Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Institut für Simulation und Grafik

Diplomarbeit

Skriptbasierte Generierung von Animationen

für die medizinische Ausbildung und Therapieplanung

Konrad Mühler

Skriptbasierte Generierung von Animationen

für die medizinische Ausbildung und Therapieplanung

Diplomarbeit

an der
Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von:	Konrad Mühler
geboren am:	07. September 1979
in:	Magdeburg
Matrikelnummer:	158192

1. Gutachter:	Prof. Dr.-Ing. Bernhard Preim
2. Gutachter:	Dr.-Ing. Stefan Schlechtweg
Betreuer:	Prof. Dr.-Ing. Bernhard Preim Dipl.-Ing. Ragnar Bade Dipl.-Ing. Arno Krüger
Zeitraum der Diplomarbeit:	08.11.2004 - 08.04.2005

Selbstständigkeitserklärung

Hiermit versichere ich, Konrad Mühler (Matrikel-Nr. 158192), die vorliegende Arbeit allein und nur unter Verwendung der angegebenen Quellen angefertigt zu haben.

Konrad Mühler

Danksagung

Mein Dank gilt an erster Stelle meinem Betreuer Prof. Dr. Bernhard Preim, der mir während der gesamten Diplomzeit und darüber hinaus eine Betreuung zuteil werden ließ, wie man sie sich nur wünschen kann. Ragnar Bade sei gleichfalls für seine intensive Betreuung meiner Arbeit wie für die konstruktive Kritik und die Fähigkeit, auch noch die kleinsten Lücken in einer scheinbar schlüssigen Argumentation zu finden, gedankt. Arno Krüger danke ich für die oft aufmunternden Worte.

Meinem langjährigen Freund Ivo Rössling bin ich dankbar für viele fruchtbare Diskussionen und seine Tipps für den mathematischen und theoretischen Teil dieser Arbeit.

Alexandra Bear, Daniel Eicke, Sandra Hartmann, Anja Kuss, Björn Meyer und allen anderen Mitdiplomanden des Grafiklabors danke ich für die tägliche, oft spielerische Motivation sowie für den mentalen Ausgleich und die Geselligkeit in oft langen Tagen und Nächten meiner Diplomzeit.

Nicht zuletzt bin ich meinen Eltern, Ute und Roland Mühler, zu Dank verpflichtet. Sie ermöglichten mir durch ihre Unterstützung in den letzen Jahren nicht nur mein Studium. Sie trugen durch ihre Akribie beim Korrekturlesen und zahlreiche Hinweise zum Schreiben wissenschaftlicher Arbeiten mit zum Gelingen dieser Arbeit bei.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Diese Arbeit beschäftigt sich mit der Entwicklung eines Konzeptes einer Skriptsprache zur Animationsgenerierung. Dabei stehen die medizinische Animation für die chirurgische Ausbildung und die klinische Therapieplanung im Zentrum der Betrachtungen. Zusätzlich wird der Versuch unternommen, das Konzept für andere Anwendungsfelder offen zu gestalten. Exemplarisch wird das Konzept als Teil der Software MeVisLab[1] implementiert und getestet.

1.1 Motivation

Seit den 80er Jahren des 20. Jahrhunderts ist ein großer Fortschritt bei der rechnergestützten Visualisierung von Daten ganz allgemein und bei medizinischen Bilddaten im Speziellen festzustellen. Waren es in der Medizin zunächst nur zweidimensionale Röntgenbilder und seit der Erfindung der Computertomographie durch Allan M. Cormack und Godfrey Hounsfield auch Schnittbilder, so entwickelten sich seit den 80er Jahren erste computergenerierte, dreidimensionale Darstellungen. Diese dreidimensionalen Bilder kamen zunächst in der Ausbildung zum Einsatz. Mitte der 90er Jahre erweiterte sich das Anwendungsfeld aber auch zunehmend auf den klinischen Bereich (vgl. dazu [Preim und Peitgen 2004]). Dabei werden 3D-Darstellungen meist zu Diagnosezwecken, für Therapieentscheidungen oder Operationsplanungen genutzt.

1.1.1 Animationen in der klinischen Anwendung

Gekoppelt ist die reine Visualisierung der Daten oft mit einer Bildanalyse. Dabei geht es darum, einzelne Strukturen aus den Daten zu extrahieren und als Objekte verfügbar zu machen. Dieser Vorgang wird als Segmentieren bezeichnet. Daran anschließend bieten sich Anwendungen wie das Vermessen von Sicherheitsabständen und Volumina oder die Vorbetrachtung von möglichen Operationssituationen oder Therapien an. Solche Anwendungen sind ohne Zweifel nur möglich, wenn die dargestellten Daten patientenindividuell sind.

[1] http://www.mevislab.de

Besonders im klinischen Alltag spielt die effiziente Präsentation der Daten eine entscheidende Rolle. Dabei bieten 3D-Darstellungen einen großen Vorteil gegenüber reinen 2D-Schichtbildern. Sie vermitteln dem Betrachter ein besseres räumliches Bild und veranschaulichen topologische Zusammenhänge zwischen den anatomischen Strukturen, die andernfalls gar nicht oder nur schwer auszumachen sind.

Einen noch besseren Eindruck der dargestellten Daten, gerade auch bei der Betrachtung von räumlichen Zusammenhängen, erhält man, wenn ein und dieselbe Szene aus unterschiedlichen Blickrichtungen betrachtet werden kann. In solchen Fällen kommen zur Zeit Anwendungen zum Einsatz, in denen der Betrachter interaktiv die Sichten der Szene manipulieren kann (z.B. der InterventionPlanner in MeVisLab [Preim u. a. 2003]). Diese Systeme haben neben der reinen Sichtmanipulation aber meist einen noch weitaus größeren Funktionsumfang. Daher ist der Trainings- und Nutzungsaufwand solcher Applikationen immens und nicht selten für den klinischen Alltag unpraktikabel.

Der Vorteil der interaktiven Darstellung, die Sicht auf die Szene frei wählen zu können, kann auch als Nachteil ausgelegt werden. So besteht die Gefahr, dass der Arzt kritische Strukturen übersieht. Eine standardisierte Animation kann diesen Nachteil vermeiden, ohne dabei den Vorteil einer räumlichen Darstellung aus unterschiedlichen Sichten aufgeben zu müssen. Eine animierte Überblicksansicht bietet den Vorteil, dass der Arzt gezwungen ist, sich zunächst alle ermittelten Strukturen von Interesse anzusehen. Dies verringert die Gefahr, dass er Strukturen nicht bemerkt. Eine Animation ließe sich daher unter dem Aspekt der räumlichen Veranschaulichung zwischen der zweidimensionalen Projektion eines dreidimensionalen Bildes und einer vollständig interaktiven Darstellung einordnen (siehe Abbildung 1.1).

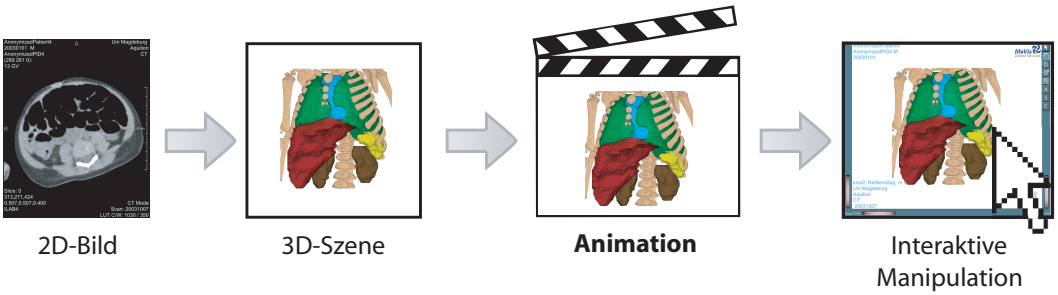


Abbildung 1.1: Einordnung einer Animation in die Möglichkeiten der Visualisierung medizinischer Daten

Im klinischen Alltag kommen schon heute Animationen zum Einsatz. Im Bereich der Leberchirurgie bietet das Centrum für Medizinische Diagnosesysteme und Visualisierung in Bremen (MeVis) Kliniken und Ärzten als Dienstleistung die Segmentierung und Analyse radiologischer Bilddaten an. Als Ergebnisse werden Bilder und Animationen bereitgestellt (siehe Abbildung 1.2). Solche Animationen beschränken sich bisher auf einfachste Rotationen der Szene um eine Achse. Hinzu kommt, dass die Erstellung solcher Animationen sehr aufwendig ist, da sie weder

adaptiv noch intuitiv erfolgt. Die Animationen müssen für jeden Patientendatensatz aufwendig von Hand erstellt werden.

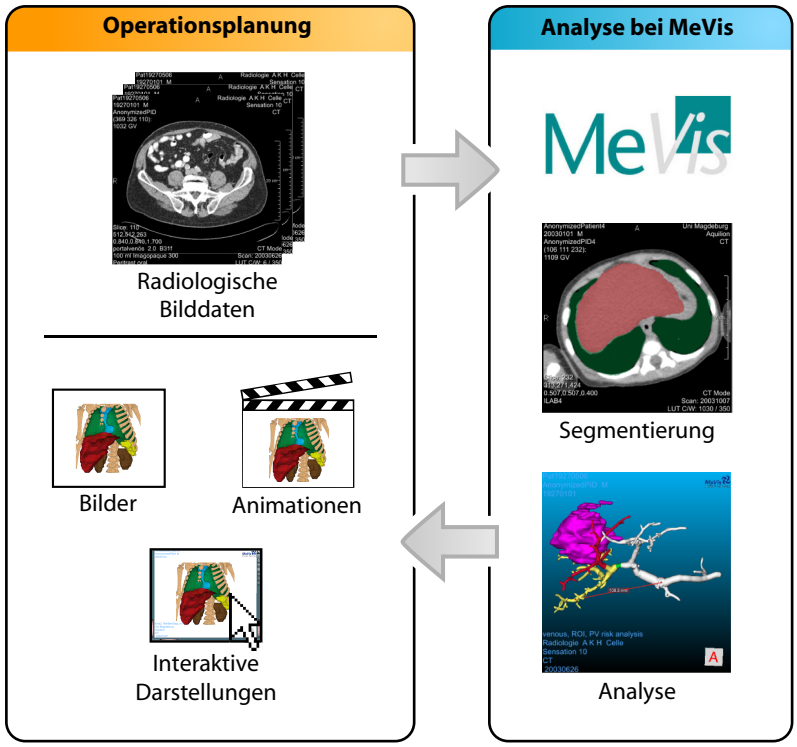


Abbildung 1.2: Schematische Darstellung des Prozesses der Segmentierung und Analyse radiologischer Daten im klinischen Alltag. Der Arzt sendet radiologische Bilddaten zu MeVis und erhält fertige Bilder und Animationen sowie in Ausnahmefällen eine interaktive Darstellung.

Daraus motiviert sich die Entwicklung eines Systems, mit dem sich medizinische Animationen möglichst einfach und effizient aus patientenindividuellen Daten erzeugen lassen. Die Adaptivität, also die Anpassung eines Systems an seine Umgebung, ist eine wesentliche Voraussetzung für ein Animationssystem, wenn es darum geht, Animationen aus patientenindividuellen Daten zu erzeugen.

1.1.2 Animationen in Ausbildung und Lehre

In der medizinischen Ausbildung war die möglichst genaue und detailgetreue Vermittlung von anatomischem Wissen immer von Bedeutung. So kamen schon sehr früh anatomische Zeichnungen wie in Abbildung 1.3 zum Einsatz. Diese Art der Darlegung anatomischen Wissens hat sich bis heute nur minimal geändert.

Die Alternative zu zwar plastisch wirkenden, aber dennoch zweidimensionalen Zeichnungen ist die direkte Arbeit an Leichen. Da diese aber nur in einem sehr begrenzten Umfang zur Verfügung stehen, wurde schon früh nach anderen Möglichkeiten gesucht. So wurde 1775 beispielsweise das Naturgeschichtliche

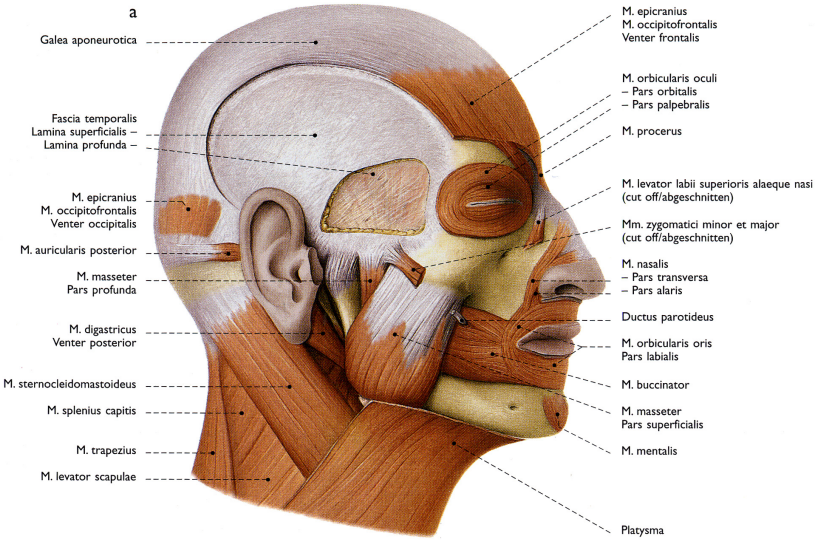


Abbildung 1.3: Anatomische Zeichnungen aus einem aktuellen Anatomieatlas [Wolf-Heidegger 2000]

Museum an der Universität Florenz gegründet. Dieses Museum legte eine Sammlung von über 1400 Wachsmodellen anatomischer Strukturen und Organe an (*La Specola*). Diese wurden aufbauend auf dem Studium von 200 Leichen angefertigt und waren sowohl Medizinstudenten wie auch der Bevölkerung zugänglich (vgl. [Mühler 2004]).

Zeigten bisherige Darstellungen in den meisten Fällen idealtypische Darstellungen gesunder oder pathologischer anatomischer Strukturen, so schufen die neu aufkommenden technischen Möglichkeiten der letzten Jahrzehnte die Voraussetzung, fallspezifische Daten zu visualisieren. Dies ermöglichte die Entwicklung von computergestützten Lernsystemen, die zusätzlich zur reinen Wissensvermittlung auch konkrete Fälle vorstellen und thematisch behandeln können. Ein solches System ist beispielsweise der LiverSurgeryTrainer für die Ausbildung von Chirurgen speziell in der Leberchirurgie [Bade u. a. 2004]. Dieses System bietet die Möglichkeit, dreidimensionale Darstellungen anatomischer Daten mit Wissen zur Operationsplanung und Therapieentscheidung zu verknüpfen. In solchen Systemen kommen Videos und Animationen zum Einsatz. Die Erstellung solcher Animationen ist, wie auch im klinischen Bereich, mit einem großen Aufwand verbunden. Dies stellt besonders unter dem Aspekt der Fallbasiertheit von neueren Lernsystemen ein Problem dar. Das ließe sich durch ein System zur Erzeugung von Animationen aus fallspezifischen Daten lösen.

Anhand dieses Beispieles leitet sich eine zweite Motivation der vorliegenden Arbeit ab: die Erstellung von medizinischen Animationen für die Ausbildung. Der Schwerpunkt liegt hier allerdings weniger auf einer möglichst effizienten Erzeugung hinsichtlich der Erstellungszeit oder dem Trainingsaufwand der Nutzer. Vielmehr ist in diesem Fall das Ziel, möglichst qualitativ hochwertige Animationen mit einem großen Maß an Freiheitsgraden seitens des Anwenders zu generieren.

Da es bei der Nutzung von Animationen in der Ausbildung immer auch um die Präsentation von Wissen geht, werden an solche Animationen spezielle Anforderungen gestellt. So kommt es darauf an, Objekte hervorzuheben, um auf Sachverhalte hinzuweisen. Dafür wurden in der Vergangenheit verschiedene Ansätze entwickelt, die es zu untersuchen und gegebenenfalls in das Konzept einzubinden gilt.

Ein weiteres Anwendungsfeld medizinischer Animationen sind Präsentationen. So sind Anwendungen im Bereich von Diskussionsbeiträgen oder Kongressen denkbar. Hier stehen weniger die Diagnose, Analyse oder die Wissensvermittlung im Vordergrund als vielmehr die Präsentation von patientenindividuellen Daten. Durch ein System zur effizienten Animationsgenerierung wäre an dieser Stelle neben der Fallbasiertheit auch ein hohes Maß an Aktualität gegeben.

1.1.3 Zielgruppen

Als Zielgruppe eines Animationssystems lassen sich zwei Gruppen ausmachen: Anfänger und Experten. Die unterschiedliche Erfahrung bezieht sich dabei auf den Umgang mit Skriptsprachen und die Animationsgenerierung. Im Gegensatz dazu haben beide Gruppen ein großes medizinisches und anatomisches Wissen. Die zwei Zielgruppen finden sich auch in den zwei Kernanwendungsfeldern wieder, aus denen sich die Notwendigkeit eines Animationssystem motiviert: die Operations- und Therapieplanung und die medizinische Ausbildung.

1.1.4 Einsatz von Skripten zur Animationsgenerierung

Kern des Animationssystems soll eine Skriptsprache sein. Skripte als Grundlage von Animationen bieten sich aus mehrerlei Gründen an. Skripte sind textbasiert, im Klartext lesbar und damit mit einem beliebigen Texteditor am Computer zu bearbeiten. Das macht sie unabhängig von Autoren- oder Präsentationssystemen, in denen später die Animationen erzeugt werden. Zusammen mit den segmentierten Daten bilden Skripte ein effizientes Konglomerat, welches sich an andere Autoren weitergeben lässt, sodass diese die Animationen nach ihren Wünschen und Erfordernissen anpassen können.

Skriptsprachen können in ihrem Aufbau und ihrer Syntax flexibel definiert werden, sodass sie sowohl den Ansprüchen von Experten wie auch den Erfordernissen von Anfängern genügen.

1.2 Aufgabenstellung

Die Aufgabe der vorliegenden Arbeit besteht in der Entwicklung und prototypischen Umsetzung eines Konzeptes zur Erstellung skriptbasierter Animationen für die medizinische Ausbildung und Therapieplanung. Skripte kommen dabei als Animationsbeschreibung zum Einsatz. Über Skripte werden durch einen Autor Anweisungen[2] an ein Animationssystem zum Verhalten der Szene und der Objekte in der Szene gegeben.

Das Skriptkonzept soll im Rahmen dieser Arbeit implementiert werden. Dies soll prototypisch in der medizinischen Bildanalysesoftware MeVisLab geschehen. Diese Software bietet ein umfassendes Paket an Funktionen zur Visualisierung von medizinischen Daten und bietet die Möglichkeit für Entwickler, eigene Erweiterungen direkt in der Software verankern zu können. Das erleichtert eine spätere Überführung der entwickelten Komponenten in eine alltägliche Nutzung.

1.3 Ergebnisse

Im Zuge dieser Arbeit ist es gelungen, die gestellte Aufgabe zu erfüllen. Es wurde eine Skriptsprache entwickelt, mit welcher Experten ebenso wie unerfahrene Nutzer Animationen erstellen können. Das Konzept wurde am Beispiel medizinischer Animationen prototypisch in der Bildanalyse Software MeVisLab implementiert. Damit ist es möglich, Animationen aus patientenindividuellen Bilddaten schnell und effizient zu generieren. Diese Animationen können sowohl in der medizinischen Ausbildung wie auch bei der Therapieplanung zum Einsatz kommen. Weiterhin wurden Anwendungsmöglichkeiten im Bereich interaktiver (medizinischer) Animationen und zur Erzeugung von Einzelbildern vorgestellt.

Es wurde gezeigt, dass die Anwendung der Skriptsprache nicht auf den medizinischen Bereich beschränkt ist, sondern universell für die Animation fast aller objektorientierten Darstellungen genutzt werden kann.

1.4 Gliederung der Arbeit

Die vorliegende Arbeit ist wie folgt gegliedert:

In **Kapitel 2** werden grundlegende Begriffe aus den Bereichen der Animation und der Skriptsprachen eingeführt und definiert. Es werden Animationstechniken verschiedener medizinischer Videos untersucht und die Datengrundlage der in

[2] Als *Anweisung* wird in dieser Arbeit im Unterschied zu einem *Befehl* die Gesamtheit aus Parametern, Befehlsbezeichnung und sonstigen Angaben, die zur Ausführung einer Anweisung notwendig sind, betrachtet. Ein *Befehl* stellt in dieser Arbeit die Bezeichnung einer *Anweisung* dar.

dieser Arbeit generierten Animationen beschrieben. Abschließend werden die Anforderungen an ein Skriptkonzept untersucht und genauer umrissen.

In **Kapitel 3** werden verwandte Arbeiten beschrieben und analysiert. Besonderes Augenmerk wird dabei auf Arbeiten zur automatischen Animationsgenerierung und auf verschiedene Skriptsprachen gelegt. Es werden zwei Softwarelösungen aus dem Bereich der medizinischen Visualisierung (MeVisLab und Amira) hinsichtlich ihrer Animationseigenschaften untersucht. Aus dem Bereich der medizinischen Lernsysteme wird der LiverSurgeryTrainer vorgestellt.

Kapitel 4 widmet sich der genauen Beschreibung der in dieser Arbeit entwickelten Skriptsprache. Es wird dabei zunächst auf den grundlegenden Gedanken der Mehrstufigkeit eingegangen, bevor die Syntax der Sprache und das Prinzip der Anweisungsersetzungen beschrieben werden. Abschließend werden ausgewählte Anweisungen vorgestellt.

Die Implementierung der Skriptsprache wird in **Kapitel 5** vorgestellt. Dabei wird zunächst auf die Grundlagen der Entwicklungsplattform MeVisLab eingegangen bevor die Applikation des AnimationScripters genauer beschrieben wird.

In **Kapitel 6** wird ein Ausblick über weitere Anwendungsmöglichkeiten und Erweiterungen der Skriptsprache gegeben. Genauer betrachtet werden dabei interaktive Animationen. Außerdem wird eine grafische Benutzerschnittstelle skizziert.

Kapitel 7 fasst die Ergebnisse der Arbeit kurz zusammen. Im **Anhang** finden sich ausgewählte Abbildungen und einige Beispiele für Skripte, die in der hier entwickelten Skriptsprache verfasst wurden.

Kapitel 2

Grundlagen und Anforderungsanalyse

2.1 Begriffe der Animation

Das Wort Animation leitet sich aus dem lateinischen Wort *animare* ab und bedeutet *zum Leben erwecken*. Eine Animation wird in dieser Arbeit wie folgt definiert:

> *Der Begriff der Animation beschreibt die zeitliche Veränderung von Bildern.*

Die Definition impliziert zwei grundlegende Charakteristika: Zum Einen das Vorhandensein von Bildern und zum Anderen eine zeitliche Komponente der Veränderung. Diese beiden Aspekte werden im Folgenden näher betrachtet.

In der Definition wird bewusst keine Aussage über den Ursprung der Bilder gemacht. Dieser kann vielfältiger Art sein. Er umfasst zweidimensionale, flächige Darstellungen genauso wie dreidimensionale Modelle. Dreidimensionale Modelle werden dabei oft in Zusammenhang mit der Bilderzeugung am Computer gebracht. Dies ist jedoch nicht immer zwingend der Fall und war auch zu Beginn der Geschichte der Animation bzw. der Trickfilme noch gar nicht möglich. Im Bereich der klassischen Trickfilme kommen ebenfalls schon 3D-Modelle als Grundlage der Bilderzeugung zum Einsatz. So basieren die Filme von Wallace&Gromit oder die Trickfilme der Plonster in der Sesamstraße auf aus Knete geformten Objekten und Figuren. Bei der Animation am Computer liegen alle Objekte im Unterschied dazu als dreidimensionale Geometriedaten vor.

Bei der Gesamtheit aller Objekte spricht man von einer Szene. Zusätzlich zu den Objekten kann eine Szene noch weitere Informationen, zum Beispiel zur Beleuchtung oder dem Hintergrund, enthalten. Eine dreidimensionale Szene wird allerdings erst durch einen Betrachter zu einem zweidimensionalen Bild. Dieser, als Projektion bezeichnete Vorgang, wird im Allgemeinen durch eine Kamera repräsentiert. Ist diese Kamera beim Trickfilm noch eine reale, so ist sie bei der Bilderzeugung am Computer eine rein virtuelle. Einen guten Überblick über die grundlegenden Techniken der Bilderzeugung am Computer bieten [Angel 2000] und [Foley 2000].

Der zweite Teil der Animationsdefintion, die zeitliche Veränderung der Bilder, erfordert eine kurze Betrachtung der menschlichen Wahrnehmung von Bewegungen. Das menschliche Auge weist eine Trägheit auf, die es Bilder nur bis zu einem bestimmten Grad als unterschiedliche statische Bilder wahrnehmen lässt. Wechseln

die Bilder vor dem Auge des Betrachters zu schnell, werden sie als Bewegung oder kontinuierliche Veränderung wahrgenommen. Nach [Mealing 1992] ist dies ab einer Bildfrequenz von 12 Bildern pro Sekunde der Fall. Man spricht in diesem Zusammenhang auch von der Framerate. Die übliche Einheit der Framerate ist *frames per second* (fps). Gebräuchlicher als der Grenzwert von 12fps sind Frameraten zwischen 18 und 24fps. Ein Frame ist ein einzelnes Bild einer Animation.

2.2 Klassifikation von Animationen

In [Masuch 2001] werden im Bezug auf den Grad der Computerunterstützung drei Arten der Animation unterschieden:

- traditionelle Animation
- computergestützte Animation
- Computeranimation

Bei der traditionellen Animation handelt es sich zum Einen um von Hand gezeichnete Zeichentrickanimationen. Zum Anderen gehören hierzu auch in der Stop-Motion-Technik aufgenommene Filme, zum Beispiel mit Knetfiguren.

Bei der computergestützten Animation kommt der Computer bei der Bearbeitung einzelner Produktionsschritte zum Einsatz. Dies kann zum Beispiel die Colorierung oder die Steuerung der Kamera sein.

Bei der Computeranimation werden die Darstellungen der Animation vollständig aus dreidimensionalen Modellen durch den Computer berechnet. Zusätzlich wird bei dieser Art der Animation noch zwischen einer online und offline gerenderten Animation unterschieden. Das Rendern beschreibt dabei den Prozess der Bildberechnung durch den Computer. Online gerenderte Animationen werden erst in dem Moment erzeugt, in dem sie angezeigt werden. Man kann daher auch von Echtzeitanimationen sprechen. Offline gerenderte Animationen werden erzeugt und gespeichert, bevor sie angezeigt werden.

Nach [Masuch 2001] wird eine Computeranimation wie folgt definiert:

> *"Die Computeranimation ist eine computerbasierte kontinuierliche Berechnung von Zuständen eines dynamischen Systems zur Darstellung sich in Raum und Zeit verändernder Strukturen durch eine Sequenz aufeinanderfolgender Bilder."*

Im Folgenden wird das Wort der Animation synonym mit dem Wort der Computeranimation verwendet.

2.3 Animationspipeline

In [Masuch 2001] wird weiterhin eine Animationspipeline vorgestellt (siehe Abbildung 2.1), jedoch leider nicht näher erläutert. Es wird nicht ausgeführt, ob sich der Transformationsbegriff nur auf Bewegungen der Kamera bezieht oder ob er eine Transformation im Sinne einer Veränderung aller zeitlich dynamischen Parameter meint.

Abbildung 2.1: Animationspipeline nach [Masuch 2001]

Daher wird nachfolgend eine abgewandelte Animationspipeline entworfen (siehe Abbildung 2.2). Diese gliedert sich in den Animationsentwurf, die Entwurfsanalyse, die Parametermanipulation, das Rendern und Speichern des Frames und die Animationsgenerierung aus den Frames. Mit Blick auf das in dieser Arbeit entwickelte Skriptkonzept entspricht die Entwurfsanalyse der in Kapitel 4 näher erläuterten mehrstufigen Skriptumwandlung. Die Schritte der Parametermanipulation, das Rendern sowie das Speichern der Frames kommt der in Kapitel 5 vorgestellten Implementierung gleich.

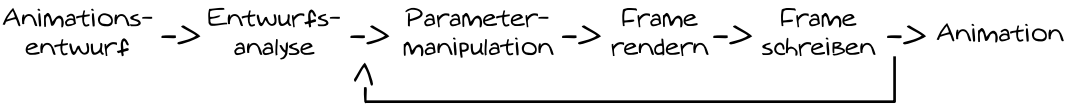


Abbildung 2.2: Entwurf einer neuen Animationspipeline

2.4 Keyframing und Keystates

Bei der Erzeugung von Computeranimationen kommt in den meisten Fällen die Technik des Keyframing zur Anwendung. Sie hat ihren Ursprung in der traditionellen Zeichentricktechnik. Dort ist es üblich, dass erfahrene Zeichner nur bestimmte Schlüsselbilder (Keyframes) zeichnen. Die Arbeit an den einzelnen Bildern zwischen diesen Keyframes werden von weniger erfahrenen Zeichnern ausgeführt. Diese Technik wurde für die Computeranimation übernommen. Hier werden die einzelnen Bilder zwischen den Keyframes vom Computer interpoliert. Dieser Vorgang wird als *in-betweening* bezeichnet. Es handelt sich dabei um ein bildbasiertes Verfahren auf 2D-Bildern.

Für Computeranimationen kann noch eine zweite Art des Keyframing ausgemacht werden, das parametrisierte Keyframing. Bei diesem Verfahren bilden dreidimensionale Szenen- und Objektbeschreibungen die Grundlage. Es werden nicht mehr ganze

Frames als Anfangs- und Endpunkt einer Interpolation begriffen, sondern stattdessen Schlüsselzustände einzelner Objekte oder Parameter definiert, die über die Zeit interpoliert werden. Man kann in diesem Zusammenhang auch von *Keystates* sprechen. Da Keystates bisher noch nicht in der Literatur eingeführt wurden, werden sie hier wie folgt definiert:

> *Keystates sind Zustände einzelner Parameter oder Objekte einer Animation, zwischen welchen über die Zeit Zwischenwerte interpoliert werden und so eine Animation erzeugt wird.*

2.5 Möglichkeiten der Generierung einer Animation

Die Person, die eine Animation plant und deren Erzeugung vornimmt oder anstößt, wird als Autor bezeichnet. Der Autor muss dabei ein gewisses Maß an Kreativität aufweisen, da er die Animation zunächst mental erstellen und dann auf dem jeweiligen System umsetzen muss. Dies wird durch eine teilweise visuelle Eingabe, bei der der Autor die aktuelle Darstellung der Animation direkt sieht, zwar erleichtert, jedoch muss der Autor auch hier eine grundlegende Vorstellung vom Ablauf einer Animation und von den Schlüsselszenen haben. Es gibt verschiedene Möglichkeiten, eine Computeranimation zu generieren. Einige der Möglichkeiten sollen im Folgenden kurz vorgestellt werden.

2.5.1 Abfilmen

Eine Möglichkeit, die der klassischen Filmproduktion nahe kommt, ist das Abfilmen einer Szene. Der Computer übernimmt dabei die Funktion einer Kamera und speichert die Ansichten des Bildschirms oder Teile davon als Bildfolge, aus der eine Animation erzeugt wird. Solche Systeme gibt es zum Beispiel in Form des Programms SnagIt von TechSmith[1]. Während die Bildschirmansichten gespeichert werden, werden vom Autor der Animation die Parameter der Szene oder der Objekte direkt verändert. Es gibt verschiedene Ausprägungen dieser Technik. So kann die Animationsaufzeichnung angehalten und fortgesetzt werden oder es werden nur Bilder für die Animation gespeichert, wenn Veränderungen auf dem Bildschirm auftreten.

Der Autor muss bei jeder Animationserzeugung sämtliche Handlungen aufs Neue ausführen, wenn er eine Animation wiederholt oder in ähnlicher Form erstellen möchte. Dies hat den großen Nachteil, dass die Animation dadurch nicht reproduzierbar ist.

2.5.2 Bild-für-Bild-Technik

Eine weitere Technik, die ebenfalls auf einer Technik des klassischen Films beruht, ist die Bild-für-Bild- oder Stop-Motion-Technik. Bei dieser werden die Bilder, aus denen

[1] http://www.techsmith.com

eine Animation besteht, einzeln und nicht in Echtzeit aufgenommen. Die Technik hat ihren Ursprung bei den Trickfilmen, wo jedes Bild einzeln gezeichnet und dann abfotografiert wird. Für eine Animationserzeugung am Computer stellt der Autor die Parameter der Szene und der Objekte für jedes Bild gesondert ein und speichert das resultierende Bild. Danach wird aus der Bilderfolge eine Animation erzeugt.

Sind dabei sämtliche Parameter numerisch einzustellen, so ist dieses Verfahren reproduzierbar. Der Autor bzw. die Visualisierungssoftware müsste sich nur die Parameter für jede Szene merken. Allerdings ist dieses Verfahren extrem aufwändig, selbst wenn man eine geringe Framerate zugrunde legt. Auch sind, wie an Software-Beispielen in den Abschnitten 3.5 und 3.6 gezeigt wird, oft nicht alle Parameter numerisch einstellbar. Kritisch ist dies bei Veränderungen der Kameraparameter. Diese lässt sich meist nur mit Hilfe der Maus direkt in der Szenendarstellung steuern. So kann bei gedrückter Maustaste zum Beispiel die Kamera um die Achsen eines Koordinatensystems rotiert werden. Auch wenn die theoretische Möglichkeit besteht, solche Manipulationen gesondert zu erfassen und zu speichern (zum Beispiel als Kamerapfad), so ist es doch immer noch ein sehr ungenaues und unpraktikables Verfahren.

2.5.3 Skript

Als dritte Möglichkeit, Animationen computergesteuert zu erzeugen, bietet sich die Verwendung von Skripten an. In Abschnitt 2.6 wird genauer auf die Definition und die Eigenschaften eines Skriptes eingegangen. An dieser Stelle ist der Ansatz ausreichend, dass ein Skript eine Folge von Anweisungen ist, die automatisch nacheinander ausgeführt werden. Auch das Skript hat seinen Ursprung im klassischen Film. Dort ist ein Skript eine Sammlung von Anweisungen für Schauspieler und Kameraführung.

Der Vorteil von Skripten liegt in der Reproduzierbarkeit der durch sie erzeugten Animationen. Dies macht das skriptgesteuerte Erzeugen von Animationen zu einem vielversprechenden Verfahren, auch und gerade unter Berücksichtigung der in Abschnitt 2.9 geschilderten Anforderungen.

Beziehen sich die Technik des Abfilmens und die Bild-für-Bild-Technik ausschließlich auf die Erzeugung von offline gerenderten Animationen, so lässt sich die Skripttechnik zusätzlich auch für online gerenderte Animationen einsetzen.

