RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

HOCHSCHULE HEILBRONN
TECHNIK WIRTSCHAFT INFORMATIK

Bachelor-Thesis

Entwicklung einer Programmkomponente zur effizienten Verwaltung und Adaption von Organmodellen als Grundlage für manuelle und semi-automatische Segmentierungsstrategien in der Strahlentherapieplanung

September 2011

Vorgelegt von: Bernd Hemmer
geboren: 10.04.1989 in Neuss

Referent: Prof. Dr. Rolf Bendl, Hochschule Heilbronn
Koreferent: Dr. Kristina Giske, DKFZ Heidelberg

Inhaltsverzeichnis

1 Einleitung

Nach der Operation ist die Strahlentherapie die erfolgreichste und meist genutzte Krebstherapie. Sie wird bei mehr als 50% der Patienten angewandt, die an Krebs erkrankt sind. Das Ziel der Strahlentherapie ist es, eine Strahlendosis zu applizieren, die hoch genug ist, um die Tumorzellen zu töten. Dies ist sowohl aus physikalischer als auch aus technischer Sicht häufig eine schwere Aufgabe, da besonders maligne Tumore oft sehr nahe an Organen lokalisiert werden, die eine hohe Sensitivität gegenüber Strahlen aufweisen. Hierzu zählen Organe wie die Augen, Sehnerven, der Hirnstamm oder die Lunge[Sch06]. Aus diesem Grunde muss die Strahlentherapie sorgfältig geplant werden. Dies ermöglicht es dem Benutzer, bestimmte Szenarien zu simulieren und die beste Konfiguration zu finden, um eine Strahlentherapie auszuführen. Das optimalste Ergebnis ist dabei, dass der Tumor 100% der benötigten Strahlendosis erfährt und das umliegende Gewebe sowie die umliegenden Organe vollständig geschont werden. Eine Methode der Strahlentherapieplanung ist die Virtuelle Strahlentherapiesimulation die auch Gegenstand dieser Arbeit ist.

Die 3D Strahlentherapieplanung basiert auf einer Serie von CT-Schichten und den definierten Konturen aller klinisch relevanten Volumen wie der des Tumors und der Risikoorgane [RB95]. Diese Volumen werden „Volumes of Interest“ genannt und mit VOI abgekürzt. Bevor die eigentliche Planung der Strahlentherapie beginnen kann, müssen die VOIs zunächst definiert werden, um ein Modell der individuellen Patientenanatomie zu erhalten. Dazu gibt es manuelle Segmentierungsverfahren, bei denen jede einzelne Kontur durch den Benutzer selbst definiert wird und semi-automatische Verfahren, die die Konturen durch verschiedene Algorithmen berechnen. Nach der Definition der klinisch relevanten Volumen beginnt die eigentliche Planung der Strahlentherapie mit ihren Strahlenfeldern und anderen Parametern, die eingestellt werden müssen.

1.1 Problemstellung

Die Erstellung eines individuellen Modells der Patientenanatomie ist sehr zeitaufwendig. Dies liegt daran, dass die Segmentierung der Strukturen, bedingt durch nicht vorhandene oder zum Teil nicht robuste Segmentierungsalgorithmen, sehr viel Zeit kosten. Dies limitiert wiederum die Zeit, die für die Optimierung des Strahlentherapieplans benötigt wird. Das ist weder vom behandelnden Arzt noch vom Patient, der die Therapie erwartet, gewünscht.

1.2 Zielsetzung

In dieser Arbeit soll eine Programmkomponente für das am DKFZ entwickelte Strahlentherapieplanungssystem VIRTUOS (VIRTUal RadiOtherapy Simulator) entwickelt werden, mit dem vordefinierte Organ- und Patientenmodelle effizient verwaltet werden, um diese als Ausgangspunkt für individuelle Segmentierungsansätze verwenden zu können. Diese Modelle werden für wichtige anatomische Strukturen erstellt. Dazu soll ein Modul erschaffen werden, welches ermöglicht erstellte Organmodelle mit ihren Oberflächendaten, sowie einer Beschreibung

und dem Namen in einer Datei zu speichern. Desweiteren soll es möglich sein Organmodelle zu laden, um diese dann in VIRTUOS dem aktuellen Patienten hinzuzufügen. Außerdem soll eine Möglichkeit geschaffen werden, um die Organmodelle in Gruppen und Kategorien zu unterteilen. Um die Organmodelle zu hinterlegen, soll ein Datei-/Ordnersystem geschaffen werden, welches eine hierarchische Speicherung von Kategorien, Gruppen und Modellen ermöglicht. Der Benutzer soll anschließend in VIRTUOS durch diese hierarchische angeordnete Struktur hindurch navigieren können, um ein oder mehrere Organmodelle zum Laden auszuwählen. Anschließend soll ein Modul geschaffen werden, dass dem Benutzer ermöglicht Organmodelle interaktiv im individuellen Datensatz positionieren, ausrichten und skalieren zu können. Dies ermöglicht es, dass Organmodelle als Grundlage für semi-automatische Verfahren, die Startkonfigurationen benötigen, genutzt oder manuell an die aktuelle Anatomie angepasst werden können.

1.3 Gliederung

Kapitel 2 Material und Methoden

Hier befinden sich zunächst detaillierte Beschreibungen zu den Materialien und Methoden, die während dieser Arbeit genutzt worden sind. Beginnend mit der Strahlentherapie geht es weiter über die Strahlentherapieplanung in den Abschnitt Segmentierung. Danach wird das Bestrahlungsplanungsprogramm VIRTUOS vorgestellt. Im letzten Abschnitt befinden sich Erläuterungen zu Softwarekomponenten, die in dieser Arbeit verwendet wurden.

Kapitel 3 Ergebnisse

In diesem Kapitel werden die Ergebnisse dargestellt. Dazu gehört eine kleine Einleitung in das neu entwickelte System mit einer anschließenden detaillierten Erläuterung, wie dieses Programmkomponente durch den Benutzer verwendet werden kann. Anschließend folgt eine Abschnitt in dem erläutert wird, wie das Modul in VIRTUOS integriert wurde und wie die internen Abläufe aussehen. Im vorletzten Abschnitt wird detailliert die Implementierung erläutert. Im letzten Abschnitt findet sich eine kleine Evaluierung der Komponente.

Kapitel 4 Diskussion und Ausblick

Hierin werden die Ergebnisse diskutiert und ein Ausblick gegeben, wie das System erweitert werden kann.

2 Material und Methoden

Dieses Kapitel beschreibt ausführlich die verwendeten Methoden und Materialen. Der erste Abschnitt beschreibt dabei die allgemeine Strahlentherapie mit ihrer Entstehung und ihrer Wirkungsweise. Der nächste Abschnitt beschäftigt sich näher mit der Strahlentherapieplanung und drei wesentlichen Aspekten, die für diese Arbeit von Relevanz sind. Danach folgt ein detaillierter Abschnitt über die Segmentierung und die verschiedenen Strategien, die zur Segmentierung verwendet werden. Im vorletzten Abschnitt wird das Bestrahlungsplanungsprogramm VIRTUOS, welches am Deutschen Krebsforschungszentrum (DKFZ) entwickelt wird, vorgestellt. Im letzten Abschnitt sind verschiedene Aspekte und Kriterien zur Entwicklung der Software erläutert.

2.1 Strahlentherapie

Die Strahlentherapie geht bis auf die Entdeckung der Röntgenstrahlen am 8.11. 1895 durch W.C. Röntgen zurück, wobei die Therapie mit ionisierenden Strahlen schon wenige Monate nach seiner Entdeckung in Wien begann[Cos11]. Heute ist die Strahlentherapie ein wesentlicher Bestandteil der Krebstherapie und gehört neben der Operation und der Chemotherapie zu den drei großen Säulen der Krebstherapie[WS07].

Ziel der Strahlentherapie ist es, wie in der Einleitung bereits erwähnt, eine so hohe Strahlendosis in den Tumor zu implizieren, dass die Tumorzellen absterben[Sch06]. Dabei werden verschiedene ionisierende Strahlen verwendet. Stand der heutigen Technik ist die Bestrahlung mit Gamma- und Röntgenstrahlung sowie Elektronen. Weiterhin wird intensiv an der Verwendung von Teilchenstrahlung in Form von Neutronen, Protonen und Schwerionen geforscht. Dafür wurden in den letzten Jahren mehrere Große Anlagen wie zum Beispiel das Heidelberger Ionenstrahl-Therapiezentrum (HIT) gebaut, um zu zeigen, dass diese Art der Therapie erfolgreich ist[Wik11e].

Grundsätzlich gibt es bei der Behandlung mit ionisierender Strahlung zwei Arten von Wechselwirkungen, die für den Organismus schädlich sind.

Zum einen gibt es dort die direkte Strahlenwirkung. Bei dieser Art von Strahlenwirkung werden ein oder mehrere Bindungselektrone aus einem Molekül herausgeschossen, wodurch dieses seine biologische Wirkung verliert und im Extremfall ein Absterben der Zelle verursacht. Häufiger ist die Entstehung von Mutationen in der dann, die ebenfalls für den Körper schädlich seien können.

Die zweite Art der Strahlenwirkung ist die indirekte Strahlenwirkung. Hier wird aus einem Wassermolekül ein Elektron ebenfalls herausgelöst, wodurch dieses Molekül zu einem hochreaktiven Radikal wird. Diese Art von Radikal reagiert dabei besonders häufig mit Nukleotidbasen, wodurch die DNA ihre Wirkung verliert und die Zelle abstirbt. Diese Art der Strahlenwirkung ist die deutlich schädlichere für den Organismus und damit auch für den Tumor und daher für die Strahlentherapie von entscheidender Bedeutung.[Ben11]

Zur Bestrahlung von Patienten gibt es mittlerweile eine Vielzahl von Geräten. Zu den bekanntesten gehört der Elektronen-Linearbeschleuniger(LINAC). Dieser kann einen Patienten sowohl mit Elektronen als auch mit ultraharter Röntgenstrahlung bestrahlen. Über ein Beschleunigerrohr wird dabei ein Elektronenstrahl beschleunigt und entweder direkt auf das zu bestrahlende Ziel geschossen oder es wird eine Wolfram-Scheibe dazwischen gelegt, wodurch beim Aufschlagen der Elektronen auf diese Scheibe Bremsstrahlung in Form von ultraharter Röntgenstrahlung entsteht, welche dann auf das zu bestrahlende Ziel geführt wird. Mit einem Elektronen-Linearbeschleuniger kann dank der Gantry, die um ihre Achse rotiert werden kann und der Couch, die frei auf dem Boden bewegt werden kann, aus nahezu jeder Position bestrahlt werden.

Abbildung 1: *Links ist ein Elektronen-Linearbeschleuniger abgebildet und rechts eine Tomotherapie-Gerät, welches auf dem gleichen Prinzip beruht, aber weniger Freiheitsgrade besitzt, dafür aber auch als CT fungieren kann[Ben11].*

Fraktionierung

Ein Vorteil, den man bei der Strahlentherapie nutzt ist, dass Tumorzellen in der Regel eine schlechtere Reparaturfähigkeit für DNA-Schäden besitzen als normale Zellen. Dadurch kann die Gesamtdosis auf tägliche Einzeldosen verteilt werden. Dadurch verringert sich die bei gleicher Dosis abgetötete Zellanzahl. Dies hat zur Folge, dass die maximale Toleranzdosis für normales Gewebe auf ein vielfaches gesteigert werden kann[Wik11e].

Weitere Techniken zur Strahlentherapie, wie zum Beispiel die Intensitätsmodulierte Strahlentherapie, sind in [WS06] zu finden.

2.2 Strahlentherapieplanung

Die Strahlentherapieplanung ist ein weiteres wichtiges Element in der Strahlentherapie. Sie dient sowohl zur Planung der gesamten Therapie als auch zur Optimierung dieser. Es gib vielfältige Möglichkeiten, die in einem Bestrahlungsplanungsprogramm enthalten seien können. Dazu gehören Mittel wie die Dosisberechnung, Einstellung der Strahlenfelder, Festlegung des Zielvolumens sowie Möglichkeiten zur Segmentierung von umliegenden Organen, die auch Risikoorgane genannt werden.

