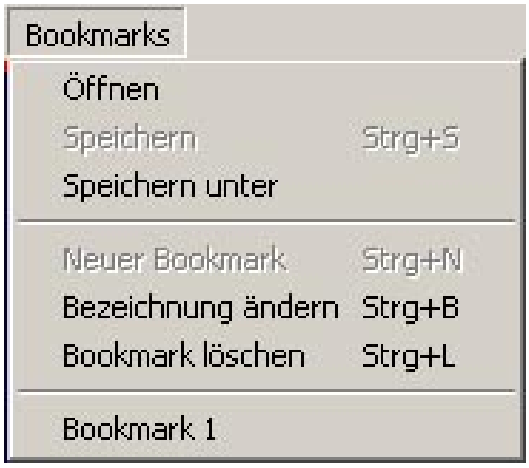

Abb. 23: Menüpunkt Bookmarks mit Untermenü

Wählt man einen der Menüpunkte aus, erscheint ein Untermenü. Bei Programmstart sind alle Menüpunkte der Untermenüs und der Slider mit den Buttons deaktiviert, bis auf zwei Punkte aus dem Menü „Datei“. Die aktivierten Punkte sind „Szene laden“ und „Beenden“.

Über „Beenden“ kann man, wie der Name schon sagt, das Programm beenden. Der Punkt „Szene laden“ ist dazu da, das 3D-Modell zu öffnen (siehe Abb. 24). Sobald ein Charakter geladen ist, werden auch andere Menüpunkte und der Slider aktiv.

Abb. 24: Geladenes Modell

Über den Punkt „Animation laden“ kann eine Animationsdatei geöffnet werden. Diese befindet sich automatisch in einer Endlosschleife und kann über den Slider und die Buttons gesteuert werden.

Der zweite Menüpunkt hat den Titel „Bearbeiten“. In seinem Untermenü befinden sich die Auswahlmöglichkeiten „Drehen/Skalieren“, „Kommentar“ und „Zeichnen“. Der Punkt „Drehen/Skalieren“ ist aktiv, sobald ein Modell geladen wird. Einen Kommentar eingeben und zeichnen kann man erst, wenn der Stop-Button gedrückt wurde. Alle drei Menüpunkte sind auch mittels

Tastenkombination anwählbar und vor dem jeweils aktiven Punkt erscheint ein Häkchen. Ist „Drehen/skalieren“ aktiv, kann man das Modell mit der linken Maustaste rotieren und mit der rechten Taste skalieren. Über den Menüpunkt „Kommentar“ kann man an der Stelle, an der man in den Viewer klickt, über ein Dialogfeld einen Text hinzufügen (siehe Abb. 25).

Abb. 25: Dialogfeld zum Einfügen eines Kommentars

Aktiviert man „Zeichnen“, steht ein Freiform-Stift zur Verfügung, mit dem man bei gedrückter Maustaste zeichnen kann (siehe Abb. 26).

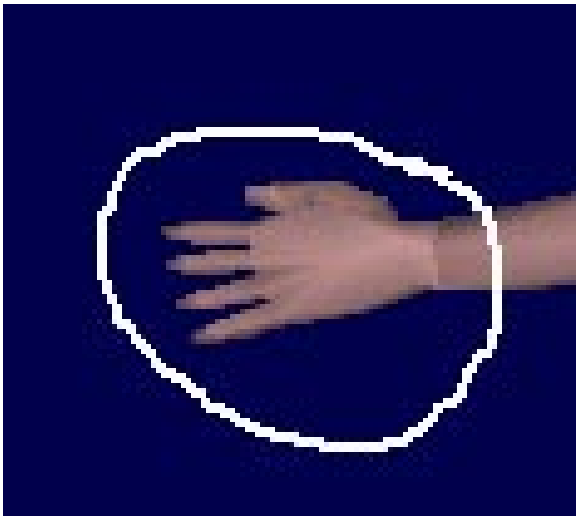

Abb. 26: Gezeichnete Markierung

Der letzte Punkt im Hauptmenü ist der Punkt „Bookmarks“. Sobald ein Modell geladen wurde, ist der Menüpunkt „Öffnen“ aktiv, damit bereits vorhandene Lesezeichen geladen werden können. Wenn der Stop-Button betätigt wurde, hat der Anwender die Möglichkeit, einen neuen Bookmark zu erstellen. Hat man ein Lesezeichen erzeugt, kann man diesem über einen Dialog einen Namen zuordnen (siehe Abb. 27).