2.5.4 Interaktive Animationen

Als interaktive Animation wird die Kombination einer interaktiven Darstellung mit Animationen bezeichnet [Preim 1997]. Es handelt sich dabei um online gerenderte Animationen. Hat der Nutzer bei offline gerenderten Animationen keine Möglichkeit, in den Ablauf der Animation einzugreifen, so werden ihm diese Möglichkeiten bei der interaktiven Animation zur Verfügung gestellt. Der Nutzer kann so beispielsweise die Fahrt einer Kamera entlang eines Pfades unterbrechen, sich selbst orientieren und dann die Fahrt der Kamera entlang des Pfades fortsetzen. Die Beschreibung

von interaktiven Animationen kann beispielsweise direkt im Animationssystem selbst oder über Skripte erfolgen.

2.6 Skript und Skriptsprache

In diesem Abschnitt werden die theoretischen Grundlagen von Skripten und Skriptsprachen behandelt.

Der Begriff des Skriptes kommt ursprünglich aus der Filmindustrie, in der ein Skript eine Folge von Anweisungen für Schauspieler und Kameraführung enthielt. Einzug in die Informatik hielt der Begriff Anfang der 70er Jahre, als die Entwickler von UNIX den Begriff des *shell scripts* prägten. Ein *shell script* ist eine Reihe von Systemanweisungen, die in einer Textdatei gespeichert und bei der Abarbeitung des Skriptes so ausgeführt werden, als seien sie nacheinander über die Tastatur eingegeben worden.

Nach [Barron 2000] haben sich drei Anwendungsfelder für Skripte in der Informatik entwickelt:

1. Skripte werden zur Entwicklung von Programmen oder Teilen von Programmen aus vorgefertigten Komponenten genutzt. Die Komponenten selbst müssen nicht in einer Skriptsprache verfasst, sondern können mit Hilfe konventioneller Programmiersprachen programmiert worden sein. Anwendung findet dieses Prinzip vor allem in Bereichen, in denen eine schnelle Entwicklungszeit von Programmen entscheidend ist. Oft werden auch graphische Benutzeroberflächen mit Hilfe von Skripten aus vorgefertigten Komponenten erzeugt (zum Beispiel bei Microsoft Visual Basic[2] oder Delphi[3]).

2. Skripte kommen bei der Manipulation, Anpassung und Automatisierung von Programmen zur Anwendung. Solche Programme müssen dafür eine Schnittstelle anbieten, die sich über eine Skriptsprache steuern lässt.

3. Skripte dienen der Automatisierung bei der Abarbeitung von Systembefehlen in Betriebssystemen. Dies ist das oben beschriebene klassische Anwendungsgebiet, über das Skripte in die Informatik eingeführt wurden.

Ein Skript wird in einer Skriptsprache verfasst. Diese definiert Befehle, Datentypen und deren Verwendung. Skriptsprachen lassen sich von Programmiersprachen abgrenzen. Der Code, der in einer Programmiersprache verfasst wurde, muss vor der eigentlichen Ausführung zunächst in eine Maschinensprache transformiert werden [Schneider 1997]. Der Vorgang gliedert sich in Kompilierung und Binden. Erst nach Abschluss dieses Prozesses kann das Programm ausgeführt werden.

Dies entfällt bei Skripten. Der Code eines Skriptes kann sofort ausgeführt werden. Man spricht bei Skripten in diesem Zusammenhang vom Interpretieren des Codes. Ein Interpreter arbeitet das Skript vom Anfang zum Ende hin seriell ab und führt

[2] http://msdn.microsoft.com/vbasic
[3] http://www.borland.de/delphi

die Anweisungen direkt aus. Im Vergleich zu Programmiersprachen hat dies ebenso Vor- wie auch Nachteilen, die teilweise von [Barron 2000] inspiriert sind:

+ Die Übersetzung und das Binden eines Programms macht es in den meisten Fällen plattformabhängig. Dies ist bei Skripten nicht gegeben, da nur der Interpreter des Skriptes an die Plattform gebunden ist, auf der er läuft. Das Skript selbst ist zumindest theoretisch plattformunabhängig.

+ Da Skripte aufgrund ihrer oben geschilderten Anwendungsfelder selten ganze Programme in ihrem vollen Funktionsumfang beschreiben müssen (und wenn sie dies tun, auf vorgefertigte Komponenten zurückgreifen können), kann die Menge der Befehle und Datentypen, die dem Autor eines Skriptes zur Verfügung stehen, stark eingegrenzt werden. Dadurch lässt sich ein höherer Abstraktionsgrad der Skriptsprache erreichen. Das hat den Vorteil einer einfacheren und leichter lesbareren Skriptsprache.

+ Der durch die Einsparung im Befehlsumfang einer Skriptsprache entfallende Overhead kann dazu genutzt werden, dem Autor auf das Anwendungsgebiet des Skriptes spezialisierte Befehle anzubieten[4].

- Durch das direkte Interpretieren von Skripten kommt es zu keiner Optimierung hinsichtlich Laufzeitverhalten und Speichernutzung. Dies macht Skripte ineffizienter und langsamer im Vergleich zu konventionellen Programmen.

- Ein Skript muss bei jeder Ausführung erneut komplett eingelesen und interpretiert werden.

+ Ein Skript bleibt für den Autor lesbar und kann jederzeit verändert werden.

In [Ecma 1999] wird eine Skriptsprache wie folgt definiert:

> *A **scripting language** is a programming language that is used to manipulate, customise, and automate the facilities of an existing system. In such systems, useful functionality is already available through a user interface, and the scripting language is a mechanism for exposing that functionality to program control. ... A scripting language is intended for use by both professional and nonprofessional programmers. To accommodate non-professional programmers, some aspects of the language may be somewhat less strict.*[5]

[4] Beispiele für eine Spezialisierung von Skriptsprachen sind:
- PHP und Perl kommen vor allem für Anwendungen im WWW zum Einsatz
- MATLAB für mathematische Berechnungen
- TCL/Tk dient der Generierung von graphischen Benutzeroberflächen

[5] Eine Skriptsprache ist eine Programmiersprache, die genutzt wird, um die Möglichkeiten eines bestehenden Systems zu manipulieren, anzupassen und zu automatisieren. In solchen Systemen ist eine nützliche Funktionsvielfalt bereits durch eine Benutzerschnittstelle gegeben. Die Skriptsprache ist ein Mechanismus, der diese Funktionen mit einer Programmkontrolle verbindet. ... Eine Skriptsprache ist für die Nutzung sowohl durch professionelle als auch unprofessionelle Programmierer gedacht. Um unprofessionellen Programmierern entgegenzukommen können einige Aspekte der Sprache weniger streng sein. (Übersetzung aus dem Englischen durch den Autor dieser Arbeit)

Hervorzuheben ist bei dieser Definition der Punkt, dass eine Skriptsprache das Ziel hat, sowohl von professionellen wie auch unprofessionellen Nutzern angewendet werden zu können.

2.7 Untersuchung von Animationstechniken

Ein erster Schritt hin zu einem Animationskonzept für medizinische Animationen ist die Untersuchung bereits existierender Animationen. Dabei geht es hauptsächlich darum, Animationstechniken zu extrahieren. Die Aufstellung der Animationstechniken in den folgenden Abschnitten ist von der Beobachtung anatomischer Animationen geprägt und inspiriert. Diese wurden teilweise in Kooperation mit klinischen Partnern oder für Operationsplanungen bei MeVis in Bremen erstellt.

Gemein war allen Videos, dass es sich um animierte 3D-Darstellungen handelte. Dargestellt wurden segmentierte Organe oder andere medizinisch interessante Strukturen wie Tumore oder Gefäßbäume.

In [Haase 2004] werden in diesem Zusammenhang verschiedene Bereiche für Animationen in der Medizin ausgemacht:

- Präsentation von Volumenmodellen
- Navigation in Volumenmodellen
- Erklärung komplexer Strukturen und Zusammenhänge
- Übergänge bei Hervorhebungswechsel und Heranführen an eine Visualisierung

Im Folgenden wird eine davon abweichende Einteilung der Animationen bzw. Teile der Animationen anhand der verwendeten Techniken vorgenommen.

2.7.1 Änderung der Kameraparameter

Die Kamera bildet innerhalb einer zu animierenden Szene ein zentrales Element. Durch die Veränderung der Position oder Blickrichtung einer Kamera erfährt der Betrachter einen räumlichen Eindruck. Indem eine Szene aus unterschiedlichen Richtungen betrachtet wird, kann die Lage der Objekte zueinander und ihre Größe besser beurteilt werden, als dies bei einzelnen Bildern der Fall wäre.

Eine erste Möglichkeit der aktiven Nutzung der Kamera ist die Präsentation der Szene für den Betrachter. Dazu wird die Kamera in den meisten Fällen um die z-Achse (Senkrechte) rotiert und so dem Betrachter ein Überblick über alle Objekte der Szene gegeben (siehe Abbildung 2.3). Die Rotation ist jedoch nicht auf die Senkrechte beschränkt und kann um jede Achse erfolgen.

Eine weitere Möglichkeit der Präsentation von Szenen ist der Zoom[6] der Kamera. Dabei kann zur besseren Sichtbarkeit oder Unterstreichung der Bedeutung eines Objektes zu diesem hingezoomt oder für einen besseren Überblick aus der Szene herausgezoomt werden (siehe Abbildung 2.4).

Wichtige Kameraparameter sind der Pfad und die Blickrichtung der Kamera (vgl. [Helbing 2004]). Auch wenn eine Rotation der Szene eine Rotation der Kamera um diese und damit eine Positionsänderung darstellt, so soll im Unterschied dazu unter einer Positionsänderung der Kamera im Folgenden die Bewegung der Kamera entlang eines Pfades betrachtet werden. Im einfachsten denkbaren Fall ist ein solcher Pfad eine lineare Strecke zwischen zwei Punkten. Die Kamera wird entlang dieses Pfades bewegt. Für die Wahl der Blickrichtung gibt es dabei zwei Möglichkeiten. Sie kann konstant bleiben oder sich verändern, indem sie sich zum Beispiel auf ein Objekt hin ausrichtet, welches im Blickpunkt der Kamera bleiben soll.

In einer fortgeschritteneren Variante der Kamerabewegung entlang eines Pfades kommen komplexere Pfade in Betracht. Dies ermöglicht unter anderem die animierte Darstellung einer virtuellen Endoskopie oder Bewegungen innerhalb von geschlossenen, röhrenförmigen Objekten wie Gefäßen oder dem Darmtrakt [Hausig 1998] (siehe Abbildung 2.5). Bei einer virtuellen Endoskopie wird die Kamerasicht dargestellt, wie sie sich einer realen Kamera am Ende eines Endoskopes bieten würde. Diese Darstellung erfordert eine Mindestgröße des Durchmessers dieser Strukturen, um einen realistischen Eindruck zu erhalten.

Eine Veränderung der Position und Blickrichtung der Kamera kann auch dazu genutzt werden, den Betrachter der Animation an eine bestimmte Sicht auf die Szene heranzuführen. Im Unterschied zu abrupten Wechseln oder Schnitten bleiben bei einer kontinuierlichen Veränderung der Kameraparameter die Kontextinformationen und die Orientierung des Betrachters in der Szene erhalten. Die Ausrichtung der Kamera zeigt dem Betrachter, welches Objekt im Moment von Bedeutung ist, indem es in den Fokus[7] der Kamera gesetzt wird. Um die Bedeutung eines Objektes im Verlauf einer Animation zu betonen ist weiterhin die Verweildauer des Objektes im Fokus der Kamera von Bedeutung. Je wichtiger ein Objekt ist oder je mehr es betont werden soll, desto länger ist die Kamera auf dieses Objekt ausgerichtet.

2.7.2 Hervorhebungen von Objekten

Ein weites Feld für medizinische Animationen mit vielen Möglichkeiten bietet die Erklärung von komplexen Strukturen und Zusammenhängen. Dabei kommt es darauf an, bestimmte Objekte oder Strukturen gegenüber anderen hervorzuheben oder zurücktreten zu lassen (vgl. [Preim und Ritter 2002] und [Dörge 2002]).

[6] Die Eigentschaft des Zooms beschreibt bei einer Kamera die Möglichkeit, Dinge optisch zu vergrößern bzw. zu verkleinern.

[7] Der Fokus der Kamera beschreibt in diesem Zusammenhang weniger der Brennpunkt der Kamera als vielmehr das Zentrum des Blickfeldes.

Das Hervorheben eines oder mehrerer Objekte zieht vielfältige Änderungen nach sich. Es sind dabei zum Einen Entscheidungen über die unterschiedlichen Objektparameter zu treffen. Zum Anderen spielen auch hier Kamerabewegung und -blickrichtung eine wesentliche Rolle. Als dritter Aspekt kommt die Frage der Sichtbarkeit der hervorzuhebenden Objekte hinzu.

In [Preim und Ritter 2002] werden drei Gruppen von Hervorhebungstechniken eingeführt:

- lokale Hervorhebungstechniken
- regionale Hervorhebungstechniken
- globale Hervorhebungstechniken

Lokale Hervorhebungstechniken verändern nur die Eigenschaften des hervorzuhebenden Objektes, während regionale Hervorhebungstechniken auch Objekte in der Umgebung des Objektes von Interesse einbeziehen. Globale Hervorhebungstechniken rufen Veränderungen an der gesamten Szene hervor.

Als Eigenschaften eines Objektes, die für eine Visualisierung im Zusammenhang mit einer lokalen Hervorhebung stehen, kommen zahlreiche Parameter in Frage. So kann die Farbe eines Objektes dieses optisch hervortreten lassen (zum Beispiel *rot*) oder seine Bedeutung für den Betrachter verringern (z.B. unauffälligere Farben im Helligkeitsbereich des Hintergrundes) (siehe Abbildung 2.6(a)). Ähnliches gilt für die Transparenz eines Objektes. Ein opakes Objekt tritt gegenüber einem transparenten in seiner Bedeutung hervor (siehe Abbildung 2.6(b)).

Eine weitere Möglichkeit ein Objekt hervorzuheben findet sich vor allem in interaktiven Darstellungen. Dabei wird um das Objekt eine *bounding box*[8] gezeichnet.

Gerade bei der Änderung der Objektparameter zeigt sich der Vorteil der Animation gegenüber statischen Darstellungen. Durch die kontinuierliche Veränderung der Objektparameter über die Zeit wird dem Betrachter diese Veränderung und damit das visuelle Ziel besser vergegenwärtigt.

Bei der Veränderung von Objekteigenschaften, besonders der Farbe, ist seitens des Autors einer Animation der Kontext der Eigenschaften der anderen Objekte zu beachten. So kann die Verwendung einer Farbe zu Hervorhebungszwecken den Betrachter irritieren oder ihre Wirkung einbüßen, wenn andere Objekte in der gleichen oder einer ähnlichen Farbe dargestellt werden.

Betrachtet man regionale Hervorhebungstechniken, so geht es in den meisten Fällen um die Sichtbarkeit eines Objektes. Ist das Objekt von Interesse ganz oder teilweise durch andere Objekte verdeckt, muss entschieden werden, ob diese Objekte für die Erklärung des Sachverhaltes oder als dessen Kontext entscheidend sind. Ist dies nicht der Fall, können sie ausgeblendet werden und so die Sicht auf das Objekt

[8] Eine *bounding box* ist eine quaderförmige Umschließung eines Objektes. Ist die *bounding box* an den Koordinatenachsen ausgerichtet, so spricht man von einer *axis-aligned bounding box*. Wird der Quader an die Ausrichtung des Objektes angepasst, spricht man von einer *oriented bounding box* (vgl. [Preim und Ritter 2002]).

des Interesses freigeben. Ist ihre Sichtbarkeit allerdings für das Verständnis oder die Orientierung in der Szene unverzichtbar, so muss eine Lösung gefunden werden, die sowohl das Objekt weiterhin sichtbar lässt, aber auch die Sicht auf das Objekt des Interesses ermöglicht.

Dafür gibt es verschiedene Ansätze. Das verdeckende Objekt von geringerem Interesse kann teilweise transparent dargestellt werden, sodass das dahinterliegende Objekt noch sichtbar ist. Ein ähnlicher Effekt kann durch einen veränderten Darstellungsstil der Objekte erreicht werden. So kann das verdeckende Objekt als Liniengitter oder Punktraster dargestellt werden, durch welches man dahinterliegende Objekte dennoch erkennt (siehe Abbildung 2.7).

Wird zu Zwecken der Hervorhebung eines Objektes die Kamera bewegt oder anders ausgerichtet, so spricht man von einer globalen Hervorhebungstechnik. Oft muss die Kamera in eine andere Position verschoben werden, damit sie einen besseren Blick auf das Objekt von Interesse erlaubt. Wird ein Objekt nicht von anderen Objekten verdeckt, kann immer noch die Möglichkeit bestehen, dass es zu weit entfernt von der Kamera und damit zu klein ist. In diesen Fällen muss die Kamera näher heranfahren, was ebenfalls in die Gruppe globaler Hervorhebungstechniken fällt.

Eine weitere Technik für die Erklärung komplexer Strukturen ist der kontinuierliche Aufbau von Objekten. Dieses Verfahren bietet sich besonders bei Gefäßen an. Die oft verzweigten, baumartigen Strukturen können so besser visualisiert werden. Dabei wächst der Gefäßbaum von einem gedachten Ursprung langsam und baut so die verzweigte Struktur auf. Der Effekt ist vergleichbar mit dem Fluten eines Kanal- oder Rohrsystems.

2.7.3 2D-Darstellungen

Ein häufig auftretender Fall in der radiologischen Diagnostik bzw. in der Therapieplanung ist das Betrachten von 2D-Schichtbildern. Eine verbreitete Art Schichtbilder am Computer zu betrachten ist das sogenannte *Slicen* oder Durchblättern. Dabei wird sich mithilfe von Maus oder Tastatur durch die Schichtbilder bewegt. Dies geschieht im Normalfall axial, wobei immer das Schichtbild der aktuell gewählten Schicht angezeigt wird. Das *Slicen* durch 2D-Schichtbilder lässt sich ebenfalls als Animation darstellen. Es sind hierbei Hervorhebungen segmentierter Strukturen durch Farbe und Transparenz möglich (siehe Abbildung 2.8).

2.7.4 Clip-Ebenen

Mit Hilfe von Clip[9]-Ebenen können beliebige Bereiche einer 3D-Ansicht flächig weggeschnitten werden. Es werden nur die Teile der Objekte angezeigt, die auf einer der zwei Seiten der Clip-Ebene liegen. Man kann dabei Objekte vom Vorgang des Clippens ausschließen. Diese Objekte werden trotz Clip-Ebene komplett dargestellt.

[9] engl.: *to clip* - abschneiden

So können Strukturen durch das Clippen von der Umgebung freigestellt werden, indem zum Beispiel umliegendes Gewebe geclippt wird, das Objekt von Interesse aber bestehen bleibt. Dieser Vorgang wird auch als selektives Clippen bezeichnet. Ein solches Clippen bietet den Vorteil, dass ein Objekt von Interesse sichtbar gemacht wird, ohne die Kontextinformation umliegender Organe komplett entfernen zu müssen (siehe Abbildung 2.9).

Aus medizinischer Sicht gibt es standardisierte Richtungen und Lagen der Clip-Ebenen, die sich durchgesetzt haben. Dazu gehört das Clippen parallel zu den Körperachsen. Allerdings gibt es technisch keine Beschränkung der Cliprichtungen. So können auch Ebenen in beliebiger Richtung zum Clippen genutzt werden.

Eine Erweiterung des einfachen Clippens ist das Clippen mithilfe mehrerer Ebenen in einer Darstellung. Dabei werden Teile von Objekten durch unterschiedliche Clip-Ebenen entfernt (siehe Abbildung 2.11(a)).

In einer animierten Darstellung werden die Clip-Ebenen kontinuierlich durch die Szene bewegt, um so Strukturen von Interesse freizulegen.

2.7.5 Kombination von 2D- und 3D-Darstellungen

Interessant ist eine Darstellung, bei der zweidimensionale Schichtbilder mit dreidimensionalen Darstellungen kombiniert werden. In einem einfachen Fall würde die 3D-Darstellung nur als Positionsanzeiger für die momentane Schicht in der 2D-Darstellung dienen. Diese Form wäre eine Weiterentwicklung der sogenannten *Fahrstuhlansicht*. Dabei würde eine Zweiteilung der Ansicht vorgenommen werden, bei der die 3D- und 2D-Darstellung nebeneinander stünden (siehe Abbildung 2.10).

In einer fortgeschritteneren Kombination beider Darstellungen wird das Schichtbild direkt in der 3D-Ansicht visualisiert. Dabei kommen die oben vorgestellten Clip-Ebenen zum Einsatz. Bei der Kombination von Schichtbildern und 3D-Ansicht wird das jeweilige Schichtbild direkt auf die Clip-Ebene projeziert und angezeigt. Verläuft eine Clip-Ebene nicht parallel zur Ebene der Schichtbilder, so kann ein virtuelles Schichtbild für die entsprechende Lage der Ebene berechnet und angezeigt werden (siehe Abbildung 2.11(b)).

2.7.6 Bewegung von Objekten

Bei technischen Darstellungen und Animationen kommt das Mittel der Explosionsdarstellung zum Einsatz. Ziel dieses Gestaltungsmittels ist es, Objekte sichtbar zu machen, die von anderen Objekten verdeckt oder umschlossen werden. Dazu werden benachbarte Objekte entlang von Achsenparallelen oder Funktionsachsen (z.B. einem Schraubengewinde) auseinander bewegt [Haase 2005]. Die Anwendung von Explosionsdarstellungen in medizinischen Animationen dagegen ist problematisch. Informationen über Lagebeziehungen der Organe und Strukturen zueinander würden verloren gehen. Auch ist es in der Anatomie nur schwer möglich, funktionale Achsen von Organen zu bestimmen.

Ein Bereich, in dem sich die Verschiebung von Objekten auch in medizinischen Animationen anbietet, ist die anschauliche Darstellung von Schnittflächen [Konrad-Verse u. a. 2004]. So ist es für einen Chirurgen bei der Resektion einer Leber nicht nur von Bedeutung, welchen Verlauf ein Schnitt auf der Oberfläche des Organs hat, sondern auch, wie die entstehende Schnittfläche zwischen dem Resektat und dem Organ beschaffen ist. Da ein Resektat aus dem Körper entfernt wird, kann der angesprochene Informationsverlust bei einer Verschiebung der beiden Objekte gegeneinander in Kauf genommen werden. Die Schnittfläche könnte, so wie in Abbildung 2.12 zu sehen, sichtbar gemacht werden.

2.7.7 Zeigegeräte

Eine Möglichkeit auf Objekte oder Sachverhalte besonders hinzuweisen ist der Einsatz von zusätzlichen Objekten als Zeigegeräte (vgl. [Dörge 2002] und [Steinicke 1996]). Zeigegeräte können in einer medizinischen Animation beispielsweise Pinzetten, Stifte oder Pfeile sein. Das Zeigegerät verweist dabei auf das Objekt, welches betont werden soll. In Animationen kann es zusätzlich genutzt werden, um Konturen eines Objektes oder Verläufe von Gefäßstrukturen abzufahren. Auch können mehrere Zeigegeräte gleichzeitig zum Einsatz kommen, zum Beispiel um den Anfangs- oder Endpunkt einer Struktur kenntlich zu machen.

2.7.8 Zusammenfassung der Animationstechniken

In den vorangegangenen Abschnitten wurden verschiedene Animationstechniken vorgestellt. Diese Aufstellung erhebt keinen Anspruch auf Vollständigkeit. Welche Animationstechniken in einem System zur Generierung von Animationen zum Einsatz kommen, hängt stark von den technischen Gegenbenheiten des Systems ab. Meist sind Animationstechniken auf diejenigen Techniken beschränkt, die eine Visualisierungssoftware für die Interaktion mit dem Nutzer bereithält.

Während die Animationstechniken im Zusammenhang mit Clip-Ebenen und die kombinierte 2D-3D-Darstellung auf den Bereich medizinischer Animationen beschränkt sind, ist eine Nutzung der vorgestellten Hervorhebungstechniken und der Einsatz von Zeigegeräten auch in Animationen anderer Bereiche denkbar.

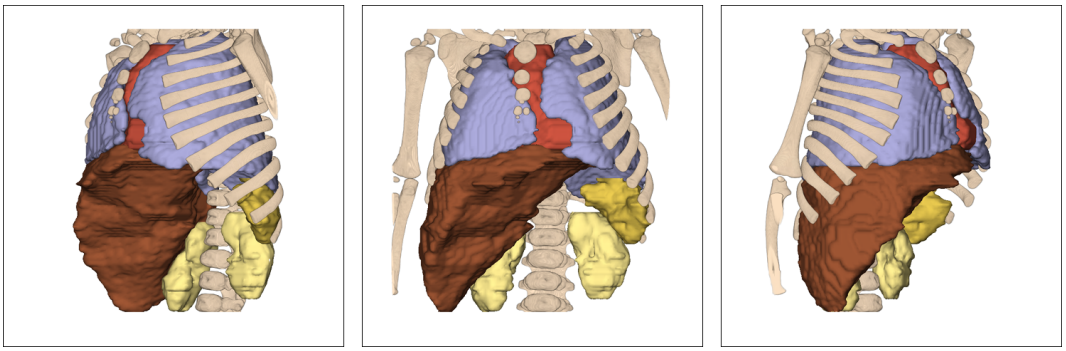


Abbildung 2.3: Beispiel einer Rotation der Kamera um die Objekte in einer Szene

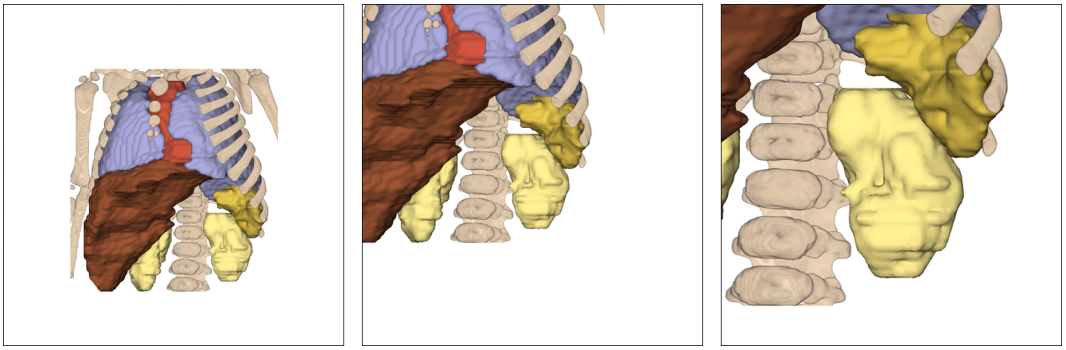


Abbildung 2.4: Zoom der Kamera aus einer Übersichtsansicht auf die linke Niere

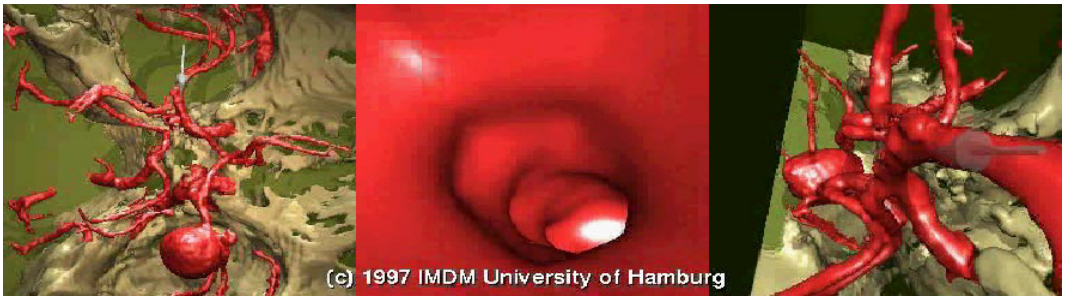
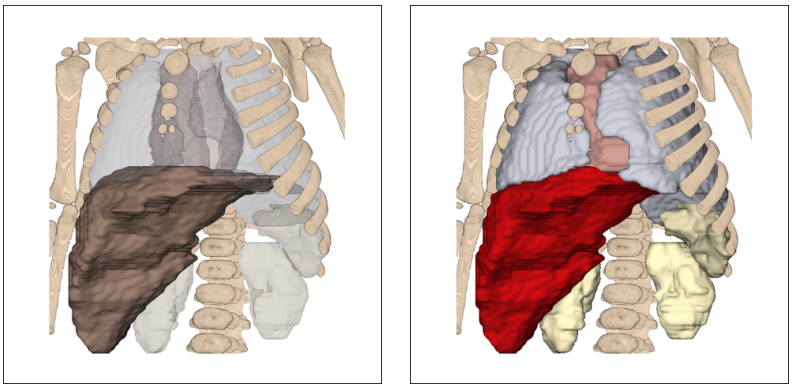
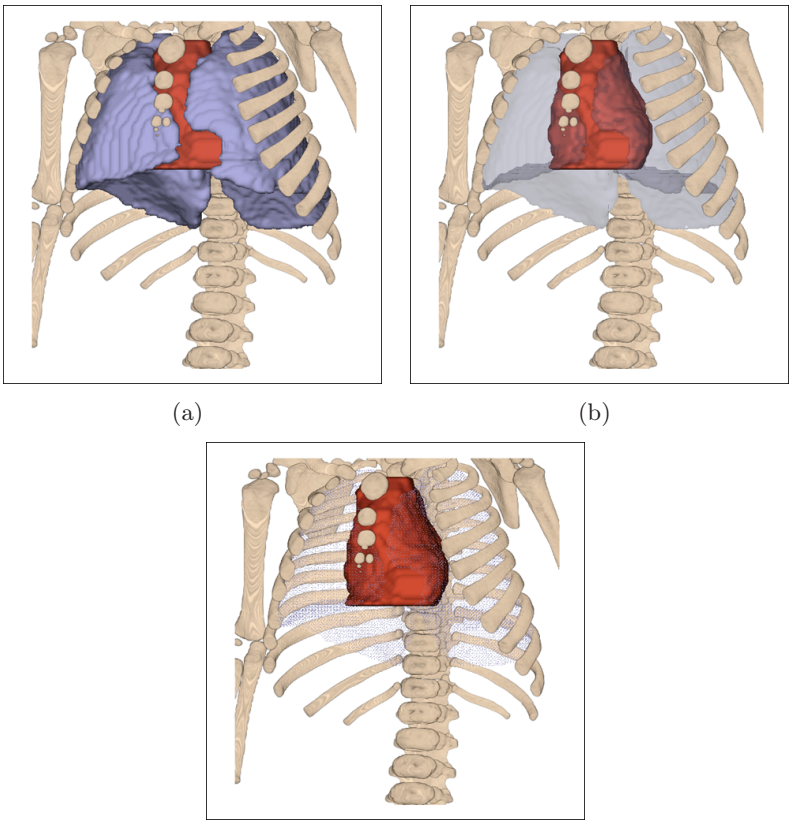


Abbildung 2.5: Sicht einer Kamera innerhalb eines Gefäßes (mittleres Bild) (aus [Hausig 1998])



(a) (b)

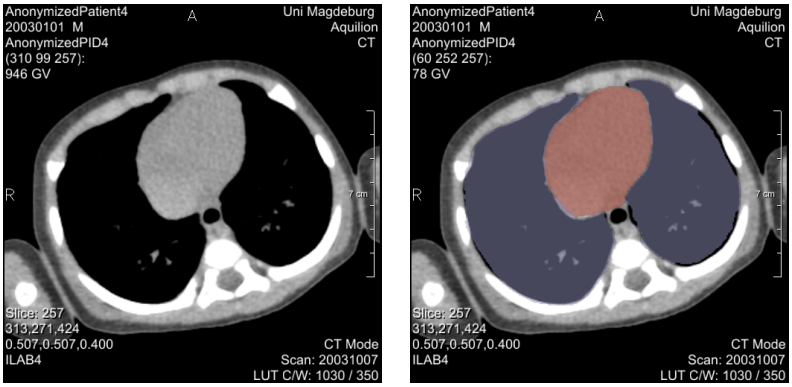
Abbildung 2.6: Hervorhebung eines Objektes durch Opazität (a) und mittels Farbgebung (b)



(a) (b)

(c)

Abbildung 2.7: Die Verdeckung eines Objektes durch ein anderes (a), die Lösung des Problems durch eine teilweise Transparenz (b) und die Darstellung des verdeckenden Objektes im Punktraster (c)



(a) (b)

Abbildung 2.8: Schichtbilder in der normalen Ansicht (a) und einer Darstellung, in der segmentierte Organe eingefärbt wurden (b)

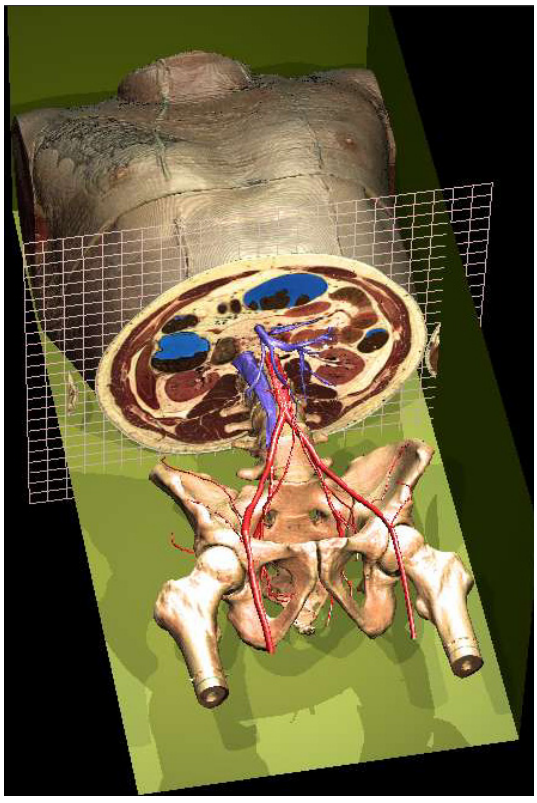


Abbildung 2.9: Selektives Clipping in einer Darstellung des VoxelMan (aus [Höhne u. a. 2003])

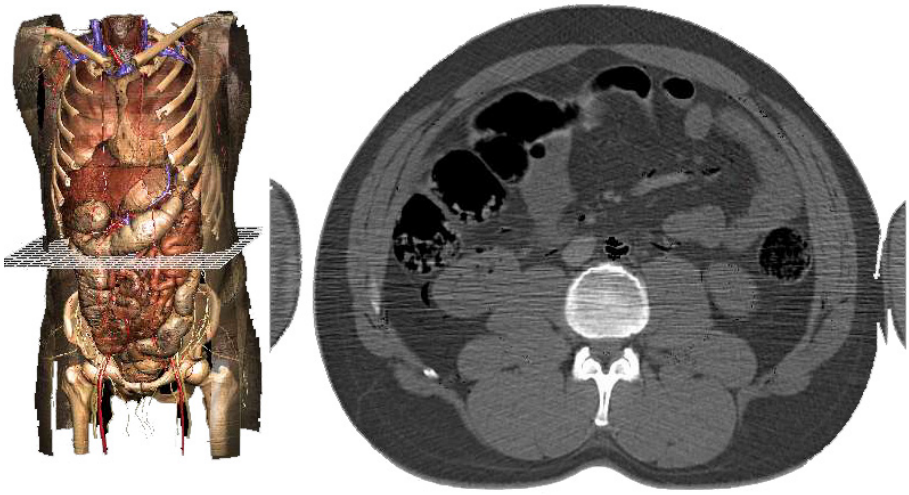
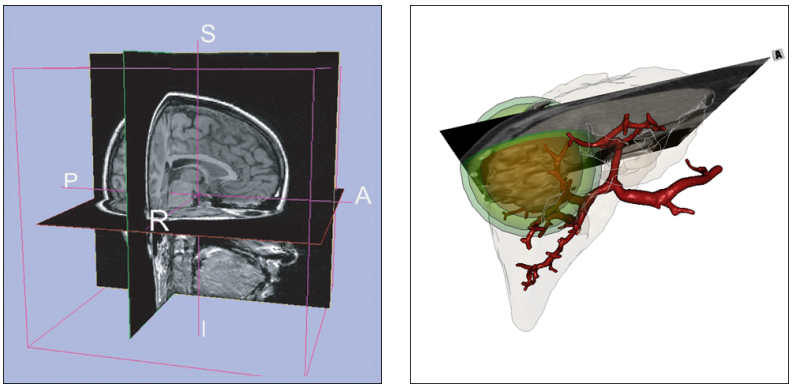


Abbildung 2.10: Darstellung eines Schichtbildes mit der gleichzeitigen Anzeige der Schichtposition und Lage in der dreidimensionalen Szene (aus [Höhne u. a. 2003]).