Die Strahlentherapieplanung beginnt mit der Aufnahme einer CT-Bildserie, auf der die Planung erfolgen soll. Eventuell mit Hilfe von weiteren Bildserien, zum Beispiel eine MRT-Bildserie oder einer CT-Aufnahme mit Kontrastmitteln, wird eine 3D-Patientenmodell erstellt, bestehend aus dem Zielvolumen und den Risikoorganen. Dieses 3D-Patientenmodell ist meist nicht nur durch die entsprechenden Konturen in den CT-Schichtbildern zu sehen, sondern auch noch in einer speziellen 3D-Ansicht. Danach erfolgt die Erstellung der Strahlenfelder. Je nach Art der Bestrahlungsplanung wird dies von dem Strahlentherapeut selbst gemacht oder durch ein Programm, nach Einngabe von spezifischen Parametern, berechnet. Letzterer Vorgang nennt sich „Inverse Strahlentherapieplanung“. Nach erfolgreicher Erstellung der Strahlenfelder beginnt dann die Dosisberechnung. In folgender Abbildung ist eine Dosisberechnung innerhalb einer Schicht zu sehen:

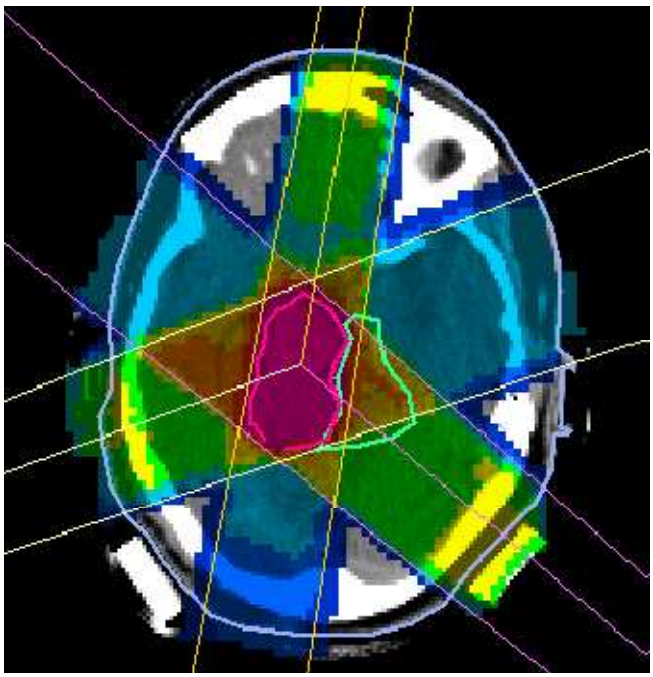

Abbildung 2: *Darstellung einer Dosisberechnung in einer CT-Schicht. Die rot gekennzeichneten Bereiche stellen eine relativ hohe Dosis dar. Die blau gekennzeichneten Bereiche stellen eine relativ niedrige Dosis dar.*

Ist die Dosisberechnung erfolgt, kann nun anhand von Dosis-Volumen-Histogrammen, die aufzeigen wie viel Prozent einer Dosis ein bestimmtes Volumen bei der Bestrahlung treffen, der Bestrahlungsplan optimiert werden. Dies wird solange in einem iterativen Prozess durchgeführt, bis der Strahlentherapeut mit dem Plan zufrieden ist.

In den folgenden Abschnitten werden Strukturen, die in einem Bestrahlungsplan vorkommen und die für die Arbeit von Relevanz sind, näher erläutert.

2.2.1 Zielvolumen

Das Zielvolumen ist das Volumen das bestrahlt werden soll. Es wird vom Strahlentherapeut anhand von bestimmten Parametern festgelegt. Der Aufbau eines Zielvolumens kann grundsätzlich in 4 Bereiche aufgeteilt werden. Der innerste Bereich wird Gross Tumor Volume genannt (GTV) und stellt den sichtbaren malignen Bereiches des Tumors dar. Das Clinical Target Volume (CTV) beinhaltet das GTV und einen weiteren Bereich, der außerhalb des GTV liegt und der bereits durch den Tumor befallen ist, aber nicht sichtbar ist. Auf das CTV folgt das Internal Target Volume (ITV). Es enthält das CTV und einen internen Abstand, welcher dafür konzipiert wurde, Bewegungen und Variationen des CTV bezüglich der Größe, Form und Position zu kompensieren. Darauffolgend kommt das Volumen, welches vom Strahlentherapeut in das Bestrahlungsprogramm eingezeichnet wird. Es heißt Planning Target Volume (PTV) und beinhaltet das ITV und einen weiteren Sicherheitsabstand, um Einstellungsfehler zu kompensieren. Das PTV wird so gezeichnet, dass sichergestellt ist, dass das CTV die vorher festgelegt Dosis komplett erhält[Ben11].

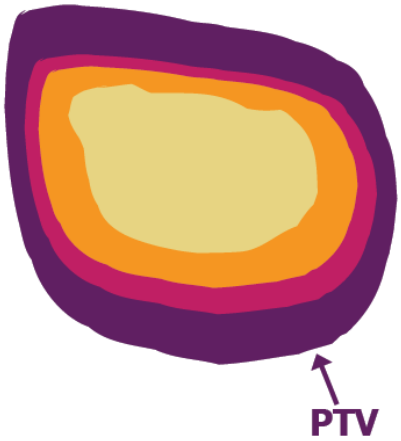

Abbildung 3: *Hier wir das Zielvolumen mit seinen vier Grundelementen schematisch dargestellt. Der beige eingefärbte Bereich stellt das GTV dar. Die orange Fläche das CTV und die pinke Fläche das ITV[Ben11].*

Das Zielvolumen wird im Regelfall komplett manuell durch den Strahlentherapeuten erstellt, da Tumore meistens nur schlecht durch Algorithmen segmentierbar sind. Dies liegt vor allem daran, dass diese auf CT-Bildern häufig nur schlecht, teilweise sogar gar nicht sichtbar sind. Der Strahlentherapeut zeichnet dann anhand von anderen Schnittbildaufnahmen, zum Beispiel einer MRT-Bildserie das PTV ein.

2.2.2 Risikoorgane

Ein weiterer Schritt um das 3D-Patientmodell eines Patienten zu erstellen ist die Definition der Risikoorgane. Risikoorgane sind, wie der Name schon sagt, die Organe, die bei einer Bestrahlung unter Risiko stehen, weil diese der implizierten Strahlung mit ausgesetzt sind. Als Grenzwerte wurden rechtlich festgelegte Toleranzdosen geschaffen, die sicherstellen, dass keine irreparablen Schäden an den umliegenden Organen entstehen. Risikoorgane sind besonders dann gefährdet, wenn der zu behandelnde Tumor von maligner Art ist. Solche Tumore grenzen sich von anderem Gewebe nicht ab sondern wachsen meist in anderes Gewebe hinein und liegen dadurch sehr nahe an Risikoorganen.

Heutzutage werden Risikoorgane in die drei Kategorien seriell, parallel und seriell-parallel unterteilt. Risikoorgane die als seriell klassifiziert sind, können ihre komplette Organfunktion, wenn auch nur

ein kleiner Teil des Organs über die erlaubte Toleranzdosis bestrahlt wird, verlieren. Ein Beispiel hierfür ist zum Beispiel der Sehnerv. Im Gegensatz dazu können Risikoorgane, die als parallel klassifiziert werden, wie zum Beispiel die Niere oder die Lunge deutlich größere regionale Schäden aushalten, ohne die komplette Funktion zu verlieren. Serial-parallel Risikoorgane stellen eine Kombination dieser beiden dar[ALG06].

Risikoorgane werden allerdings nicht von Strahlentherapeuten segmentiert, sondern dies wird entweder durch besonders geschulte Zivildienstleistende oder von Medizinisch-technischen Radiologieassistenten durchgeführt. Dafür gibt es in einem Bestrahlungsplanungsprogramm meist mehrere Methoden die auf manuelle oder semi-automatische Segmentierungsstrategien setzen.

2.2.3 Adaptive Strahlentherapie

Ein häufiges Problem, dass bei der Strahlentherapie vorkommt, ist die Bewegung der Anatomie des Körpers. Innerhalb einer Therapiesitzung kann dies die Atembewegung bei der Behandlung eines Lungentumors sein oder zwischen verschiedenen Therapiesitzungen die verschiedenen Füllstände von Blase und Darm sein, welche die umliegenden Organe verschieben.

Bevor Konzepte der Adaptiven Strahlentherapie (Image Guided Radio Therapy – IGRT) in die Überlegung zur Kompensation solcher Veränderung der Anatomie eingebunden wurden, wurde das Zielvolumen durch Sicherheitszonen vergrößert, um auf jeden Fall den Tumor komplett zu treffen. Dies führte zu hohen Belastungen des gesunden Gewebes und dies wiederum zu Komplikationen nach der Therapie. Die Adaptive Strahlentherapie sieht vor, das PTV, mit seinem kompletten Volumen und damit die Therapie an die Veränderung der Anatomie anzupassen[Ben11].

Der Ansatz der Adaptiven Strahlentherapie unterteilt sich in vier weitere Kategorien, die sich in ihrer Komplexität, aber auch in der Optimierung der Therapie steigern. Die Kategorie 1 sieht vor, dass die Planung einer Therapie auf mehreren CT-Serien durchgeführt. Diese CT-Serien werden zu unterschiedlichen Zeitpunkten durchgeführt und ermöglichen so das Erkennen von Organbewegung. Dies ermöglicht die Minimierung von Sicherheitsbereichen. Die nächste Steigerung dazu ist die Kategorie 2, die es ermöglicht, Lageveränderung direkt vor einer Fraktion zu erkennen. Dazu wird direkt vor einer Therapiesitzung ein Kontroll-CT durchgeführt und mit dem Planungs-CT verglichen. Organbewegungen können dadurch erkannt und im optimalen Fall direkt im Bestrahlungsplan korrigiert werden. Die Kategorie 3 der Adaptiven Strahlentherapie sieht vor, entweder bei der Planung der Therapie gleich mehrere Pläne zu erstellen um für jede Situation den besten Auswählen zu können oder nach der Durchführung des Kontroll-CTs einen neuen Bestrahlungsplan zu erstellen. Die letzte Kategorie 4 ist gerade für die Kompensation von Bewegung, die durch die Atembewegung entsteht gut geeignet. Insgesamt gibt es zwei Ansätze, die dies ermöglichen, die jedoch dafür eine Online-Bildgebung während der Therapie benötigen. Das Prinzip „Gating“ sieht vor, die Bewegung eines Zielvolumens innerhalb einer Fraktion zu erkennen und nur dann zu bestrahlen, wenn sich das Zielvolumen wieder an einer bestimmten Position befindet. Das zweite Prinzip „Tracking“ bedeutet die Verfolgung des Zielvolumens innerhalb der Bestrahlung durch Online-Bildgebung und die Anpassung des Strahls an die momentane Position des Zielvolumens[Ben11].

Um diese Art der Strahlentherapie zu ermöglichen, müssen zum einen die Bestrahlungsplanungsprogramme eine schnelle Planung, in Form von schnellen Segmentierungs- und Berechnungsalgorithmen, unterstützen. Zum anderen muss es mit den Linearbeschleunigern möglich sein, eine Bildgebung während der Bestrahlung zu ermöglichen oder kurz vor der Bestrahlung. Dazu

werden entweder CTs in den Raum des Linearbeschleunigers integriert, um Bilder direkt vor der Therapiesitzung zu ermöglichen, oder an den Linearbeschleuniger werden Detektorpanels angebaut, die eine Online-Bildgebung ermöglichen. Weiterhin gibt es noch die Möglichkeit, eine weitere Röntgenröhre sowie eine Detektorplatte an einen Beschleuniger anzubringen, um Röntgenaufnahmen zu machen. Folgende Abbildungen zeigen solche Systeme:

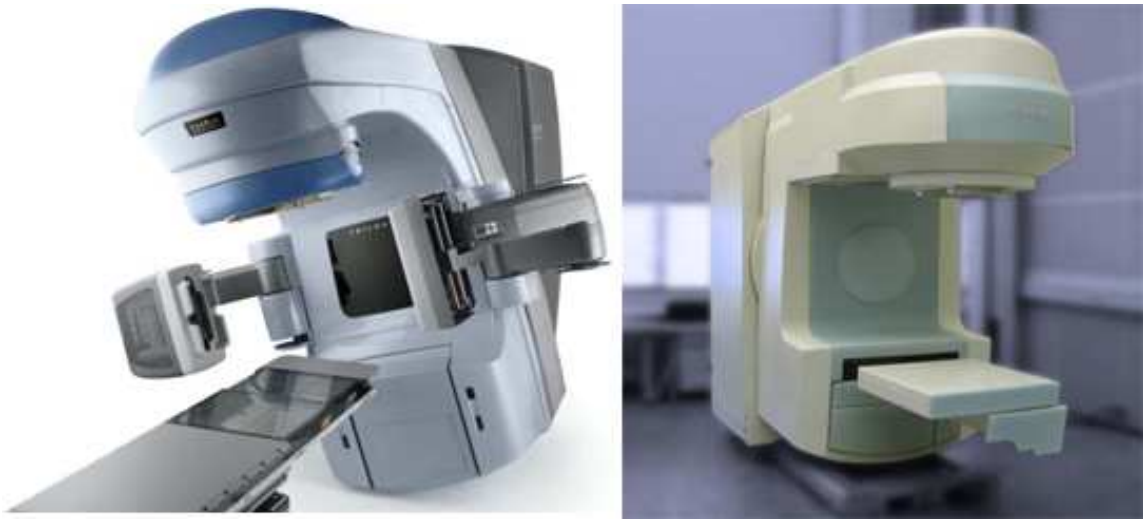

Abbildung 4: *Links ist ein Linearbeschleuniger mit einer weiteren Röntgenröhre und einem Detektorpanel zu sehen. Rechts ist ein Linearbeschleuniger mit einer ausfahrbaren Detektorplatte zu sehen[Ben11].*

2.3 Segmentierung

Segmentierung bedeutet allgemein die Einteilung eines Bildes in örtlich zusammenhängende Bereiche. Je nach der abstrakten Ebene, auf der ein Segmentierungsverfahren ansetzt, unterscheidet man methodisch die pixel-, kanten- und textur- bzw. regionen-orientierte Verfahren. Darüber hinaus existieren Segmentierungsverfahren, die sich aus Kombination einzelner Ansätze ergeben[Des11].

Die Segmentierung von medizinischen Bildern geschieht dabei entweder über die digitalisierten Grauwertbilder, berechnet durch CT- oder MRT-Aufnahmen, oder basiert direkt auf den Werten, die von einem CT oder MRT ausgegeben werden. Besonders die Hounsfield-Werte, die durch ein CT pro Pixel in einer Schicht ausgegeben werden, sind ein wichtiges Element. Die Hounsfield-Skala beschreibt die Dichte, die ein Pixel in einer CT-Aufnahme hat und reicht von -1024 bis 3071. Der Wert -1024 steht dabei für Luft. Dies bietet den Vorteil, dass durch die möglichen 4096 Werte eine CT-Aufnahme eine höhere Varianz bietet als ein Grauwertbild, das lediglich durch 256 Werte dargestellt werden kann. Zusätzlich können Werte auf der Hounsfield-Skala bestimmten Gewebearten zugeordnet werden.