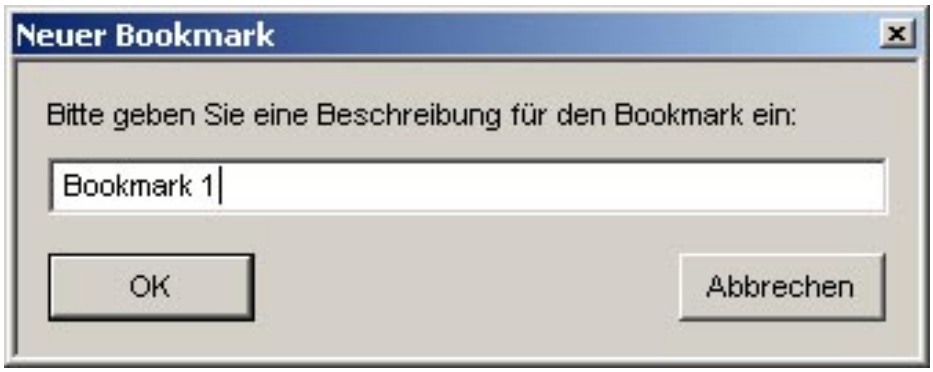

Abb. 27: Dialogfeld zum Benennen eines Bookmarks

Dieser wird dann in dem Menü „Bookmarks“ angezeigt und über seinen Namen anwählbar (siehe Abb. 28).

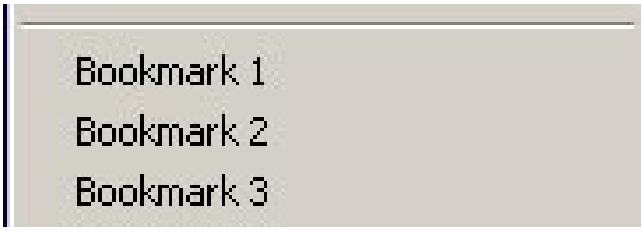

Abb. 28: Anwählbare Bookmarks

Gab es für die geladene Szene bisher noch keine Lesezeichen, wird der Menüpunkt „Speichern unter“ aktiv. Wird eine bereits vorhandene Datei verändert, hat man die zusätzliche Möglichkeit, den Punkt „Speichern“ anzuwählen.

Sobald Bookmarks bestehen, kann der Anwender den Namen im Nachhinein noch über „Bezeichnung ändern“ umbenennen (siehe Abb. 29)

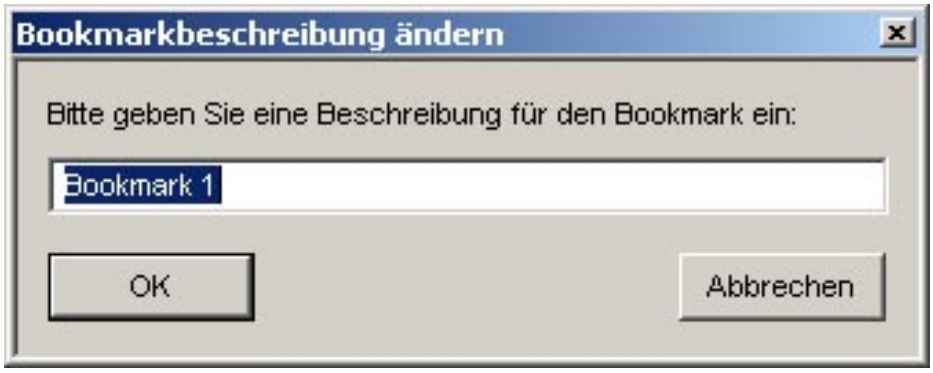

Abb. 29: Dialogfeld zum Ändern der Bookmarkbezeichnung

oder ein nicht mehr benötigtes Lesezeichen löschen (siehe Abb. 30).