(a) (b)

Abbildung 2.11: Visualisierung von Schichtbildern direkt in einer dreidimensionalen Ansicht. Im linken Bild (a) kommen drei Ebenen gleichzeitig zum Einsatz (aus [Gering 1999]). Das rechte Bild (b) zeigt die Projektion eines Schichtbildes auf eine Clip-Ebene.

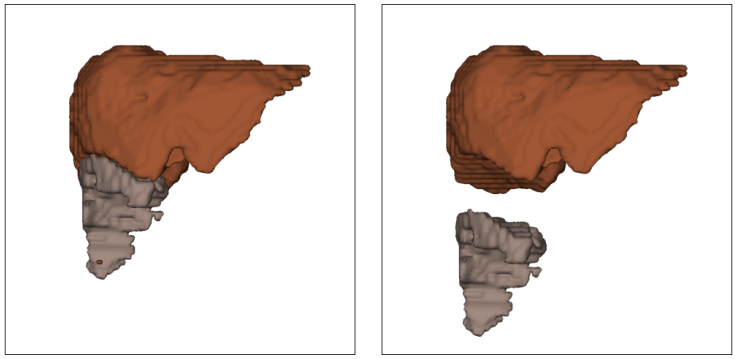


Abbildung 2.12: Durch das Verschieben eines Resektats wird die Schnittfläche einer Resektion sichtbar gemacht.

2.8 Datengrundlage der Animationen

Im Rahmen dieser Arbeit wurden statische Daten aus Computer-Tomographie- und Magnet-Resonanz-Tomographie-Messungen (CT und MRT) verwendet. Die Datensätze wurden segmentiert, sodass die einzelnen Organe und interessante Regionen wie Metastasen als Objekte vorliegen. Dazu kam unter anderem das Tool HepaVision zum Einsatz [Bourquain u. a. 2002]. HepaVision ist speziell zur Segmentierung der Leber und umliegender Strukturen wie der Gefäße am Centrum für Medizinische Diagnosesysteme und Visualisierung in Bremen (MeVis) entwickelt worden.

Bei der Segmentierung werden teils automatisch, teils von Hand durch einen Experten Organe oder Regionen identifiziert. Die Segmentierung wird bei einer späteren Visualisierung in der Oberflächendarstellung wiedergegeben und erst dort als solche sicht- und fassbar. Bei der Oberflächendarstellung werden die Oberflächen segmentierter Objekte mithilfe von Triangulation aus den Volumendaten ermittelt (vgl. [Hinz u. a. 2001]).

Im Zuge der Visualisierung können jedem Objekt Eigenschaften wie Farbe, Transparenz oder seine Position im Raum zugeordnet werden. Bei der Visualisierung der Daten werden alle Objekte in einer Szene zusammengefasst und in dreidimensionaler Form dargestellt. Dabei kommt neben der Oberflächendarstellung auch die Volumenvisualisierung zum Einsatz. Einen Überblick über die Volumenvisualisierng aus Sicht der Medizin bietet [Preim u. a. 2000]. Eine Szene enthält weiterhin eine Kamera, die die Projektion der dreidimensionalen Szene auf eine zweidimensionale Fläche übernimmt (siehe Abbildung 2.13).

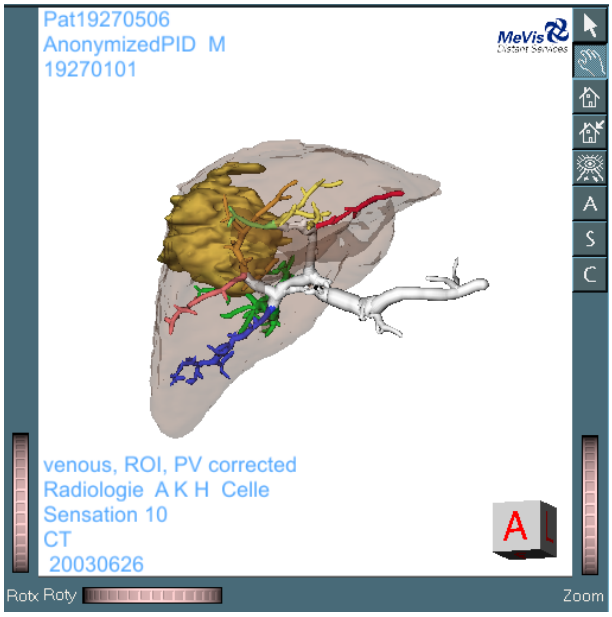


Abbildung 2.13: Darstellung einer 3D-Szene mit der Oberflächenvisualisierung einer Leber mit verschiedenen Objekten wie Gefäßen und einem Tumor

2.9 Anforderungen an ein Animationssystem

Anhand der bisher dargestellten Erkenntnisse im Zusammenhang mit Systemen der Therapie- und Operationsplanung sowie der Datengrundlage werden im Folgenden eine Reihe von Anforderungen an ein skriptbasiertes Animationssystem formuliert.

Mit Blick auf eine spätere Nutzung des Systems sind zwei Benutzergruppen auszumachen: professionelle und unprofessionelle Anwender. Die Begriffe professionell und unprofessionell werden hier im Sinne eines Programmierverständnisses und der Erfahrung im Umgang mit Skriptsprachen verstanden. Der unprofessionelle Anwender wird exemplarisch durch einen Arzt repräsentiert, der das System als Teil eines Gesamtsystems zur Operations- und Therapieplanung nutzt. Schon heute kommen solche Systeme in Form des InterventionPlanners [Preim u. a. 2003] zum Einsatz. Ziel ist es dabei, an aktuellen Daten einen patientenindividuellen Eindruck der relevanten Bereiche für eine Operation oder Therapie zu bekommen. Mithilfe von zusätzlichen Vermessungsdaten von Volumina und Abständen kann außerdem eine Risikoabschätzung vorgenommen werden. Teil einer solche Planung können standardisierte Animationen sein, die auf patientenindividuellen Daten basieren. Eine Präsentation der Daten zur Planung und Beratung wird zur Zeit zwar auch an 3D-Visualisierungen vorgenommen, jedoch bedarf es eines hohen Grades an Interaktivität mit dem 3D-Modell. Gerade bei immer wiederkehrenden standardisierten Sichten und Darstellungen kann dies hinderlich sein. Dafür und für die Darstellung eines Überblicks über die Daten sind vorgefertigte Animationen besser geeignet.

Aufgrund der immer noch knapp bemessenen Zeit zur Einzelanalyse und -planung sind hier abspielbare Animationen von Vorteil. Aus dieser Zeitbegrenzung resultiert auch die Notwendigkeit einer möglichst schnellen Generierung der Animationen durch einen Autor. Dabei spielen eine kurze Einabreitungszeit und ein intuitives Verständnis eine wichtige Rolle. Des Weiteren sollen Animationen schnell reproduzierbar und ohne großen Anpassungsaufwand auf eine Vielzahl von patientenindividuellen Daten anwendbar sein.

Als exemplarisch für einen Experten kann der Autor eines Lernsystems wie dem LiverSurgeryTrainer (siehe Abschnitt 3.7) angenommen werden. Hier geht es um die Einarbeitung neuer Fälle in Form von Animationen in das System. Solch ein Autor ist professioneller als ein Arzt anzusehen, da er zum Einen ein tieferes technisches Verständnis für das System besitzt. Auch steht ihm mehr Zeit für die Einarbeitung und die Auseinandersetzung mit der Materie zur Verfügung. Zum Anderen stellt er größere Anforderungen an das System. Dies betrifft vor allem die Individualisierung, die sich in spezialisierteren Funktionen und Manipulationsmöglichkeiten ausdrückt.

Unabhängig von der Nutzergruppe lassen sich weitere Anforderungen an ein Animationssystem ermitteln. Mit Hilfe des Systems soll es möglich sein, die Eigenschaften der Objekte, der Szene und der Kamera einer 3D-Visualisierung medizinischer Daten, wie sie in Abschnitt 2.8 beschrieben wurden, in der Art zu manipulieren, dass dabei automatisch Animationen generiert werden. Als Grundlage dafür soll ein Skript dienen, welches vom Autor verfasst wird.

Zusammenfassend lassen sich zwei Kernanforderungen an ein skriptbasiertes Animationssystem für die medizinische Ausbildung und Therapieplanung formulieren:

1. **Flexibilität**

   Mit der Skriptsprache soll eine Manipulation der verschiedenen Objektparameter wie auch der Parameter der Szene (z.B. Kameraposition und -blickrichtung) möglich sein.

2. **Geeignete Benutzungsschnittstelle**

   Die Manipulationen sollen für den professionellen Autor detailliert und speziell und für den unprofessionellen Autor einfach und allgemein erfolgen können.

Kapitel 3

Analyse verwandter Arbeiten

In diesem Kapitel werden bisherige Arbeiten und Lösungsansätze vorgestellt, die für die vorliegende Arbeit relevant sind. Die ersten Abschnitte konzentrieren sich dabei auf Arbeiten zur Animationsgenerierung. Anschließend wird genauer auf zwei Softwarelösungen, AMIRA und MEVISLAB, eingegangen. Der vorletzte Abschnitt geht kurz auf medizinische Lernsysteme, insbesondere den LIVERSURGERYTRAINER ein. Der letzte Abschnitt fasst die vorgestellten Ansätze zusammen und extrahiert für diese Arbeit wesentlich Aspekte.

3.1 Grundlegende Arbeiten aus dem Bereich der Animationsgenerierung

Zur Problematik der Animationsgenerierung wurden in den letzten 15 Jahren vermehrt Arbeiten veröffentlicht. Dies hängt mit der technischen Entwicklung hin zu leistungsfähigeren Computern auch in der Grafikerstellung und damit der Animationsgenerierung zusammen. Besonderes Augenmerk wurde in vielen Arbeiten dabei auf die automatische Animationsgenerierung gelegt.

So stellten [André u. a. 1996] einen Ansatz vor, der die Erstellung von Präsentationen von Informationen ermöglichte. Das WIP-Konzept (Wissensbasierte-Informations-Präsentation) geht dabei von Präsentationszielen (auch als Kommunikationsziele bezeichnet) aus, die mit Hilfe einer Wissensbasis in Präsentationsanweisungen überführt werden. Die Überführung erfolgt mithilfe von Dekompositionsregeln[1]. Die Ziele werden hierarchisch in einer Baumstruktur aufgeschlüsselt. Am Ende des Baumes, in dessen Blättern, befinden sich elementare Anweisungen, während die Wurzel des Baumes ein Präsentationsziel darstellt. Eingeführt wurde dieser Ansatz an technischen Darstellungen, die in Form geometrischer dreidimensionaler Modelle vorlagen.

Auf dem WIP-Konzept baut das BETTY-System [Butz 1994] auf. Dieses setzt auf die Animation einzelner Ziele (z.B. durch Kamerabewegungen), um so einen besseren räumlichen Eindruck der Darstellung zu vermitteln. Aus den Kommunikationszielen und den Umgebungsparametern wird ein Skript generiert, wobei mithilfe der

[1] Dekompositionsregeln beschreiben die Vorgehensweise beim Aufschlüsseln von Kommunikationszielen.

Wissensbasis Entscheidungen über Kameraposition, Zooms, Bewegungen und Schnitte automatisch getroffen werden. Besonderes Augenmerk wird bei diesem Prozess auf die Bewegungen und Zooms der Kamera gelegt.

Bei der Umwandlung der Kommunikationsziele in elementare Animationsanweisungen kam bei Betty ein Modell verschiedener Abstraktionslevel zum Einsatz, welches schon in [Zeltzer 1991] vorgestellt wurde. Dabei wird die Überführung der Kommunikationsziele in direkte Anweisungen in vier Level eingeteilt (siehe Abbildung 3.1):

- **Task-Level**
  Der Task-Level dient der Beschreibung der Kommunikationsziele.
- **Functional-Level**
  Der Functional-Level stellt die Ziele in Techniken des klassischen Filmherstellungsprozesses dar.
- **Procedural-Level**
  Im Procedural-Level werden die Funktionen des vorherigen Levels in Grundoperationen wie Bewegung, Zoom oder Schnitte überführt.
- **Machine-Level**
  Dieser letzte Level enthält die Grundoperationen als explizite Beschreibungen in der endgültigen Geometrie.

Abbildung 3.1: Vier Level bei der Überführung von Kommunikationszielen in elementare Animationsanweisungen nach [Zeltzer 1991]

Schon 1993 stellten [Karp und Feiner 1993] mit Esplanade (*Expert System for Planning Animation Design and Editing*) einen Ansatz zur wissensbasierten Animationsplanung vor. Als Eingabe dienen auch hier Kommunikationsziele. Hinzu kommen kausale Beziehungen zwischen den möglichen Aktionen und Objekten (z.B. *'Drücken von Türklinke A öffnet Tür B'*) sowie Informationen, welche Aktionen mit welchen Objekten ausgeführt werden können. Eine Animation wird nach [Karp und Feiner 1993] hierarchisch in Sequenzen, Szenen und Aufnahmen zerlegt (siehe Abbildung 3.2). Ein Skript, welches zur Eingabe der Kommunikationsziele und Aktionen dient, wird dahingehend analysiert, welche Aktionen sich in einer Aufnahme und welche Aufnahmen sich in einer Szene zusammenfassen lassen.

Den Systemen Betty und Esplanade ist gemein, dass sie auf einer hierarchischen Aufschlüsselung von abstrakten Kommunikationszielen hin zu elementaren Animationsanweisungen aufbauen. Der Unterschied beider Systeme liegt in der Aufschlüsselung der Kommunikationsziele mittels Dekompositionsregeln. Beim Esplanade-System werden alle Elemente einer Ebene der Hierarchie ermittelt und zunächst komplett berechnet, bevor die Umwandlungen in der nächsten Ebene erfolgen (*breadth-first*). Dadurch können gegenseitige Beeinflussungen der Ziele untereinander berücksichtigt werden. Bei Betty wird dagegen zunächst versucht, ein Ziel komplett mit

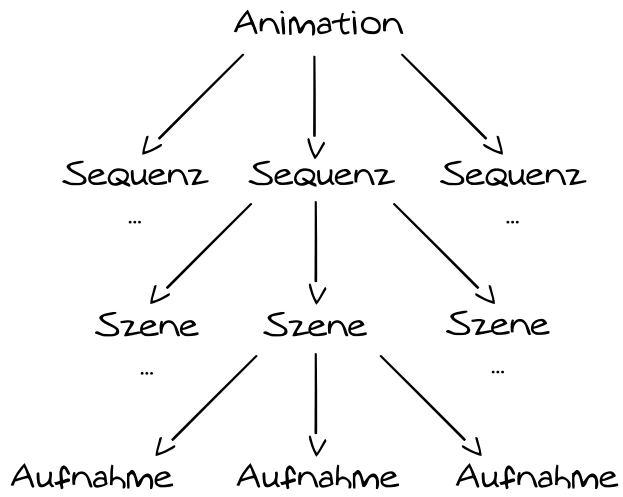


Abbildung 3.2: Aufteilung einer Animation nach [Karp und Feiner 1993] in Sequenzen, Szenen und Aufnahmen

möglichst wenigen Anweisungen für Änderungen aufzuschlüsseln (*depth-first*). Damit sind bei den folgenden Aufnahmen die Details der davor gelegenen Aufnahmen bekannt und können entsprechend berücksichtigt werden.

Nachteile beider Systeme sind zum Einen die fehlende Flexibilität bezüglich kleiner Änderungswünsche für eine Animation. Die Animation lässt sich ausschließlich über Kommunikationsziele steuern und nur die Änderung von ganzen Zielen bzw. die Kreation neuer Kommunikationsziele kann eine Veränderung an der Animation bewirken. Zum Anderen ist die Definition von Kommunikationszielen in den Skripten äußerst komplex und daher für unprofessionelle Autoren nicht handhabbar.

3.2 Die Animationskomponente des ZoomIllustrators

Ein Animationssystem, welches ebenfalls auf der Idee von [Zeltzer 1991], der levelorientierten Zerlegung von Kommunkationszielen bzw. Anweisungen basiert, wurde am Institut für Simulation und Graphik der Otto-von-Guericke-Universität Magdeburg entwickelt (vgl. [Steinicke 1996], [Preim u. a. 1996] und [Preim 1997]). Das Animationssystem ist Teil des Hypermediasystems ZoomIllustrator, welches gleichfalls in Magdeburg entstand [Preim und Strothotte 1996].

Der ZoomIllustrator verbindet eine interaktive, dreidimensionale Darstellung mit textuellen Annotationen. Der Nutzer des Systems hat die Möglichkeit, die dreidimensionale Darstellung mehrerer Objekte (zum Beispiel der Bestandteile eines Fußes wie in Abbildung 3.3) interaktiv zu betrachten. Dabei werden die Bezeichnungen der Objekte, die über Linien mit diesen verbunden sind und so auf sie verweisen, automatisch angepasst. Als Vorbild dienten den Entwicklern anatomische Zeichnungen wie in Abbildung 1.3. Zielgruppe des Systems sind Medizinstudenten,

für die die interaktiven Darstellungen anatomischer Objekte eine Erweiterung des vorherrschenden Lehrmaterials in Form von Anatomieatlanten sind.

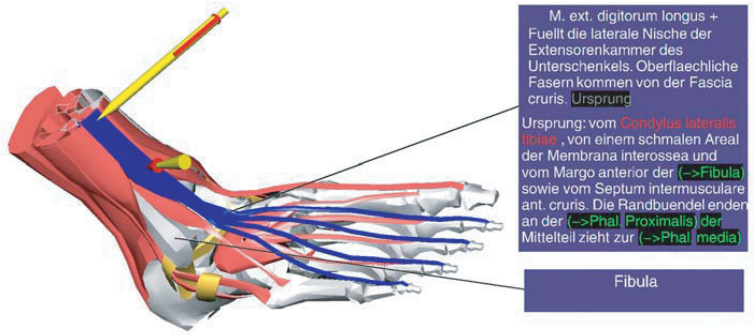


Abbildung 3.3: Ausschnitt aus einer Darstellung eines Fußes im ZoomIllustrator. Einzelne Objekte werden durch Textannotationen hervorgehoben und ein Zeigegerät in Form eines Stiftes kommt zum Abfahren eines längeren Muskels zum Einsatz.

Aufgabe des Animationssystems als Teil des ZoomIllustrators ist es, innerhalb dieser Darstellungen verschiedene Vorgänge zu animieren. Das Kernziel der Arbeit von [Steinicke 1996] ist nicht die Erstellung fertiger Animationen. Vielmehr sollen mit einer eigens entwickelten Skriptsprache nur kurze Abschnitte der interaktiven Darstellung animiert werden. Der Nutzer kann dabei interaktiv in die Animation eingreifen, indem er diese zum Beispiel anhält und die Kamera selbstständig weiter bewegt. Eine Möglichkeit für einen animierten Vorgang kann zum Beispiel die Hervorhebung einzelner Objekte sein. Wählt der Nutzer ein Objekt aus, so wird die Kamera in eine optimale Sichtrichtung zu diesem bewegt und eventuell verdeckende Objekte werden ausgeblendet. Die optimale Sichtrichtung inklusive der Sichtbarkeitsinformationen werden zuvor anhand des dreidimensionalen Modells einmalig ermittelt und gespeichert.

Das Animationssystem ist zusammen mit dem ZoomIllustrator darauf ausgelegt, einen festen Satz von Gitternetzmodellen mit exakt darauf abgestimmten Animationsskripten zur Verfügung zu haben. Die Gitternetzmodelle sind dabei idealtypische Darstellungen anatomischer Objekte.

Die Verhaltensweisen des Systems bei der Ausführung von Anweisungen wie *"Erkläre Objekt X"* oder *"Betone Objekt Y"* werden in Skripten definiert. Skripte können neben der rein textuellen Eingabe auch interaktiv generiert werden (*Teach-In* Modus). Dabei werden Aktionen, die der Nutzer in der Szene vornimmt, als Skriptanweisungen ausgegeben und können durch Kopieren in ein Skript integriert werden. Dies ist ob der nicht intuitiven Skriptsprache (siehe Listing 3.1) ein Vorteil, wird aber auch schnell zum Nachteil, wenn es darum geht, zum Beispiel Kamerapfade mit mehreren Stützpunkten anzugeben[2]. Ein weiterer Nachteil der interaktiven Skripterzeugung ist die Entstehung von Ungenauigkeiten, wie sie bei der Eingabe von Kamerapositionen

[2]Der Autor spricht in diesem Zusammenhang von 1-2 Stunden Zeitaufwand zur Festlegung eines Pfades von 10 Stützpunkten [Steinicke 1996].

und -blickrichtungen direkt in der Szenenansicht mittels einer Maus unvermeidlich sind. Ein weiteres Problem stellt die fehlende Möglichkeit dar, Anweisungen zeitlich parallel oder überlappend auszuführen. Anweisungen können nur strikt nacheinander ausgeführt werden.

Die Anweisungen werden in verschiedene Basisanweisungen zerlegt und sofort ausgeführt, das heißt animiert. Der gewählte Ansatz der Zerlegung von abstrakten Anweisungen in Basisfunktionen erscheint vielversprechend und ähnelt dem in dieser Arbeit entworfenen Skriptkonzept. Allerdings ist es bei der von [Steinicke 1996] entwickelten Skriptsprache nicht möglich, Parameter von einer abstrakten Anweisung an Basisfunktionen zu übergeben.

Das Skript in Listing 3.1 wurde in der Skriptsprache von [Steinicke 1996] verfasst. Darin wird ein Objekt selektiert und in den darauffolgenden 10 Sekunden transformiert (gedreht und skaliert). Anschließend wird in 4 Sekunden die Transparenz zweier weiterer Objekte verändert, bevor das Objekt mit einer komplexen `explain`-Anweisung erklärt wird. Die `explain`-Anweisung bildet die zentrale Anweisung der Skriptsprache. Der Autor kann mit ihr die Animation einer Erklärung eines Objektes oder einer Objektgruppe initiieren. Er hat dabei die Möglichkeit, die Animationstechnik und den zeitlichen Ablauf zu bestimmen. Besonderen Wert wird in diesem Zusammenhang auf den Einsatz von Zeigegeräten wie virtuellen Stiften oder Pinzetten gelegt, mit denen ein Objekt unterstützend zur Erklärung abgefahren werden kann.

```
Select['group#12'];
SaveIV[];
Transform[1.2,30,20,#,#,10];
ChangeMaterial[{'group0#32', 'group0#33'},#,#,#,#,0.75,4];
Explain['Musculus extensor digitorum longus'];
```

Listing 3.1: Beispielskript in der von [Steinicke 1996] entwickelten Skriptsprache

Die von [Steinicke 1996] entwickelte Skriptsprache ist sehr speziell auf das System des ZoomIllustrators zugeschnitten. Eine Nutzung der Skriptsprache in anderen Bereichen oder eine Überführung auf andere Systeme erscheint daher nicht praktikabel. Die Erstellung von Skripten oder neuen Anweisungen sowie die Einbringung neuer Gitternetzmodelle in das System erscheinen sehr aufwändig, auch wenn sie im Rahmen dieser Arbeit nicht näher in der Praxis untersucht werden konnten. Der Aufwand wiegt aber insoweit weniger schwer, als dass die Schnelligkeit bei der Skripterstellung keine vordergründige Rolle im System spielt und der ZoomIllustrator im Vergleich zu Systemen mit dem Ziel einer eher patientenindividuellen Datengrundlage nur über ein sehr begrenztes Repertoire an Gitternetzmodellen verfügen muss. Auch bei diesem System fällt die Nutzung einer levelbasierten Aufsplittung von Anweisungen als Vorteil bei der Skripterstellung auf.

3.3 Das Animationssystem Alice

Ein weiteres Animationssystem, welches nicht die Erstellung von fertigen Animationen, sondern das interaktive Animieren von Szenen zum Ziel hat, ist das System Alice [Conway 1997]. Alice wurde speziell für Computeranfänger und Kinder konzipiert. Szenen werden durch das Einlesen vorgefertigter Objekt generiert. Die Animation der gesamten Szene, aber auch einzelner Objekte kann dabei über eine grafische Benutzerschnittstelle erfolgen, die wesentlich einfacher angelegt ist als Szenenmanipulationen in Systemen professionellerer Ausrichtung (siehe Abbildung 3.4). Zusätzlich können über eine Kommandozeile Animationsanweisungen gegeben werden, die sofort ausgeführt werden oder mit Maus- und Tastatureingaben verknüpft werden können.

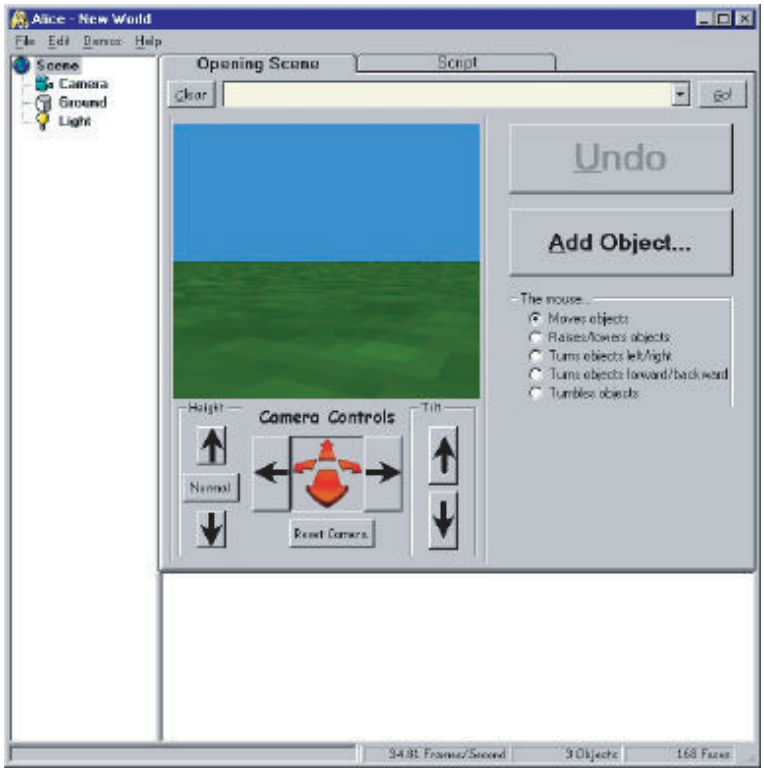


Abbildung 3.4: Grafische Oberfläche zur Manipulation und Animation einer Szene in Alice (aus [Conway 1997])

Jede Anweisung wird mit einer festen Geschwindigkeit ausgeführt, was die Angabe von Ausführungslängen oder -zeitpunkten überflüssig macht. Dies vereinfacht die Skriptsprache für die Zielgruppe der Anfänger und Kinder.

Bei Alice stechen zwei weitere Eigenschaften heraus, die mit Blick auf die vorliegende Arbeit von Interesse sind:

- Alice bietet einen festen Satz von einfachen Befehlen (wie `move`, `turn`, `roll` oder `align`), die ihre Mächtigkeit erst durch die Erweiterung um Parameter erhalten.
- In Alice werden sämtliche Bewegungen oder Ortsangaben koordinatenfrei vorgenommen und relativ zu existierenden Objekten beschrieben (z.B. `left`, `right`, `up`, `forward`, `toptobottom`).

Auch wenn die Skriptsprache speziell für Anfänger und Kinder entwickelt wurde, so stellt auch sie Anforderungen an das grundlegende Verständnis von Anweisungen, Parametern und Objektbezeichnungen. So zeigt die Notation der Anweisungen, wie sie in Listing 3.2 zu erkennen sind, starke Züge der Notation in einer objektorientierten Programmiersprache wie Java auf.

```
doinorder(
        # recoil
        bunny.resize(toptobottom, 0.5, likerubber),

        # leap
        dotogether(
                bunny.resize(toptobottom, 2, likerubber),
                bunny.move(up, 1)
        ),

        # fall
        bunny.move(down, 1),

        # bounce
        bunny.resize(toptobottom, 0.5, likerubber),
        bunny.resize(toptobottom, 2, likerubber)
)
```

Listing 3.2: Beispiel eines Animatonsskriptes in Alice aus [Conway 1997]. Auffällig sind die Verwendung von relativen Orts- und Richtungsbezeichnungen sowie die Notation in einer objektbezogenen Schreibweise, wie sie in Programmiersprachen wie Java üblich ist.

3.4 Kommerzielle Skriptsprachen zur Generierung von Animationen

Eine große Verbreitung haben Skriptsprachen in Multimedia-Autorenwerkzeugen erfahren. Sie werden dort zwar auch zur Animationsgenerierung genutzt, verfügen jedoch über weit mehr Funktionen und haben ein größeres Aufgabenspektrum abzudecken. Das Mehr an Aufgaben macht sie um ein großes Maß komplexer und umfangreicher als reine Animationsskriptsprachen. Eine gute Übersicht bietet hier [Hintze 2003].

Zwei Vertreter dieser Skriptsprachen sind ActionScript und Lingo, die in den Programmen Flash und Director von Macromedia[3] zum Einsatz kommen. Funktionen, die in ActionScript verfasst sind, können mit Ereignissen wie Mausklicks oder Benutzerinteraktionen in einem Interface verknüpft werden. In Flash generierte Szenen können mit ActionScript animiert werden. Dabei handelt es sich in den meisten Fällen um zweidimensionale Animationen flächiger, vektorieller Darstellungen.

Lingo verfolgt ein Konzept, bei dem zwischen einer einsteigerfreundlichen Variante und einer Variante für Experten unterschieden wird [Welsch 1997]. Die Syntax der Einsteigervariante ist wortreich und orietentiert sich an natürlichsprachlichen Ausdrücken. Die Syntax der Expertenvariante ähnelt dagegen mehr klassischen Programmiersprachen (siehe Listing 3.3). Beide Syntaxvarianten können bei Lingo frei kombiniert werden.

Anweisung im wortreichen Syntax für Einsteiger:

```
set the forecolor of sprite 10 to 155
```

Anweisung im kompakteren Syntax für Experten:

```
sprite(10).foreColor = 155
```

Listing 3.3: Beispielanweisungen der Skriptsprache Lingo in der Variante für Einsteiger und Experten

Als dritte Skriptsprache eines Autorensystems soll an dieser Stelle MEL (Maya Embeded Language) erwähnt sein. MEL ist eine Modellierungssprache für die 3D-Entwicklungsumgebung Maya der Firma Alias[4]. MEL dient der Erzeugung von Szenen und Objekten auf der Grundlage genauer Geometrieberechnungen. Neben der Aufgabe der Gestaltung von Benutzerschnittstellen bietet die Skriptsprache auch die Möglichkeit Szenen zu animieren.

Zusammenfassend lässt sich sagen, dass die drei vorgestellten Skriptsprachen für Multimedia-Autorenwerkzeuge in ihrer Art sehr komplex und im Aufbau und der Syntax Programmiersprachen wie Java oder C++ sehr ähnlich sind. So weisen

[3] http://www.macromedia.com
[4] http://www.alias.com

allesamt Kontrollkonstrukte wie Schleifen und `if`-Anweisungen auf. Auch besteht bei vielen die Möglichkeit, Variablen zu definieren. Eigenschaften, die bei einer reinen Animationsskriptsprache nicht zwingend notwendig sind.

3.5 Animationserzeugung mit MeVisLab

3.5.1 Möglichkeiten der Animationserzeugung in MeVisLab

In der Bildanalysesoftware MeVisLab kommt das Modul `SoMovieScripter` für die Animationsgenerierung zum Einsatz. Da die Implementierung des in dieser Arbeit entwickelten Konzeptes ebenfalls in MeVisLab erfolgt ist, werden die Grundlagen zum Aufbau und der Funktionsweise dieser Software in Abschnitt 5.1 gesondert behandelt.

Beim `SoMovieScripter` handelt es sich um ein Modul, welches in bestehende MeVisLab-Netzwerke eingebettet werden kann. In Kombination mit einem Viewer[5] zum Betrachten der Szenen (`SoExaminerViewer`) verfügt der `SoMovieScripter` über drei Möglichkeiten eine Animation zu erzeugen:

- Die Szene kann um die drei Koordinatenachsen in beide Richtungen rotiert werden.
- Es kann mittels eines Skriptes eine Animation erzeugt werden.
- Es können einzelne Rotation um eine frei wählbare Achse in einem bestimmten Zeitabschnitt (zwischen zwei bestimmten Frames) innerhalb der Gesamtanimation auf Knopfdruck direkt ausgeführt werden.

Aufgrund der Bedeutung für diese Arbeit soll im Folgenden genauer auf die skriptgesteuerte Animationserzeugung eingegangen werden.