Die Segmentierung von medizinischen Bildern erlaubt es uns, aus einer CT-Bildserie einen 3D Patientenmodell zu erstellen. Dies erst ermöglicht eine auf den Patienten abgestimmte Planung der Strahlentherapie zu erstellen.

In den nachfolgenden Abschnitten finden sich nähere Erläuterungen zu Begriffen der Segmentierung und zu verschiedenen Segmentierungsstrategien.

2.3.1 Volume of Interest

Grundsätzlich besteht ein 3D-Patientenmodell aus mehreren „Volume of Interests“ (VOIs). Ein „Volume Of Interest“ stellt ein Volumen dar, das für den Strahlentherapeut von Relevanz innerhalb der Strahlentherapie bzw. der Strahlentherapieplanung ist. Dies bedeutet, dass VOIs sowohl das Zielvolumen als auch Risikoorgane definieren. Weiterhin werden oft sogenannte Hilfs-VOIs, die beim Erstellen der Strahlenfelder hilfreich sind hinzugefügt.

Das Pendant zum VOI in einem zwei-dimensionalen Raum stellt die Region of Interest (ROI) dar. Eine ROI ist das Ergebnis der zwei-dimensionalen Segmentierung einer Region, die für den Strahlentherapeut von Interesse ist.

Das Erstellen eines 3D-Patienmodells kann auf verschiedene Weise durchgeführt werden. Eine einfache Methode ist das Bilden von Konturlinienstapeln. Die mit Hilfe von Segmenterungsverfahren extrahierten und über mehrere Schichten hinweg zu den VOIs zusammengefassten ROIs bilden so zusammen das gewünschte Modell der Patientenanatomie[Fis96].

Die 3D-Darstellung eines VOI kann nicht nur durch einen Konturlinienstapel realisiert werden. Häufig wird aus den segmentierten Daten ein trianguliertes Modell mit Hilfe von Algorithmen wie der Delauney-Triangulation oder dem Marching-Cube Verfahren erstellt. Ein trianguliertes VOI ist meist viel detaillierter als ein VOI, welches nur aus Konturlinien besteht. In folgender Abbildung sind mehrere VOIs im Kopfbereich zu sehen.

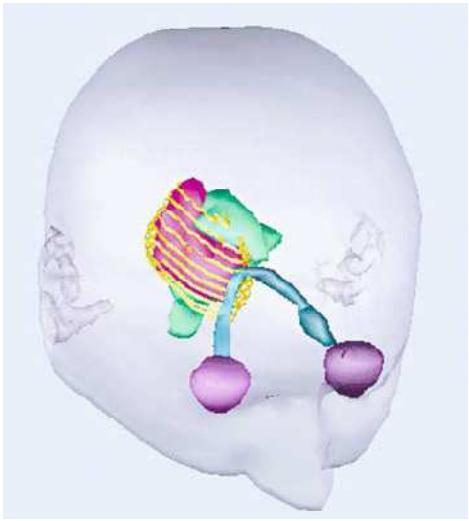

Abbildung 5: *Darstellung eines 3D-Patientmodells mit mehreren VOIs[DKF05].*

Das rote, durch gelbe Linien umgebene VOI ist das Zielvolumen. Die umliegenden VOIs stellen die Risikoorgane dar.

2.3.2 Manuelle Segmentierung

Ein sehr weit verbreitetes Verfahren, um VOIs zu extrahieren ist die manuelle Segmentierung. Dabei wird per Hand in einer Schicht eine ROI definiert. Klassisch gesehen heißt dies, dass der Benutzer jeden Punkt der Konturlinie, die die ROI umgibt, manuell setzen muss. Häufig gibt es in den Bestrahlungsplanungsprogrammen allerdings Methoden, um dies zu vereinfachen. Als Beispiel wird hier das Bestrahlungsplanungsprogramm VIRTUOS, welches am DKFZ entwickelt wird, genommen.

In diesem ist es möglich, Konturlinien von einer CT-Schicht in die darunter oder darüber liegende Schicht zu kopieren. Desweiteren ist es möglich, einfache geometrische Figuren wie zum Beispiel einen Kreis in eine Schicht zu legen, um ihn anschließend zu vergrößern oder zu verkleinern. Gerade das Kopieren der Konturlinien ist eine Funktion, die häufig gebraucht wird, da die Unterschiede eines VOI zwischen zwei ROIs meist nicht sehr groß sind[Rüp09]. Ebenfalls eine Funktion, die sehr nützlich ist, wurde in der Arbeit [Rüp09] implementiert. In dieser Arbeit wurde ein Toolkit zur 3D-Bearbeitung von Konturlinien erstellt. Beim Bearbeiten eines Punktes innerhalb einer Konturlinie werden umliegende Punkte nach mathematischen Modellen mit bearbeitet.

Ein großer Nachteil der manuellen Segmentierung ist jedochoffensichtlich. Der Zeitaufwand, um ein komplettes VOI zu segmentieren ist zu hoch und hängt sehr von der Erfahrung des Benutzers ab. Ebenso ist die Qualitätder Segmentierung stark von der Erfahrung des Benutzers abhängig. Andererseits bietet gerade dann die manuelle Segmentierung eine gute Unterstützung zu automatisierten Segmentierungsalgorithmen, wenn deren Ergebnisse verbessert werden müssen. Deshalb ist es zwingend, die Möglichkeit zur manuellen Segmentierung in ein Bestrahlungsplanungsprogramm zu integrieren.

2.3.3 Semi-automatische Segmentierung

Da die manuelle Segmentierung sehr viel Zeit in Anspruch nimmt, sind im Laufe der Zeit einige semi- und voll-automatische Segmentierungsalgorithmen entwickelt worden, die die Segmentierung einfacher und auch schneller machen. Dabei gibt es eine Vielzahl von Ansätzen, die aber alle auf die in Kapitel 2.3 erwähnten Ebenen zurück geführt werden können. Semi-automatische Segmentierungsalgorithmen unterscheiden sich von voll-automatischen Varianten dadurch, dass sie bestimmte Parameter benötigen. Dies sind meist Parameter, wie Schwellwerte, Saatpunkte oder Startkonturen. In diesem Abschnitt werden einige Algorithmen, welche auch in VIRTUOS implementiert wurden, vorgestellt.

Schwellwert

Eine Segmentierung mit einem Schwellwertverfahren ist eine einfache aber nicht robuste Variante. Sie kann dann gut eingesetzt werden wenn, sich die segmentierenden Strukturen gut voneinander abheben[Ben11]. Die Segmentierung selbst geht sehr schnell. Es wird jedes Pixel hinsichtlich seines Wertes untersucht. Wenn der Wert einen Schwellwert überschreitet, unterschreitet oder zwischen zwei Schwellwerten liegt, wird er der entsprechenden Region zugeordnet. Gerade bei Bildern mit wenig Kontrast, vielen Artefakten oder viel Rauschen ist der Einsatz dieses Algorithmus nicht sinnvoll. Deshalb werden bei dieser Methode vorverarbeitende Filter eingesetzt, die das Bild glätten, wie zum Beispiel ein Medianfilter.

Kantendetektion

Ein weitere Variante, um Regionen zu segmentierten, ist die Suche nach Kanten. Dies bedeutet, dass starke Veränderungen hinsichtlich des Grauwertes gesucht werden. Kantenorientierte Segmentierungsverfahren sind daher nur in solchen Fragestellungen einsetzbar, in denen die Objekte mit klaren Umrandungen dargestellt werden[Des11]. Dies bedeutet, dass die Bilder an den Kanten einen hohen Kontrast haben müssen. Die meisten Algorithmen funktionieren auf diese Weise. Zunächst wird entlang einer Bildzeile eine Kante gesucht. Danach wird diese Linie verfolgt, bis diese am Startpunkt der Kante wieder auskommt. Zur Kantenverfolgung wird der Umkreis des letzten Kantenpunktes nach Punkten mit ähnlichen Werten durchsucht. Der Punkt mit der höchsten Ähnlichkeit wird dann der Kante hinzugefügt. Auch bei diesem Verfahren werden vor dem eigentlich Verfahren Filter angewandt, um die Qualität des Ergebnisses zu erhöhen.

Volume/Region Growing

Dieser Algorithmus ist den Schwellwertverfahren zu zuordnen. Vor der Durchführung wird daher ein Medianfilter angewandt, um die Qualität des Ergebnisses zu erhöhen. Ein Medianfilter glättet eine Region, in dem dieser einen stark abweichenden Pixelwert durch den Median der Pixelwerte, die in dieser Region enthalten sind, ersetzt. Danach muss der Benutzer einen Saatpunkt setzen. Von diesem Saatpunkt aus werden alle umliegenden Punkte dahingehend untersucht, ob sie anhand eines durch den Benutzer vorher festgelegten Ähnlichkeitskriteriums zu der zu segmentierenden Region hinzugefügt werden müssen oder nicht. Bei allen neu hinzugefügten Punkten wird nun ebenfalls die Umgebung nach ähnlichen Punkten untersucht. Dies wird so lange durchgeführt bis keine neuen Punkte gefunden werden. Der Unterschied zwischen Volume- und Region-Growing besteht darin, dass beim Volume-Growing auch Punkte in der jeweils unteren und oberen Schicht untersucht werden.

2.3.4 Modellbasierte Segmentierung

Diese Art von Segmentierung wird sowohl den semi- als auch den voll-automatischen Segmentierungsstrategien zugeordnet, beinhaltet allerdings selbst vieleVarianten, die für die Arbeit von hoher Relevanz sind.

Mit diesen Segmentierungsansätzen wird versucht, die menschliche Wahrnehmung also Vorbild zu sehen. Der Mensch erkennt viele Dinge aufgrund seiner Erfahrung und ordnet sie so zu.

Abbildung 6: *Darstellung verschiedener Modelle einer Tasse[MH11]*

Der Mensch weiß dank seiner Erfahrung, dem so genannten A-priori-Wissen, dass dies Oberflächen von verschiedenen Tassen sind. Würde der Computer nun alle diese 5 Tassen kennen und müsste eine dieser Tassen segmentieren, so reicht ein einfacher Vergleich mit allen 5 Tassen aus, um diese zu identifizieren und zu segmentieren. Dies bringt zum einen den Vorteil, dass der Computer weiß, um was für ein Objekt es sich handelt, andererseits ist das Objekt auch segmentiert. Nachfolgend werden mehrere Algorithmen erläutert die, auf A-priori-Wissen aufbauen.

Aktive Konturen

Diese Segmentierungsstrategie baut darauf, dass sich eine Kontur an Grauwertinformation anpassen kann[Ben11]. Der Benutzer setzt an der Stelle, an der die entsprechende Kontur angepasst werden soll, eine Startkontur. Entweder zeichnet der Benutzer diese Kontur per Hand oder er wählt aus bereits vordefinierten Konturen eine aus. Durch bestimmte Bildmerkmale, die als externe Kräfte gesehen werden, wird die Kontur angezogen. Stärke und Reichweite der Anziehung hängt zum Beispiel vom Betrag des Gradienten ab. Die Kante besitzt dabei eine gewisse Steifigkeit, die regelt, inwiefern sich diese Kontur deformieren lässt[Ben11]. Eine Optimierung findet bis zu einem gewissen Kräftegleichgewicht statt.

Statistische Formmodelle

Statistische Formmodelle sind eine populäre Methode um den Segmentierungsprozess in der Medizinischen Bildverarbeitung zu automatisieren[MS07]. Dieser Algorithmus baut auf Trainingsdatensätze, die zu einem gemeinsamen mittleren Formmodell zusammengefasst werden. Dieses Formmodell kann sich in dem Rahmen der Trainingsdatensätze verändern und dadurch an Bildinformationen anpassen.

Zunächst müssen Trainingsdatensätze segmentiert und dann deren Oberflächen parametrisiert werden. Dies lässt sich an Abbildung 6 genauer beschreiben. Parametrisierung bedeutet, dass die Landmarke die die linke obere Ecke der ersten Tasse beschreibt die gleiche Landmarke ist die die linke obere Ecke der zweiten Tasse beschreibt. Nach erfolgter Parametrisierung wird aus allen Datensätzen ein gemeinsames Formmodell errechnet.

Möchte der Benutzer nun ein Objekt, das durch das Formmodell beschrieben wird, in einem Datensatz segmentieren, so muss er das Formmodell platzieren und danach passt es sich an die Grauwertinformationen der Bildserie im Rahmen der zu Grunde liegenden Trainingsdatensätze an.

Atlasbasierte Segmentierung

Bei der Segmentierung mit Hilfe von Atlanten wird von einem segmentierten Referenzdatensatz, dem Atlas, die Segmentierung auf den Patientendatensatz übertragen[MS09]. Dafür muss zunächst ein Atlas erstellt werden. Dazu werden alle Organe von einem vorhandenen Bilddatensatz segmentiert und in einer Datenbank gespeichert. Es kann dabei durchaus hilfreich sein, mehrere Datensätze in einem Atlas zu speichern, um eine hohe Varianz zu erlangen. Danach ist es möglich, jeweils einen Datensatz auf einen anderen Patienten zu übertragen. Da nicht jeder Patient die exakt gleichen anatomischen Strukturen aufweist, muss zwischen dem Atlasdatensatz und dem Patientendatensatz eine Registrierung durchgeführt werden. Die einfache Variante wäre dabei eine rigide Registrierung, die allerdings Schwächen im Bezug auf die Deformation der Organe im Patientendatensatz aufweist. Daher wird die elastische Registrierung bevorzugt.