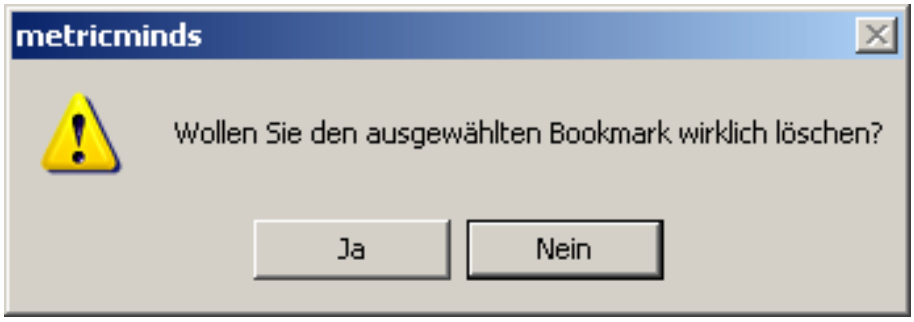

Abb. 30: Abfrage zum Löschen eines Bookmarks

Ein Lesezeichen wird auch dann erzeugt, wenn man die Zeichenfunktion wählt und anfängt zu zeichnen, oder wenn man die Kommentar-Funktion auswählt und beim Eingabedialog auf „Ok“ geht.

Hat man Lesezeichen erstellt und vergessen, abzuspeichern, wird beim Beenden des Viewers oder vor dem Laden einer neuen Datei ein Dialog angezeigt, der fragt, ob man die Änderungen an der Bookmarkdatei noch abspeichern will (siehe Abb. 31).

Abb. 31: Abfrage beim Schließen des Viewers

Wurde bereits eine Datei angelegt, werden die Änderungen einfach abgespeichert. Gibt es noch keine Datei, wird der „Speichern unter“ - Dialog angezeigt und nach dem Speichern wird das Programm beendet.

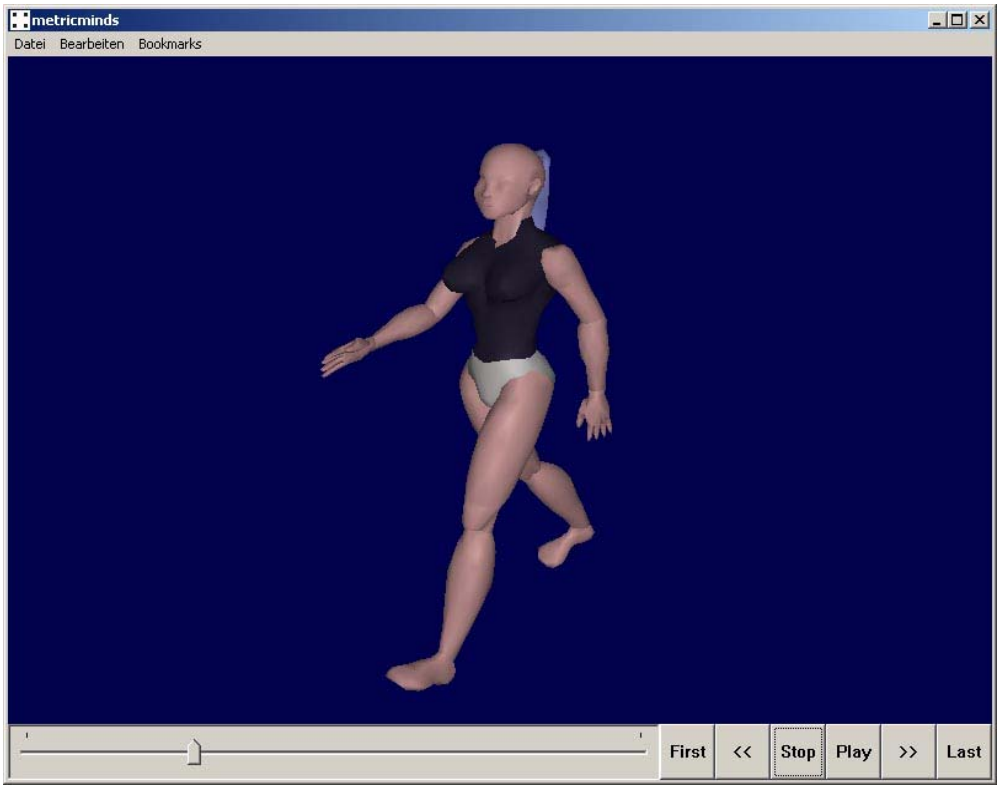

Abb. 32: Der fertige Viewer

4.4 Zusammenfassung

In diesem Kapitel wurde die Realisierung und die Funktionsweise des in dieser Diplomarbeit erstellten Viewers eingegangen.