3.5.2 Skriptgesteuerte Animationserzeugung

Alle Ausdrücke der Skripte des `SoMovieScripter` werden geklammert und von einem `$`-Zeichen angeführt (siehe Listing 3.4). Ein Ausdruck besteht aus einer Zeitbereichsangabe, dem Befehl und eventuell zum Befehl gehörenden Parametern. Der zeitliche Bereich, für den der Befehl gelten soll, kann in Frames, Sekunden oder als prozentualer Anteil an der Gesamtanimation angegeben werden.

Der Umfang an Befehlen beinhaltet Befehle zum Rotieren der Kamera sowie zum Interpolieren von Werten. Zur Rotation der Kamera um die drei Achsen stehen die Befehle `rotateX`, `rotateY`, `rotateZ` und `rotateXAndBack`, `rotateYAndBack`, `rotateZAndBack` zur Verfügung. Die letzten drei Befehle rotieren die Kamera in der ersten Hälfte des Zeitbereichs um die angegebene Gradzahl in die Hinrichtung und in der zweiten Hälfte in die Rückrichtung. Die Angabe des Winkels erfolgt als

[5] Als Viewer wird hier ein Modul zur Betrachtung von 2D-Bildern und 3D-Szenen bezeichnet.

Parameter des Befehls und in Grad. Zusätzlich kann die Kamera mit dem Befehl `rotateAxis` um eine frei definierbare Achse rotiert werden (siehe auch Listing 3.5).

Neben der Rotation können Variablen über Zeitbereiche interpoliert werden. Zur Verfügung stehen hier *Integer*-Werte, Gleitkommazahlen, Vektoren, Rotationsvektoren und Ebenen. Pro Variablenart stehen jeweils acht Variablen zur Verfügung. Diese können über das Netzwerk mit den Variablen anderer Objekte verknüpft und so während der Animation neben der Kamerarotation weitere Werte innerhalb des Netzwerkes verändert werden.

```
$(Zeit Befehl Parameter)

$(0-3s rotateY 180)
```

Listing 3.4: Aufbau einer Skriptanweisung im `SoMovieScripter` sowie eine Beispielanweisung

```
$(0-25% rotateYAndBack 60)
$(25-50% rotateYAndBack -60)
$(50-75% rotateXAndBack -60)
$(75-100% rotateXAndBack 60)
$(75-100% int0 1 0)
```

Listing 3.5: Ein Beispielskript des `SoMovieScripter`, bei dem zunächst die Kamera um verschiedene Achsen rotiert wird. Mit der letzten Anweisung (einer Interpolation) wird ein Objekt ausgeblendet.

3.5.3 Zusammenfassung des Skriptansatzes in MeVisLab

Zusammenfassend lässt sich sagexn, dass der `SoMovieScripter` nur für die Erstellung von einfachen Rotationen eine praktikable Möglichkeit bietet. Durch die Interpolation verschiedener Variablenwerte ist zwar theoretisch auch die Manipulation anderer Parameter möglich. Dies erfordert aber ein hohes Maß an Wissen und viel Erfahrung im Bau von Netzwerken in MeVisLab. Hinzu kommt, dass die Datensätze bzw. die darin enthaltenen Objekte für den Anwender nicht offen als Netze vorliegen und so keine Verknüpfung von Objektparametern mit dem `SoMovieScripter` möglich ist.

Aufgrund des sehr unflexiblen und nur schwer erweiterbaren Konzeptes des `SoMovieScripters` wurde im Rahmen dieser Arbeit von einer Erweiterung dieses Moduls abgesehen und stattdessen ein neues Konzept entwickelt und implementiert.

3.6 Animationserzeugung in Amira

3.6.1 Grundlagen zu Amira

Amira[6] ist ein Werkzeug zur 3D-Visualisierung verschiedenster Daten. Anwendung findet Amira vor allem in der Medizin, der biologischen Forschung, in den Geowissenschaften und bei Visualisierungen im Ingenieurbereich. Im Rahmen dieser Arbeit waren vor allem die Anwendungsmöglichkeiten im medizinischen Bereich von Interesse.

Amira-Anwendungen bestehen aus Objektnetzen. Die Objekte nehmen unter anderem Funktionen des Ladens von Daten, der Datenverarbeitung (z.B. arithmetische Operationen auf den Daten oder Anwendung von Filtern) und der Visualisierung wahr. Die Objekte sind durch ein Netzwerk miteinander verbunden und werden dementsprechend einander zugeordnet. So ist zum Beispiel das Objekt zum Laden eines CT-Schnittbilddatensatzes (`reg005.surf`) in Abbildung 3.5 mit einem Objekt zur Visualisierung eines Oberflächenrenderings (`SurfaceView2`) dieser Daten verbunden.

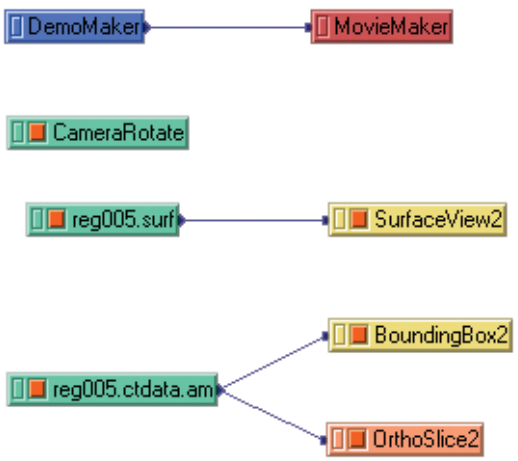


Abbildung 3.5: Amira-Netzwerk mit Objekten zur Oberflächenvisualisierung und Schnittbilddarstellung. Zusätzlich enthält das Netzwerk das Objekt `DemoMaker` zur Erzeugung von Animationen.

Des Weiteren verfügt jedes Objekt über eine Reihe von *ports*, über die einzelne Parameter der Objekte verfügbar und veränderbar sind. Auf die Daten der *ports* jedes Objektes kann über das dazugehörige graphische Interface oder über TCL[7]-Kommandos in einem Konsolenfenster zugegriffen werden. Die Portwerte können aber auch durch andere Objekte automatisch verändert werden.

[6] http://www.amiravis.com

[7] *Tool Command Language:* TCL ist eine Skriptsprache, deren Kommandos im Aufbau denen von Kommandozeileninterpretern (z.B. MS-DOS) sehr ähnlich sind.

3.6.2 Möglichkeiten der Animationserzeugung in Amira

Bei der Erzeugung von Animationen kommt bei Amira das Objekt des `DemoMakers` zum Einsatz. Erweitert um das Objekt des `MovieMakers` können die generierten Animationen auch als Video oder Bildfolge exportiert werden. Einzelne Objekte können entweder durch das Objekt `Animate` oder speziell für Rotation und Translation durch die Objekte `ObjectRotate` und `ObjectTranslate` animiert werden.

Das Objekt des `DemoMakers` dient der automatischen Steuerung einzelner Objekte und Netzwerke in Amira mit dem Ziel, eine Animation zu generieren. Der `DemoMaker` hat Zugriff auf die *ports* aller im Netzwerk vorhandenen Objekte und kann deren Werte direkt verändern. Der Nutzer muss dazu im `DemoMaker` eine Liste von Ereignissen[8] (*event list*) erzeugen, die im Verlauf der Animation abgearbeitet werden sollen.

Ereignisse beschreiben im `DemoMaker` die diskrete Veränderung des Wertes eines *ports* zu einem Zeitpunkt oder die kontinuierliche Veränderung eines Wertes über einen Zeitraum. Beide Zeitangaben erfolgen, genauso wie die Länge der gesamten Animation, in Zeiteinheiten. Erst später findet eine Umrechnung in Sekunden statt.

Erweitert um die Objekte `CameraPath` und `CameraRotate` können Position und Ausrichtung der Kamera während der Animation gesteuert werden. Gerade das Objekt `CameraPath` bietet dem Nutzer umfangreiche Möglichkeiten den Weg der Kamera zu definieren. Dazu kann der Nutzer im Viewer von Amira die Kamera an die gewünschten Positionen navigieren und diese dann als einen Keyframe zum Kamerapfad hinzufügen. Während der Animation wird dann von der Kamera dieser Pfad abgefahren, wobei diese dabei ihre Sicht auf die Mitte der Szene beibehält. In einem gesonderten Fenster wird zudem der aktuelle Kamerapfad mit den einzelnen definierten Kamerapositionen aus einer externen Sicht dargestellt (siehe Abbildung 3.6). Das Objekt `CameraRotate` bietet die Möglichkeit die Kamera um die drei Achsen zu rotieren.

Das Objekt `Animate` kann mit jedem Objekt verbunden werden, das über *ports* mit veränderlichen Zahlenwerten verfügt. Dies gilt auch und vor allem für Objekte zur Visualisierung. Das `Animate`-Objekt kann jeweils den Wert eines *ports* über die Zeit verändern und diesen so animieren. Zur Steuerung der Geschwindigkeit einer Animation ist zusätzlich ein `Time`-Objekt nötig. Die Ausgabe einer Animation aus dem `Animate`-Objekt ist wie auch beim `DemoMaker` unter Zuhilfenahme des `MovieMakers` möglich.

Mit Hilfe des `MovieMakers` kann eine Animation aus Amira exportiert werden. Der Export kann sowohl in eine Video-Datei als auch als Sequenz von Einzelbildern erfolgen.

[8] Ein Ereignis ist ein Vorfall, *"der im System, beim Objekt eine Aktivität bzw. Zustandsänderung auslöst."* [Fischer und Hofer 2004]

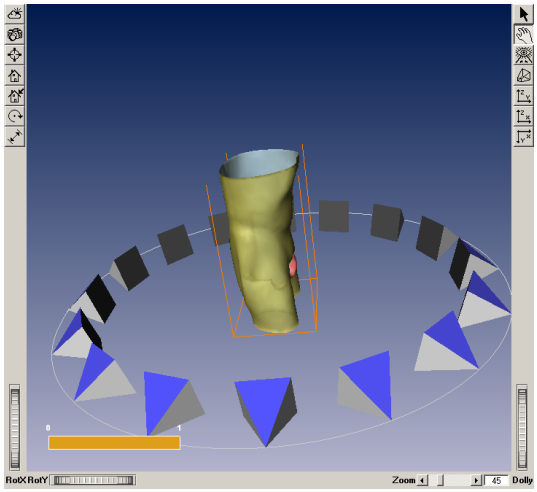


Abbildung 3.6: Darstellung eines Kamerapfades mit den einzelnen Kamerapositionen in einem Amira-Viewer

Der `MovieMaker` bietet unterschiedliche Einstellungsmöglichkeiten. So kann die vorher nur in Zeiteinheiten definierte Länge einer Animation durch Angabe von Framezahl und Framrate festgelegt werden. Die Auflösung des Videos kann ebenfalls frei definiert werden.

Als Ergebnis steht am Ende der Erzeugungskette entweder ein Video, welches die 3D-Animation entsprechend der Ereignisliste wiedergibt, oder eine dementsprechende Sequenz von Einzelbildern.

3.6.3 Skripte in Amira und deren Nutzung zur Animationsgenerierung

Neben der Steuerung von Amira über die grafische Benutzeroberfläche können fast alle Funktionalitäten von Amira auch über TCL-Kommandos gesteuert werden. Amira hat die Menge an TCL-Kommandos um einen eigenen Satz an Befehlen erweitert, das Amira Script Interface. Dieses ermöglicht, wie im Listing 3.6 zu sehen, den direkten Zugriff auf jedes Objekt im Netzwerk und dessen *ports*. Über globale Kommandos können zusätzlich neue Objekte zum Netzwerk hinzugefügt oder von ihm entfernt werden.

```
<object> <port> <port-command> <arguments>
```

Listing 3.6: Struktur einer Skriptanweisung aus dem Amira Script Interface, mit der ein direkter Zugriff auf alle Objekte und deren Eigenschaften möglich ist.

Skripte in Amira können sowohl eine Reihe von Befehlen zur Steuerung eines schon bestehenden Netzwerkes enthalten als auch sämtliche Kommandos zum Erzeugen einen neues Netzwerkes. Amira selbst speichert alle Netzwerke in Form von Skripten.

Diese werden beim Laden eines Netzwerkes ausgeführt und das Netzwerk wird so erstellt.

Die Nutzung von Amira-Skripten zum Erstellen von Animationen erscheint dagegen nicht praktikabel. Der mögliche Zeitgewinn durch die Nutzung von wieder verwendbaren Skripten oder die mögliche Einfachheit gegenüber den anderen hier vorgestellten Methoden im grafischen Interface von Amira wird durch die Komplexität der Skriptsprache zunichte gemacht. Das Laden von neuen Datensätzen in ein schon bestehendes Netzwerk macht sowohl im grafischen Interface als auch auf der Ebene eines Skriptes faktisch die Konstruktion eines komplett neuen Netzwerkes nötig, da sämtliche Objekte von der Visualisierung bis zur Ereignisliste im `DemoMaker` neu erstellt und mit den neuen Daten verknüpft werden müssten.

3.6.4 Zusammenfassung der Animationsmöglichkeiten in Amira

Im Gesamtblick bietet Amira eine gute Möglichkeit 3D-Animationen aus medizinischen Daten zu erzeugen. Der Ansatz der Ereignislisten erscheint grade bei der Darstellung von parallel ablaufenden Veränderungen sehr sinnvoll. Auch die Nutzung von Zeiteinheiten als unabhängige Zeitgröße ermöglicht sowohl eine variable Animationslänge und -geschwindigkeit wie auch eine pseudo-prozentuale Definition aller Ereignisse[9].

Ein großes Manko ist die fehlende Möglichkeit einmal erstellte Ereignislisten des `DemoMaker` abzuspeichern. Der Umweg über Skripte ist theoretisch möglich. Da jedoch für jeden neuen Datensatz faktisch ein gesamtes Netzwerk neu generiert werden muss (sei es nun in Skriptform oder grafisch), ist ein praktischer Einsatz der Skripte zum Generieren von Animationen nicht möglich.

3.7 Arbeiten zu medizinischen Lernsystemen

Die Anwendungsgebiete von Animationen in der medizinischen Ausbildung sowie in der Operations- und Therapieplanung wurden bereits im Abschnitt 1.1.1 beschrieben. In diesem Abschnitt wird ein System zur Ausbildung von Leberchirurgen vorgestellt.

Computergestützte Lernsysteme (*computer based training, CBT*[10]) erhalten in der medizinischen Ausbildung ein immer stärkeres Gewicht. Im Bereich der Ausbildung während des Medizinstudiums kommt es vor allem auf die Vermittlung von Grundkenntnissen an. Auch wenn dabei im Bereich der Anatomieausbildung mit Leichen und Präparaten konkrete Fälle vorliegen (wenn auch immer noch in nicht ausreichendem Maße), so ist die idealtypische Darstellung der anatomischen Strukturen immer noch die am meisten verbreitete. Dies drückt sich auch in der

[9] Genau dann, wenn die Länge der Animation auf 100 Zeiteinheiten festgelegt wird, können die Angaben der Zeitpunkte der Ereignisse als prozentuale Anteile an der Gesamtanimation betrachtet werden.

[10] CBT: *"Computer based training ist eine Ausbildungsform, bei der Computer nicht als Lerngegenstand, sondern als Lehrmittel eingesetzt werden."* [Haag 1998]

starken Nutzung von Anatomieatlanten aus. Mit den Anwendungen des VoxelMan fand dieser Lernansatz seine Umsetzung in computergestützten Lernsystemen (vgl. [Höhne u. a. 2003] und [Tiede 1999]). Der VoxelMan bietet eine hochaufgelöste dreidimensionale Ansicht der Anatomie des Menschen. Als Datengrundlage dienten Fotografien und radiologische Bilddaten einer in dünne Schichten geschnittenen männlichen Leiche.

Ein anderer Ansatz, der des problemorientierten Lernens (POL), wird bei einer Reihe anderer medizinischer Lernsysteme verfolgt. In der Ausbildung von Medizinern herrscht eine starke Abhängigkeit zum Einen von aktuell verfügbaren Fällen und zum Anderen von Experten zur Vermittlung des Spezialwissens. Hier bieten sich Lernsysteme an, die diese Abhängigkeiten auflösen und dem Lernenden die Möglichkeit eröffnen, sich orts- und zeitunabhänig weiterzubilden. Um dieses Ziel zu erreichen, müssen die Lernsysteme auf der Basis von realen, patientenindividuellen Fällen aufgebaut sein. Beispiele für solche Systeme sind das CASUS-Programm der Ludwig-Maximilians-Universität in München [Fischer u. a. 1996] und das CAMPUS-System der Universität Heidelberg [Haag 1998].

An der Otto-von-Guericke Universität Magdeburg wurde der LiverSurgeryTrainer entwickelt ([Bade u. a. 2004] und [Mirschel 2004]). Hauptziele dieser Lernsoftware sind die praxisnahe Vermittlung und das Training von Therapieentscheidungen und Operationsplanungen, speziell in der Leberchirurgie. Dem LiverSurgeryTrainer dienen reale, patientenindividuelle Fälle als Grundlage. Der LiverSurgeryTrainer stellt die Bilddaten mit Bildanalyseergebnissen, 3D-Modellen, kommentierten Operationsvideos sowie Diagnoseinformationen und Patientendaten für den Lernenden dar. Diesem wird die Möglichkeit gegeben die Planung Schritt für Schritt nachzuvollziehen und eigene Entscheidungen zu treffen. Sie werden bewertet und dem Lernenden wird so ein Feedback gegeben.

Der große Vorteil des Systems liegt in seiner Fallbasiertheit. Das macht es jedoch nötig, immer wieder aktuelle Fälle in das System einzupflegen. Dies ist bei den dreidimensionalen Daten immer noch mit einem großen Aufwand verbunden. Der LiverSurgeryTrainer beinhaltet die Möglichkeit sich die dreidimensionalen radiologischen Bilddaten jedes Falles interaktiv zu betrachten (siehe Abbildung 3.7), wie es beispielsweise auch im InterventionPlanner [Preim u. a. 2003] möglich ist und in Abschnitt 5.2.1 näher beschrieben wird.

Die interaktive Darstellung wird durch Animationen unterstützt. So werden Hervorhebungswechsel hin zu einem vom Nutzer gewählten Objekt animiert dargestellt. Diese Animationen müssen vom Autor für jeden neuen Fall, der in den LiverSurgeryTrainer eingepflegt wird, direkt im Autorensystem[11] programmiert werden.

Auch wenn in diesem Fall nicht das Erstellen einer fertigen Animation in Form eines Videos das Ziel ist, besteht beim LiverSurgeryTrainer dennoch das Problem Animationen einfach zu erzeugen. Dies motiviert die Entwicklung eines Konzeptes zur Animationsgenerierung. Ein solches System muss flexibel sein und dem Autor,

[11] Der LiverSurgeryTrainer wurde im Autorensystem Macromedia Director 8.5 entwickelt.

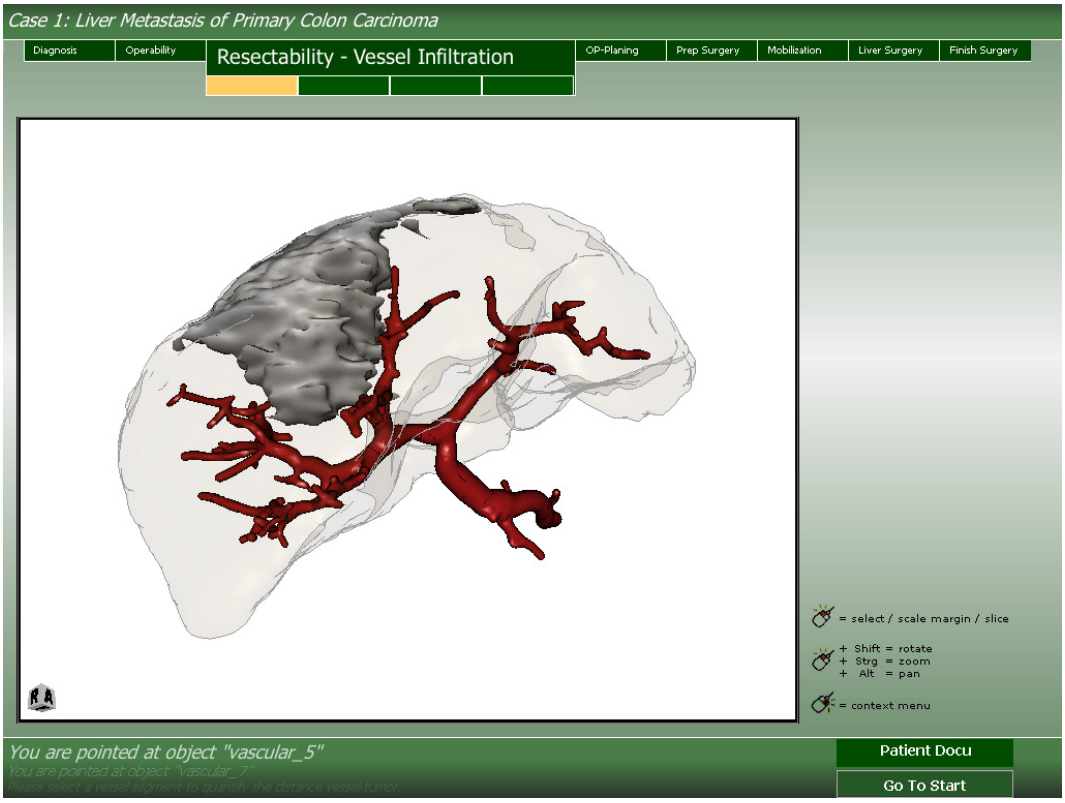


Abbildung 3.7: Screenshot des LiverSurgeryTrainers mit einer interaktiven Darstellung.

in diesem Fall demjenigen, der die Fälle in den LiverSurgeryTrainer einpflegt, ein großes Maß an Freiheit und individuellen Gestaltungsmöglichkeiten bieten.

3.8 Zusammenfassung der Arbeiten

Im Folgenden sollen Aspekte des letzten Kapitels, die für die vorliegende Arbeit von Bedeutung sind, zusammengefasst werden.

Als Erstes ist dabei der Ansatz der Zerlegung von komplexen Animationsanweisungen in einfachere Basisanweisungen zu nennen, wie er von [Zeltzer 1991] mit dem System von verschiedenen Ebenen beschrieben wurde. Der Ansatz findet sich in vielen Arbeiten wieder , so auch in der Animationskomponente des ZoomIllustrators [Steinicke 1996]. Dieser Animationskomponente liegt zwar eine komplizierte und stark an den ZoomIllustrator angepasste Skriptsprache zugrunde. Dennoch ist die interaktive Erzeugung von Skriptanweisungen hier hervorzuheben.

ALICE [Conway 1997] bietet mit der Möglichkeit, Positionen und Bewegungen koordinatenfrei angeben zu können, einen Ansatz, der vor allem für weniger erfahrene Anwender einer Skriptsprache interessant und vielversprechend erscheint. Einen

weiteren Ansatz für eine Skriptsprache für Einsteiger bietet Lingo mit seiner sehr stark natürlichsprachlich geprägten Syntax. Hier sticht die freie Kombinationsmöglichkeit der einfachen mit der komplizierten Syntax für Experten hervor. Dennoch ist festzuhalten, dass beim Einsatz von Skripten zur Animationsgenerierung ein gewisses Grundverständnis vom Aufbau einer Anweisung und deren Wirken vorhanden sein muss.

Mit dem `SoMovieScripter` in MeVisLab und Amira präsentieren sich zwei Softwarelösungen, die besonders im Bereich medizinischer Animationen von Interesse sind. Allerdings wirkt das Konzept des `SoMovieScripters` zu sehr beschränkt. Amira bietet an dieser Stelle wesentlich mehr Möglichkeiten, Animationen zu generieren. Jedoch fällt hier das Fehlen eines Skriptansatzes auf, der reproduzierbare Animationen verschiedener Datensätze ermöglicht.

Kapitel 4

Entwicklung eines Skriptkonzeptes

4.1 Theoretisches Konzept

Im Folgenden wird ein Konzept einer Skriptsprache entwickelt. Wie in Abschnitt 2.6 festgestellt wurde, sind Skriptsprachen stark auf ihre Anwendungsfelder zugeschnitten. Im Fall dieser Arbeit ist das die Generierung von Animationen anhand medizinischer Daten.

Im Rahmen dieser Arbeit wurde das Konzept einer Verarbeitungskette entwickelt, welches alle Schritte von der Skripterstellung bis zur fertigen Animation umfasst. Das Schema in Abbildung 4.1 zeigt eine solche Verarbeitungskette. Den Ausgangspunkt bildet ein Skript, welches die Abläufe und Darstellungen innerhalb der Animation beschreibt. Zur Erzeugung und Darstellung der einzelnen Bilder der Animation wird eine Software vorausgesetzt, die die medizinischen Daten aufbereitet und visualisiert. Diese Software muss eine Schnittstelle (Skriptschnittstelle) besitzen, die es einem Autor über Skript ermöglicht, Veränderungen an der Visualisierung der Daten vorzunehmen. Eine Skriptschnittstelle übernimmt die Aufgabe Anweisungen eines Skriptes in eine bildweise Repräsentation und damit in direkte Anweisungen zur Manipulation der Visualisierung umzuwandeln.

Ist nur eine Schnittstelle zur allgemeinen Manipulation der Visualisierung vorhanden (z.B. durch Nutzerinteraktionen), so muss eine Skriptschnittstelle geschaffen werden. Die Software muss weiterhin die Möglichkeit bieten, die erzeugten Bilder zu speichern. Aus diesen Bildern wird dann im letzten Schritt der Verarbeitungskette ein Animation erzeugt.

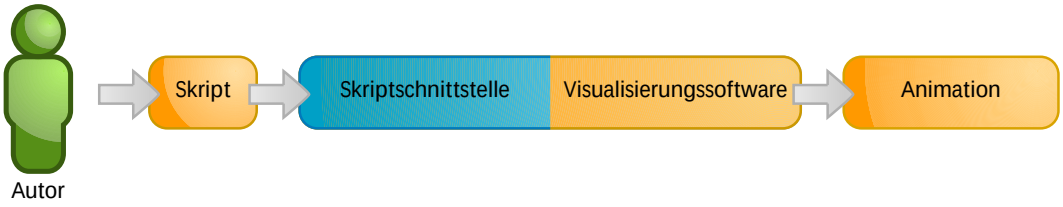


Abbildung 4.1: Verarbeitung eines Skriptes über die Skriptschnittstelle einer Visualisierungssoftware zur Animation

In Abschnitt 2.9 wurden zwei wesentliche Anforderungen an eine Skriptsprache zur Generierung von medizinischen Animationen beschrieben: Die Möglichkeit zur Manipulation von Objekten und Szene (*Flexibilität*) und die Nutzung der Skriptsprache sowohl für professionelle wie auch unprofessionelle Autoren (*Geeignete Benutzungsschnittstelle*).

Die Anforderung der Flexibilität lässt sich vereinfachen, wenn Szene und Kamera ebenfalls als Objekte mit einer Reihe von Parametern betrachtet werden. So reduziert sich die Lösung des Problems der Flexibilität auf die Möglichkeit Objektparameter zu verändern. Eine Veränderung von Parametern impliziert aber auch eine zeitliche Komponente. Es ist hier zwischen einer Veränderung zu einem bestimmten Zeitpunkt und der Veränderung über einen Zeitraum zu unterscheiden. Beides muss in der Skriptsprache berücksichtigt werden.

Zur Lösung des Problems einer geeigneten Benutzungsschnittstelle, sowohl für professionelle wie auch unprofessionelle Autoren, wurden im Rahmen dieser Arbeit zwei Möglichkeiten in Betracht gezogen.

4.1.1 Paralleler Ansatz

Ein Ansatz bestand darin, zwei völlig unabhängige Skriptsprachen zu entwickeln. Dies bietet die Möglichkeit, jede Skriptsprache individuell an die Bedürfnisse der beiden Autorengruppen anzupassen. Man hat zwei Befehlssätze, wobei die Befehle für den unprofessionellen Autor eine größere Abstraktheit aufweisen als die für den professionellen Autor. Auch existiert für den unprofessionellen Autor ein völlig anderer Ansatz des Skript- und Anweisungsaufbaus.

Jeder Befehlssatz greift für sich unabhängig auf die Funktionen zu, die von der Skriptschnittstelle angeboten werden (siehe Abbildung 4.2). Eine detaillierte Skriptsprache für den professionellen Autor macht möglichst alle dort verfügbaren Daten und Funktionen für den Autor zugänglich. Eine allgemeiner gehaltene Skriptsprache dagegen abstrahiert viele der Daten vor dem Autor. Zum Beispiel kann der Autor mit einer allgemeineren Skriptsprache nicht mehr die genaue Position der Kamera bestimmen, sondern nur noch aus einer Reihe von Sichten wählen (z.B. *axial*, *sagittal*, *coronal*).

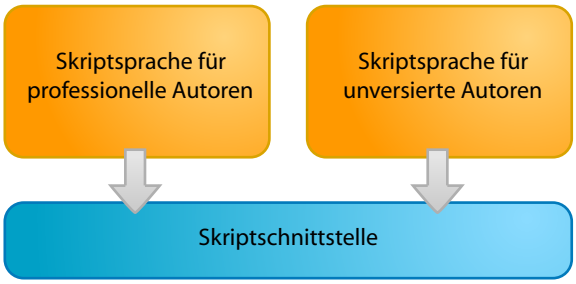


Abbildung 4.2: Paralleler Zugriff zweier unabhängiger Skriptsprachen auf die gleiche Schnittstelle

Der Nachteil zweier unabhängiger Skriptsprachen besteht in der fehlenden Kompatibilität. So lässt sich ein Skript nur entweder in der sehr abstrakten oder in der sehr detaillierten Skriptsprache verfassen. Um in einer Animation, die mithilfe eines abstrakten Skriptes erstellt wurde, eine Detailveränderung vornehmen zu können, die nur über einen Befehl der detaillierten Skriptsprache möglich wäre, müsste der Autor zunächst das gesamte Skript in ein Skript der detaillierten Skriptsprache überführen.

4.1.2 Hierarchischer Ansatz

Um die Nachteile zweier paralleler Skriptsprachen zu vermeiden wurde im Rahmen dieser Arbeit ein anderer Ansatz verfolgt. Anstatt zwei unabhängige Skriptsprachen mit ihrem jeweils eigenen Befehlssatz zu entwickeln wurde eine Skriptsprache mit einem einzigen Befehlssatz entworfen. Um dabei dennoch beiden oben genannten Autorengruppen eine für sie passende Benutzerschnittstelle bieten zu können enthält der Befehlssatz Anweisungen unterschiedlicher Abstraktion. Es gibt detaillierte, atomare Anweisungen, die möglichst jede Funktion, die von der Schnittstelle angeboten wird, abbilden. Auf diesen Anweisungen setzen abstraktere Anweisungen für die unprofessionellen Autoren auf (siehe Abbildung 4.3).

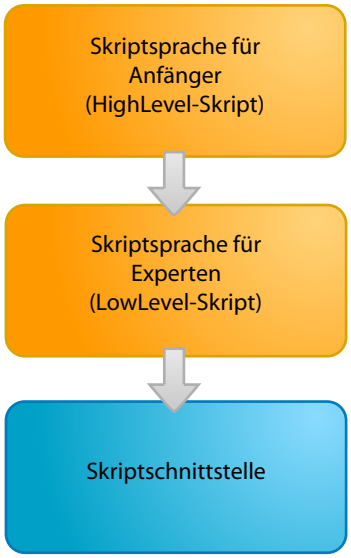


Abbildung 4.3: Hierarchischer Skriptansatz. Abstrakte Anweisungen (*HighLevel*) bauen auf atomaren Anweisungen (*LowLevel*) auf.

Die Abstraktion wirkt sich auf alle der in Abschnitt 4.2.2 genauer beschriebenen Teile der Anweisungen aus. Es besteht die Möglichkeit mehrere Befehle zu einem abstrakteren Befehl zusammenzufassen. Objekte können zu Objektgruppen kombiniert und die Parameter der Befehle abstrakter definiert werden.

Die höhere, abstraktere Ebene wird im weiteren Verlauf als *HighLevel*-Ebene bzw. *HighLevel*-Skript bezeichnet, die detaillierte, atomare Ebene als *LowLevel*-Ebene

bzw. *LowLevel*-Skript. Beide Ebenen treten als eigenständige Skripte zutage, die jedoch in derselben Skriptsprache verfasst werden.

Im Zuge der Animationserstellung wird das *HighLevel*-Skript in ein *LowLevel*-Skript umgewandelt. Ziel des Umwandlungsprozesses ist es, in Form des *LowLevel*-Skriptes eine Sammlung atomarer, das heißt unteilbarer Anweisungen zu haben, die in ihren Befehls- und Objektbezeichnungen die gegebenen Funktionen und Objekte der Skriptschnittstelle direkt repräsentieren.

Der große Vorteil des hier entwickelten Konzeptes aufeinander aufbauender Skripte mit einem gemeinsamen Befehlssatz liegt in der Kombinationsmöglichkeit beider Skripte. Es ist möglich, Anweisungen der *HighLevel*-Ebene mit denen der *LowLevel*-Ebene zu kombinieren. Bei einer Umwandlung eines *HighLevel*-Skriptes in ein *LowLevel*-Skript würden die im *HighLevel*-Skript vorhanden *LowLevel*-Anweisungen erhalten bleiben und nur die *HighLevel*-Anweisungen in *LowLevel*-Anweisungen umgewandelt werden. Es ist so möglich, Details betreffende Anweisungen in einem abstrakten *HighLevel*-Skript vorzunehmen, ohne das gesamte *HighLevel*-Skript zunächst in eine *LowLevel*-Skript überführen zu müssen.

Nach der in Abschnitt 2.6 vorgestellten Definition eines Skriptes müssten die Anweisungen von einem Interpreter der Reihe nach direkt ausgeführt werden. Dabei gerät man aber mit dem Aufbau der hier entwickelten Skripte in Konflikt. Durch die als unbedingt notwendig erachtete Anforderung, Objektparameter auch über einen Zeitraum verändern zu können, kann es zu Parallelen und Überschneidungen der Anweisungen kommen. Eine rein serielle Abarbeitung eines Skriptes kommt daher nicht in Frage.

Aus diesem Grund wurde zwischen die Schicht des *LowLevel*-Skriptes und der Schnittstelle zum umgebenden Programm die Schicht der Frameliste eingeführt. Diese Schicht beinhaltet für jeden Frame der Animation die Liste der in diesem Frame auszuführenden Anweisungen. Eine solche Frameliste wird seriell frameweise abgearbeitet. Für jeden Frame werden die entsprechenden Anweisungen ausgeführt.