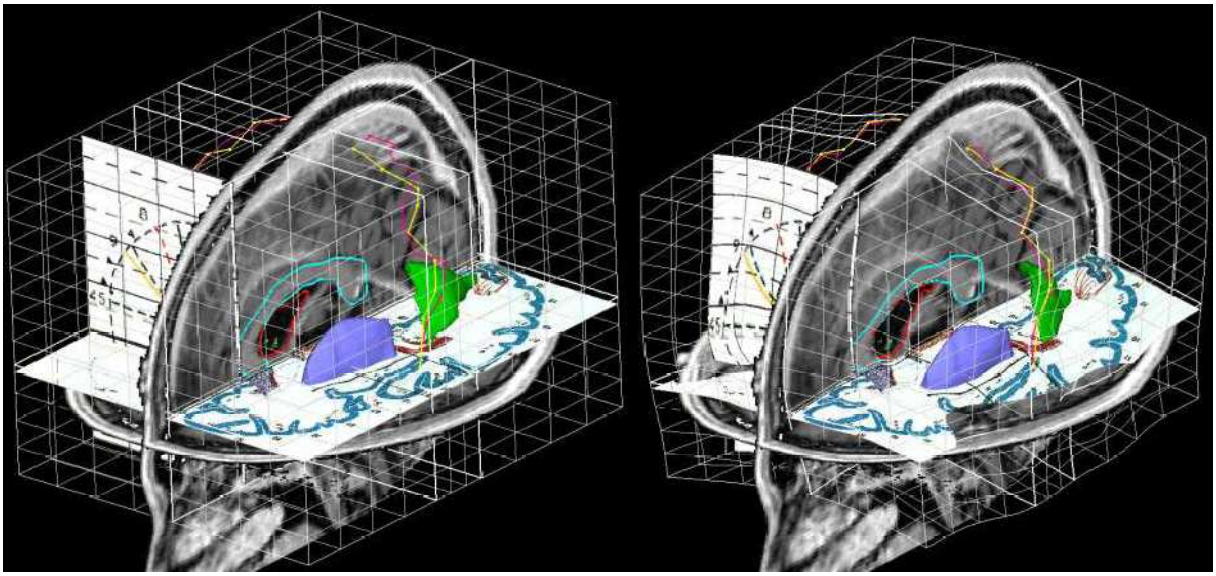

Abbildung 7: *Darstellung der Anpassung eines Atlasdatensatz auf einen Patientendatensatz mit Hilfe der elastischen Registrierung[Eis11].*

2.4 VIRTUOS

Das Programm VIRTUOS (VIRTUal RadiOtherapy Simulator) dient zur virtuellen Simulation von Strahlentherapie und wird am Deutschen Krebsforschungszentrum (DKFZ) entwickelt. Die Ursprünge dieses Programms liegen in der Arbeit [Ben91]. Seitdem wurde das Programm sowohl durch viele wissenschaftliche Arbeiten als auch durch die Arbeitsgruppe Therapieplanung – Entwicklung stetig erweitert. Auch diese Arbeit wurde deshalb für VIRTUOS entwickelt und in das Programm integriert. Die Sprache des Programms ist English.

Seit 1991 wird VIRTUOS auch in der Strahlentherapieklinik des DKFZ als Therapieplanungssystem eingesetzt, wodurch es alle Methoden beinhaltet, eine komplette Therapieplanung durchzuführen[DKF11b]. Es sind zahlreiche manuelle und automatische Segmentierungsmethoden implementiert, die es ermöglichen, ein 3D-Patienmodell zu erstellen. Weiterhin können mit Hilfe verschiedenster Ansichten auf die CT-Schichten als auch auf das Patientenmodell Strahlenfelder erstellt werden und es ist möglich die Dosisverteilung zu berechnen. Darüber hinaus ist es möglich, mit dieser Software verschiedene Bestrahlungspläne zu vergleichen.

2.4.1 Graphische Benutzeroberfläche

Die graphische Benutzeroberfläche von VIRTUOS basiert auf dem Toolkit Motif, die unter dem X-Window System genutzt wird. Zwar ist die Bibliothek nicht mehr die aktuellste, jedoch ist sie nach IEEE 1295 ein Standard und ist häufig in Software zu finden, die ihren Ursprung in den 90er Jahren hat. Zusätzlich wird Motif auch noch heute durch das Portal MotifZone weiterentwickelt[Wik11c].

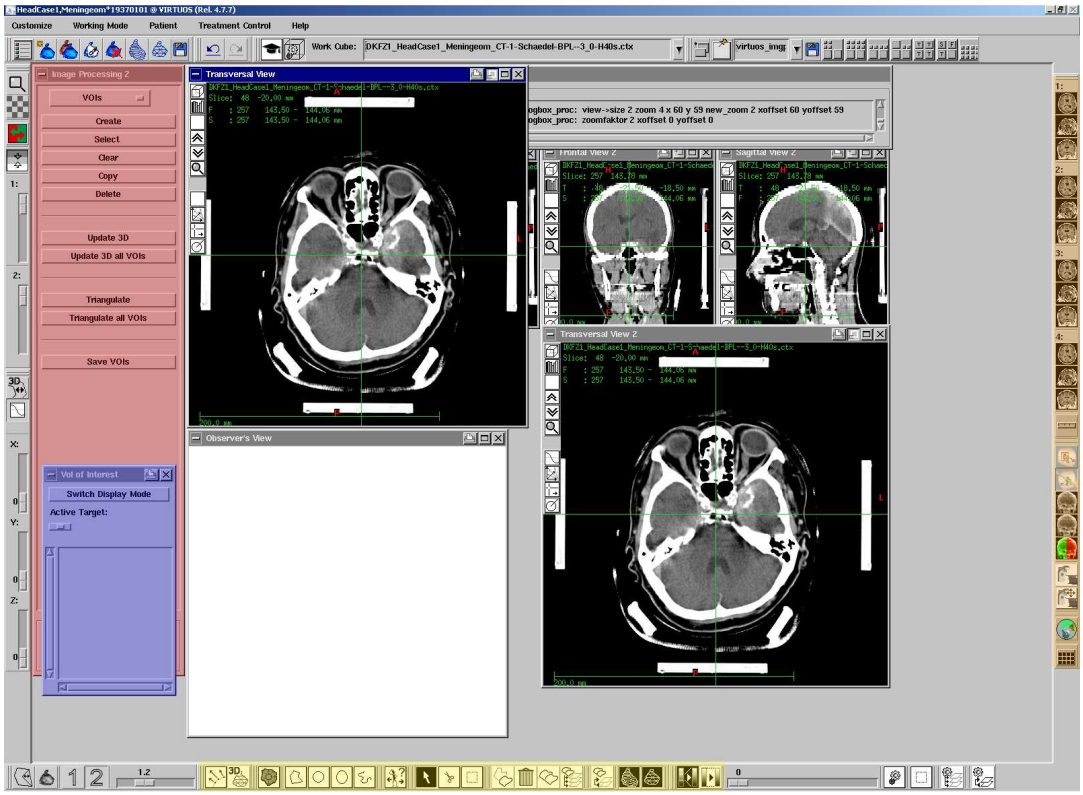

Abbildung 8: *VIRTUOS Benutzeroberfläche direkt nach dem Start des Programms.*

VIRTUOS besteht grundsätzlich aus drei großen Modi. Meist wird zum Start der Modus ImageProcessing geladen. Dieser Modus stellt alle Funktionen, die zur Bildverarbeitung in das Programm implementiert wurden, zur Verfügung. Um zwischen den verschiedenen Funktionen, die dieser Modus bietet, zu navigieren gibt es das Fenster, das im rot markierten Bereich zu sehen ist. In diesem Fall ist die Funktion zum Verwalten der VOIs offen. Das Fenster im blau markierten Bereich enthält eine VOI-List mit allen VOIs die der geladene Datensatz enthält. Auf diesem Bild erscheint kein VOI. Die am rechten Rand orange eingefärbte Leiste dient zur Verwaltung der geöffneten Ansichten. Damit ist es möglich, zwischen verschiedenen CT-Ansichten wie Frontal, Sagittal und Transversal oder auch 3D-Ansichten des Patientenmodells in den geöffneten Ansichtsfenstern zu wechseln. Am unteren Rand ist eine gelb eingefärbte Toolliste zu finden, die mehrere Methoden zur manuellen Bildbearbeitung enthält. Sie stellt Funktionen wie das Zeichnen eines Kreises oder das Kopieren einer Kontur in die nächste oder vorherige Schicht zu Verfügung.

Um zwischen den verschiedenen Modi zu wechseln können, müssen im Feld Working Mode der Menüleiste die anderen Modi aktiviert werden. Das Laden und Speichern von Patientendatensätzen ist unter dem Menüpunkt Patient möglich.

2.4.2 Volume of Interest – Modul

In der Arbeit [Fis96] von Hans-Jörg-Fischer wurde in VIRTUOS ein Modul zur Verwaltung und Manipulation von VOIs implementiert und integriert. Das Modul trägt heute den Namen VoiManSTL und hat seitdem auch schon zahlreiche Änderung durchlaufen, ist aber im Wesentlichen noch genauso aufgebaut wie in der Diplomarbeit beschrieben. Nähere Erläuterungen, die in diesem Abschnitt nicht erwähnt werden, können daher in dieser Diplomarbeit nachgeschlagen werden.

Aufbau

Um ein Volume Of Interest darzustellen, wurde in der Diplomarbeit die Klasse VOI geschaffen. Diese Klasse enthält die Basisinformationen wie zum Beispiel den Namen der VOIs. Zusätzlich ist es möglich, dass eine VOI mehrere untergeordnete „Volumes“ besitzt. Diese werden allerdings nicht direkt in der VOI hinterlegt sondern, lediglich mit dem Namen referenziert. Da ein VOI über mehrere Methoden dargestellt werden kann, besitzt sie je nach Bedarf eine ContourVoi, FacetVoi und eine VoxelVoi.

Die erste und einfachste Darstellungsform eines VOI besteht in der Darstellung eines Konturlinienstapels. Diese Repräsentationsform wird durch die Klasse ContourVoi abgedeckt. Eine ContourVoi enthält mehrere Schichten, die wiederum mehrere Konturen besitzen können.

Die triangulierte Darstellung eines VOI wird durch die Klasse FacetVoi repräsentiert. Auch diese Klasse enthält mehrere Schichten, die wiederum mehrere Dreicke enthalten, die die Oberfläche des VOI repräsentieren. Ein Dreieck enthält selbst keine Punkte sondern verweist dabei auf die Punkte innerhalb einer Kontur.

Ein Bilddatensatz besteht immer aus sehr kleinen Würfeln den sogenannten Voxeln. Deshalb hält eine VoxelVoi pro Schicht alle Voxel die in einem VOI vorkommen.

Da ein 3D-Patientenmodell immer aus mehr als einem VOI besteht, gibt es noch eine weitere Klasse die alle VOIs zusammenfasst. Diese Klasse heißt VoiModel und repräsentiert ein komplettes 3D-

Patientenmodell. Neben den VOIs enthält es noch andere wichtige Daten, die für das Modell von Relevanz sind. Folgende Abbildung verdeutlicht den Aufbau des Moduls.

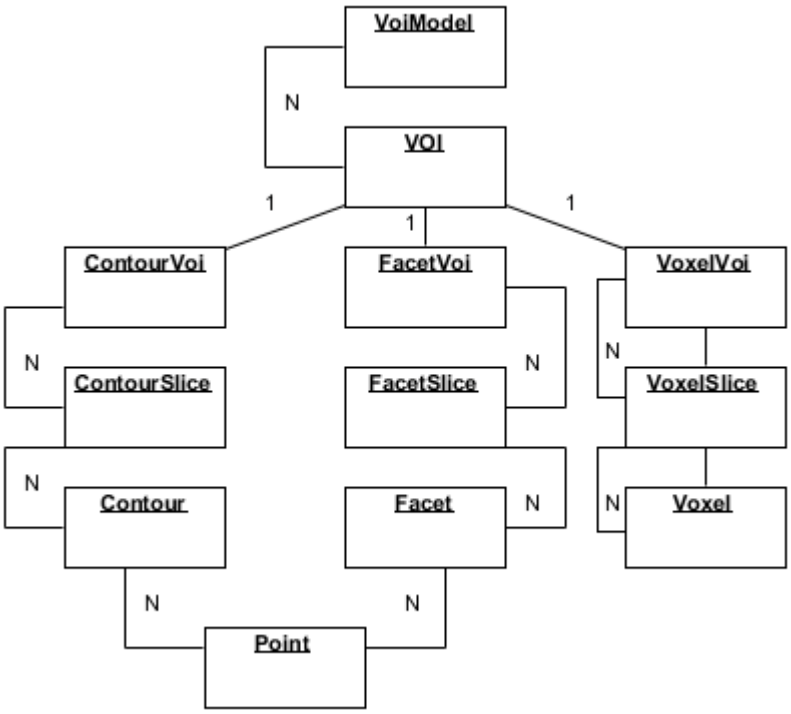

Abbildung 9: *Darstellung des Aufbaus des VoiManSTL Moduls.*

Der Zugriff auf dieses Modul erfolgt dabei zentral über die Schnittstelle voi_man. Diese hält immer das aktuelle VoiModel und führt alle Aktionen aus, die von VIRTUOS an das Modul weitergegeben werden.