In dem Kapitel „Die Erstellung des Viewers“, wurde beschrieben, wie der Viewer entstanden ist und wie einzelne Funktionen arbeiten. Dazu wurden unter anderem auch Ausschnitte aus einzelnen Funktionen verwendet und anschließend deren Aufgabe erläutert. Das nächste Kapitel heißt „Die Funktionsweise und Oberfläche des Viewers“ und beschreibt, wie der Titel schon sagt, die Oberfläche des

Viewers und wie dieser funktioniert. Anhand einzelner Screenshots wird in diesem Kapitel beschrieben, welche Funktionen der Viewer hat und wie er aussieht. Weiterhin wird mit Hilfe dieser Screenshots erläutert, wie der im Rahmen dieser Diplomarbeit erstellte Viewer funktioniert.

5 Abschlussbetrachtung

Ziel dieser Diplomarbeit war es, eine bereits vorhandene Freeware-Engine zu verwenden und diese zu modifizieren, um damit 3D-Modelle und Motion Capture Daten darzustellen.

Hier sollte noch mal erwähnt werden, dass es sich bei den 3D-Modellen ausschließlich um Charaktere handelt, die ursprünglich mit 3D Studio Max erstellt wurden.

Das Ziel, die Motion Capture Daten direkt zu laden, konnte leider nicht, wie in Kapitel 3.3 bereits beschrieben, erreicht werden, da bei dem Kombinieren, des gewünschten Animationsformats mit dem 3D-Modell Probleme auftraten.

Die dafür gefundene Alternative bedeutet zwar, dass mehr Arbeitsschritte als gewünscht aufgebracht werden müssen, um die Bewegungsdateien zu laden. Das Hauptziel, die Kommunikation zwischen Auftragnehmer und Auftraggeber zu erleichtern, wurde aber erfüllt.

Der Auftraggeber hat die Möglichkeit, die Bewegungen direkt auf seinem 3D-Modell zu sehen und kann unmissverständliche Angaben machen, was zu verbessern ist.

Für die Zukunft wäre es allerdings sinnvoll, solche Exporter, wie einer bereits für 3D Studio Max besteht, auch für Maya und andere Animationsprogramme zu realisieren. Damit wäre man nicht nur auf ein Animationsformat beschränkt und jeder Kunde könnte von Anwendungen wie dem, in dieser Diplomarbeit entwickelten, Viewer profitieren.

6 Literaturverzeichnis

[1] Vicon Motion Systems:
„Vicon 8 The Manual", Vicon Motion Systems

[2] Burggraf, Lorenz:
"Jetzt lerne ich OpenGL", München: Markt und Technik Verlag, 2003

[3] Woo, Mason; Neider, Jackie; Davis, Tom:
"OpenGL Programming Guide", 2. Auflage, Addison-Wesley Verlag, 1998

[4] OpenGL Homepage:
http://www.opengl.org

[5] cal3d Projekt-Homepage:
http://sourceforge.net/projects/cal3d

[6] c3d Homepage:
http://www.c3d.org

[7] Motion Lab Systems:
"C3D Format User Guide", Baton Rouge, LA: Motion Lab Systems, Inc., 2003

[8] Miller, Phil; u. a.:
„3D Studio Max R3", München: Markt und Technik Verlag, 2000

[9] Unreal Pictures, Inc.:
„Character Studio R2"; Autodesk, Inc.; Juni 1998

[10] MSDN-Bibliothek von Microsoft
http://msdn.microsoft.com/library/default.asp

[11] Petzold, Charles:
„Windows-Programmierung“, 5. Auflage, Unterschleißheim: Microsoft Press Deutschland, 2000