Im engeren Sinne kann der Vorgang des Überführens eines *LowLevel*-Skriptes in eine Frameliste durchaus als Kompilierung und der Begriff des Skriptes für die *HighLevel*- und *LowLevel*-Skripte als nicht zutreffend angesehen werden. Dennoch wird hier weiter von dem Begriff des Skriptes gesprochen. Die *HighLevel*- und *LowLevel*-Skripte weisen für den Autor nach außen alle Eigenschaften eines Skriptes auf. Auch wenn die bildweise Repräsentation nach außen hin nicht sichtbar ist, so stellt sie als bloße Folge von Objektparameterveränderungen (die nur anders angeordnet und aufgeschlüsselt sind als beim ursprünglichen Skript) ein eigenes Skript dar. Dieses kann seinerseits nun seriell und direkt abgearbeitet werden.

4.2 Syntax und Aufbau der Skriptsprache

Nachdem im vorigen Abschnitt der konzeptionelle Aufbau in drei Schichten dargelegt wurde, werden in diesem Abschnitt die Syntax der im Rahmen dieser Arbeit entwickelten Skriptsprache sowie der Aufbau der Skripte vorgestellt.

4.2.1 Aufbau eines Skriptes

Sowohl *HighLevel*- als auch *LowLevel*-Skript gliedern sich in zwei Bereiche: einen Initialisierungsteil und einen Anweisungsteil. Im Initialisierungsteil werden Einstellungen vorgenommen, die die Länge der Animation betreffen und die im Abschnitt 4.2.2 näher beleuchtet werden. Im Anweisungsteil folgen die eigentlichen Anweisungen zur Animationsgenerierung, wobei jede Anweisung in einer neuen Zeile steht. Einige Beispiele für Skripte finden sich im Anhang C.

4.2.2 Syntax der Anweisungen

Die Anweisungen der Skriptsprache gliedern sich in vier Bereiche (siehe Listing 4.1). Jede Anweisung besteht dabei aus einer Zeitangabe, die den Zeitraum oder Zeitpunkt der Ausführung der Anweisung bestimmt, einer Objektbezeichnung, die das Objekt festlegt, auf das sich die Anweisung bezieht, der Befehlsbezeichnung und möglichen Parametern. Die vier Bereiche sind jeweils durch ein Leerzeichen getrennt (Beispiel einer Anweisung siehe Listing 4.2). Jeder Bereich kann bei der Umwandlung eines *HighLevel*-Skriptes in ein *LowLevel*-Skript Veränderungen unterworfen sein. Im Folgenden wird daher auf die einzelnen Bereiche genauer eingegangen.

```
Time Object Command [Parameter]
```

Listing 4.1: Einteilung einer Skriptanweisung

```
[0,10] 'Liver' setColor blue
```

Listing 4.2: Beispiel einer Skriptanweisung

Zeitangaben

Es gibt zwei verschiedene Möglichkeiten der Zeitangabe: die Angabe eines Zeitpunktes oder die eines Zeitbereiches. Die Zeitangabe erfolgt in eckigen Klammern, wobei bei der Angabe eines Zeitbereiches dessen Start- und Endwert durch ein Komma

getrennt wird (siehe Listing 4.3). Bei der Angabe eines Zeitpunktes wird die Anweisung zu diesem ausgeführt. Im Fall eines Zeitbereiches wird versucht die durch die Anweisung hervorgerufene Änderung in diesem Zeitbereich zu interpolieren. Ist dies nicht möglich, zum Beispiel weil es sich um eine diskrete Zustandsänderung handelt, wird die Anweisung zu Beginn des Zeitbereiches ausgeführt.

```
[0,10] ...
[5] ...
[start,10] ...
[start,end] ...
```

Listing 4.3: Möglichkeiten der Angabe von Zeitpunkten und Zeiträumen für eine Anweisung

Die Angabe der Zeiten erfolgt in abstrakten Zeiteinheiten. Das erlaubt eine größere Flexibilität im Hinblick auf Länge und Framerate einer Animation. So hätte die Angabe der Zeiten auch in Sekunden, Frames oder einer prozentualen Einheit erfolgen können. Jedoch haben alle drei Möglichkeiten größere Nachteile. Bei einer Nutzung von Sekunden könnte dem Wunsch des Autors, eine Animation bei gleichbleibendem Ablauf lediglich in ihrer Gesamtzeit zu verkürzen oder zu verlängern (also die Geschwindigkeit der Animation zu ändern) nicht ausreichend Rechnung getragen werden. In solchen Fällen müsste der Autor sämtliche Zeitangaben neu überdenken und ändern.

Die großen Nachteile in der Nutzung von Frames als Maß für die Zeitangabe liegen in der Unverständlichkeit dieses Zeitmaßes für viele Autoren und in der fehlenden Flexibilität hinsichtlich der Änderung der Framerate einer Animation. Eine veränderliche Framerate kommt zum Beispiel bei der Erstellung von Animationen auf schwächeren Rechnern zum Tragen, wo eine geringere Framerate einen wesentlich geringeren Berechnungsaufwand mit sich bringt.

Prozentuale Angaben als Anteil an der Gesamtzeit einer Animation haben ähnlich wie die Nutzung von Sekunden einen großen Nachteil, wenn es um die Veränderung der Gesamtlänge einer Animation geht. Sind sie im Fall einer oben beschriebenen Geschwindigkeitsänderung problemlos nutzbar, so treten erhebliche Probleme auf, wenn es darum geht, weitere Anweisungen an das Ende einer Animation anzufügen und diese so zu verlängern. In einem solchen Fall müsste der Autor auch hier sämtliche Zeitangaben anpassen.

Die Umrechnung der abstrakten Zeiteinheiten in Sekunden und darauf folgend in konkrete Frames erfolgt mit Hilfe zweier im Initialisierungsteil durch den Autor festgelegten Parameter: der Länge der Animation in Sekunden (`LengthSeconds`) und der Länge der Animation in Zeiteinheiten (`LengthTimeUnit`) (siehe Listing 4.4). Es bestünde die Möglichkeit auch die Framerate im Initialisierungsteil anzugeben. Jedoch wird diese im Rahmen der in der Arbeit erfolgten Implementierung über die

Benutzerschnittstelle im Programm direkt eingegeben. Dies ermöglicht eine rechnerabhängige Framerateeinstellung unabhängig vom Skript.

```
[Ini]
LengthTimeUnit=14
LengthSeconds=10
...
```

Listing 4.4: Initialisierungsteil eines Skriptes mit der Definition der Länge der Animation in Zeiteinheiten und Sekunden

Mithilfe der beiden Zeitparameter im Initialisierungsteil hat der Autor die Möglichkeit die Gesamtlänge in Sekunden unabhängig von den Zeitangaben in den einzelnen Anweisungen festzulegen. Damit ist der oben diskutierte Fall der Geschwindigkeitsänderung möglich. Auch kann der Autor durch eine gemeinsame Erhöhung von `LengthTimeUnit` und `LengthSeconds` weitere Anweisungen zum Skript hinzufügen, ohne die schon bestehenden Abschnitte in ihrem Ablauf oder ihrer Geschwindigkeit zu beeinflussen.

Die Angabe der Zeiten der Anweisung erfolgt bis auf einen kleinen Unterschied im *HighLevel*-Skript und im *LowLevel*-Skript identisch. Auf der Ebene des *HighLevel*-Skriptes besteht zusätzlich die Möglichkeit zwei Schlüsselwörter - `start` und `end` - in den Zeitangaben zu verwenden (siehe Listing 4.3). Bei der Umwandlung des *HighLevel*-Skriptes in ein *LowLevel*-Skript würden diese beiden Schlüsselwörter durch `0` bzw. `LengthTimeUnit` ersetzt. Dadurch ergibt sich ein Vorteil durch eine verbesserte Lesbarkeit des Skriptes und auch eine leicht verbesserte Anwendung der Skriptsprache für den unprofessionellen Autor.

Objektbezeichnungen

An zweiter Stelle innerhalb einer Anweisung erfolgt die Bezeichnung des Objektes, auf welches die Anweisung angewendet werden soll. Um Objektbezeichnungen mit Leerzeichen zu ermöglichen wird diese in Hochkommata gesetzt. Auf der *LowLevel*-Ebene entsprechen die Objektbezeichnungen den Bezeichnungen, die den Objekten bei der Segmentierung der Daten zugewiesen wurden bzw. die sie innerhalb des Datensatzes haben. Im Folgenden werden diese Bezeichnungen als atomare, weil unteilbare, Objektbezeichnungen bezeichnet. Im *HighLevel*-Skript können dagegen andere Bezeichnungen gewählt werden. Diese werden anhand von Ersetzungsvorschriften in atomare Bezeichnungen überführt. Dabei kann eine einfache Ersetzung erfolgen (z.B. kann die Bezeichnung `'Liver'` durch `'Liver, segm'` substituiert werden). Oder eine Bezeichnung kann durch mehrere Bezeichnungen ersetzt werden (z.B. `'Lung'` durch `'Lung left'` und `'Lung right'`). Erfolgt eine Ersetzung hin zu mehreren Objektbezeichnungen, so wird die Anweisung für das jeweilige Objekt im Ganzen kopiert. Aus einer Anweisung für das Objekt

`'Lung'` würden im Ersetzungsvorgang daher je eine Anweisung für `'Lung left'` und `'Lung right'` entstehen. Mithilfe dieser Ersetzungen ist es möglich, auf der Ebene des *HighLevel*-Skriptes eine Anweisung für ganze Objektgruppen oder alle Objekte anzuwenden.

Zusätzlich zu den atomaren Bezeichnungen der Objekte im Datensatz gibt es zwei systemseitige Bezeichnungen: `'Cam'` und `'System'`. Diese kommen bei Anweisungen zur Anwendung, die eine Manipulation der Kamera (z.B. eine Rotation) oder der gesamten Szene (z.B. eine Änderung des Hintergrundes) ermöglichen.

Die Ersetzungsvorschriften werden in einer XML-Struktur abgelegt (siehe Listing 4.5). Die Auszeichnungssprache XML hat sich als Standard zur strukturierten Repräsentation von Daten etabliert. Sie bietet den Vorteil, dass Daten mithilfe von XML in einer frei definierbaren Struktur abgelegt werden können und dabei sowohl maschinell zu verarbeiten als auch vom Menschen lesbar sind. XML bietet sich zudem für die hier entwickelten Ersetzungsvorschriften an, da sie eine hierarchische Strukturierung ermöglicht.

Der entscheidende Vorteil, die Ersetzungsvorschriften nicht intern im Programm zu definieren, sondern extern und frei editierbar abzulegen, liegt in der Möglichkeit für Entwickler und Autoren, vom Programm unabhängig neue abstrakte Anweisungen kreieren zu können. So kann der Autor zum Beispiel mehrere Anweisungen zu einer zusammenfassen und das Skript dadurch besser lesbar machen. Die Wiederverwendung bestimmter Anweisungsblöcke in unterschiedlichen Skripten wird ebenfalls vereinfacht.

Ersetzung von Befehlen

Während eine Anweisung die Gesamtheit aus Zeitangabe, Objektbezeichnung, Befehlsbezeichnung und Parameter darstellt, wird im Weiteren als *Befehl* nur die eigentliche Befehlsbezeichnung betrachtet. Die Befehle des *LowLevel*-Skriptes sind atomar, das heißt, sie sind direkte Repräsentationen von Befehlen der Skriptschnittstelle. Eine Aufstellung elementarer Befehle findet sich im Anhang B. Auf diesen Befehlen aufbauend werden die *HighLevel*-Befehle definiert. Ähnlich den Objektbezeichnungen kommt es auch hier zu Ersetzungen von *HighLevel*-Befehlen in einen oder mehrere andere Befehle. Ebenso können die Ersetzungen hierarchisch strukturiert werden, sodass nicht jeder *HighLevel*-Befehl sofort auf einen atomaren *LowLevel*-Befehl abgebildet werden muss.

Die Ersetzungsvorschriften für Befehle sind ebenfalls in einer XML-Struktur abgelegt. Da die Ersetzung eines Befehls durch einen oder mehrere Befehle aber Auswirkungen auf die gesamte Anweisung (z.B. die Zeitangaben oder Objektbezeichnungen) haben kann, gibt es einige Unterschiede zum Prozess der Objetkbezeichnungsersetzungen.

Die Ersetzungsvorschrift umfasst eine oder mehrere komplette Anweisungen. Als Zeitangabe für die ersetzenden Anweisungen kann wahlweise eine konkrete Zeit, wie auch im Skript, angegeben werden oder aber durch ein `T` die Zeitangabe aus der zu ersetzenden Anweisung übernommen werden. Ähnliches gilt für die

Anweisung mit einer zu ersetzenden Objektbezeichnung:

```
[0,10] 'Lung' on
```

Auszug aus der XML-Struktur mit den Ersetzungsvorschriften:

```
<object name="Lung">
        <object>Lung left</object>
        <object>Lung right</object>
</object>
<object name="Lung left">
        <object>kind2, LiLungeSeg, m Int</object>
</object>
<object name="Lung right">
        <object>kind2, ReLungeSeg, m Int</object>
</object>
```

Zwei Anweisungen als Ergebnis der Ersetzungen:

```
[0,10] 'kind2, LiLungeSeg, m Int' on
[0,10] 'kind2, ReLungeSeg, m Int' on
```

Listing 4.5: Auszug aus einer XML-Struktur zur Definition der Ersetzungsvorschriften für Objektbezeichnungen

Objektbezeichnung, die entweder konkret eine Objektbezeichnung oder aber ein `O` als Platzhalter darstellen kann. Bei einem Platzhalter `O` würde die Objektbezeichnung aus der zu ersetzenden Anweisung genutzt werden. Mit Unterstützung dieser Notation kann die Objektbezeichnung bei der Ersetzung auch als Parameter in die neue Anweisung übernommen werden (siehe Listing 4.6). Mithilfe eines `P` ist es außerdem möglich, auch die in der zu ersetzenden Anweisung eventuell angegebenen Parameter in die neue Anweisung zu übernehmen. Bei der Entwicklung der atomaren und *HighLevel*-Anweisungen ist darauf zu achten, dass die Platzhalter `T`,`O` und `P` nicht als Parameter in Frage kommen.

Die Ersetzungsvorschriften stellen sich komplexer dar als die eigentliche Syntax der Skriptsprache. Sie ist für den Anfänger zwar einseh- und veränderbar, jedoch nicht auf ihn zugeschnitten. Vielmehr soll hier dem professionellen Autor sowie den Entwicklern die Möglichkeit gegeben werden, programmunabhängig neue Befehle auf der Basis der atomaren Befehle zu erzeugen. Diese können dann wahlweise vom professionellen Autor selbst genutzt oder vom Entwickler dem unprofessionellen Autor zur Verfügung gestellt werden.

Beispiel 1:
Ausgangsanweisung:

```
[0,10] 'Liver' on
```

Auszug aus der XML-Struktur mit den Ersetzungsvorschriften:

```
<command commandStr="on">
        <command>T 0 setVisible true</command>
        <command>T 0 setColor red</command>
</command>
```

Ergebnis nach der Ersetzung:

```
[0,10] 'Liver' setVisible true
[0,10] 'Liver' setColor red
```

Beispiel 2:
Ausgangsanweisung:

```
[0,10] 'Liver' view sagittal
```

Auszug aus der XML-Struktur mit den Ersetzungsvorschriften:

```
<command commandStr="view">
        <command>T 0 on</command>
        <command>T 'Cam' move 0 P 1.0</command>
</command>
```

Ergebnis nach der Ersetzung:

```
[0,10] 'Cam' move 'Liver' sagittal 1.0
```

Listing 4.6: Zwei Beispiele für die Ersetzung eines Befehls durch einen anderen

Ersetzung von Parametern

Als Parameter einer Anweisung werden all' diejenigen Angaben gewertet, die nach der Befehlsbezeichnung folgen. Wird mehr als ein Parameter angegeben, erfolgt die Trennung durch Leerzeichen. Enthält ein einzelner Parameter ein oder mehrere Leerzeichen, so wird dieser Parameter in Hochkommata notiert.

Auch für die Parameter sind Ersetzungen beim Umwandlungsprozess der Skripte vorgesehen. Dabei wird zwischen zwei Arten der Ersetzung unterschieden: der Einzelwertersetzung und den Geschwindigkeitsangaben.

Bei Ersetzungen, die auf der ersten Art basieren, wird lediglich ein Austausch der Zeichenkette des Parameters durch eine andere Zeichenkette vorgenommen. So kann der Parameter `red` durch den RGB-Wert `255,0,0` ersetzt werden. Die Definition dieser Ersetzungsvorschriften erfolgt in der XML-Struktur für die Befehlsersetzungen. Die Ersetzung ist befehlsabhängig und wird in der Struktur als zusätzliches Element der Befehlsersetzung angegeben (siehe Listing 4.7).

```
<command commandStr="setColor">
        ...
  <parameter paramStr="red" singleValue="255,0,0" />
  <parameter paramStr="yellow" singleValue="255,255,0" />
  <parameter paramStr="blue" singleValue="0,0,255" />
  <parameter paramStr="green" singleValue="0,255,0" />
</command>
```

Listing 4.7: Beispiel der Definition einer einfachen Parameterersetzung in XML

Handelt es sich bei der Parameterersetzung um eine Einzelwertersetzung, so wird der zu ersetzende Wert mit `singleValue` angegeben. Eine zweite Art der Parameterersetzung bedingt eine zusätzliche Berechnung. Bei Anweisungen, die einen absoluten Zahlenwert als Parameter erfordern (z.B. die Rotation um eine bestimmte Gradzahl), kann als Parameter auch ein Geschwindigkeitswert zum Einsatz kommen (z.B. `slow`). Die Notation in der Ersetzungsvorschrift erfolgt dann als `valuePerSecond`, also als Absolutwert pro Sekunde (siehe Listing 4.8).

Bei der Umwandlung eines *HighLevel*-Skriptes in ein *LowLevel*-Skript werden bei der Parameterersetzung mit Geschwindigkeitsparametern zusätzliche Berechnungen vorgenommen. Hier gilt es zwischen zwei Fällen der Berechnung zu unterscheiden: der Absolutwertberechnung und der Endzeitberechnung. Erfolgt die Angabe eines Geschwindigkeitsparameters ohne weiteren Zusatz(z.B. `'slow'`), so wird mithilfe des Zeitbereiches der Anweisung und der Zeitdaten des Initialisierungsteils des Skriptes ein Absolutwert berechnet, der den Parameter ersetzt.

Erfolgt zusätzlich zur Geschwindigkeits- noch eine Absolutwertangabe (z.B. `'slow 180'`, was bei einer Rotation die langsame Drehung um 180 Grad bedeutet), so ist in Abhängigkeit des `valuePerSecond` des Parameters die Endzeit der Anweisung variabel. Diese wird neu berechnet und gesetzt. Der Parameter wird durch den angegebenen Absolutwert ersetzt.

Die Neuberechnung der Endzeit hat den Nachteil, dass bei der Erstellung des *HighLevel*-Skriptes dem Autor die Endzeit einer Anweisung nicht bekannt ist. So kann er sie nicht mit nachfolgenden Anweisungen synchronisieren. Dies wäre mit einem ereignisbasierten Ansatz, wie er in Abschnitt 3.6 im Zusammenhang mit der Software Amira vorgestellt wurde, besser möglich.

Bei der Geschwindigkeitsangabe wurde als Zeiteinheit eine Sekunde gewählt und nicht die sonst verwendete abstrakte Zeiteinheit. Das bringt Vor- und Nachteile mit sich. Ein Vorteil besteht darin, dass die Geschwindigkeitsangabe unabhängig von der am Ende tatsächlichen Geschwindigkeit der Animation ist. Unabhängig von den Zeiteinstellungen im Initialisierungsteil und der Framerate wird zum Beispiel eine Rotation mit einer Geschwindigkeitsangabe `slow` mit einem `valuePerSecond` von 5 (Grad pro Sekunde) in der fertigen Animation immer mit 5 Grad pro Sekunde erfolgen. Aus Sicht des Autors kann aber genau dieser Vorteil auch zum Nachteil

gereichen, wenn der Autor eine Geschwindigkeitsänderung für sämtliche Anweisungen einer Animation, also auch die mit Geschwindigkeitsparametern, erreichen möchte.

```
<command commandStr="rotate">
        ...
  <parameter paramStr="slow" valuePerTimeUnit="10" />
</command>
```

Listing 4.8: Beispiel der Definition eines Geschwindigkeitsparameters in XML

4.2.3 Grammatik der Sprache

Der Skriptsprache liegt eine kontextfreie Grammatik G zu Grunde. Die Grammatik beschreibt die Syntax der einzelnen Anweisungen und beinhaltet nicht die Definition der Ersetzungsvorschriften. Im Folgenden ist die Grammatik der Skriptsprache kurz mit der Regelmenge P in Backus-Naur-Form (BNF) dargestellt:

$G = (V, \Sigma, P, S)$
$V = \{ANWEISUNG, ZEIT, OBJEKT, BEFEHL, PARAMETER, ZAHL\}$
$\Sigma = \{string, float, integer\}$
$S = ANWEISUNG$

Die Regelmenge P:

$$
\begin{array}{lll}
ANWEISUNG & \rightarrow & ZEIT\ OBJEKT\ BEFEHL\ PARAMETER \\
ZEIT & \rightarrow & [ZAHL] \mid [ZAHL, ZAHL] \\
OBJEKT & \rightarrow & 'string' \\
BEFEHL & \rightarrow & string \\
PARAMETER & \rightarrow & PARAMETER\ PARAMETER \mid VEKTOR \\
 & & \mid ZAHL \mid string \mid \epsilon \\
VEKTOR & \rightarrow & ZAHL, ZAHL, ZAHL \\
ZAHL & \rightarrow & float \mid integer
\end{array}
$$

Die Regelmenge P beschreibt die Syntax der zulässigen Anweisungen. Wie im vorigen Abschnitt beschrieben wird eine *ANWEISUNG* dabei zu *ZEIT OBJEKT BEFEHL* und *PARAMETER* abgeleitet. Ein *PARAMETER* kann dabei aus mehreren Parametern bestehen, einen Vektor darstellen, eine Zahl oder ein String sein.

4.2.4 Frameliste

Durch die Art der sich zeitlich überschneidenden Anweisungen ist das serielle Abarbeiten eines *LowLevel*-Skriptes nicht direkt möglich. Daher muss es vor der endgültigen Animationsgenerierung in eine bildweise Repräsentation überführt

werden. Wie schon oben angedeutet wird als eine solche Repräsentation eine Frameliste erzeugt.

Eine Frameliste ist eine hierarchische Liste. Jedem Frame der Animation ist dabei eine Liste von Anweisungsobjekten zugeordnet. Ein einzelnes Anweisungsobjekt beinhaltet unter anderem Informationen zum ausführenden Befehl, dem zugeordneten Objekt und den Parametern. Bei der Umwandlung eines *LowLevel*-Skriptes in eine Frameliste wird jede Anweisung den Frames der Liste zugeordnet, über die sie sich laut ihrer Zeitangaben erstreckt.

4.3 Betrachtung ausgewählter Anweisungen

Im Folgenden werden einige Anweisungen der Skriptsprache vorgestellt. Dabei wird es zunächst um die Bewegung der Kamera, die Rotation um ein Objekt und die Veränderung von Objekteigenschaften gehen. Abschließend werden Ideen und Vorschläge für abstrakte *HighLevel*-Anweisungen vorgestellt.

4.3.1 Manipulation der Kamera

Koordinatenfreie Ortsangaben

Sind Anweisungen mit Ortsbezug, wie die Bewegung einer Kamera, auszuführen, sind Positionsangaben unerlässlich. Eine Möglichkeit dafür stellt die Angabe von dreidimensionalen Koordinaten dar. Dies ist jedoch mit erheblichen Nachteilen verbunden. Mit dem Einsatz fester Koordinaten würde man die Nutzbarkeit eines Skriptes auf einen bestimmten Datensatz beschränken und es unmöglich machen, dieses Skript auch auf andere Datensätze anzuwenden. Desweiteren sind Nutzer der Skriptsprache in den meisten Fällen mit der Koordinatenangabe und der Orientierung in dreidimensionalen Koordinatensystem nicht vertraut und überfordert. Zwar können sich Autoren die Position, an der sich beispielsweise eine Kamera befinden soll, mental vorstellen. Jedoch sind die Koordinaten einer Position dem Autor nur selten bekannt.

Aus diesen Gründen wurde ein koordinatenfreier Ansatz zur Positionsangabe in dem vorliegenden Skriptkonzept entwickelt. Die Angabe von Positionen erfolgt dabei über die einzelnen Objekte und in Relation zu ihnen. Als Koordinate eines Objektes gilt dabei das Zentrum der *bounding box*. Die Umwandlung von Objektbezeichnungen in Koordinaten erfolgt erst in der Skriptschnittstelle direkt bei der Animationsgenerierung.

Achsen zur Rotation werden auf *LowLevel*-Ebene dennoch als Vektoren angegeben. Durch Parameterersetzungen können Achsen aber auf einfache Weise durch textuelle Bezeichnungen ersetzt werden.

Beschleunigen und Abbremsen der Kamera

Im Normalfall werden die Parameter einer Anweisung im angegebenen Zeitbereich linear interpoliert. Dies hat bei Bewegungen der Kamera, seien es Rotationen oder lineare Bewegungen, ein unnatürliches Verhalten zur Folge: Die Geschwindigkeit der Kamera ist vom ersten bis zum letzten Augenblick konstant. Physikalisch unterliegt die Kamera aber am Beginn und am Ende einer Bewegung einer Beschleunigung. Dieses Verhalten lässt sich mithilfe von Bézier-Kurven simulieren. Eine solche Kurve kann dabei als Kennlinie des Weges über der Zeit fungieren (siehe Abbildung 4.4). Eine genauere Betrachtung von Bézier-Kurven speziell unter dem Aspekt der Kameraparameter findet sich in [Helbing 2004]. Das beschleunigte Verhalten der Kamera kann bei Anweisungen zur Bewegung oder Rotation der Kamera global ein- und ausgeschaltet werden.

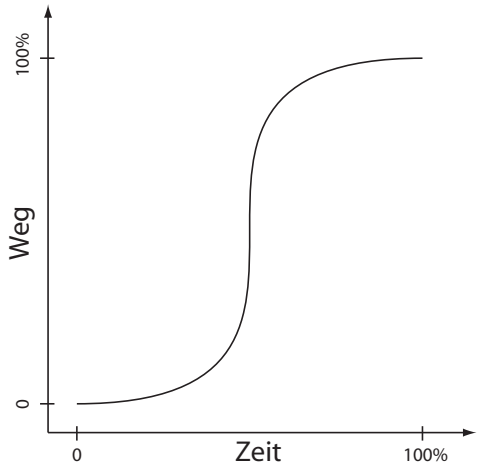


Abbildung 4.4: Darstellung des zurückgelegten Weges der Kamera über dem Zeitbereich einer Anweisung. Zu erkennen ist das beschleunigte und abgebremste Verhalten der Kamera.

Bewegung der Kamera

Die `move`-Anweisung positioniert die Kamera. Die `move`-Anweisung hat vier Parameter: Objektbezeichnung, zwei Rotationswinkel und einen Skalierungsfaktor (siehe Listing 4.9). Als Objektbezeichnung ist die festgelegte Objektbezeichnung `'Cam'` vorgeschrieben. Der erste Parameter bezeichnet das Objekt, auf welches Bezug genommen wird. Es können Objektgruppen, wie bei den Objektbezeichnungen für Anweisungen üblich (siehe Abschnitt 4.2.2), verwendet werden. Die Ersetzung der Objektbezeichnung erfolgt erst bei der unmittelbaren Ausführung der Anweisung.

Die Positionierung erfolgt durch die Angabe von zwei Winkeln und eines Skalierungsfaktors für den Abstand. Der normale Abstand (Skalierungsfaktor = 1) entspricht einer Darstellung des Objektes, sodass es in seiner kompletten Ausdehnung sichtbar ist. Als Maß für die Größe eines Objektes wird die größte Ausdehnung der

```
Zeit 'Cam' move Objekt Winkel_sagittal Winkel_axial [Skalierung]

Beispiele 1 und 2:
[0,10] 'Cam' move 'Liver' 0 0
[0,10] 'Cam' move 'Liver' 90 -90 1
```

Listing 4.9: Parameter der `move`-Anweisung. Die Beispiele zeigen die zwei Anweisungen zur Positionierung der Kameras, wie sie in Abbildung 4.5 zu sehen sind.

bounding box genutzt. Mit Blick auf eine spätere Implementierung, in der eine orthogonale Kamera zur Anwendung kommt, entspricht bei normalem Abstand die Höhe des *ViewVolumes*[1] der größten *bounding-box*-Ausdehnung. Die Entfernung zum Objektmittelpunkt ist gleich dem Radius einer gedachten Kugel, die die *bounding box* und damit das Objekt exakt umschließt. Bei einer Skalierung findet eine Vergrößerung bzw. Verkleinerung der Höhe des *ViewVolumes* statt. Die Angabe des Skalierungsfaktors ist optional.

Die beiden Winkel entsprechen der Rotation der Kamera in der sagittalen bzw. axialen Eben um die gegebene Gradzahl. Dabei wird zuerst die Rotation um die sagittale und daraufhin um die axiale Achse durchgeführt (siehe Abbildung 4.5). Damit lässt sich jede Position auf einer Kugel beschreiben, die die *bounding box* des Objektes umschließt. Der Radius der Kugel entspricht dem skalierbaren Abstand der Kamera vom Objektmittelpunkt.

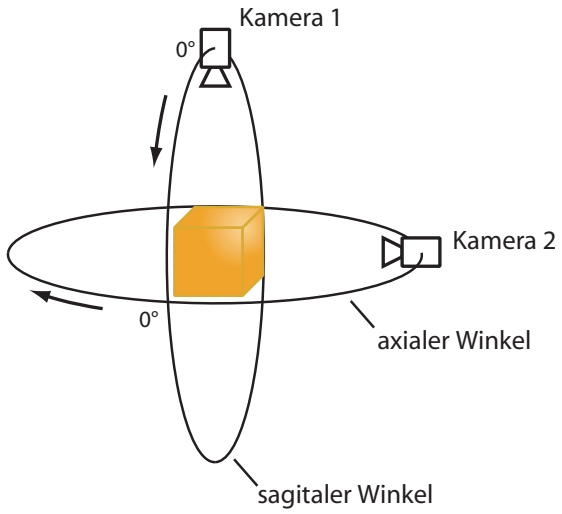


Abbildung 4.5: Darstellung der Rotationswinkel, wie sie zur Positionierung der Kamera mithilfe der `move`-Anweisung benutzt werden. Dabei wird zuerst um den sagittalen und anschließend um den axialen Winkel rotiert. Die beiden Anweisungen zur Positionierung von *Kamera 1* und *2* finden sich in im Listing 4.9.

[1] siehe dazu [Angel 2000]

Rotation der Kamera

Die Rotation der Kamera umfasst auf der *LowLevel*-Ebene mehrere Anweisungen, von denen einige hier vorgestellt werden sollen. Die zentrale Anweisung zur Rotation ist die `rotate`-Anweisung. Die Rotation wird dabei wie auch bei der `move`-Anweisung in Relation zu einem Objekt bzw. einer Gruppe von Objekten ausgeführt. Als Bezugspunkt der Rotation gilt ebenfalls der Mittelpunkt der *bounding box*.

Die `rotate`-Anweisung wird über drei Parameter gesteuert (siehe Listing 4.10). Mit dem ersten Parameter wird das Objekt bezeichnet, um das die Rotation erfolgen soll. Hier gelten die gleichen Konventionen wie für die `move`-Anweisung. Als zweiter Parameter folgt ein Achsenvektor, der zusammen mit dem Mittelpunkt der *bounding box* die Achse bildet, um die die Kamera rotiert wird. Als dritter Parameter wird der zu rotierende Winkel angegeben. Während der Rotation wird die Blickrichtung der Kamera in der Art angepasst, dass diese immer auf das Objekt (den Mittelpunkt von dessen *bounding box*) ausgerichtet ist, um das sie rotiert.

```
Zeit 'Cam' rotate Objekt  Achse  Winkel

Beispiel:
[0,10] 'Cam' rotate 'Liver' 0,0,-1 180
```

Listing 4.10: Parameter der `rotate`-Anweisung. Das Beispiel rotiert die Kamera axial (um die negative Z-Achse) um 180° um die Leber.

Weitere Rotationsanweisungen sind `rotateX`, `rotateY`, `rotateZ`, `rotateVirtualX`, `rotateVirtualY`, `rotateVirtualZ` und `rotateCamX`, `rotateCamY`, `rotateCamZ`. Die Anweisungen `rotateX`, `rotateY` und `rotateZ` rotieren die Kamera um die globalen Achsen des Weltkoordinatensystems. Mit `rotateVirtualX`, `rotateVirtualY` und `rotateVirtualZ` wird die Kamera um gedachte Achsen eines Koordinatensystems rotiert, dessen Ursprung im momentanen lookAt-Punkt[2] liegt. Die Z-Achse dieses Koordinatensystems ist aus der aktuellen Sicht heraus gerichtet, die Y-Achse nach oben und die X-Achse nach rechts. Mit den Befehlen `rotateCamX`, `rotateCamY` und `rotateCamZ` wird die Kamera um Achsen rotiert, die ihren Ursprung im Zentrum der Kamera haben. Als Parameter für alle drei Gruppen von Rotationsanweisungen wird der zu rotierende Winkel in Grad angegeben.

4.3.2 Veränderung von Objekteigenschaften

Im Rahmen dieses Konzeptes lassen sich viele Anweisungen in die Gruppe der direkten Veränderung von Objekteigenschaften einordnen. Allen diesen Anweisungen ist

[2] Der lookAt-Punkt ist der Punkt, auf den sich die Blickrichtung der Kamera ausrichtet. Er kann über die Anweisung `setLookAt` gesondert gesetzt werden.

gemein, dass sie einen Parameter eines bestimmten Objektes direkt verändern. Damit lassen sich alle einfachen Parameter manipulieren, die von der Skriptschnittstelle angeboten werden. Als einfache Parameter werden Zahlenwerte, Vektoren oder Zeichenketten verstanden. Die Anweisungen haben sämtlichst nur einen Parameter. Einige sollen im Folgenden exemplarisch vorgestellt werden.

Transparenz

Die Anweisung `setTransparency` ändert die Transparenz eines Objektes (siehe Listing 4.11). Als Parameter wird eine Fließkommazahl zwischen 0 und 1 angegeben, wobei 1 komplett transparent und 0 vollständig opak entspricht.

```
Zeit Objektbezeichnung  setTransparency Transparenzwert

Beispiel:
[0,10] 'Heart' setTransparency 0.5
```

Listing 4.11: Parameter der `setTransparency`-Anweisung. Das Beispiel setzt die Transparenz des Herzens auf 50%.