Homogene Koordinaten

Von zentraler Bedeutung ist die Beschreibung der einzelnen Punkte. Diese werden sowohl in VIRTUOS als auch im VoiManSTL Modul durch homogene Koordinaten ausgedrückt. Dies bedeutet, dass ein Punkt um die Koordinate 1 erweitert wird. Ein Punkt wird dann so ausgedrückt:

$$P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Dies bringt einen entscheidenden Vorteil mit sich. Komplexe Transformationen können als Kombination einfacher Transformationen ausgedrückt werden. Einfache Transformationen können mit folgenden Matrizen ausgeführt werden:

$$T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

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

$$R_y = \begin{pmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_z = \begin{pmatrix} \cos y & -\sin y & 0 & 0 \\ \sin y & \cos y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$S = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Eine komplexe Transformation kann nun durch Multiplikation der entsprechenden Matrizen, in die vorher die entsprechenden Werte eingesetzt wurden, berechnet werden. Die folgende Gleichung berechnet eine Transformation in der ein Punkt P verschoben, um die X-Achse rotiert und skaliert wird:

$$P' = \big((T * R_x) * S\big) * P$$

2.4.3 VDX – Dateiformat

Um VOIs zu hinterlegen, existiert das VDX-Dateiformat. Eine VDX-Datei enthält immer ein komplettes VoiModel mit den dazugehörigen VOIs. Die aktuelle Version ist die Version 2. Folgendes Beispiel zeigt einen Ausschnitt aus solch einer Datei:

```
contours
reference_frame
 origin 0.000000 0.000000 0.000000
 point_on_x_axis 1.000000 0.000000 0.000000
 point_on_y_axis 0.000000 1.000000 0.000000
 point_on_z_axis 0.000000 0.000000 1.000000
number_of_slices 5

slice 0
slice_in_frame 299.500
thickness 5.000  reference  start_pos 297.000  stop_pos 302.000
number_of_contours 1
contour 1
internal false
number_of_points 41
 278.028 142.228 299.500  0.000 0.000 0.000
 276.315 142.640 299.500  -0.320 -0.902 -0.289
```

In einer VDX-Datei werden alle Daten ohne Hierarchie untereinander geschrieben. Am Anfang der Datei stehen Daten, die dem zum VoiModel gehören. So ist dort sowohl die Anzahl der VOIs hinterlegt als auch die Versionsnummer, in der diese Datei geschrieben ist. Danach folgen die Daten der VOIs. Pro VOI wird zuerst die ContourVoi niedergeschrieben. Dies ist im obigen Beispiel zu sehen.

Das Schlüsselwort „contours“ leitet dies ein. Pro ContourSlice also pro Schicht werden alle Konturen Punkt für Punkt geschrieben. Danach folgt die FacetVoi. Auch dort werden pro Schicht die einzelnen Dreiecke geschrieben. Nach dem Schreiben des ersten VOI folgt dann das nächste VOI bis keine mehr vorhanden sind. Jede Datei ist durch eine Checksumme gesichert. Eine Änderung, die nicht von VIRTUOS durchgeführt wurde, wird so direkt durch VIRTUOS beim Prüfen dieser Checksumme bemerkt. In diesem Fall wird die Datei nicht geladen.

2.4.4 OCTOPUS-Version

OCTOPUS ist eine spezielle Erweiterung von VIRTUOS für die Behandlung von Augentumoren mit Protonen. Die Erweiterung wurde in einer nationalen Zusammenarbeit mit dem Hahn-Meiner Institut in Berlin entwickelt. Am Hahn-Meiner Institut wurde die erste Therapieanlage, die mit Protonen arbeitet, installiert[DKF11a].

OCTOPUS unterstützt 4 grundlegende Funktionen, die für solch eine Software üblich sind. Zum einen kann das Auge des Patienten komplett modelliert werden, zum anderen können bestimmte Behandlungsparameter angegeben werden. Weiterhin kann wie in VIRTUOS ein präzise Berechnung der Dosisverteilung durchgeführt werden. Ebenso wie in VIRTUOS können auch in dieser Erweiterung Pläne evaluiert und bei nicht zufrieden stellenden Ergebnisse der Plan überarbeitet werden.

Besonders die erste Funktion ist für die Arbeit von großer Bedeutung, da hier Methoden implementiert wurden um VOIs und ihre untergeordneten VOIs zu verschieben, rotieren und zu skalieren.

2.5 Softwareentwicklung

In diesem Abschnitt finden sich einige Erläuterungen zu Elementen die in der entwickelten Software verwendet wurden.

2.5.1 Programmiersprache C++

Da VIRTUOS in der Programmiersprache C bzw. C++ geschrieben wurde, liegt es nahe, dass auch die entwickelte Software in C++ geschrieben wurde.

C++ ist eine objektorientierte Sprache, die in ISO standardisiert wird. Sie wurde ab 1979 von Bjarne Stroustrup als Erweiterung zur Programmiersprache C entwickelt[Wik11a]. Dadurch, dass C++ als Aufsatz zu C entwickelt wurde, ist es möglich in C++ auch C zu benutzen und zu schreiben. Dies ist besonders für VIRTUOS wichtig, da dort sowohl C als auch C++ benutzt wird. Weiterhin ist C++ unabhängig von Plattformen. Dadurch ist es möglich, sowohl UNIX/LINUX als auch Windows als Betriebssystem zu verwenden.

Obwohl es mit C++ möglich ist, eine Programmierung auf einer hohen Abstraktionsebene durchzuführen kann, mit dieser Programmiersprache auch sehr maschinennah programmiert werden. Dies führt insgesamt bis heute zu einer sehr hohen Beliebtheit dieser Programmiersprache.

2.5.2 MVC – Entwurfsmuster

Die Architektur einer Software folgt immer einem grundlegenden Konzept. Ohne ein Konzept, welches die grundsätzliche Konfiguration festlegt und dokumentiert, entsteht sehr schnell ein Chaos bei der Entwicklung, da nicht nachvollziehbar ist, wo welche Implementierungen zu finden sind. Dies verlangsamt nicht nur die Entwicklung des Projekts sondern erschwert auch die Wartung und Revisionierung.

Solch ein Konzept stellt das MVC-Entwurfsmuster dar. Es unterteilt den Aufbau einer Software in Controller-, View- und Model-Klassen.

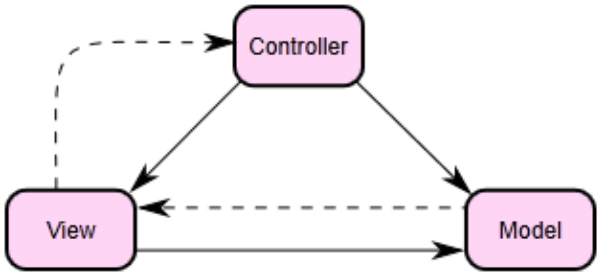

Abbildung 10: *Darstellung des MVC-Entwurfsmuster. Die gestrichelten Pfeile stellen indirekte Assoziationen dar und die durchgezogenen Pfeile stellen direkte Assoziationen da[Wik11d].*

Controller-Klassen sind diejenigen Klassen, die die Steuerung des Programms beinhalten und die Anweisungen an die entsprechenden Klassen weitergeben. Die Controller-Klassen sind auch die einzigen Klassen die sowohl die Model-Klassen als auch die View-Klassen kennen. Die View-Klassen beinhalten alle Implementierung bezüglich der graphischen Benutzeroberfläche. Die View-Klassen kennen die Model-Klassen, da diese auf der Benutzeroberfläche dargestellt werden müssen. Die

Model-Klassen enthalten die darzustellenden Daten und sind von den beiden anderen Klassen unabhängig.

Solch ein Programmentwurf fördert die Wartung sowie die Implementierung von Erweiterungen. Außerdem ist es möglich einzelne Komponenten dieser Implementierung in anderen Programmen wieder zu verwenden.

2.5.3 Metasprache XML

Die Extensible Markup Language (XML) dient sowohl zur Darstellung von hierarchischen Strukturen, als auch zur Definition von weiteren eingeschränkten Sprachen. Auch wird die Sprache zum Austausch von Daten verwendet. Die erste Definition dieser Sprache wurde vom World Wide Web am 10. Februar 1998 herausgegeben und seitdem stetig verbessert und erweitert[Wik11b].