Farben

Die Anweisung `setColor` setzt die Farbe eines Objektes. Als Parameter wird der Anweisung ein dreielementiger Vektor übergeben (siehe Listing 4.12). Er repräsentiert die jeweiligen Anteile von rot, grün und blau in einem Wertebereich zwischen 0 und 255. Die drei Werte sind durch Kommata getrennt.

```
Zeit Objektbezeichnung  setColor RGB-Farbwert

Beispiel:
[0,10] 'Liver' setColor 242,101,34
```

Listing 4.12: Parameter der `setColor`-Anweisung. Das Beispiel setzt die Farbe der Leber auf orange.

Sichtbarkeit

Mithilfe von `setVisible` kann ein Objekt angezeigt oder ausgeblendet werden. Als Parameter wird entweder `true` oder `false` angegeben (siehe Listing 4.13). Da sich

diese Eigenschaft im Unterschied zu den numerische Werten nicht interpolieren lässt, wird der Wert sofort im ersten Frame der Anweisung gesetzt. Dies gilt für alle Anweisungen, die Änderungen von Objekteigenschaften mit Zeichenketten als Parameter vornehmen.

```
Zeit Objektbezeichnung  setVisible true/false

Beispiel:
[0,10] 'Liver' setVisible true
```

Listing 4.13: Parameter der `setVisible`-Anweisung. Das Beispiel setzt die Sichtbarkeitseigenschaft der Leber auf sichtbar.

4.3.3 Abstrakte Anweisungen

Ein großer Vorteil des in dieser Arbeit entwickelten Skriptkonzeptes ist die Möglichkeit abstrakte Anweisungen auf Basis der atomaren Anweisungen definieren zu können. Da diese Ersetzungsvorschriften vom Programm unabhängig in XML-Dateien abgelegt sind, hat sowohl der Entwickler wie auch der Autor die Möglichkeit, neue Anweisungen zu erschaffen oder bestehende zu verändern. Einige dieser abstrakten Anweisungen, die bereits umgesetzt wurden und dem Autor einer Animation zur Verfügung stehen, sollen nachstehend vorgestellt werden.

Die `view`-Anweisung

Die `view`-Anweisung zeigt ein Objekt aus einer gewünschten Richtung (siehe Abbildung 4.6). Diese wird zusammen mit einem Skalierungsfaktor als Parameter übergeben. Im Listing 4.14 ist die Umwandlung zu erkennen. Dabei wird zunächst das Objekt sichtbar gemacht[3] und dann die Kamera an die gewünschte Position im Bezug auf das Objekt bewegt. Die in Listing 4.14 gezeigte Ersetzung entspricht noch nicht der kompletten Umwandlung der `view`-Anweisung in *LowLevel*-Anweisungen, da die `on`-Anweisung selbst eine abstrakte Anweisung ist, deren Umwandlung weiterer Ersetzungsvorgänge bedarf.

Die `emphasize`-Anweisung

Die `emphasize`-Anweisung hebt ein Objekt hervor (siehe Abbildung 4.7). Sie stellt eine Erweiterung der `view`-Anweisung in der Hinsicht dar, dass sie zusätzlich zu einem

[3] Dies bedeutet nicht, dass eine Überprüfung hinsichtlich störender anderer Objekte vorgenommen wird. Es wird hier lediglich die grundsätzliche Sichtbarkeitseigenschaft (`setVisible true`) gesetzt (vgl. dazu auch Abschnitt 6.2 im Ausblick).

Ausgangsanweisung:

```
[0,10] 'Lung' view sagittal 1
```

Ergebnis nach der Ersetzung:

```
[0,10] 'Lung' on
[0,10] 'Cam' move 'Lung' sagittal 1
```

Ersetzungsvorschrift in XML:

```
<command commandStr ="view">
  <command>T 0 on</command>
  <command>T 'Cam' move 0 P</command>
</command>
```

Listing 4.14: Die `view`-Anweisung - Ersetzungsvorschrift mit Beispiel

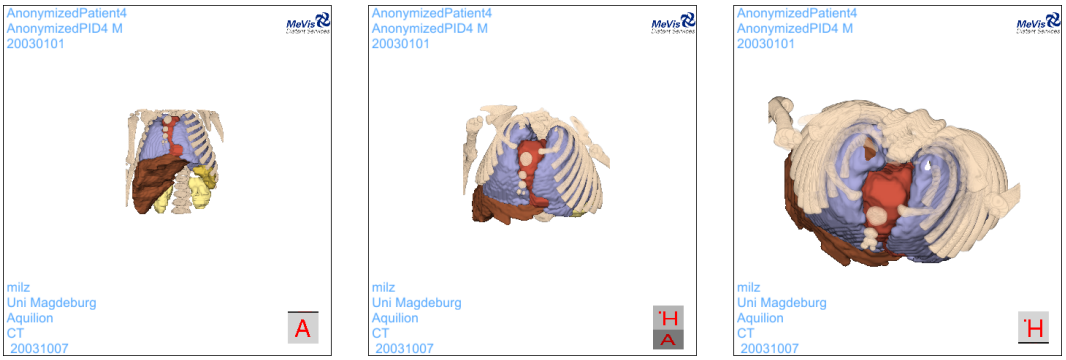


Abbildung 4.6: Drei Auschnitte aus einer Animation, in welcher die `view`-Anweisung für die Lunge angewendet wurde

Sichtbarmachen des Objektes und einer Kamerabewegung auch noch die Farbe und die Transparenz des Objektes verändert. Das Objekt wird rot gefärbt und fast opak dargestellt. In Listing 4.15 ist wiederum die Ersetzungsvorschrift mit einem Beispiel zu sehen.

Ähnlich aufgebaut ist die `deemphasize`-Anweisung, bei der lediglich Farbe und Transparenz auf Standardwerte gesetzt werden und so die Hervorhebung des Objektes aufgelöst wird.

4.4 Zusammenfassung des Skriptkonzeptes

Das entwickelte Skriptkonzept basiert auf einem mehrstufigen Ansatz. Dabei wird ein HighLevel-Skript mit abstrakten Anweisungen in ein LowLevel-Skript mit atomaren Anweisungen umgewandelt. Die Umwandlung erfolgt anhand von Ersetzungsvorschriften für Objektbezeichnungen, Befehle und Parameter. Die Ersetzungsvorschriften werden extern in einer XML-Struktur definiert. Dies ermöglicht es Nutzern

Ausgangsanweisung:

```
[0,10] 'Lung' emphasize
```

Ergebnis nach der Ersetzung:

```
[0,10] 'Lung' view front
[0,10] 'Lung' setColor red
[0,10] 'Lung' setTransparency low
```

Ersetzungsvorschrift in XML:

```
<command commandStr ="emphasize">
  <command>T 0 view front</command>
  <command>T 0 setColor red</command>
  <command>T 0 setTransparency low</command>
</command>
```

Listing 4.15: Die `emphasize`-Anweisung - Ersetzungsvorschrift mit Beispiel einer Ersetzung.

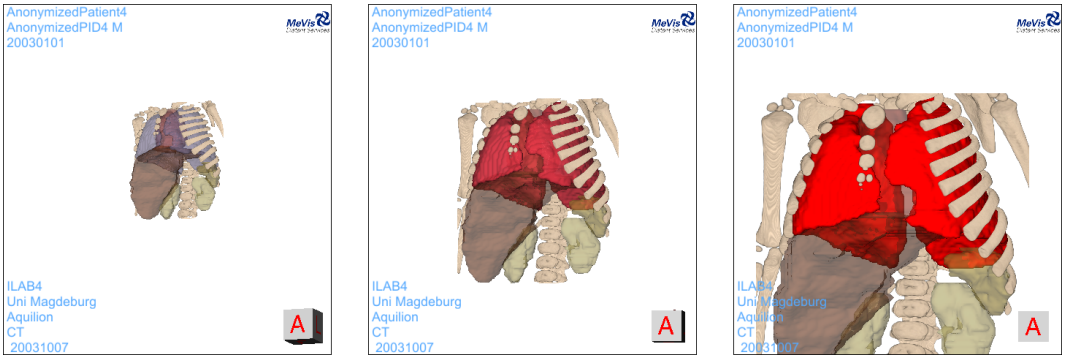


Abbildung 4.7: Drei Ausschnitte aus einer Animation, in welcher die `emphasize`-Anweisung für die Lunge angewendet wurde

der Skriptsprache eigene abstrakte Anweisungen zu definieren und bestehende Anweisungen zu modifizieren. Mithilfe eines beschränkten Befehlssatzes an atomaren *LowLevel*-Befehlen lässt sich so ein mächtiger Anweisungsumfang der Skriptsprache erreichen.

Der parallele Einsatz von abstrakten und atomaren Anweisungen erlaubt es sowohl unerfahrenen Nutzern wie Experten, mit der Skriptsprache Animationen nach ihren Vorstellungen zu beschreiben.

Anweisungen können zu einem bestimmten Zeitpunkt oder über einen Zeitraum ausgeführt und so animiert dargestellt werden. Als Zeiteinheit wurden abstrakte Zeiteinheiten gewählt.

Die Angabe von Positionen oder Blickrichtungen erfolgt koordinatenfrei und in Relation zu Objekten. Dies macht die Skripte adaptiv und für unterschiedliche Datensätze wiederverwendbar.

Die Anwendung der Skripte ist nicht auf den medizinischen Bereich beschränkt. Der Ansatz ermöglicht die Nutzung für die Animationsgenerierung in allen objektbasierten grafischen Darstellungen. Es wird dabei lediglich eine Schnittstelle zur Umsetzung der atomaren *LowLevel*-Anweisungen in eine Visualisierung vorausgesetzt.

Kapitel 5

Implementierung

Das im vorigen Kapitel vorgestellte Skriptkonzept wurde im Rahmen dieser Arbeit prototypisch implementiert. Die Grundlage dafür bot die Software MeVisLab des Centrums für Medizinische Diagnosesysteme und Visualisierung (MeVis) in Bremen. MeVisLab ist eine Forschungs- und Entwicklungsplattform mit einer Spezialisierung im Bereich der medizinischen Bildverarbeitung und Visualisierung. Im Vergleich zu anderen Plattformen wie Amira[1], die ebenfalls zur Disposition standen, bietet MeVisLab zwei große Vorteile:

1. MeVisLab ist auf die Verarbeitung, Analyse und Visualisierung medizinischer Daten spezialisiert.
2. MeVis bietet externen Entwicklern die Möglichkeit, eigene Erweiterungen, sogenannte Module, für MeVisLab zu entwickeln. Dazu wurde im Rahmen einer Forschungskooperation der Otto-von-Guericke-Universität Magdeburg eine spezielle Entwicklerversion bereitgestellt.

In diesem Kapitel werden einige Aspekte der Implementierung vorgestellt sowie wesentliche, im Rahmen dieser Arbeit entwickelte Module beschrieben.

5.1 Entwicklungsumgebung MeVisLab

5.1.1 Grundlagen

Die Grundlage der Entwicklung von Anwendungen in MeVisLab bildet ein Konzept der grafischen Programmierung. Die Grundbausteine sind dabei Module, die zu Netzwerken verknüpft werden können. Als Verknüpfung sind dabei zwei Möglichkeiten vorgesehen. So kann jedes Modul über mehrere Ein- und Ausgänge verfügen. Diese können miteinander verknüpft werden und ermöglichen so den Austausch von Bildern oder anderen Datenstrukturen. Desweiteren kann jedes Modul über unterschiedliche Parameter (Felder) verfügen. Diese repräsentieren einfache Datentypen (wie Zeichenketten oder Zahlen) und dienen neben der Abfrage von Informationen zu den Modulen auch dem Datenaustausch zwischen diesen. Dazu

[1] http://www.amiravis.com

können Felder verschiedener Module über Feldverbindungen miteinander verknüpft werden.

Bei den Modulen werden drei Arten unterschieden:

- OpenInventor-Module
- ML[2]-Module
- Makromodule

ML-Module dienen primär der Berechnung von Bilddaten und werden daher auch oft als Bildverarbeitungsmodule bezeichnet. Ein einfaches Beispiel ist das Modul `Threshold`, welches als Eingabe ein Grauwertbild erhält und anhand eines Schwellwertes eine binäre Maske berechnet. ML-Module können neben der Bildberechnung aber auch andere Aufgaben übernehmen, zum Beispiel die Ausführung von Skriptanweisungen. OpenInventor-Module übernehmen in MeVisLab-Netzwerken die Funktionen der Visualisierung und Interaktion mit dem Nutzer. Sie basieren auf dem OpenSource-Projekt der Grafikbibliothek OpenInventor[3]. Genau wie ML-Module werden OpenInventor-Module in C++ programmiert. OpenInventor-Module können mit ML-Modulen verknüpft und so Applikationen[4] geschaffen werden.

Makromodule fassen in MeVisLab Netzwerke zusammen. So kann ein einzelnes Makromodul eine komplette Anwendung repräsentieren. Mithilfe der MeVisLab eigenen *Module Description Language* (MDL) können sowohl die GUI (grafische Benutzeroberfläche) als auch die Verbindungen des gekapselten Netzwerkes nach außen hin definiert und beschrieben werden. Zusätzlich kann mit der Skriptsprache JavaScript das dynamische Verhalten der GUI beschrieben werden.

Neben den ca. 400 bisher vorhandenen Modulen kommen in MeVisLab unterschiedliche zusätzliche Bibliotheken zum Einsatz:

1. Die MeVis Image Library[5] zum Einlesen verschiedener (medizinischer) Bildformate
2. LaPack[6] für mathematische Berechnungen
3. Xerces[7] für die Verarbeitung von XML-Daten
4. Der GigaVoxelRenderer[8] zur Erzeugung von Volumenvisualisierungen

Hervorzuheben ist die Bibliothek Xerces, da sie im Rahmen dieser Arbeit für die Auswertung der XML-Ersetzungsvorschriften zum Einsatz kam. Die freie Verfügbarkeit der Bibliothek ermöglicht zusätzlich eine externe Nutzung der Skriptumwandlung.

[2] MeVis Image Processing Library
[3] http://oss.sgi.com/projects/inventor
[4] Als Applikation wird dabei kein eigenständiges Programm verstanden, sondern ein in MeVisLab laufendes Teilprogramm.
[5] http://www.mevis.de
[6] http://www.netlib.org/lapack
[7] http://xml.apache.org/xerces-c
[8] http://www.mevis.de

In Abbildung 5.1 ist die Grundsturktur von MeVisLab schematisch zusammengefasst.

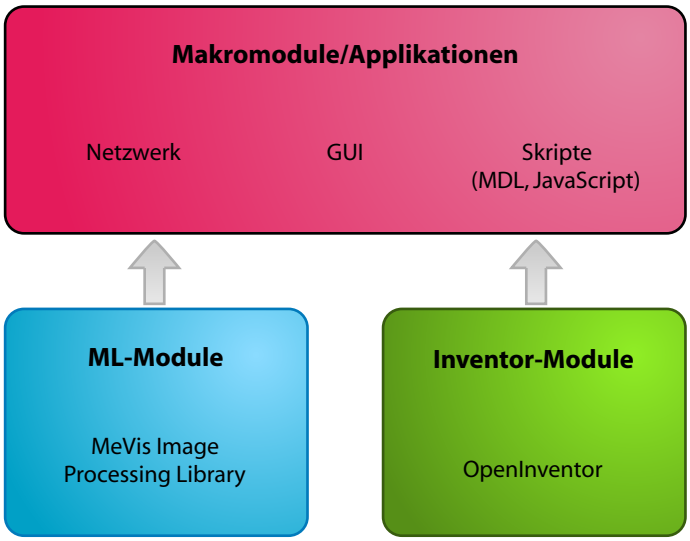


Abbildung 5.1: Schematische Darstellung des Aufbaus von MeVisLab

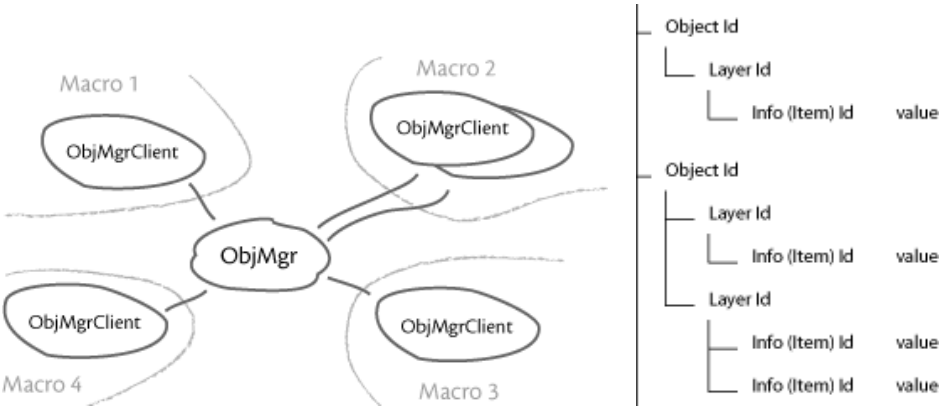
5.1.2 Das Konzept des ObjectManagers

Zur Verwaltung der Daten und Eigenschaften von Objekten aus segmentierten Datensätzen wurde in MeVisLab das Konzept des ObjectManagers entwickelt. Der ObjectManager repräsentiert dabei eine hierarchische Datenstruktur, in der alle Informationen zu den Objekten gespeichert werden. Über ObjectManager-Clients (*Clients* genannt) können Makros oder Applikationen in MeVisLab auf diese Daten zugreifen und sie manipulieren. Verändert ein *Client* Daten in der Datenstruktur, so wird eine Nachricht an alle anderen *Clients* verschickt, die mit dem ObjectManager verbunden sind (siehe Abbildung 5.2(a)). Dies hat den großen Vorteil, dass diese Clients sofort auf mögliche Änderungen reagieren können.

Die hierarchische Struktur des ObjectManagers gliedert sich in drei Ebenen: die `ObjectID`, die `LayerID` und die `InfoID` (siehe Abbildung 5.2(b)). Bei der `ObjectID` handelt es sich um eine eindeutige Bezeichnung für jedes Objekt in der Datenstruktur. Die `LayerID` bezeichnet eine Gruppe von logisch zusammengehörenden Werten und die `InfoID` bezeichnet jeweils ein einzelnes Datenfeld. Alle Daten werden als Zeichenkette abgelegt.

5.2 Applikation des AnimationScripters

Als Anknüpfungspunkt für eine Skriptschnittstelle innerhalb von MeVisLab wurde der InterventionPlanner [Preim u. a. 2003] gewählt. Der InterventionPlanner stellt eine eigenständige Applikation in MeVisLab dar. Er wurde speziell



(a) (b)

Abbildung 5.2: Schematische Darstellungen des Konzeptes des ObjectManagers mit verschiedenen *Clients* (a) und der hierarchischen Datenstruktur mit den Ebenen der `ObjectID`, `LayerID` und `InfoID` (b) (aus [Ritter 2005])

als Softwareassistent für die Leberoperationsplanung entwickelt und verfügt über einen großen Umfang an Funktionen. Interessant sind die Funktionen vor allem unter dem Gesichtspunkt der Anbindung einer Skriptschnittstelle, da sich der InterventionPlanner gerade wegen des Reichtums an Funktionen anbietet.

5.2.1 Funktionen des InterventionPlanners

Den Kern des InterventionPlanners bildet ein 3D-Viewer, um dessen Manipulationsmöglichkeiten eine Reihe weiterer Funktionen gruppiert wurden. Dieser Viewer wird durch das Inventor-Modul `SoExaminerViewer` repräsentiert [Wernecke 1994]. Zu den Grundfunktionen des `SoExaminerViewers` zählen die Manipulation der Kamera mithilfe der Maus in der Darstellung. Der `SoExaminerViewer` bietet zudem die Möglichkeit einzelne Frames zwischenzuspeichern und daraus eine fertige Animation in Form eines Videos zu generieren.

Für die Auswahl der Objekte steht dem Nutzer eine tabellarische Übersicht zur Verfügung, in welcher die Objekte entsprechend ihrer Strukturzugehörigkeit, zum Beispiel Gefässe oder Tumore, ausgewählt sowie ein- oder ausgeblendet werden können. Zu jedem Objekt können grundlegende Parameter über Auswahllisten, Schieberegler oder Eingabefelder angegeben und verändert werden. Dazu zählen Farbe, Transparenz, Darstellungsqualität und Darstellungsstil von Objekten. Die Objekte werden jeweils in der Oberflächendarstellung visualisiert. Zusätzlich kann ein Volumerendering hinzugeschaltet werden. Die Parameter des Volumenrendering können numerisch eingegeben werden.

Weiterhin bietet der InterventionPlanner dem Nutzer die Möglichkeit die Lagebeziehungen einzelner Objekte, zum Beispiel eines krankhaften Objekt zu

umliegenden vitalen Strukturen, genau zu bestimmen. Dazu stehen dem Nutzer Werkzeuge zur Vermessung von Abständen und Winkeln zur Verfügung. Die Volumina segmentierter Objekte werden automatisch berechnet.

Desweiteren können im InterventionPlanner 2D- und 3D-Ansichten segmentierter Daten synchron betrachtet werden. Der Nutzer hat die Möglichkeit die Visualisierung eines Datensatzes sowohl in der 3D-Ansicht wie auch in der Schichtbilddarstellung zu betrachten. Veränderungen an Objekten in den Darstellungen sind dabei immer in beiden Ansichten sofort zu erkennen. So wird die Farbe eines in der 3D-Ansicht selektierten Objektes in der 2D-Ansicht als farbige Fläche im Schichtbild eingeblendet. Ebenso wird in der Schichtbildansicht automatisch die zu einem selektierten Objekt gehörende Schicht[9] angezeigt.

Der InterventionPlanner bietet außerdem die Möglichkeit Clip-Ebenen, wie sie in Abschnitt 2.7.4 beschrieben wurden, durch die dreidimensionale Darstellung zu legen. Diese verlaufen achsenparallel und können nur interaktiv, in der Szene verschoben, und nicht numerisch positioniert werden.

Zusammenfassend lässt sich sagen, dass der InterventionPlanner ein umfassendes Werkzeug zur interaktiven Betrachtung segmentierter Datensätze ist. Allerdings weist er das Manko einer fehlenden Reproduzierbarkeit der Darstellungen auf. Dies liegt in der fehlenden Möglichkeit begründet, entscheidende Parameter, wie die der Kamera, nicht numerisch, sondern nur interaktiv mit der Maus eingeben zu können. Der InterventionPlanner bietet aber dennoch aufgrund seiner hohen Flexibilität, was die Veränderung von Parametern der Szene und der Objekte betrifft, eine gute Grundlage zur Implementierung des in dieser Arbeit entwickelten Skriptkonzeptes.

5.2.2 Beschreibung des AnimationScripters

Der AnimationScripter ist ein um die Skriptschnittstelle erweiterter InterventionPlanner. Abbildung A.1 im Anhang zeigt die Oberfläche des AnimationScripters, die in großen Teilen der des InterventionPlanners ähnelt. Für den Nutzer werden die Erweiterungen durch zwei neue Funktionen sichtbar:

- Es können Skripte eingelesen werden, aus denen direkt Animationen generiert werden.
- Durch die Eingabe von Skriptanweisungen in einer Kommandozeile können Einzelbilder erzeugt werden.

In den beiden folgenden zwei Abschnitten werden diese Erweiterungen kurz eingeführt, bevor in den darauf folgenden Abschnitten die Struktur und Funktionsweise der Skriptschnittstelle skizziert werden.

Im Idealfall würde der AnimationScripter in seiner Funktion als Skriptschnittstelle sämtliche für den Nutzer im InterventionPlanner verfügbaren Funktionen

[9] Im Standardfall ist dies die mittlere Schicht aller Schichten, die das Objekt beinhalten.

auch als Skriptanweisungen zur Verfügung stellen. Dies ist im Fall des AnimationScripters nicht möglich. Dies liegt zum Einen im dem zeitlich begrenzten Rahmen dieser Diplomarbeit begründet. Zum Anderen ist es technisch zur Zeit noch nicht möglich, bestimmte Funktionen des InterventionPlanners in der Skriptschnittstelle umzusetzen, da diese Funktionen im InterventionPlanner selbst noch experimentell verankert und für externe Entwickler nicht verfügbar sind. Dazu zählen die Nutzung von *clipping planes* und das Volumenrendering.

Generierung von Videos

Bei der Animationsgenerierung kann der Autor wahlweise ein *HighLevel*-Skript oder ein schon generiertes *LowLevel*-Skript einlesen. Soll die Animation aus einem *HighLevel*-Skript erzeugt werden, so ist zusätzlich die Angabe der XML-Dateien mit den Ersetzungsvorschriften für Objektbezeichnungen, Anweisungen und Parameter erforderlich.

Nach dem Einlesen des Skriptes wird der Vorgang der Animationsgenerierung gestartet. Da für jeden Frame eine komplette Generierung des Bildes seitens des `SoExaminerViewers` veranlasst werden muss, findet dieser Prozess nicht in Echtzeit statt[10]. Der Zeitaufwand hängt dabei stark von der Anzahl der dargestellten Objekte und der Menge der auszuführenden Anweisungen ab.

Dem Nutzer wird an dieser Stelle, dem Ende des Erzeugungsprozesses aller Frames, zusätzlich die Möglichkeit angeboten, entweder ein Video erzeugen zu lassen oder die Frames als Einzelbilder zu speichern und sie so für spätere Anwendungszwecke, wie die Einbindung in schriftliche Arbeiten oder als Standbilder, weiter zu verwenden.

Generierung von Einzelbildern

Neben der Generierung von fertigen Animationen können mithilfe der Skriptanweisungen auch Einzelbilder erzeugt werden. Dabei wird die jeweilige Anweisung durch den Autor in einer Kommandozeile eingegeben und sofort ausgeführt. Ein Screenshot dieses Teiles des AnimationScripters findet sich in Abbildung 5.3. Die Art der Anweisung (*HighLevel* oder *LowLevel*) spielt dabei keine Rolle. Sämtliche in der Skriptsprache vorgesehenen oder durch Ersetzungsvorschriften definierten Anweisungen können ausgeführt werden. Dabei entfällt der Teil der Anweisung zur Angabe von Zeitpunkt oder Zeitraum.

Damit bei stets wiederkehrenden Darstellungen nicht immer sämtliche Anweisungen von Hand eingegeben werden müssen, können statische Bilder per Skript erzeugt werden. Auch hierbei werden die Zeitangaben vernachlässigt.

[10] Auf dem genutzten Rechner (3,2 GHz mit 1GB RAM) benötigte die Erzeugung einer 30 sekündigen Animation ca. 2 Minuten

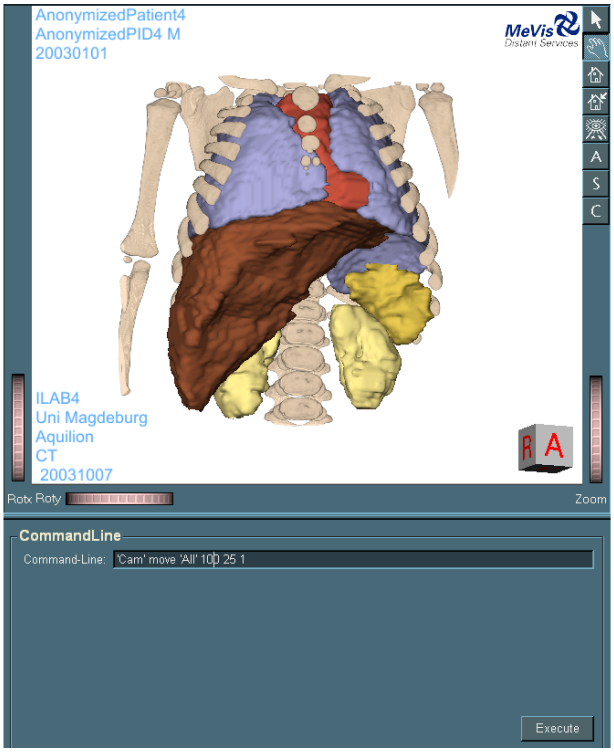


Abbildung 5.3: Ausschnitt aus der Oberfläche, wie sie sich zur Erzeugung von Einzelbildern mit der Kommandozeile präsentiert.

Das Netzwerk des AnimationScripters

Das Netzwerk des AnimationScripters basiert auf dem Netzwerk des InterventionPlanners. Die Skriptschnittstelle wird durch das Modul `UMDAnimationWrapper` repräsentiert. Dieses ist in das Unternetzwerk `IPView` des AnimationScripters eingebunden. Das `IPView`-Modul übernimmt die Aufgabe der Darstellung aller Objekte. In Abbildung A.2 im Anhang ist ein Ausschnitt des `IPView`-Netzwerkes mit Teilen der in dieser Arbeit implementierten Module zu sehen.

Das Modul `UMDAnimationWrapper` kapselt als Makromodul die drei Module `UMDAnimation`, `SoUMDChangeTrigger` und `BackgroundRamp` (siehe Abbildung 5.4). Das ML-Modul `UMDAnimation` übernimmt den Großteil der Animationsgenerierung und beinhaltet alle wesentlichen Funktionen der Skriptschnittstelle. Es wird im folgenden Abschnitt genauer beschrieben.

Das Inventor-Modul `SoUMDChangeTrigger` dient der Synchronisation zwischen dem Animationsmodul und dem `SoExaminerViewer` der Anwendung. Das Modul `SoExaminerViewer` ist für die endgültige Darstellung der Szene verantwortlich. Als drittes Modul kommt ein Modul der Klasse `BackgroundRamp` zum Einsatz, welches die Gestaltung von Farbverläufen im Hintergrund der Szene ermöglicht (Abbildung A.2 (D)).

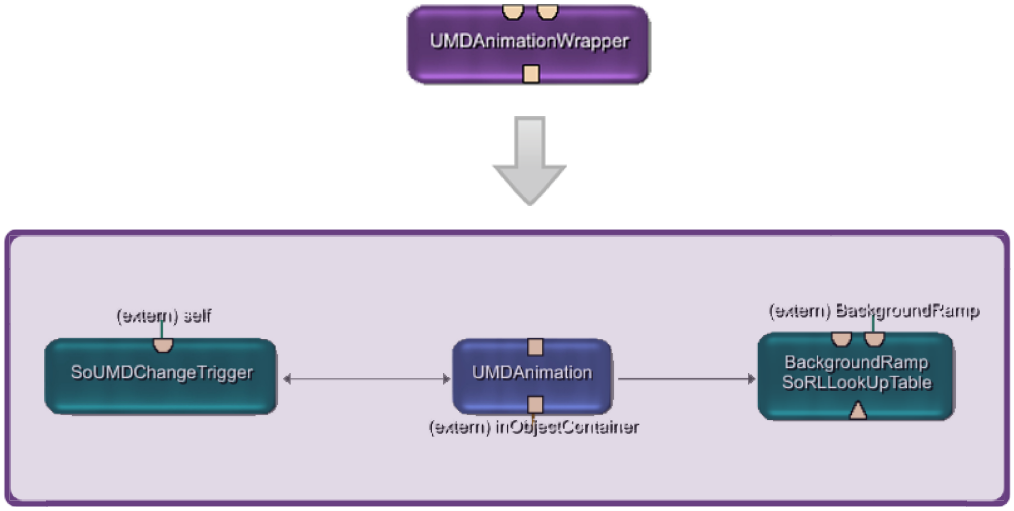


Abbildung 5.4: Das im Rahmen dieser Arbeit entwickelte Modul `UMDAnimationWrapper`.

Zusätzlich wurde ein Modul zur Berechnung der *bounding box* implementiert und im Netzwerk des AnimationScripters integriert. Das Modul `UMDBoundingBox` übernimmt dabei die einmalige Berechnung der *bounding box* eines Objektes, nachdem es geladen wurde, und ermöglicht so die spätere Ermittlung von Position und Mittelpunkt des Objektes, beispielsweise für die Kamerapositionierung. Desweiteren wurden Erweiterungen in den Skriptdateien zu verschiedenen Modulen vorgenommen, um die Benutzerschnittstelle zu erweitern und anzupassen.

Zu Test- und Entwicklungszwecken wurden Module zur Erzeugung eines Koordinatensystems in das `IPView`-Netzwerk integriert (siehe Abbildung A.2 (E)). Damit ist es möglich, über Skriptbefehle ein Koordinatensystem in der Szene einzublenden.

Das Animationsmodul `UMDAnimation`

Das Animationsmodul `UMDAnimation` liest eine *LowLevel*-Skript ein, extrahiert daraus die zur Manipulation der Darstellung notwendigen Funktionen und führt diese aus. Sämtliche Funktionen, die zum Parsing[11] eines Skriptes benötigt werden, wurden in einer externen Bibliothek gekapselt, die beim Start von MeVisLab dynamisch eingebunden wird. Dies bietet den großen Vorteil, dass Skripte unabhängig von der implementierten Skriptschnittstelle und dem dazugehörigen Visualisierungssystem umgewandelt und analysiert werden können. Die Analyse eines Skriptes umfasst dabei die Umwandlung eines *LowLevel*-Skriptes in eine Frameliste, wie sie in

[11] Ein *Parser* ist ein Programm, das eine syntaktische Analyse durchführt [Schneider 1997]. Als Parsing wird hier die Umwandlung eines *HighLevel*-Skriptes in ein *LowLevel*-Skript und die darauffolgende Analyse dieses Skriptes verstanden.

Abschnitt 4.2.4 beschrieben wurde und wie sie von einer Skriptschnittstelle direkt verarbeitet werden kann.

Die Frameliste wird vom Modul `UMDAnimation` Anweisung für Anweisung seriell abgearbeitet. Dabei lassen sich drei Gruppen von Anweisungen ausmachen:

1. Anweisungen zur Veränderung von Objekteigenschaften
2. Anweisungen zur Manipulation von Szenenparametern
3. Anweisungen zur Videogenerierung

Anweisungen, die Objektparameter wie Farbe oder Transparenz verändern, werden über das Nachrichtensystem des ObjectManagers ausgeführt (Abbildung A.2 (A)). Über die Veränderung werden alle betroffenen Komponenten des Netzwerkes des AnimationScripters informiert, unter anderem auch der `SoExaminverViewer`, der daraufhin ein Rendern der Szene veranlasst.

Szenenanweisungen dienen der Veränderung von Parametern der Szene, wie Kameraposition und -ausrichtung oder Art und Farbe des Hintergrundes. Alle Szenenparameter können direkt an den Feldern des `SoExaminerViewers` abgefragt und manipuliert werden. Daher ist für Anweisungen dieser Art das Modul `UMDAnimation` (bzw. das übergeordnete Netzwerk `UMDAnimationWrapper`) über eine Vielzahl direkter Feldverbindungen mit dem `SoExaminerViewer` verbunden (Abbildung A.2 (B)). Veränderungen an diesen Feldern, ausgelöst beispielsweise durch eine Anweisung zur Positionierung der Kamera, werden sofort dargestellt.