Sowohl hierarchische Strukturen als auch Sprachen werden über XML-Schemas definiert. So wird zum Beispiel die Sprache XHTML durch ein bestimmtes XML-Schema definiert. Folgendes Beispiel zeigt eine einfache XML-Datei:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<verzeichnis>
   <titel>Wikipedia Städteverzeichnis</titel>
   <eintrag>
      <stichwort>Genf</stichwort>
      <eintragstext>Genf ist der Sitz von ...</eintragstext>
   </eintrag>
   <eintrag>
      <stichwort>Köln</stichwort>
      <eintragstext>Köln ist eine Stadt, die ...</eintragstext>
   </eintrag>
</verzeichnis>
```

In der ersten Zeile ist eine XML-Deklaration zu sehen, um XML-Version, Zeichenkodierung und Verarbeitbarkeit ohne Dokumenttypdefinition zu spezifizieren[Wik11b]. Weiterhin ist zu sehen, dass jedes Element, oder auch „Tag“, mit einem Start-Tag `<name_des_tags>`beginnt und mit einem End-Tag `</name_des_tags>` endet. Befindet sich kein Inhalt zwischen Start-Tag und End-Tag so wird der Tag abgekürzt `<name_des_tags />`. Zu beachten ist, dass jedes XML-Dokument mindestens ein Wurzelelement beinhalten muss.

Um ein Tag einem anderen Tag unterzuordnen, muss das übergeordnete Element das untergeordnete Element beinhalten. Im Beispiel ist das Element `titel` dem Element `verzeichnis` untergeordnet. Dies ermöglicht die Darstellung einer Hierarchie in Form einer Baumstruktur. Zusätzlich zu den bisherigen Definitionen kann ein Tag ein Attribut enthalten. Dafür wird in den Start-Tag der Attributname geschrieben und nach einem Gleichzeichen der entsprechende Wert hinzugefügt `<name_des_tags attribut="wert">`.

3 Ergebnisse

Dieses Kapitel beschreibt die Ergebnisse der vorliegenden Arbeit. Zum einen wird das im Rahmen dieser Arbeit entwickelte Modul „Orghandler“, welches zur Verwaltung von Organmodellen eingesetzt wird, näher erläutert. Ebenso wird erläutert, wie dieses Modul in das hauseigene Programm des DKFZ „VIRTUOS“ integriert wurde und wie es genutzt werden kann. Weiterhin wird erläutert, wie das Modul zur Adaption von Organmodellen in VIRTUOS integriert wurde.

Kapitel 3.1 gibt dabei eine kurze Einführung über die entwickelte Komponente. Das darauffolgende Kapitel 3.2 enthält detaillierte Erläuterungen zu den verwendeten Datenstrukturen und deren Erweiterungen. Anschließend wird in Kapitel 3.3 näher auf die entwickelte Benutzeroberfläche und auf ihre Verwendung eingegangen. In Kapitel 3.4 wird näher auf die Integration des OrgHandlers eingegangen, sowie auf die internen Abläufe des entwickelten Systems. In Kapitel 3.5 wird schlussendlich ausführlich auf die Implementierung eingegangen, um anschließend in Kapitel 3.6 auf die Evaluierung dieses Moduls einzugehen.

3.1 Einführung OrgHandler

Aufgabe dieser Arbeit war es, eine Programmkomponente zur effizienten Verwaltung von Organmodellen zu entwickeln. Weiterhin soll es möglich sein, diese Organmodelle zu adaptieren, um eine Grundlage für manuelle- und semi-automatische Segmentierungsstrategien im Rahmen der Strahlentherapieplanung zu bilden.

Um Organmodelle effizient zu verwalten, wurde im Rahmen dieser Arbeit ein Modul mit dem Namen „OrgHandler“ entwickelt, welches die in der Einleitung erstellten, Anforderungen erfüllt. Der OrgHandler ermöglicht das Anlegen, Abspeichern und Laden von Kategorien von Organkategorien, Gruppen von Organmodellen und Organmodellen selbst. Organmodelle werden im OrgHandler durch „Volume Of Interests“ (VOIs) beschrieben. Ein VOI ist eine dreidimensionale Datenstruktur um die Oberfläche einer Struktur darzustellen, die für den Strahlentherapeut von Relevanz ist(siehe 2.3.1).

Um eine Abhängigkeit zwischen Adaption und Verwaltung der Organmodelle zu vermeiden, wurde die Komponente zur Adaption der Modelle unabhängig vom OrgHandler entwickelt. Die Implementierung dieser Komponente erfolgte direkt in VIRTUOS und ermöglicht das Rotieren, Skalieren und Verschieben von VOIs.

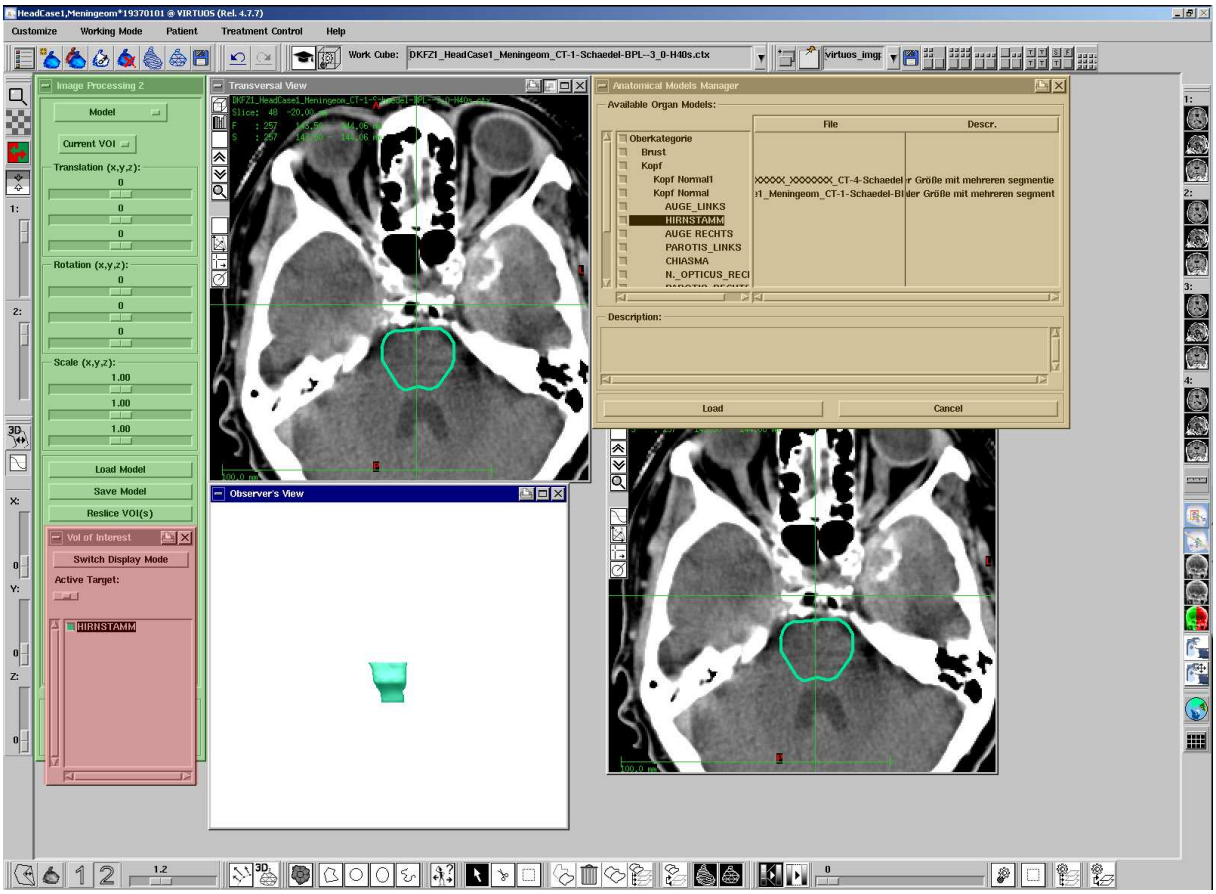

Abbildung 11: *Screenshot mit der VIRTUOSoberfläche und dem aktivierten Oberfläche für das Laden und Speichern von Organmodellen.*

Der grün markierte Bereich stellt die Oberfläche zur Verfügung, um VOIs zu verschieben, zu rotieren oder zu skalieren. Die beiden Schaltflächen „Load Model“ und „Save Model“ öffnen den orange markierten Dialog, in dem die Liste von Organmodellen mit den dazugehörigen Beschreibungen zu finden ist. Je nachdem, über welche Schaltfläche der Dialog geöffnet wird, ist im orangenen Dialog eine Schaltfläche „Load“ oder eine Schaltfläche „Save“ zu sehen. Die Schaltfläche „Reslice Voi“ im grün markierten Bereich dient dazu, die ContourVoi anhand der triangulierten Oberfläche, die in der FacetVoi gespeichert ist, neu zu berechnen (siehe 2.4.3). Im roten Bereich ist die Liste aller VOIs zu sehen. Sie ist Bestandteil des Hauptprogramms VIRTUOS.

3.2 Datenstrukturen

Die Speicherung von Organmodelle erfordert eine effiziente Datenstruktur, die es ermöglicht Daten geordnet abzulegen und diese ebenso geordnet auszulesen.

Grundsätzlich gibt es mehrere Alternativen, solche Datenstrukturen zu realisieren, wie zum Beispiel Datenbanken oder Datei/Ordner-Systeme. Da eine effiziente Anbindung einer Datenbank den Rahmen dieser Arbeit zu weit ausdehnen würde, wurde die Implementierung der Datenstruktur des OrgHandlers mit einem Datei/Ordner-System realisiert. Dies führt weiterhin dazu, dass das bereits existierende VDX-Dateiformat, welches in VIRTUOS genutzt wird, um ein VoiModel in einer Datei zu speichern (siehe 2.4.4), zum Speichern und Laden der Organmodelle genutzt werden kann. Ein VoiModel beinhaltet eine oder mehrere VOIs. Das bedeutet, dass ein VoiModel eine Organgruppe darstellt und ein oder mehrere Organmodelle beinhaltet, die durch eine VOI beschrieben werden. Desweiteren ist es notwendig, Organgruppen zu kategorisieren. Deshalb wurde als oberste Hierarchieebene noch die Organkategorie eingeführt. Eine Organkategorie kategorisiert Organgruppen die zu einer bestimmten Region des Körpers gehören, zum Beispiel dem Körperstamm. In den folgenden Kapiteln wird nun erläutert, wie die Datenstruktur der Programmkomponente aufgebaut ist und inwiefern das bereits vorhandene VDX-Dateiformat integriert wurde.

3.2.1 Integration des VDX-Dateiformats

Um Organmodelle in einer Datei zu speichern, muss eine Dateispezifikation erstellt werden, die den Aufbau der Datei genau beschreibt. Dies ermöglicht eine einfache Implementierung in das Modul und eine einfache Wartung.

Da bereits mit der VDX-Spezifikation ein Dateiformat vorhanden ist, in dem Oberflächenstrukturen in Form von VOIs abgespeichert werden können, wurde dieses Dateiformat und dessen Implementierung in das entwickelte Modul integriert. Da die reine Integration des Dateiformats nicht ausreichte, um alle Informationen, die für die entwickelte Komponente benötigt werden, in einer VDX-Datei abspeichern zu können, wurde die Spezifikation der VDX-Datei erweitert.

Erweiterung der VDX-Spezifikation

Bisher war es in der aktuellen Version „2“ und in der für die Octopus Version von VIRTUOS (siehe 2.4.5) genutzten Version „3“ der VDX-Spezifikation nicht möglich, anzugeben auf welchem Datensatz das VoiModel definiert wurde. Das Abspeichern des Namens der zugrundeliegenden Bildserie erlaubt beim Laden eine Plausibilitätskontrolle die verhindern kann, dass Daten, die nicht zusammenpassen, zusammen dargestellt werden.

Die VDX-Spezifikation wurde um folgende Datenfelder erweitert: „File“, „Dicom Series Id“, „Frame of Reference UID“. Das Datenfeld „File“ enthält den Dateinamen des Bilddatensatz, auf dem das VoiModel basiert. Die „Dicom Series Id“ referenziert den verwendeten Bilddatensatz anhand der UID, die pro Bilddatensatz eindeutig ist. Das Datenfeld „Frame of Reference UID“ enthält die ID des Koordinatensystems, in dem Bild- und Konturdaten definiert sind. Desweiteren ist es für den Benutzer sehr angenehm, wenn er zu jedem Organmodell eine Beschreibung hat. Aus diesem Grunde wurde pro VOI und für das gesamte VoiModel jeweils das Datenfeld „description“ eingeführt. Im Zuge

der Erweiterung der VDX-Spezifikation wurden noch weitere Datenfelder hinzugefügt, die allerdings für die Arbeit irrelevant sind und nur für zukünftige Entwicklungen von VIRTUOS gedacht sind.

Mit diesen Erweiterungen ist es nun möglich, Organmodelle, welche als VOI vorliegen, in einer Datei zu speichern.

3.2.2 Umstellung VDX zu VDX-XML

Die aktuelle Spezifikation der VDX-Datei sieht vor, dass die Inhalte der spezifizierten Datenfelder ohne sichtbare Hierarchie untereinander geschrieben werden (siehe 2.4.4). Dies erschwert zum einen das Überprüfen der Korrektheit einer Datei und zusätzlich ist es schwer, Erweiterungen einzuführen. Deshalb wurde im Rahmen dieser Arbeit die bisherige erweiterte VDX-Spezifikation auf eine XML-Spezifikation portiert. Jedes spezifizierte Datenfeld wird nun durch einen XML-Tag dargestellt. Durch eine XML basierte Spezifikation in einer Schema-Datei ist eine automatisierte Syntaxprüfung möglich, bevor die Datei eingelesen oder nach dem sie geschrieben wird. Dadurch wird das Lesen der Dateien einfacher, da von einer korrekten Dateistruktur ausgegangen werden kann.

Aufbau der XML-Datei

Die neue VDX-Spezifikation beruht jetzt auf zwei Hauptbereichen. Der erste Bereich in einer VDX-Datei ist nun der „Header“. In diesem sind alle Daten angegeben die sich auf das VoiModel beziehen, wie zum Beispiel zugehöriger Patient oder zugehöriger Bilddatensatz. Weiterhin werden alle VOIs die das VoiModel beinhaltet, hierarchisch angegeben. Dabei kann jedes VOI ein oder mehrere (Unter-) VOIs beinhalten. Folgendes Beispiel zeigt den Ausschnitt aus einem Header-Abschnitt:

```
<header>
  .
  <based_on>
    <file/>
    <dicom_series_id/>
    <frame_of_reference_uid/>
  </based_on>
  <approved_by/>
  <trafo_rel_to_voi_symm_system_without_scale>
    <row>+1.000000 +0.000000 +0.000000 +0.000000</row>
    <row>+0.000000 +1.000000 +0.000000 +0.000000</row>
    <row>+0.000000 +0.000000 +1.000000 +0.000000</row>
    <row>+0.000000 +0.000000 +0.000000 +1.000000</row>
  </trafo_rel_to_voi_symm_system_without_scale>
  <voi name="Lungen">
    <key>empty</key>
    <type>10</type>
    <description/>
    <voi name="Lunge_links">
      <key>empty</key>
      <type>2</type>
      <description>linker Lungenflügel</description>
    </voi>
    <voi name="Lunge_rechts">
      <key>empty</key>
      <type>2</type>
      <description>rechter Lungenflügel</description>
    </voi>
  </voi>
  .
</header>
```

Der Abschnitt „based_on“ beinhaltet die oben genannten neuen Datenfelder, welche den Bilddatensatz angeben, auf dem das VoiModel basiert. Matrizen werden Zeile für Zeile gespeichert. In dem oben genannten Beispiel ist die Matrix „trafo_rel_to_symm_without_scale“ eine 4x4 Matrix. Weiterhin werden alle Fließkommazahlen als solche mit dem dazugehörigen Vorzeichen gespeichert. Ganze Zahlen werden ebenfalls als solche und mit dem dazugehörigen Vorzeichen gespeichert. Nach dem Abschnitt „Header“ folgt nun der zweite Abschnitt „Body“. Dieser beinhaltet alle Oberflächendaten der einzelnen VOIs. Die Aufteilung dieses Abschnittes ist an die Datenstruktur einer VOI angelehnt (siehe 2.4.3). Beinhaltet eine VOI eine CountourVoi so werden alle ContourSlices, die die ContourVoi enthält, in dem XML-Tag „slice“ gespeichert. Wobei pro ContourSlice ein XML-Tag „slice“ mit der zugehörigen Nummer erstellt wird. In jedem „slice“ werden dann die dazugehörigen Contours in „contours“ abgelegt. Ähnlich verläuft es mit dem Abspeichern einer FacetVoi. Beinhaltet ein VOI eine FacetVo,i so wird diese in dem XML-Tag „facets“ hinterlegt. Pro FacetSlice werden in „slab“ alle Facets in jeweils einem „facet“ Tag abgespeichert, welche die Konturen aus zwei benachbarten Konturen verbinden. Folgendes Beispiel zeigt den Ausschnitt aus einem Abschnitt „Body“:

```
<body>
  <voi name="Lunge">
    <center_of_voi>+0.000000 +0.000000 +0.000000 +1.000000</center_of_voi>
    <slice number="0">
      <dicom_image_id>not implemented yet</dicom_image_id>
      <position_in_frame>+76.000000</position_in_frame>
      <start>+74.500000</start>
      <stop>+77.500000</stop>
      <contours>
        <contour number="1">
          <internal>false</internal>
          <number_of_points>13</number_of_points>
            <point no="0">+36.9959 +34.1930 +76.0000 +0.0000 +0.0000 +0.000000</point>
            <point no="1">+32.5120 +39.7990 +76.0000 +0.0000 +0.0000 +0.000000</point>
            .
        </contour>
        .
      </contours>
    </slice>
    .
    <facets>
      <slab number="0">
        <connecting>0 1</connecting>
        <number_of_facets>153</number_of_facets>
        <facet number="0">
          <normal>-0.777000 -0.777000 -0.777000</normal>
          <connections no_of_points="3">+1 +0 +3;+0 +0 +4;+0 +0 +3</connections>
        </facet>
        .
      </slab>
      .
    </facets>
  </voi>
</body>
```

3.2.3 Ordnerstruktur

Um eine Hierarchie zwischen Gruppen von Organmodellen herzustellen, wurde für die entwickelte Programmkomponente eine Ordnerstruktur spezifiziert, die es ermöglicht Organkategorien sowie Organgruppen darzustellen. Eine Organkategorie soll dabei Gruppen von Organen kategorisieren. So sollen zum Beispiel in der Kategorie „Kopf“ alle Organe hinterlegt werden, die im Kopf vorkommen. Eine Gruppe von Organen vereint alle Organe, die vom selben Bilddatensatz abstammen. Eine Organkategorie kann dabei sowohl eine Organkategorie als auch eine Organgruppe beinhalten. Der Ordner einer Organgruppe beinhaltet dabei die entsprechende VDX-Datei und wenn nötig den referenzierten Bilddatensatz. Sowohl die Organgruppe als auch die Organkategorie werden durch den entsprechenden Ordnernamen betitelt. Folgende Abbildung erläutert die Ordnerstruktur:

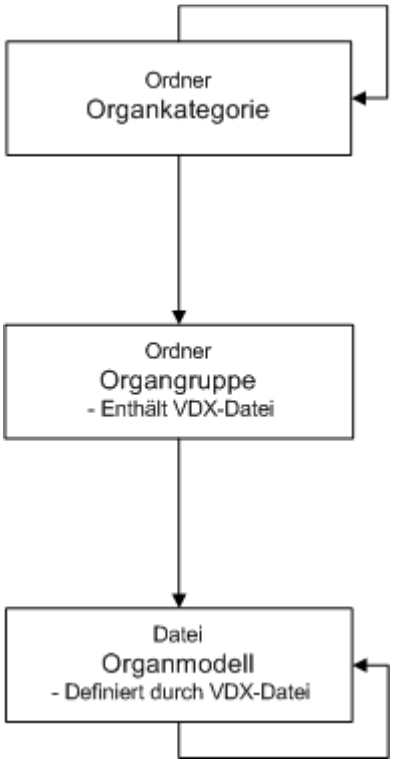

Abbildung 12: *Darstellung der Ordnerstruktur.*

Die Pfeile in diesem Diagramm sollen eine „1 zu N“ Beziehungen darstellen. Das heißt jede Struktur von der der Pfeil ausgeht kann eine, keine oder mehrere Strukturen enthalten auf die der Pfeil zeigt.

Beispiel

Vorausgesetzt die Daten von VIRTUOS werden in dem Ordner „C:\Daten“ gespeichert und der Hauptordner von der entwickelten Komponente heißt „orgHandler“, so würde alle Daten die vom OrgHandler erzeugt werden in dem Ordner „C:\Daten\orgHandler“ gespeichert. Eine Organgruppe mit dem Namen „Schmaler-Kopf“ und der übergeordneten Organkategorie „Koepfe“ würde in folgendem Ordner hinterlegt „C:\Daten\orgHandler\Koepfe\Schmaler-Kopf“. Die VDX-Datei, die die Organmodelle definiert und den Dateinamen des Bilddatensatz enthält hätte folgenden Pfad und Namen „C:\Daten\orgHandler\Koepfe\Schmaler Kopf\Schmaler-Kopf.vdx.xml“.

3.3 Workflow der Benutzeroberfläche

Ebenfalls sehr wichtig für eine Programmkomponente ist der einfache und flüssige Workflow der Benutzeroberfläche. Muss der Anwender durch viele komplizierte Ansichten navigieren, bevor er überhaupt ein Ergebnis hat oder ist die Benutzeroberfläche generell zu kompliziert, verliert der Benutzer sehr schnell den Überblick und wird die Programmkomponente nicht nutzen. Dieses Kapitel beschreibt den Workflow, den der Benutzer durchgehen muss, um Organmodelle zu laden, zu speichern und diese zu adaptieren.

3.3.1 Modell laden und speichern

Organmodelle zu laden und zu speichern ist ein zentraler Bestandteil des Verwaltungsmoduls. Um dies zu realisieren, wurde eine neue Dialogbox geschaffen, mit deren Hilfe durch die Liste mit Modellen, Kategorien und Gruppen navigiert wird.

Modelle laden

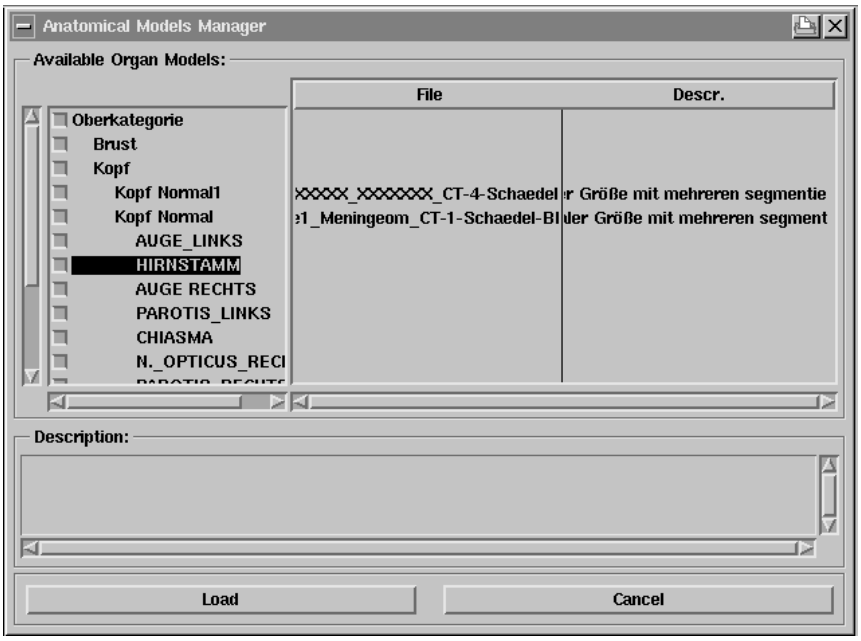

Abbildung 13: *Dialogbox mit der Liste aller Objekte. Rechts daneben befindet sich der Beschreibungstext der Gruppen „Kopf Normal“ und „Kopf Normal 1“, sowie die zugehörigen Namen der Bilddatensätzen.*

Um ein Organmodell zu laden, muss der Benutzer einen Bilddatensatz geladen haben. Ohne einen Bilddatensatz ist es nicht möglich, ein Organmodell zu laden. Möchte der Benutzer nun ein Modell laden, so muss er zunächst die in Abbildung 13 gezeigt Dialogbox öffnen. Dazu klickt er mit der linken Maustaste auf die Schaltfläche „Load“ welche in Abbildung 11 im grün markierten Bereich zu sehen ist. Nun öffnet sich die Dialogbox und der Benutzer kann durch die Liste navigieren. Mit einem einfachen Klick der linken Maustaste wird ein Listenelement markiert, wodurch es schwarz hinterlegt wird. Mit einem Doppelklick der linken Maustaste wird das entsprechende Listenelement geöffnet. Das heißt, sofern dieses Element untergeordnete Elemente besitzt, werden diese nun auch leicht nach rechts versetzt unter dem markierten Listenelement angezeigt. In Abbildung 13 ist das Listenelement „Kopf Normal“ geöffnet, weshalb die untergeordneten Listenelemente „AUGE_LINKS“, „AUGE_RECHTS“, „PAROTIS_LINKS“ und folgende angezeigt werden. Hat man sich für eine

Organgruppe oder ein Organmodell entschieden, so muss dieses markiert werden und dazu auf die Schaltfläche „Load“ geklickt werden. Danach wird das markierte Objekt in den aktuellen Bilddatensatz geladen. Danach kann die Dialogbox geschlossen werden.

Modelle speichern

Der Ablauf um Modelle abzuspeichern ist ebenfalls sehr simpel gehalten. Möchte der Benutzer ein Organmodell speichern, muss zunächst ein Bilddatensatz geladen sein. Dieser sollte entweder bereits segmentierte VOIs beinhalten oder der Benutzer muss selbst welche segmentieren. Pro Organmodell muss ein VOI existieren. Möchte der Benutzer nun die erstellten VOIs als Organmodelle speichern, so muss er auf die Schaltfläche „Save“ klicken, welche in Abbildung 11 in dem grün markierten Bereich zu sehen ist. Dabei öffnet sich wieder die gleiche Dialogbox wie sie in Abbildung 13 zu sehen ist. Lediglich die Schaltfläche „Load“ wurde durch die Schaltfläche „Save“ ersetzt. Nun kann der Benutzer durch die Liste navigieren und die entsprechende Organkategorie aussuchen, unter der er die Organmodelle speichern möchte. Die Organmodelle werden dabei zusammen als Gruppe hinterlegt. Hat der Benutzer die gewünschte Kategorie erreicht, so klickt er auf die Schaltfläche „Save“. Es öffnet sich eine weitere Dialogbox, in der der Benutzer den Namen der Organgruppe eingeben muss. Mit „Ok“ bestätigt er die Eingabe. Daraufhin öffnet sich eine weitere Dialogbox, die den Benutzer auffordert, eine Beschreibung für die Organgruppe einzugeben. Nach erfolgter Eingabe klickt der Benutzer erneut auf „Ok“ und die Organgruppe wird abgespeichert. Nach erfolgreicher Speicherung taucht die neue Organgruppe in der Liste auf und die Dialogbox kann geschlossen werden.

3.3.2 Adaption von Modellen

In diesem Kapitel wird der Workflow zum Transformieren von Organmodellen bzw. zum Transformieren von VOIs beschrieben. Dieses Modul ist technisch gesehen unabhängig von der Verwaltung der Organmodelle und kann also auch bei VOIs eingesetzt werden, die vom Benutzer selbst erstellt und nicht durch den OrgHandler geladen wurden. Um VOIs verschieben und transformieren zu können, muss in der ImageProcessing-Einheit der Bereich Model ausgewählt sein. Dann öffnet sich die folgende Ansicht:

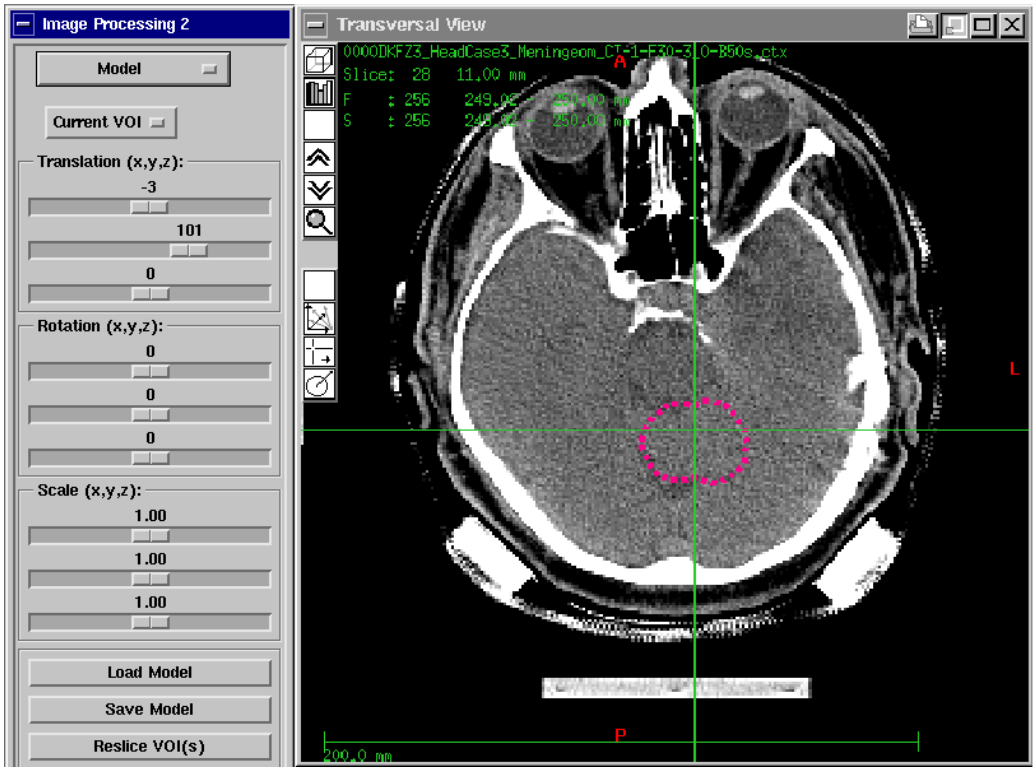

Abbildung 14: *Darstellung der Benutzeroberfläche zum Transformieren und Verschieben von VOIs. Links die Schieberegler zum Einstellen der Werte. Rechts zeigt eine CT-Schicht mit einer Kontur der transformierten VOI.*

Hat der Benutzer ein VOI in der VOI-Liste markiert, so kann dieses nun über die Schieberegler verschoben und transformiert werden. In der Gruppe „Translation“ befinden sich die Regler zum verschieben. Der oberste führt eine Translation in Richtung der X-Achse. Der zweite Regler eine Translation in Richtung der Y-Achse und der dritte Regler eine Translation in Richtung der Z-Achse durch. Die zweite Gruppe von Schiebereglern dient zur Rotation eines VOI. Der erste Schieberegler rotiert das VOI um dessen Schwerpunkt entlang der X-Achse. Der zweite Schieberegler führt eine Rotation des VOI um seinen Schwerpunkt entlang der Y-Achse aus und der dritte ebenfalls um den Schwerpunkt des VOI entlang der Z-Achse. Die letzte Gruppe von Schiebereglern dient zur Skalierung der VOI. Der erste skaliert das VOI um seinen Schwerpunkt entlang der X-Achse. Der zweite und dritte Schieberegler jeweils entlang der Y- und Z-Achse.