Die dritte Gruppe der Anweisungen, die Videoanweisungen, basieren als einzige nicht auf Anweisungen des *LowLevel*-Skriptes, sondern werden im Zuge der Erstellung der Frameliste in diese eingefügt. Die zwei wichtigsten Anweisungen aus dieser Gruppe sind die Anweisungen zum Speichern eines Frames und zur Erzeugung des Videos. Erstere Anweisung wird ausgeführt, nachdem alle Anweisungen zu einem Frame abgearbeitet wurden. Die Anweisung zur Erzeugung eines Videos wird ausgeführt, nachdem alle Frames erzeugt und gespeichert wurden.

Synchronisation des Renderprozesses mit der Anweisungsausführung

Der getrennte Ablauf der Prozesse der Anweisungsabarbeitung und des Rendering macht eine Synchronisation beider nötig (siehe Abbildung A.2 (C)). Dafür wurde das Modul `SoUMDChangeTrigger` entwickelt. Es wird nach der Ausführung einer Anweisung seitens des Modules `UMDAnimation` in einen aktiven Zustand versetzt und überprüft daraufhin zyklisch den `SoExaminerViewer` auf den Status des Renderingprozesses hin. War der Renderingprozess erfolgreich, so übermittelt das Modul diese Nachricht an das Modul `UMDAnimation` , welches daraufhin in der Abarbeitung seiner Frameliste fortfahren kann. Dieses Mittel der Synchronisation musste entwickelt werden, da der InterventionPlanner bzw. MeVisLab im Bereich des Renderings nicht über ein Nachrichtensystem verfügt, wie es beispielsweise beim ObjectManager zum Einsatz kommt.

5.3 Zusammenfassung der Implementierung

Das in dieser Arbeit entwickelte Skriptkonzept wurde exemplarisch in der Software MeVisLab implementiert. Diese bietet den großen Vorteil, dass Entwickler eigene Erweiterungen in Form von Modulen und Applikationen in der Software einbringen können.

Die Skriptschnittstelle wurde als Teilapplikation AnimationScripter implementiert. Der Autor einer Animation hat hier die Möglichkeit, Skripte und XML-Ersetzungsvorschriften einzulesen und sich daraus auf Basis eines patientenindividuellen Datensatzes eine Animation bzw. eine Folge von Einzelbildern erzeugen zu lassen. Zusätzlich können über eine Kommandozeile Anweisungen direkt eingegeben und ausgeführt werden. Dies ermöglicht die Erstellung von definierten und reproduzierbaren Einzelbildern.

Kapitel 6

Ausblick

6.1 Skripte für Interaktive Animationen

Das hier entwickelte Skriptkonzept kann neben der Erstellung von fertigen Videos auch für interaktive Animationen genutzt werden, wie sie im Abschnitt 2.5.4 beschrieben wurden. In einem System, in dem interaktive Animationen zum Einsatz kommen (z.B. dem LiverSurgeryTrainer, Abschnitt 3.7), können einzelne Animationsabläufe als Skript gespeichert werden. Diese Skripte können dann direkt ausgeführt werden, wenn der Nutzer bestimmte Interaktionen ausführt. Ein Beispiel hierfür ist die Anweisung `emphasize`, wie sie in Abschnitt 4.3.3 beschrieben wurde. Eine solche Anweisung kann ausgeführt werden, wenn der Nutzer in der Darstellung ein Objekt selektiert. Daraufhin würde die Kamera in eine Position gebracht werden, aus der das Objekt sichtbar ist und die Objekteigenschaften (z.B. Farbe oder Transparenz) würden entsprechend der Hervorhebungstechnik geändert.

6.2 Sichtbarkeitstests

Eine andere Erweiterungsmöglichkeit, die vor allem bei interaktiven Animationen zum Tragen käme, aber auch bei fertigen Videos sinnvoll wäre, sind Sichtbarkeitstests. Bisher bleibt es dem Autor einer Animation überlassen, dafür Sorge zu tragen, verdeckende Objekte auszublenden oder in ihrer Darstellungsart so zu verändern, wie es in Abschnitt 2.7.2 vorgestellt wurde. Die Manipulation von verdeckenden Objekten kann einerseits durch eine gesonderte *LowLevel*-Anweisung erfolgen oder andererseits automatisch bei allen Kamerabewegungen mit der Ausrichtung auf ein Objekt erfolgen.

Mithilfe von Sichtbarkeitstest erschließt sich auch eine neue Animationstechnik, die bisher noch nicht beobachtet und daher auch in Abschnitt 2.7 nicht beschrieben wurde. Bei einer interaktiven Navigation des Nutzers können direkt Objekte ausgeblendet werden, die die Sicht auf das derzeitige Objekt von Interesse verdecken[1]. Dies kann mithilfe von Skripten und einer Anweisung erfolgen, die die Transparenz entsprechender Objekte innerhalb einer kurzen Zeitspanne verändert. Objekte,

[1] Das Objekt von Interesse kann der Nutzer zuvor beispielsweise durch Selektion bestimmt haben.

die die Sicht nicht mehr verdecken, können anschließend, wiederum animiert, eingeblendet werden.

6.3 Verbesserung des Kamerapfades

Die Bewegung der Kamera von einer Position zur nächsten erfolgt in der derzeitigen Implementierung noch linear. Dabei wird nicht kontrolliert, ob es zu Kollisionen der Kamera mit Objekten der Szene kommt. Eine Kollisionserkennung wäre aber sinnvoll, um den Autor von Überlegungen zum Verlauf des Pfades der Kamera zu entlasten. In diesem Zuge ist auch eine Veränderung des Pfades abweichend von einem linearen Verlauf denkbar. [Buckley 1994] beschreibt einen Ansatz, bei dem Bézier-Kurven genutzt werden, um einen Kamerapfad mithilfe verschiedener Stützpunkte zu generieren. So können Kamerapfade generiert werden, die für den Betrachter natürlicher und ohne abrupte Richtungsänderungen erfolgen. [Helbing 2004] erweitert diesen Ansatz um eine Kollisionserkennung und eine damit verbundene Anpassung des Pfades.

6.4 Grafische Skripteingabe

Da der Entwurf und die Eingabe von Skripten immer noch einen gewissen, wenn auch geringen Grad an Grundkenntnissen der Programmierung erfordert, lohnen Überlegungen über eine einfachere und intuitivere Erzeugung von Skripten. Dies kann gerade für unerfahrene Nutzer eine große Hilfe sein. Im Rahmen dieser Arbeit wurden dazu zwei Ideen entwickelt, die im weiteren Verlauf kurz vorgestellt werden: die Erzeugung von Skripten über eine grafische Benutzerschnittstelle und über die Eingabe von Keystates in der Darstellung der Szene.

6.4.1 Erzeugung von Skripten über eine grafische Benutzerschnittstelle

Eine grafische Benutzerschnittstelle würde in der Hierarchie des Skriptkonzeptes (Abschnitt 4.1.2) oberhalb des *HighLevel*-Skriptes als neue Stufe ansetzen. Als Ergebnis würde eine solche Schnittstelle ein *HighLevel*-Skript erzeugen, welches wiederum nachträglich durch den Nutzer editiert werden könnte. Einen ersten Entwurf für eine solche Schnittstelle speziell für medizinische Animationen lieferte [Mehnert 2005] (siehe Abbildung A.3). Ein ähnlicher Ansatz wurde bei dem Entwurf einer möglichen Oberfläche im Rahmen dieser Arbeit verfolgt.

Da im Vergleich zum textuellen Skript eine Vereinfachung der Eingabe das Ziel einer grafischen Oberfläche ist, wurde das Design dementsprechend einfach gehalten. Zentrum der Darstellung, wie sie sich in Abbildung A.4 im Anhang zeigt, ist eine Zeitleiste. Die Idee dazu entstammt Programmen wie Macromedia Director[2]

[2] http://www.macromedia.com

oder Adobe Premiere[3]. Auf der Zeitleiste werden von links nach rechts Aktionen eingetragen. Die Länge der Aktionen in der späteren Animation wird dabei über die horizontale Ausdehnung der Aktion auf der Zeitleiste bestimmt. Auf der Zeitleiste können mehrere Aktionen eingetragen werden, die sich zeitlich überlappen oder parallel verlaufen.

Dem Autor wird eine Reihe von möglichen Aktionen angeboten, aus denen er jeweils eine wählen kann. Aktionen repräsentieren dabei einzelne Skriptbefehle. In gleicher Form werden alle verfügbaren Objekte des zugrundeliegenden Datensatzes angezeigt. Aus Objekt und Aktion wird in einem zusätzlichen Feld eine Anweisung generiert, die per *Drag&Drop*[4] in die Zeitleiste gezogen werden kann (siehe Abbildung A.4). Mögliche Parameter können je nach Aktion über ein zusätzliches Fenster abgefragt werden.

6.4.2 Interaktive Eingabe von Keystates

Eine zweite Möglichkeit, die Erzeugung einer Animationsvorlage in Form eines Skriptes zu vereinfachen und von der textuellen Eingabe zu abstrahieren ist das Festlegen von Keystates direkt in der interaktiven Darstellung. Der Autor kann dabei sämtliche Einstellungen für Objekte und Kamera in der Szene vornehmen und die jeweilige Sicht als Zustand speichern. Zur Erzeugung einer Animation aus diesen Keystates müssen die zeitlichen Längen zwischen den Keystates angegeben werden. Aus diesen Informationen kann dann automatisch ein Skript generiert werden. Dieses könnte dann zusätzlich angepasst und editiert werden.

Dieser Ansatz fördert vor allem das Vorstellungsvermögen des Autors bezüglich der fertigen Animation. Ein großer Nachteil liegt dabei in der Interaktion und der damit ungenauen Ausrichtung der Kamera. Auch müsste der Autor zu jeder Sicht ein oder mehrere Objekte als Relation angeben, damit die Kameradaten im Bezug auf diese gespeichert werden und eine Erzeugung von Anweisungen mit dem geforderten Objektbezug möglich ist.

6.5 Evaluation

Aus zeitlichen Gründen konnte eine Evaluation der Ergebnisse der Arbeit nicht durchgeführt werden. Die Durchführung einer Evaluation wäre jedoch besonders für zwei Aspekte der Arbeit sinnvoll: die Animationserstellung und die Beurteilung der erzeugten Animationen.

Bei der Evaluation der Animationserstellung kann zunächst die Verständlichkeit der Skriptsprache für verschiedene Nutzergruppen untersucht werden. Dabei ist

[3] http://www.adobe.com

[4] Unter *Drag&Drop* wird das Aufgreifen eines Objektes mittels Mausklick, das Bewegen dieses Objektes bei gedrückter Maustaste und das abschließende Fallenlassen des Objektes an einer gewünschten Stelle verstanden.

zu analysieren, wie schnell ein Nutzer die Skriptsprache erlernen kann und ob erfahrene Nutzer vom Angebot der selbstständigen Erweiterung des *HighLevel*-Befehlssatzes Gebrauch machen. Die Nachvollziehbarkeit und Praktikabilität der XML-Ersetzungsvorschriften ist ebenfalls in Zusammenarbeit mit möglichen Nutzern zu untersuchen.

Die Evaluation der fertigen Animationen kann eine Analyse der verwendeten Animationstechniken beinhalten. Dabei ist zu prüfen, ob die eingesetzten Techniken zum Erreichen des Zieles der Animation beitragen oder diesem eher abträglich sind. Für statische Visualisierungen im Bereich der medizinischen Ausbildung haben [Mirschel 2004] und [Tietjen 2004] mit ihren Arbeiten ein gutes Beispiel einer solchen Evaluation gegeben.

Kapitel 7

Zusammenfassung

Abschließend werden die Ergebnisse der Arbeit kurz zusammengefasst.

Es wurde eine Skriptsprache zur Generierung von medizinischen Anwendungen entwickelt. Der dabei gewählte mehrstufige Ansatz ermöglicht es, sowohl Anfängern wie auch Experten Animationen nach ihren Wünschen und Vorstellungen zu erstellen. Dabei kommen Anweisungen verschiedener Abstraktionsstufen zum Einsatz. Die Skriptsprache ist weitestgehend koordinatenfrei. Alle Positionsangaben erfolgen in Bezug auf Objekte in der Szene. Dies erleichtert die Anwendung der Skriptsprache für die Autoren von Animationen enorm bzw. macht den praktischen Einsatz erst möglich.

Das Skriptkonzept wurde exemplarisch am Beispiel medizinischer Animationen implementiert. Als Datengrundlage der Animationen dienten segmentierte anatomische Daten, in denen die zu visualisierenden Strukturen als Objekte vorliegen. Damit können sowohl Animationen für die Therapieplanung wie auch für medizinische Lernsysteme in der Ausbildung generiert werden. Für die Implementierung kam die Software MeVisLab zum Einsatz. Es wurde innerhalb dieser Software die Applikation AnimationScripter entwickelt, mit der Skripte, die in der hier entworfenen Skriptsprache verfasst wurden, eingelesen werden können. Zusammen mit den entsprechenden patientenindividuellen Daten kann daraus eine Animation im Form eines Videos generiert werden. Im Zuge der Entwicklung des AnimationScripters wurde sowohl ein Parser zur Umwandlung und Analyse der Skripte als auch die komplette Skriptschnittstelle zur Software MeVisLab implementiert.

Die Ergebnisse konnten zusätzlich um einige nicht in der Aufgabenstellung geforderte Punkte erweitert werden. Das entwickelte Skriptkonzept ermöglicht es nicht nur medizinische Animationen zu erzeugen. Aufgrund seines stark objektbezogenen Charakters ist es zudem möglich, mithilfe der Skriptsprache nahezu jede objektorientierte grafische Darstellung zu animieren. Dabei ist lediglich die entsprechende Schnittstelle zur visualisierenden Software zu entwickeln.

Desweiteren bietet das Konzept eine sehr gute Möglichkeit zur Erzeugung von statischen Einzelbildern. Dies ist von Vorteil, wenn es darum geht, Objekte oder Szenen aufgrund fest vorgegebener Parameter zu generieren. Das können zum Beispiel definierte Sichten auf eine Szene oder bestimmte Parameter von Objekten wie deren Farbe sein.

Ausblickend konnte gezeigt werden, dass die hier entwickelte Skriptsprache ohne Probleme auch für interaktive Animationen eingesetzt werden kann. In diesem Zusammenhang wurde eine Animationstechnik entwickelt, die die Sichtbarkeit eines Objektes auch während einer interaktiven Navigation in der Szene sicherstellt. Außerdem wurde eine grafische Oberfläche zur vereinfachten Eingabe von Animationsanweisungen skizziert und beschrieben.

Literaturverzeichnis

[André u. a. 1996] André, Elisabeth ; Müller, Jochen ; Rist, Thomas: WIP/PPP: Automatic Generation of Personalized Multimedia Presentations. In: *Proc. of Multimedia 96, 4th ACM International Multimedia Conference*. Boston, MA, 1996, S. 407–408

[Angel 2000] Angel, Edward ; Angel, Edward (Hrsg.): *Interactive Computer Graphics - A top-down approach with OpenGL*. Addison Wesley, 2000

[Bade u. a. 2004] Bade, Ragnar ; Mirschel, Sebastian ; Oldhafer, Karl J. ; Preim, Bernhard: Ein fallbasiertes Lernsystem für die Behandlung von Lebertumoren. In: *Bildverarbeitung für die Medizin* (2004), S. 438–442

[Barron 2000] Barron, David W.: *The world of scripting languages*. John Wiley & Sons, Ltd., 2000

[Bourquain u. a. 2002] Bourquain, H. ; Schenk, A. ; Link, F. ; Preim, B. ; Prause, G. ; Peitgen, H.-O.: HepaVision2: A software assistant for preoperative planning in living-related liver transplantation and oncologic liver surgery. In: Lemke, H. U. (Hrsg.): *Computer Assisted Radiology and Surgery (CARS 2002)*, 2002, S. 341–346

[Buckley 1994] Buckley, Colm: Bézier Curves for Camera Motion. In: *2nd Irish Computer Graphics Workshop*, 1994

[Butz 1994] Butz, Andreas: Betty: Planning and Generating Animations for the Visualization of Movements and Spatial Relations. In: *Advanced Visual Interfaces (Proceedings of AVI '94, Bari, Italy)*, ACM Press, 1994

[Conway 1997] Conway, M. J.: *Alice: Easy-to-Learn 3D Scripting for Novices*, Faculty of the School of Engineering and Applied Science at the University of Virginia, Diss., 1997

[Dörge 2002] Dörge, Christina: *Techniken zur interaktiven Hervorhebung von Objekten in medizinischen 3d-Visualisierungen*, Universität Bremen- Fachebreich 3 - Mathematik & Informatik, Diplomarbeit, August 2002

[Ecma 1999] Ecma: ECMAScript Language Specification / Ecma International. 1999. – Forschungsbericht

[Fischer u. a. 1996] Fischer, M. ; Schauer, S. ; Gräsel, C. ; Baehring, Th. ; Mandl, H. ; Gärtner, R. ; Scherbaum, W. ; Scriba, P.C.: Modellversuch

CASUS: Ein computergestütztesAutorensystem für die problemorientierte Lehre in der Medizin. In: *Zeitschriftfür ärztliche Fortbildung* (1996)

[Fischer und Hofer 2004] Fischer, Peter ; Hofer, Peter: *Lexikon der Informatik*. SmartBooks, 2004

[Foley 2000] Foley, James D.: *Computer graphics : principles and practice*. 2. ed. Addison-Wesley, 2000

[Gering 1999] Gering, David T.: *A System for Surgical Planning and Guidance using Image Fusion and Interventional MR*, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Diplomarbeit, 1999

[Haag 1998] Haag, Martin: *Plattformunabhängige, adaptive Lehr-/Lernsysteme für die medizinische Aus- und Weiterbildung*, Medizinische Fakultät der Ruprecht-Karls-Universität Heidelberg, Diss., 1998

[Haase 2004] Haase, Tina: *Automatische Animationen in einem fallbasierten medizinischen Lernsystem*, Otto-von-Guericke Universität Magdeburg, Laborpraktikum, März 2004

[Haase 2005] Haase, Tina: *Explosionsanimationen in virtuell-interaktiven Trainingsszenarien*, Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, Diplomarbeit, März 2005

[Hausig 1998] Hausig, Nils: *Entwurf und Implementation eines Werkzeugs zur Generierung von Animationen durch automatische Navigation in schlauchförmigen Hohlstrukturen*, Fachbereich Informatik, Universität Hamburg, Diplomarbeit, 1998

[Helbing 2004] Helbing, Ralf: *Ein Erweitertes Kameramodell - Methoden und Werkzeuge für die dynamische Kamerasteuerung in interaktiven Systemen*, Otto-von-Guericke Universität Magdeburg, Diss., 2004

[Höhne u. a. 2003] Höhne, Karl H. ; Pflesser, Bernhard ; Pommert, Andreas ; Priesmeyer, Kay ; Riemer, Martin ; Schiemann, Thomas ; Schubert, Rainer ; Tiede, Ulf ; Frederking, Hans ; Gehrmann, Sebastian ; Noster, Stefan ; Schumacher, Udo: *VOXEL-MAN 3D Navigator: Inner Organs. Regional, Systemic and Radiological Anatomy - 3 CD-ROMs*. Springer-Verlag Electronic Media, Heidelberg, 2003

[Hintze 2003] Hintze, Jana: *3D-Animations-Skripting für nicht-professionelle Benutzer*, Otto-von-Guericke Universität Magdeburg, Diplom, April 2003

[Hinz u. a. 2001] Hinz, M. ; Pohle, R. ; Tönnies, K. D.: Interpretation medizinischer Schichtdaten durch interaktiv gesteuerte 3D-Bildanalyse. In: *Simulation und Visualisierung*, 2001, S. 375–384

[Karp und Feiner 1993] Karp, Peter ; Feiner, Steven: Automated presentation planning of animation using task decomposition with heuristic reasoning. In: *Graphics Interface 1993, Toronto, Canada, May 17-21*, 1993, S. 118–127

[Konrad-Verse u. a. 2004] Konrad-Verse, Olaf ; Preim, Bernhard ; Littmann, Arne: Virtual Resection with a Deformable Cutting plane. In: *Simulation und Visualisierung 2004*, 2004

[Masuch 2001] Masuch, Maic: *Nicht-photorealistische Visualisierungen: von Bildern zu Animationen*, Universität Magdeburg, Diss., 2001

[Mealing 1992] Mealing, Stuart ; Mealing, Stuart (Hrsg.): *The art and science of computer animation.* Oxford : Intellect, 1992

[Mehnert 2005] Mehnert, Annika: *Authoring von Animationen für die Ausbildung von Chirurgen*, Hochschule Magdeburg-Stendal, Diplomarbeit, 2005

[Mühler 2004] Mühler, Konrad: *VoxelMan*, Otto-von-Guericke Universität Magdeburg, Seminararbeit, Oktober 2004

[Mirschel 2004] Mirschel, Sebastian: *Erstellung eines Prototypen für ein fallbasiertes Lernsystem in der Leberchirurgie*, Otto-von-Guericke Universität Magdeburg, Diplomarbeit, April 2004

[Preim u. a. 2003] Preim, B. ; Hindennach, M. ; Spindler, W. ; Schenk, A. ; Littmann, A. ; Peitgen, H.-O.: Visualisierungs- und Interaktionstechniken für die Planung lokaler Therapien. In: *SimVis2003*, 2003, S. 237–248

[Preim und Peitgen 2004] Preim, B. ; Peitgen, H.-O.: Medizinische Visualisierung: Methoden und Anwendungen in der Ausbildung und Therapieplanung. In: *IT und TI* (2004)

[Preim und Ritter 2002] Preim, B. ; Ritter, F.: Techniken zur interaktiven Hervorhebung von Objekten in medizinischen 3d-Visualisierungen. In: *SimVis2002*, 2002, S. 187–200

[Preim 1997] Preim, Bernhard: *Interaktive Illustrationen und Animationen zur Erklärung komplexer räumlicher Zusammenhänge*, Otto-von-Guericke-Universität Magdeburg, Diss., September 1997

[Preim u. a. 1996] Preim, Bernhard ; Ritter, Alf ; Steinicke, Gunnar: Gestaltung von Animationen zur Erklärung komplexer 3D-Modelle. In: *Proc. of Simulation und Animation für Planung, Bildung und Präasentation.* Magdeburg, Februar 1996, S. 255–266

[Preim u. a. 2000] Preim, Bernhard ; Spindler, Wolf ; Peitgen, Heinz-Otto: Interaktive medizinische Volumenvisualisierung - ein Überblick. In: *Simulation und Visualisierung*, 2000, S. 69–88

[Preim und Strothotte 1996] Preim, Bernhard ; Strothotte, Thomas: Illustrating and Annotating 3D Geometric Models for Explanations: System Architecture. In: *Proc. 12th European Conference on Artificial Intelligence.* Budapest, August 1996, S. 1–8

[Ritter 2005] Ritter, Felix: *MeVisLab Module Reference*, April 2005

[Schneider 1997] Schneider, Hans-Jochen ; Schneider, Hans-Jochen (Hrsg.): *Lexikon der Informatik und Datenverarbeitung*. R. Oldenbourg Verlag, München, Wien, 1997

[Steinicke 1996] Steinicke, Gunnar: *Gestaltung von Animationen zur Erklärung komplexer 3D-Modelle*, Otto-von-Guericke Universität Magdeburg, Diplomarbeit, 1996

[Tiede 1999] Tiede, Ulf: *Realistische 3D-Visualisierung multiattributierter und multiparametrischer Volumendaten*, Fachbereich Informatik, Universität Hamburg, Diss., 1999

[Tietjen 2004] Tietjen, Christian: *Evaluierung und Modifikation von Methoden zur Generierung von Liniengrafiken in der medizinischen Visualisierung*, Otto-von-Guericke-Universität Magdeburg, Diplomarbeit, 2004

[Welsch 1997] Welsch, Norbert ; Welsch, Norbert (Hrsg.): *Multimedia-Programmierung mit Lingo*. 2. Auflage. Berlin [u.a.] : Springer, 1997

[Wernecke 1994] Wernecke, Josie: *The Inventor Mentor - Programming object-oriented 3D graphics with Open Inventor*. Addison-Wesley, 1994

[Wolf-Heidegger 2000] Wolf-Heidegger ; Köpf-Maier, P. (Hrsg.): *Atlas of Human Anatomy*. Bd. 2. 5. Auflage. Karger, 2000

[Zeltzer 1991] Zeltzer, D.: Task-Level Graphical Simulation: Abstraction, Representation, and Control. In: Badler, N. I. (Hrsg.) ; Barsky, B. A. (Hrsg.) ; Zeltzer, D. (Hrsg.): *Making Them Move: Mechanics, Control, and Animation of Articulaited Figures*. San Mateo, CA : Morgan Kaufmann, 1991, S. 3–33

Abbildungsverzeichnis

Listings

Anhang A

Abbildungen

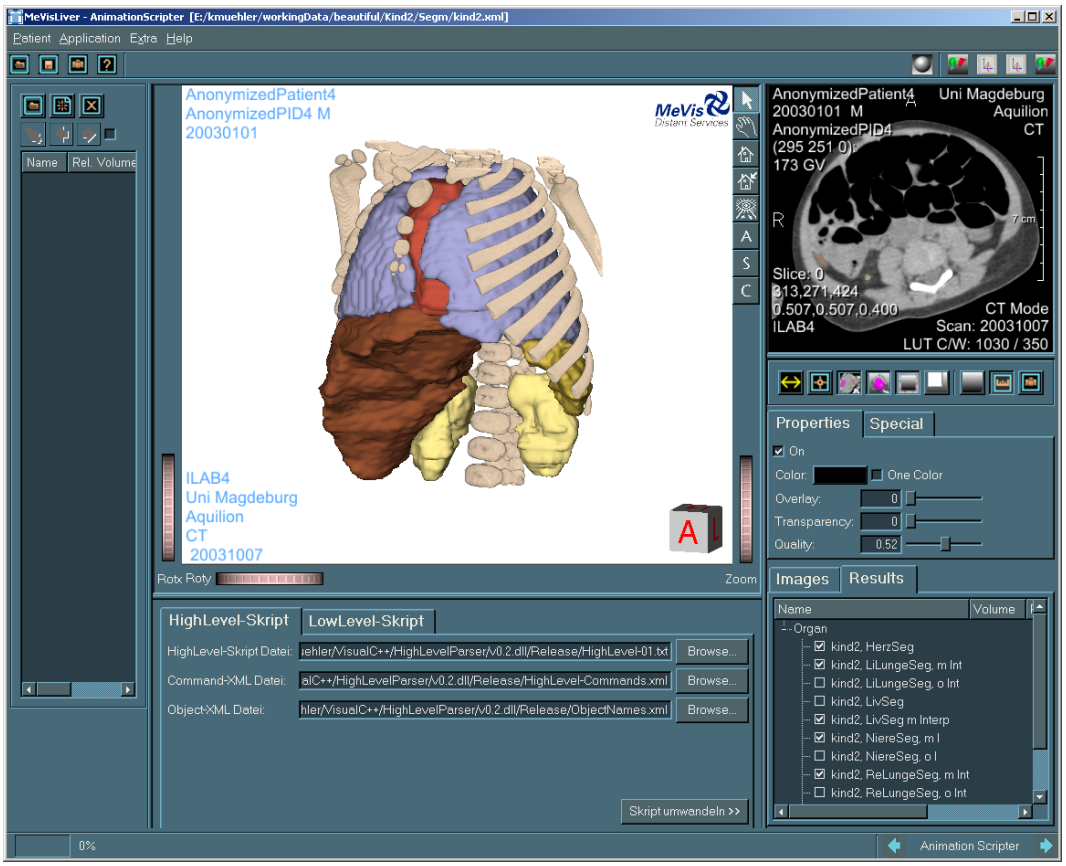


Abbildung A.1: Oberfläche des AnimationScripters als Erweiterung des InterventionPlanners. Zu erkennen sind die zentrale dreidimensionale Ansicht mit der darunterliegenden Möglichkeit ein Skript sowie die XML-Dateien einzulesen. Rechts oben ist das synchronisierte Schichtbild und unten rechts die Liste der segmentierten Objekte zu erkennen.

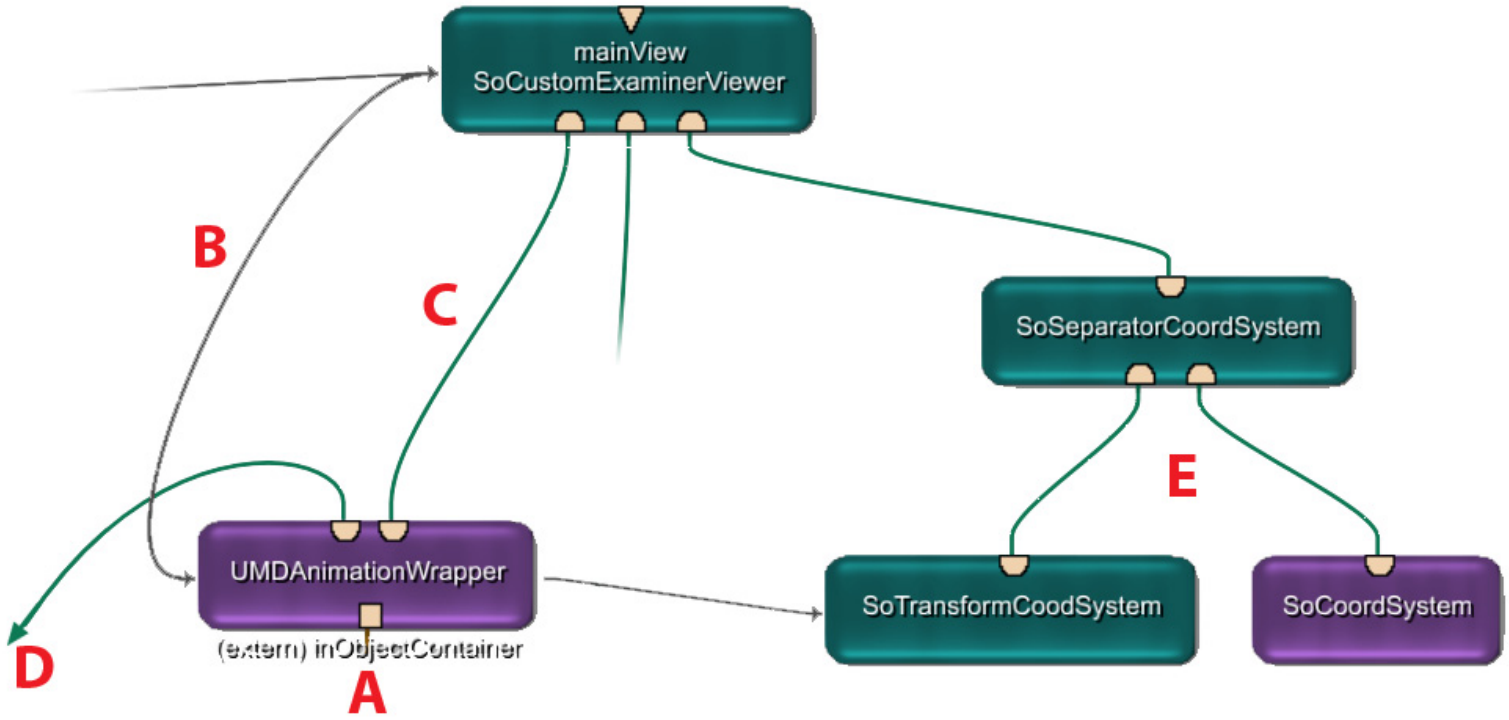


Abbildung A.2: Ausschnitt aus dem Netzwerk des `IPView`-Moduls.
(A) - Verbindung zum ObjectManager, (B) - Feldverbindungen für Szenen- und Videoanweisungen, (C) - Verbindung zur Synchronisation der Abläufe, (D) - Verbindung zur Erzeugung eines Hintergrundes, (E) - Module zur Visualisierung eines Koordinatensystems

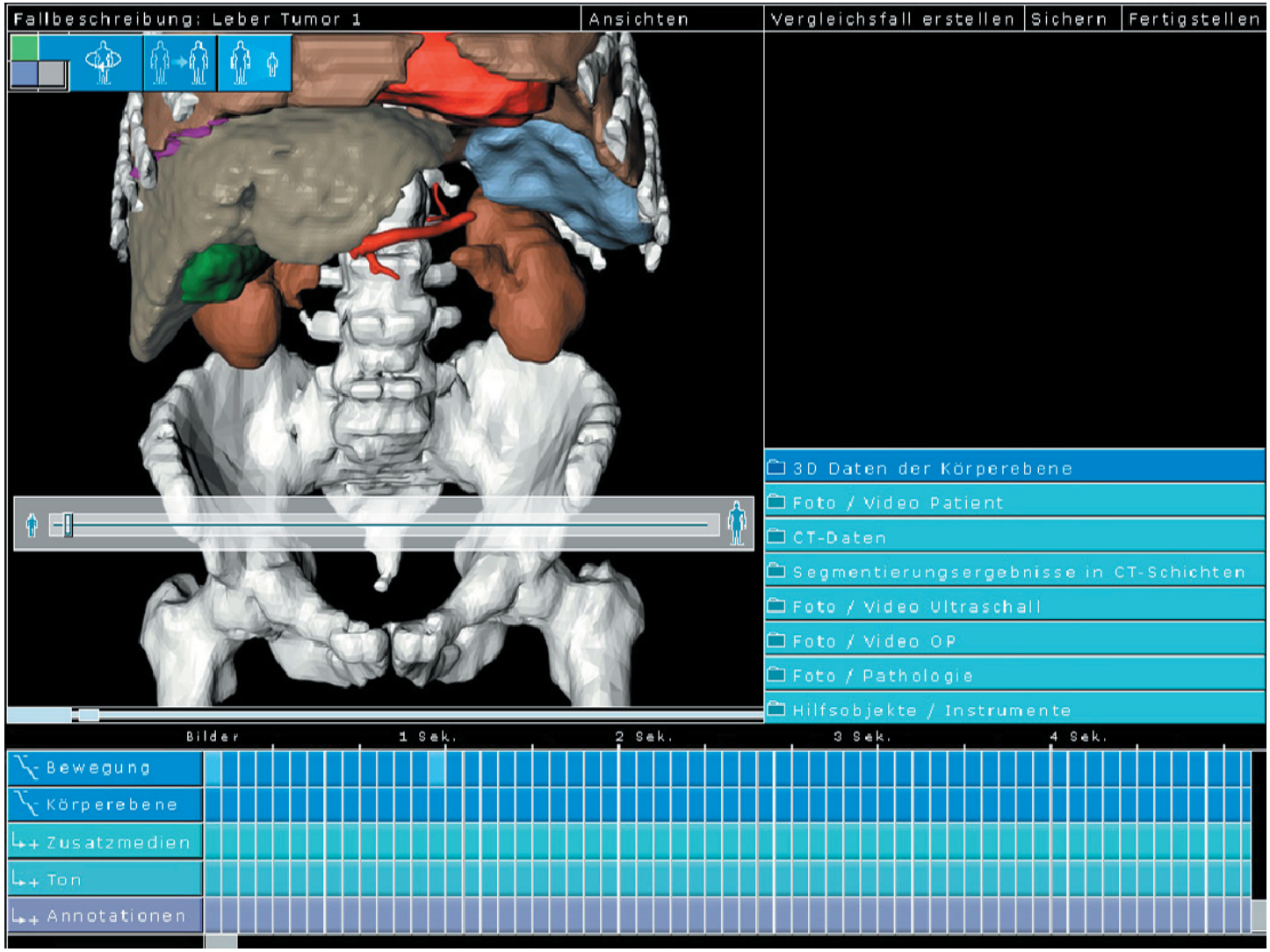


Abbildung A.3: Screenshot eines Designentwurfs für eine Oberfläche eines Autorensystems zur Generierung von Animationen für die medizinische Ausbildung (aus [Mehnert 2005])

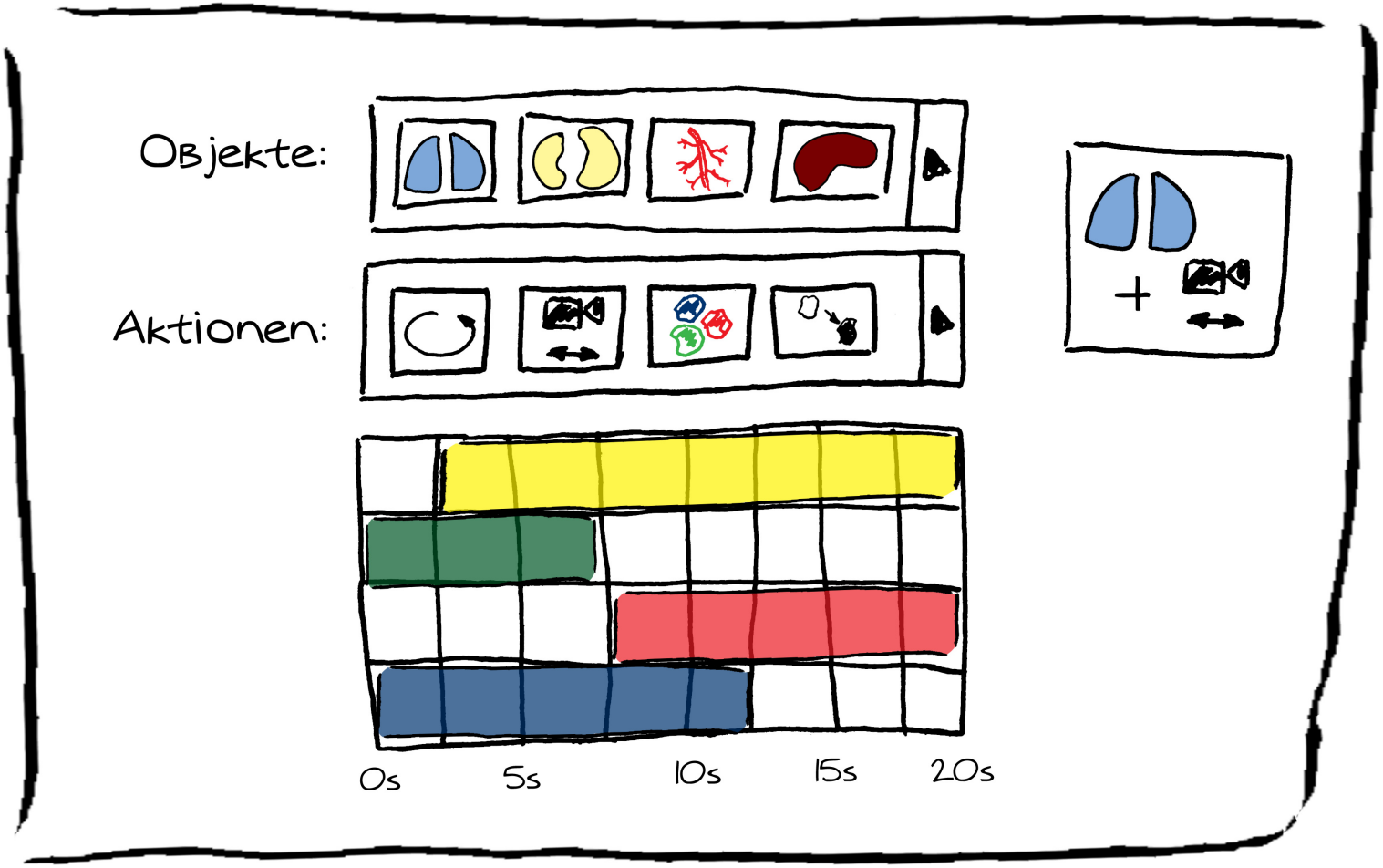


Abbildung A.4: Skizze einer Oberfläche, wie sie als Interface zur Erzeugung von *HighLevel*-Skripten zum Einsatz kommen könnte.

Anhang B

LowLevel-Anweisungen

***LowLevel*-Anweisung**	**Beschreibung**
`move` *`object a_sagi a_axial zoom`*	Bewegt die Kamera. Die Position ergibt sich dabei aus der Rotation in der sagittalen Ebene und daraufhin in der axialen Ebene um den Mittelpunkt des angegebenen Objektes und dem Zoomfaktor.
`rotate` *`object direction angle`*	Rotation der Kamera um den Mittelpunkt eines Objektes. *Direction* bezeichnet die Rotationsachse und *angle* den zu rotierenden Winkel.
`rotateX` *`angle`* `rotateY` *`angle`* `rotateZ` *`angle`*	Rotiert die Kamera um den angegebenen Winkel um eine Parallele zu einer Koordinatenachse und den momentanen `LookAt`-Punkt.
`rotateCamX` *`angle`* `rotateCamY` *`angle`* `rotateCamZ` *`angle`*	Rotiert die Kamera um ihren eigenen Mittelpunkt.
`rotateVirtualX` *`angle`* `rotateVirtualY` *`angle`* `rotateVirtualZ` *`angle`*	Rotiert die Kamera um Achsenparallelen zur momentanen Sicht und den aktuellen `LookAt`-Punkt. Dabei zeigt die X-Achse immer nach rechts, die Y-Achse nach oben und die Z-Achse aus dem Bild heraus.
`setBackground` *`R,G,B [ramp]`*	Setzt die Hintergrundfarbe der Szene auf einen RGB-Wert. Wird der Parameter *ramp* angegeben, so wird der Hintergrund im Verlauf von dieser Farbe zu schwarz dargestellt.
`setColor` *`R,G,B`*	Setzt die Farbe eines Objektes auf den angegebenen RGB-Wert, wobei die Werte sich jeweils im Bereich zwischen 0 und 255 bewegen.

`setCoorSize` *value*	Blendet ein Koordinatensystem der angegebenen Größe am jeweiligen LookAt-Punkt ein. Ein Wert von 0 blendet das Koordinatensystem aus.
`setCreateVideo` *true*/*false*	Erlaubt oder verhindert das Erzeugen eines Videos am Ende einer Animationsgenerierung.
`setLookAt` *object*	Richtet die Blickrichtung der Kamera auf ein Objekt aus.
`setQuality` *value*	Setzt die Darstellungsqualität eines Objektes auf einen angegebenen Wert zwischen 0 und 1.
`setSelected` *true*/*false*	Selektiert oder deselektiert ein Objekt.
`setStyle` *Filled*/*Lines*/*Points*	Setzt den Darstellungsstil auf eine gefüllte, eine Gitternetz oder eine gepunktete Darstellung.
`setTransparency` *value*	Setzt die Transparenz eines Objektes auf einen angegebenen Wert zwischen 0 und 1.
`setVisible` *true*/*false*	Setzt die Sichtbarkeitseigenschaft eines Objektes.
`setWatchUpVector` *true*/*false*	Schaltet die automatische Korrektur der Kamera für eine aufrechte Sicht ein oder aus.

Tabelle B.1: Übersicht über alle *LowLevel*-Anweisungen, die im Rahmen der Arbeit im AnimationScripter implementiert wurden

Anhang C

Beispiele

C.1 Animation 1

In dieser Animation fährt die Kamera zunächst an die rechte Niere heran. Da die Leber diese verdeckt, wird sie in einer gepunkteten Darstellung angezeigt. Anschließend wir das Herz mithilfe der `emphasize`-Anweisung rot gefärbt und die Kamera zum Herz hin bewegt. Die Transparenz der Lungenflügel wird so verändert, dass das Herz gut sichtbar ist.

Ausschnitte der Animation sind in Abbildung C.1 zu erkennen. Das *HighLevel*-Skript zu dieser Animation findet sich in Listing C.1. Die für diese Animation entscheidenden Ausschnitte aus den XML-Dateien für die Objektbezeichnungen und die Anweisungsersetzungen sind in den Listings C.2 und C.3 dargestellt.

```
[Ini]
LengthTimeUnit=15
LengthSeconds=15

[Script]
[0] 'System' init
[0,5] 'Kidney_left' view front 1
[3] 'Liver' setStyle Points
[5,10] 'All' view front 1
[8] 'Liver' setStyle Filled
[10,15] 'Heart' emphasize
[10,15] 'Lung' setTransparency high
```

Listing C.1: *HighLevel*-Skript zur Erzeugung der Animation, wie sie in Abschnitt C.1 beschrieben wird.

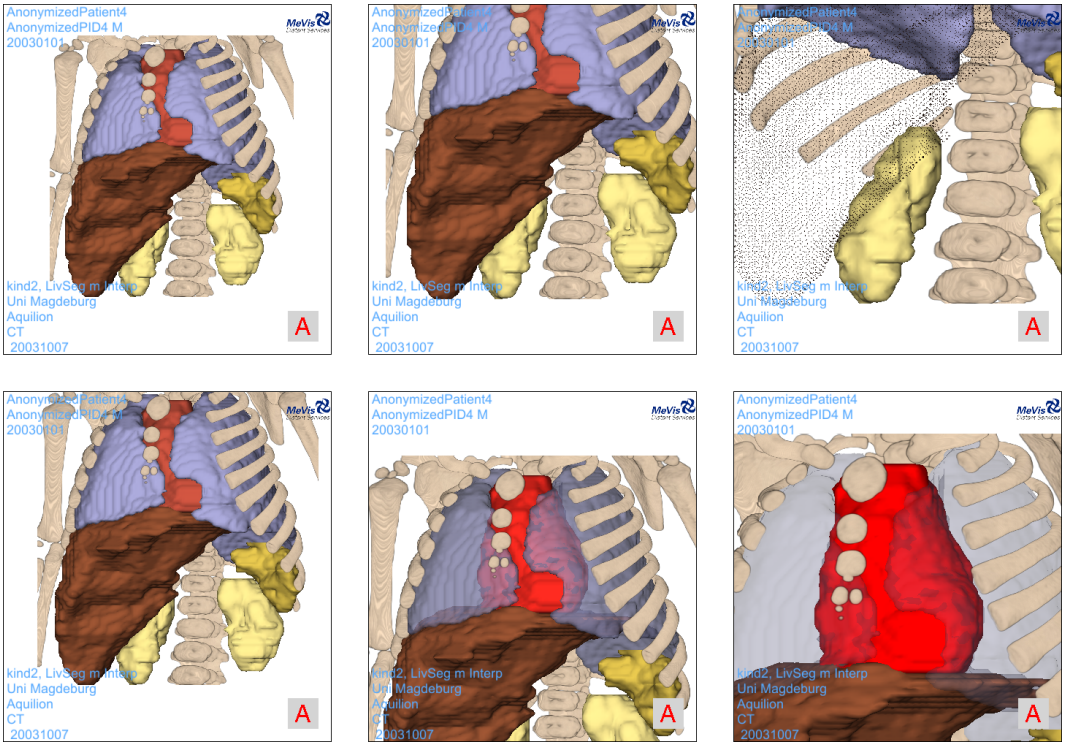


Abbildung C.1: Ausschnitte einer Animation, wie sie mithilfe des *HighLevel*-Skriptes in Listing C.1 erzeugt wurde.

```
<ObjectNames>
 <object name="Kidney">
  <object>Kidney_left</object>
  <object>Kidney_right</object>
 </object>
 <object name="Kidney_left">
  <object>kind2, NiereSeg, m I</object>
 </object>
 <object name="Kidney_right">
  <object>kind2, ReNiereSeg, m I</object>
 </object>

 <object name="Lung">
  <object>Lung_left</object>
  <object>Lung_right</object>
 </object>
 <object name="Lung_right">
  <object>kind2, ReLungeSeg, m Int</object>
 </object>
 <object name="Lung_left">
  <object>kind2, LiLungeSeg, m Int</object>
 </object>
```

```
 <object name="Milt">
  <object>milz</object>
 </object>

 <object name="Heart">
  <object>kind2, HerzSeg</object>
 </object>

 <object name="Liver">
  <object>kind2, LivSeg m Interp</object>
 </object>

 <object name="All">
  <object>Kidney</object>
  <object>Lung</object>
  <object>Liver</object>
  <object>Heart</object>
  <object>Milt</object>
 </object>
</ObjectNames>
```

Listing C.2: XML-Struktur für die Objektersetzungen. Das entsprechende *HighLevel*-Skript dazu findet sich in Listing C.1.

```
<HighLevel>
...
 <command commandStr="on">
  <command>T O setVisible TRUE</command>
 </command>
 <command commandStr="off">
  <command>T O setVisible FALSE</command>
 </command>
...
 <!-- View -->
 <command commandStr ="view">
  <command>T O on</command>
  <command>T 'Cam' move O P</command>
 </command>

 <!-- Komplexe Hervorhebungen -->
 <command commandStr ="emphasize">
  <command>T O view front 1</command>
  <command>T O setColor red</command>
  <command>T O setTransparency opaque</command>
 </command>
```

```
<command commandStr ="deemphasize">
 <command>T 0 setColor green</command>
 <command>T 0 setTransparency normal</command>
</command>

<!-- Laden eines Objekte -->
<command commandStr="load">
 <command>T 0 on</command>
 <command>T 0 off</command>
</command>

<!-- Grundinitialisierung -->
<command commandStr="init">
 <command>T 'System' setBackground white</command>
 <command>T 'System' setWatchUpVector true</command>
 <command>T 'System' setCreateVideo true</command>
 <command>T 'All' on</command>
 <command>T 'Lung' setColor LungColor</command>
 <command>T 'Heart' setColor HeartColor</command>
 <command>T 'Liver' setColor LiverColor</command>
 <command>T 'Milt' setColor MiltColor</command>
 <command>T 'Kidney' setColor KidneyColor</command>
 <command>T 'All' setTransparency opaque</command>
 <command>T 'Cam' move 'All' front 1</command>
</command>


<!-- Farbwerte -->
<command commandStr="setColor">
 <parameter paramStr="red" singleValue="255,0,0" />
 ...
</command>

<command commandStr="setBackground">
 <parameter paramStr="white" singleValue="255,255,255" />
 ...
</command>

<!-- Transparenzen -->
<command commandStr="setTransparency">
 <parameter paramStr="normal" singleValue="0.5" />
 <parameter paramStr="high" singleValue="0.8" />
 <parameter paramStr="low" singleValue="0.2" />
 <parameter paramStr="opaque" singleValue="0" />
 <parameter paramStr="transparent" singleValue="1" />
</command>
```

```
...

 <command commandStr="move">
  <parameter paramStr="top" singleValue="0 0" />
  <parameter paramStr="bottom" singleValue="180 0" />
  <parameter paramStr="right" singleValue="90 90" />
  <parameter paramStr="left" singleValue="90 -90" />
  <parameter paramStr="front" singleValue="90 0" />
  <parameter paramStr="back" singleValue="90 180" />

  <parameter paramStr="sagittal" singleValue="right" />
  <parameter paramStr="axial" singleValue="top" />
  <parameter paramStr="coronal" singleValue="back" />

  <parameter paramStr="slow" valuePerTimeUnit="5" />
  <parameter paramStr="fast" valuePerTimeUnit="15" />
 </command>

</HighLevel>
```

Listing C.3: Auszug aus der XML-Struktur für die Befehls- und Parameterersetzungen. Das entsprechende *HighLevel*-Skript dazu findet sich in Listing C.1.

C.2 Animation 2

In dieser Animation wird eine Leber mit einem Tumor dargestellt. Die Kamera bewegt sich im ersten Teil der Animation an den Tumor heran, wobei die umgebende Leber ausgeblendet wird und nur der Tumor selbst und der Gefäßbaum sichtbar bleiben. Anschließend werden die Segmente der Leber eingeblendet, die vom Tumor direkt betroffen sind. Zum Abschluss wird die Kamera einmal um den Tumor und die Segmente rotiert.

Ausschnitte der Animation sind in Abbildung C.2 zu erkennen. Das *HighLevel*-Skript zu dieser Animation findet sich in Listing C.4. Die für diese Animation entscheidenden Ausschnitte aus den XML-Dateien für die Objektbezeichnungen und die Anweisungsersetzungen sind in den Listings C.5 und C.6 dargestellt.

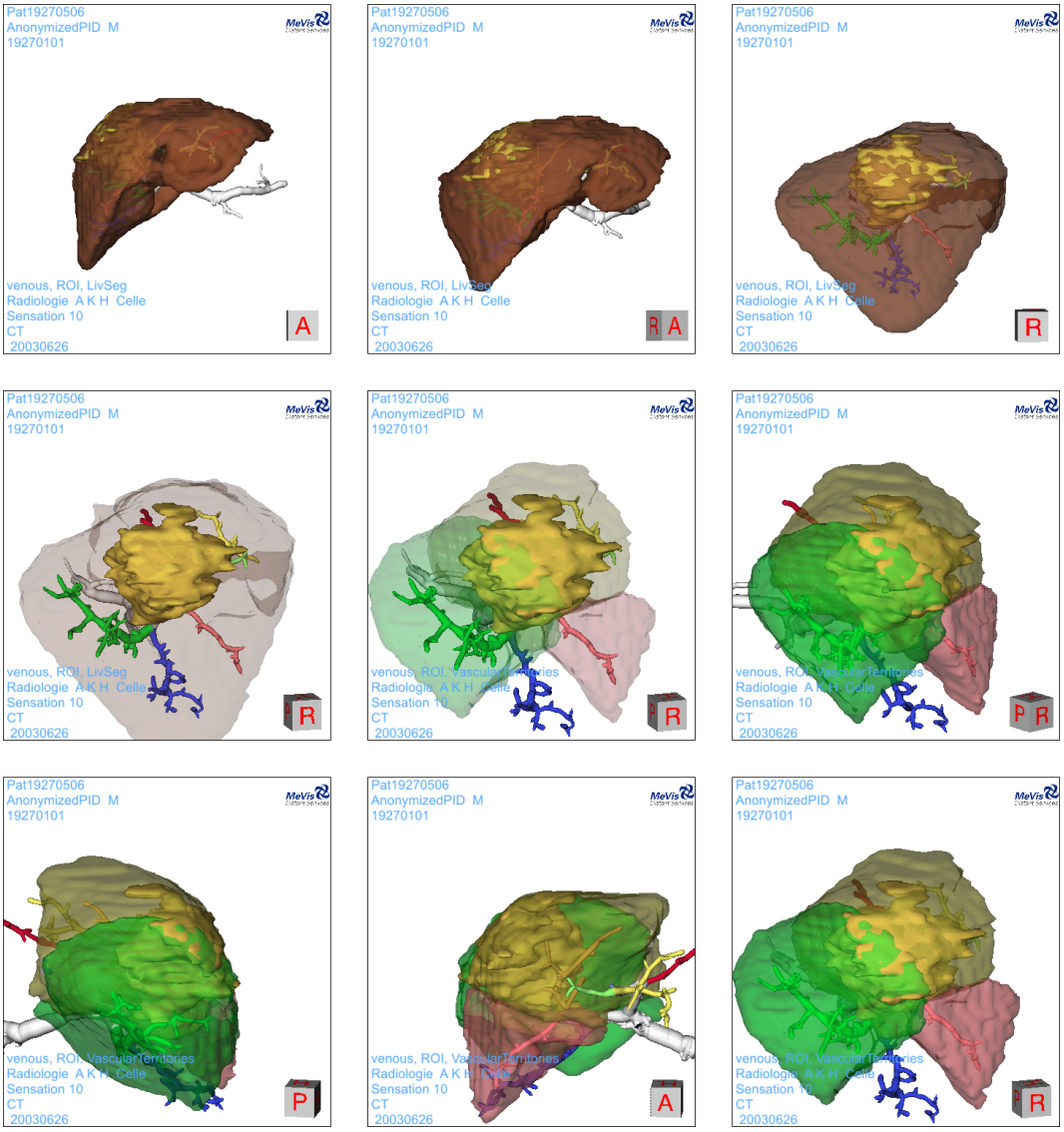


Abbildung C.2: Ausschnitte einer Animation, wie sie mithilfe des *HighLevel*-Skriptes in Listing C.4 erzeugt wurde.

```
[Ini]
LengthTimeUnit=30
LengthSeconds=30

[Script]
[0] 'System' init
[0,5] 'Cam' move 'Tumor' right 2
[5,9] 'Cam' move 'Tumor' 75 110 1.25
[5,9] 'Liver' blendOut
[10] 'Territory_IV' on
[10] 'Territory_IV' setTransparency transparent
[10] 'Territory_VII' on
[10] 'Territory_VII' setTransparency transparent
[10] 'Territory_VIII' on
[10] 'Territory_VIII' setTransparency transparent
[10,15] 'Territory_IV' setTransparency normal
[10,15] 'Territory_VII' setTransparency normal
[10,15] 'Territory_VIII' setTransparency normal
[15,30] 'Cam' rotate 'Tumor' axial 360
```

Listing C.4: *HighLevel*-Skript zur Erzeugung der Animation, wie sie in Abschnitt C.1 beschrieben wird.

```
<ObjectNames>
 <object name="Liver">
  <object>venous, ROI, LivSeg</object>
 </object>

 <object name="Tumor">
  <object>venous, ROI, TumSeg_</object>
 </object>

 <!-- Territories -->
 <object name="Territory_II">
  <object>ter_II</object>
 </object>
 <object name="Territory_III">
  <object>ter_III</object>
 </object>
 <object name="Territory_IV">
  <object>ter_IV</object>
 </object>
 <object name="Territory_IVb">
  <object>ter_IVb</object>
 </object>
 <object name="Territory_VI">
  <object>ter_VI</object>
 </object>
 <object name="Territory_VII">
  <object>ter_VII</object>
 </object>
 <object name="Territory_VIII">
  <object>ter_VIII</object>
 </object>

 <object name="Territories">
  <object>Territory_II</object>
  <object>Territory_III</object>
  <object>Territory_IV</object>
  <object>Territory_IVb</object>
  <object>Territory_VI</object>
  <object>Territory_VII</object>
  <object>Territory_VIII</object>
 </object>

 <object name="All">
  <object>Liver</object>
  <object>Tumor</object>
```

```
 </object>

</ObjectNames>
```

Listing C.5: XML-Struktur für die Objektersetzungen. Das entsprechende *HighLevel*-Skript dazu findet sich in Listing C.4.

```
<HighLevel>
...
 <!-- Objekte ein/ausblenden -->
 <command commandStr="blendOut">
  <command>T 0 setTransparency transparent</command>
 </command>
 <command commandStr="blendIn">
  <command>T 0 setTransparency opaque</command>
 </command>
...
 <!-- Grundinitialisierung -->
 <command commandStr="init">
  <command>T 'System' setBackground white</command>
  <command>T 'System' setWatchUpVector true</command>
  <command>T 'Territories' load</command>
  <command>T 'Territories' setTransparency 1</command>
  <command>T 'Liver' on</command>
  <command>T 'Liver' setTransparency low</command>
  <command>T 'Liver' setColor LiverColor</command>
  <command>T 'Vessels_Analysis' on</command>
  <command>T 'Tumor' on</command>
  <command>T 'Tumor' setColor TumorColor</command>
  <command>T 'Tumor' setTransparency opaque</command>
  <command>T 'Cam' move 'Liver' front 1</command>
 </command>
...
</HighLevel>
```

Listing C.6: Auszug aus der XML-Struktur für die Befehls- und Parameterersetzungen. Das entsprechende *HighLevel*-Skript dazu findet sich in Listing C.4. Anweisungen, die schon in Listing C.3 definiert wurden, werden hier nicht mehr mit aufgeführt.

C.3 Animation 3

Dieser Animation liegt ein HNO-Datensatz zugrunde. Es wurden verschiedene Muskeln sowie mehrere Lymphknoten segmentiert. Die Kamera bewegt sich zunächst in auf die rechte Seite des Kopfes und fährt dann auf einen Lymphknoten zu. Da dieser vom Muskel *Sternocleidomastoideus* verdeckt wird, wird dieser parallel ausgeblendet. Der Lymphknoten wird mit einer Rotfärbung hervorgehoben. Hier zeigt sich ein Problem, welches schon im Abschnitt 2.7.2 angesprochen wurde. Da auch die *Carotis* in ihrer roten Färbung erkennbar ist, verliert sich der Effekt der Hervorhebung des Lymphknotens etwas. Nach dieser Hervorhebung wird die Kamera in einer frontale Sicht auf den Tumor gefahren, dieser hervorgehoben und die Transparenz des verdeckenden Knochens verändert. Danach wird die Szene einmal axial um den Tumor rotiert.

Ausschnitte der Animation sind in Abbildung C.3 zu erkennen. Das *HighLevel*-Skript zu dieser Animation findet sich in Listing C.7. Die für diese Animation entscheidenden Ausschnitte aus den XML-Dateien für die Objektbezeichnungen und die Anweisungsersetzungen sind in den Listings C.8 und C.9 dargestellt.

```
[Ini]
LengthTimeUnit=35
LengthSeconds=35

[Script]
[0] 'System' init
[0] 'Cam' move 'All' front 1.25
[0,5] 'Cam' rotate 'All' axial 90
[5,8] 'lymphknoten 3' view right 5
[5,8] 'lymphknoten 3' setColor red
[5,8] 'SternoRight' blendOut
[10,13] 'SternoRight' view right 1
[10,13] 'lymphknoten 3' setColor LymphknotenColor
[10,13] 'SternoRight' blendIn
[13,20] 'Cam' move 'Tumor' front 1.5
[20,25] 'Knochen' setTransparency high
[20,25] 'Tumor' setColor red
[25,35] 'Cam' rotate 'Tumor' axial 360
```

Listing C.7: *HighLevel*-Skript zur Erzeugung der Animation, wie sie in Abschnitt C.3 beschrieben wird.

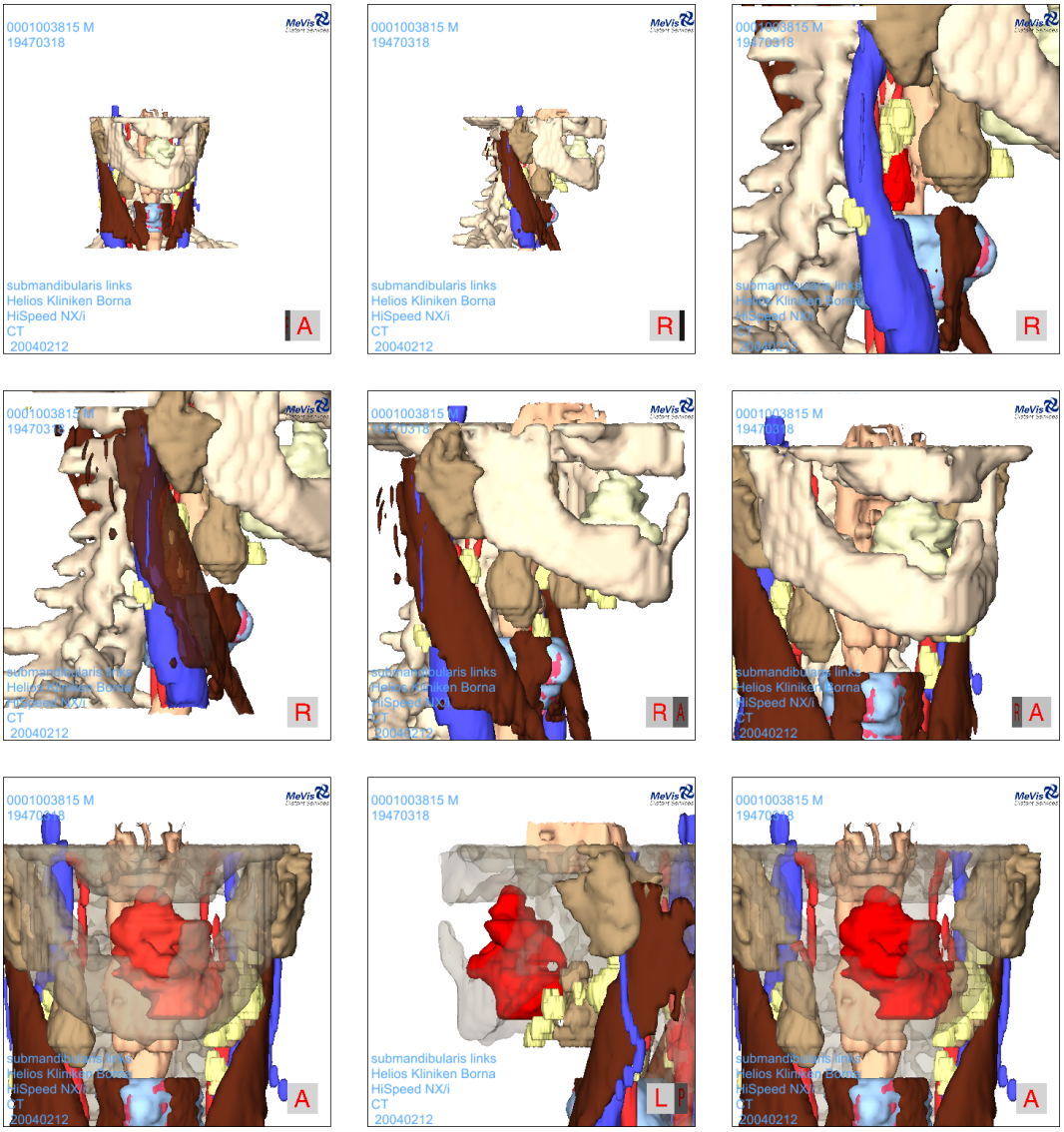


Abbildung C.3: Ausschnitte einer Animation, wie sie mithilfe des *HighLevel*-Skriptes in Listing C.7 erzeugt wurde.

```
<HighLevel>
...
 <!-- Grundinitialisierung -->
 <command commandStr="init">
  <command>T 'All' on</command>
  <command>T 'All' setTransparency opaque</command>
  <command>T 'System' setBackground white</command>
  <command>T 'System' setWatchUpVector true</command>
  <command>T 'AllLymphknoten' setColor LkColor</command>
  <command>T 'Knochen' setColor KnochenColor</command>
  <command>T 'Kehlkopf' setColor KehlkopfColor</command>
  <command>T 'Tumor' setColor TumorColor</command>
  <command>T 'PharynxTrachea' setColor PTColor</command>
  <command>T 'Parotis' setColor ParotisColor</command>
  <command>T 'Submandibularis' setColor SMColor</command>
  <command>T 'Sterno' setColor SternoColor</command>
  <command>T 'Omo' setColor OmoColor</command>
  <command>T 'Jugularis' setColor JugularisColor</command>
  <command>T 'Carotis' setColor CarotisColor</command>
 </command>
...
</HighLevel>
```

Listing C.8: XML-Struktur für die Objektersetzungen. Das entsprechende *HighLevel*-Skript dazu findet sich in Listing C.7.

```
<ObjectNames>
 <object name="AllLymphknoten">
  <object>lymphknoten 1</object>
  <object>lymphknoten 2</object>
  <object>lymphknoten 3</object>
  <object>lymphknoten 4</object>
  <object>lymphknoten 5</object>
  <object>lymphknoten 6</object>
  <object>lymphknoten 7</object>
  <object>lymphknoten 8</object>
  <object>lymphknoten 9</object>
  <object>lymphknoten 10</object>
  <object>lymphknoten 11</object>
  <object>lymphknoten 12</object>
  <object>lymphknoten 13</object>
  <object>lymphknoten 14</object>
 </object>

 <object name="PharynxTrachea">
```

```
 <object>pharynx und trachea</object>
</object>

<object name="CarotisRight">
 <object>carotis externa rechts</object>
 <object>carotis interna rechts</object>
</object>

<object name="CarotisLeft">
 <object>carotis links</object>
</object>

<object name="Carotis">
 <object>CarotisRight</object>
 <object>CarotisLeft</object>
</object>

<object name="JugularisRight">
 <object>jugularis externa rechts</object>
 <object>jugularis interna rechts</object>
</object>

<object name="JugularisLeft">
 <object>jugularis externa links</object>
 <object>jugularis interna links</object>
</object>

<object name="Jugularis">
 <object>JugularisRight</object>
 <object>JugularisLeft</object>
</object>

<object name="OmoRight">
 <object>omohyoideus rechts</object>
</object>

<object name="OmoLeft">
 <object>omohyoideus links</object>
</object>

<object name="Omo">
 <object>OmoRight</object>
 <object>OmoLeft</object>
</object>

<object name="ParotisRight">
```

```
 <object>parotis rechts</object>
</object>

<object name="ParotisLeft">
 <object>parotis links</object>
</object>

<object name="Parotis">
 <object>ParotisRight</object>
 <object>ParotisLeft</object>
</object>

<object name="SternoRight">
 <object>sternocleidomastoideus rechts</object>
</object>

<object name="SternoLeft">
 <object>sternocleidomastoideus links</object>
</object>

<object name="Sterno">
 <object>SternoRight</object>
 <object>SternoLeft</object>
</object>

<object name="SubmandibularisRight">
 <object>submandibularis rechts</object>
</object>

<object name="SubmandibularisLeft">
 <object>submandibularis links</object>
</object>

<object name="Submandibularis">
 <object>SubmandibularisRight</object>
 <object>SubmandibularisLeft</object>
</object>

<object name="All">
 <object>AllLymphknoten</object>
 <object>Tumor</object>
 <object>Kehlkopf</object>
 <object>Knochen</object>
 <object>PharynxTrachea</object>
 <object>Ringknorpel</object>
 <object>Schildknorpel</object>
```

```
  <object>Carotis</object>
  <object>Jugularis</object>
  <object>Omo</object>
  <object>Parotis</object>
  <object>Sterno</object>
  <object>Submandibularis</object>
 </object>
</ObjectNames>
```

Listing C.9: Auszug aus der XML-Struktur für die Befehls- und Parameterersetzungen. Das entsprechende *HighLevel*-Skript dazu findet sich in Listing C.7. Anweisungen, die schon in Listing C.3 und C.6 definiert wurden, werden hier nicht mehr mit aufgeführt.