Wichtig in diesem Zusammenhang ist die Schaltfläche „Reslice VOI(s)“. Konturen werden auf Basis der Schichtbilder definiert. Alle Punkte einer Kontur, die auf einem transversalen Bild definiert wurden, haben deshalb die gleiche z-Koordinate. Der Abstand zwischen aufeinanderfolgenden Schichtbildern ist i.d.R. größer als die Pixelgröße in einem Schichtbild. Über eine Triangulation wird zwischen den Konturlinien eine Oberfläche approximiert.

Um die Definition einer Kontur (ihren Verlauf) zu ändern, muss wieder die Schicht in der die Kontur definiert wurde, dargestellt werden. Wenn ein VOI rotiert wird oder in z-Richtung verschoben wird, so werden die schichtweise definierten Konturen gegenüber den Schichten der Bildserie verkippt. Da die Konturen nicht mehr in einer darstellbaren Schicht liegen, könnten Sie nicht mehr editiert werden. Um auch individuell die Form anpassen zu können, muss deshalb ein transformiertes VOI an den Positionen der Schichtbilder neu geschnitten werden. Diese Operation kann mit diesem Knopf gestartet werden. Durch das Klicken dieser Schaltfläche werden aus den triangulierten Oberflächendaten neue Konturen berechnet. Zusätzlich werden alle Schieberegler auf ihre Standardwerte gesetzt, wodurch ein Zurücksetzen des VOI nicht mehr möglich ist. Diese Schaltfläche sollte nach Beendigung der Adaptierung aktiviert werden.

3.4 Integration in VIRTUOS

Die Integration des OrgHandlers in VIRTUOS war eine weitere sehr wichtige Aufgabe. Dazu wurde das Modul ModelManBox implementiert. In diesem Modul befindet sich zum einen die Steuerung der Dialogbox mit der Liste aller Organmodelle, Organgruppen und Organkategorien, als auch der Zugriff auf den OrgHandler. In diesem Modul wird eine Instanz der Klasse OrgHandler ebenfalls als Singleton gehalten. Das Modul ModelManBox selbst ist in der ImageProcessing-Einheit von VIRTUOS implementiert. Die ImageProcessing-Einheit beinhaltet alle Funktionalitäten zum Manipulieren von Bilddaten, die in VIRTUOS implementiert wurden.

Da für die Adaption von VOIs kein eigenes externes Modul vorgesehen war, wurde dieses Modul direkt in VIRTUOS integriert. Hierbei konnte auf große Teile der Implementierung eines vorherigen Moduls zur Adaption von VOIs zurückgegriffen werden. Diese Implementierung war zum großen Teil in der Octopus-Version von VIRTUOS zu finden (siehe 2.4.5) sowie in dem VoiManSTL Modul(siehe 2.4.3).

3.4.1 Modelle verwalten

Die Integration des OrgHandlers erforderte grundsätzlich die Implementierung von drei Abläufen. Der erste Ablauf ist die Initiierung des kompletten Moduls. Die weiteren Abläufe sind das Laden eines Organmodells oder einer Organgruppe und das Speichern eines Organmodells oder einer Organgruppe.

Initiierung

Wird VIRTUOS gestartet, werden automatisch alle Module der ImageProcessing-Einheit initiiert. Deshalb wurde eine Routine erstellt, die dies für das entwickelte Modul macht. Bei der Initiierung ist es am sinnvollsten, wenn alle Organmodelle sowie Organgruppen und Organkategorien geladen werden, da direkt eine Liste aller Objekte erstellt werden kann. Um die Startzeiten nicht unnötig in die Länge zu ziehen, werden die Organmodelle allerdings ohne Oberflächendaten geladen. So ist dem Programm bekannt, welche Organmodelle es gibt; es blockiert aber nicht unnötig Speicher mit Oberflächendaten, die ist eventuell nicht braucht.

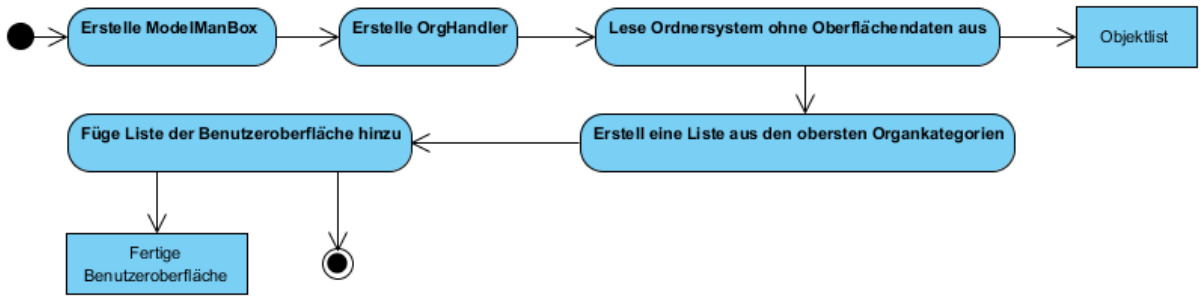

Abbildung 15: *Schematisch dargestellter Ablauf zur Initiierung der Modellvewaltung. Die abgerundeten Rechtecke stellen Aktivitäten und die eckigen Rechtecke stellen Objekte dar.*

Bei der Initialisierung der ImageProcessing-Einheit wird das Instanz der Klasse ModelManBox erstellt. Bei dieser Erstellung wird die Benutzeroberfläche in Form des Dialogs, der die Organmodellliste beinhaltet, instanziiert (siehe Abbildung 11. Orange markierter Bereich). Bei der Erstellung des Moduls wird der OrgHandler initialisiert. Dies hat zur Folge, dass der OrgHandler das komplette Ordnersystem - bis auf die Oberflächendaten- ausliest und somit eine komplette Liste mit Kategorien, Gruppen und Modellen hält.

Da die angezeigte Organmodellliste im Dialog hierarchisch geordnet ist, gibt der OrgHandler eine Liste mit den Organkategorien, die auf der höchsten Ebene liegen, zurück. Danach werden diese in die Liste des Dialogs eingefügt. Damit ist die Erstellung der ModelManBox abgeschlossen.

Modell laden

Ein weiterer wichtiger Ablauf zur Verwaltung der Modelle besteht darin, Modelle auszuwählen und diese dann in das aktuelle VOIModel einzufügen und der Bildserie zu überlagern.

Das vom Benutzer ausgewählte Listenelement wird hinsichtlich seines Typs kontrolliert. Ist das ausgewählte Element eine Organkategorie, so wird nicht fortgefahren, da es nicht sinnvoll ist ganze Kategorien zu laden. Wählt der Nutzer eine Organgruppe aus, so wird über alle Organmodelle die in der Organgruppe hinterlegt sind, iteriert. Bei jedem Iterationsschritt wird dann so fortgefahren, als sei direkt ein Organmodell ausgewählt worden

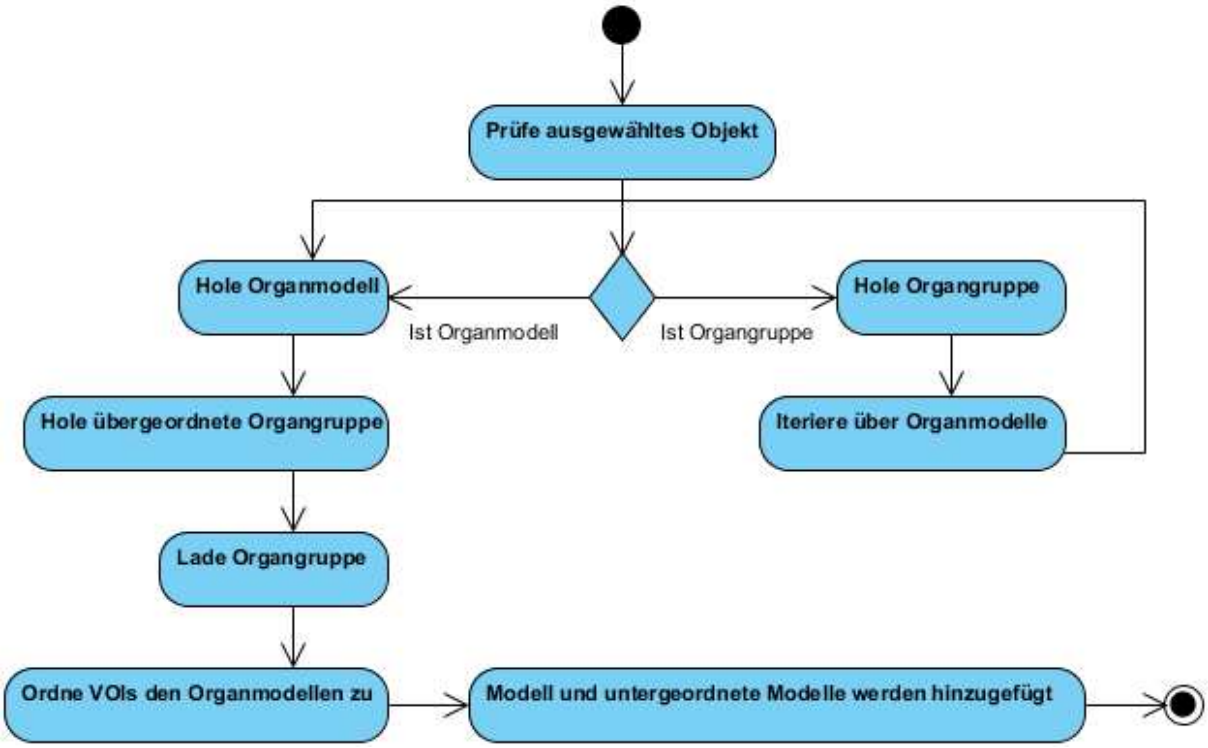

Abbildung 16: *Schematische Darstellung des Ablaufes zum Laden eines Organmodells und einer Organgruppe. Die abgerundeten Rechtecke stellen Aktivitäten und die eckigen Rechtecke stellen Objekte dar.*

Ist das Listenelement ein Organmodell, so werden die Oberflächendaten in Form des referenzierten VOI benötigt. Da ein VOI immer einem VoiModel untergeordnet ist und auch nur komplette VoiModels in Form einer VDX-Datei abgespeichert werden können, muss die komplette VDX-Datei geladen werden, in die das VOI abgelegt ist. Deshalb wird zu dem ausgewählten Organmodell die

übergeordnete Organgruppe geholt. In dieser Gruppe ist der entsprechende Name der VDX-Datei hinterlegt, die dann geladen wird. Ist dies geschehen, so werden alle geladenen VOIs den entsprechenden Organmodellen zugeordnet. Die Zuordnung wird über den VOI- bzw. Organmodellnamen realisiert. Nach der Zuordnung ist das ausgewählte Organmodell komplett geladen und kann nun in den aktuellen Bilddatensatz eingefügt werden. Sollten dem Organmodell selbst noch weitere Organmodelle untergeordnet sein, so werden diese ebenso geladen.

Dafür wird die im Organmodell hinterlegte VOI dem aktuellen VoiModel hinzugefügt. Da das Koordinatensystem des aktuellen Bilddatensatzes nicht mit dem Koordinatensystem Bilddatensatz übereinstimmt, auf dem das VOI basiert, befindet sich das VOI irgendwo im Raum und muss deswegen neu positioniert werden. Dazu wird der Schwerpunkt des VOI berechnet. Anhand dieses Schwerpunkts wird das VOI nun um die Differenz verschoben. Zunächst wird das VOI in Z-Richtung auf die Nullebene verschoben. Dies entspricht der Verschiebung um den Wert der Z-Koordinate des VOI-Schwerpunktes in Richtung des Koordinatenursprungs. Danach folgt die Verschiebung der VOI innerhalb der Ebene. Dazu muss der Mittelpunkt der Ebene berechnet werden. Dazu wird sowohl die X- als auch die Y-Schichtdimension des Bilddatensatzes herangeholt.

$$M_x = (Dim_x * PixDist)/2$$

$$M_y = (Dim_y * PixDist)/2$$

Um die X- und Y-Koordinate des Mittelpunkts der Ebene zu bekommen, muss die Dim_x und die Dim_y mit dem Pixelabstand, der durch das bildgebende Gerät vorgegeben ist, multipliziert werden und dann durch zwei geteilt werden.

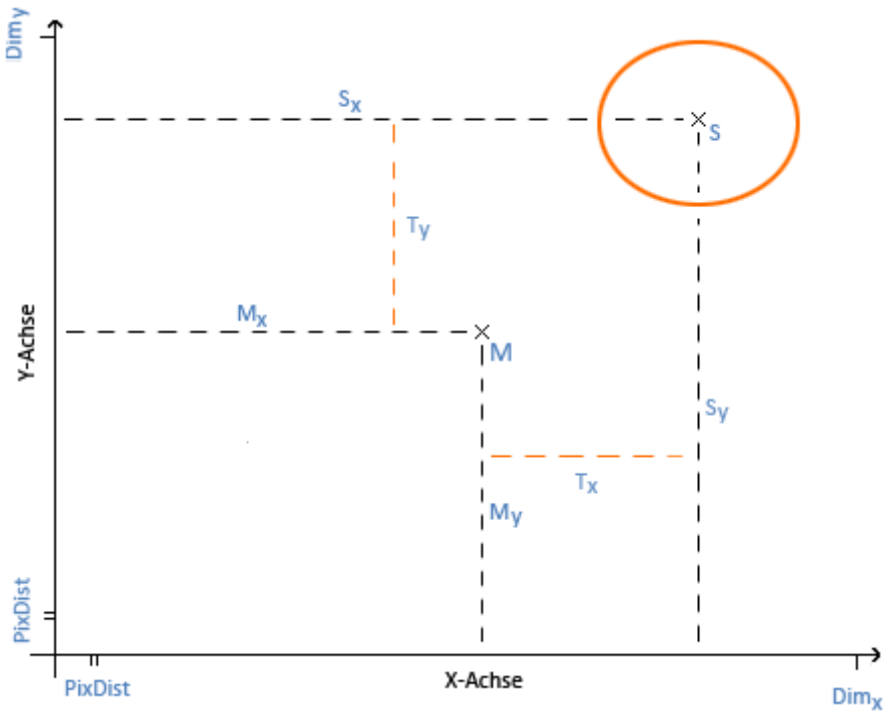

Abbildung 17: *Darstellung zur Verschiebung des VOI in einem 2D Koordinatensystem der Nullebene. Die orange gefärbte Ellipse stellt eine Kontur des VOI innerhalb der Ebene dar.*

Anhand des Mittelpunktes und des Schwerpunktes kann nun die Verschiebung auf der X- und Y-Achse der VOI berechnet werden. Dazu muss von der X-Koordinate des Mittelpunktes die X-Koordinate des Schwerpunktes subtrahiert werden. Genauso verfährt man mit den Y-Koordinaten der jeweiligen Punkte:

$$T_x = M_x - S_x$$

$$T_y = M_y - S_y$$

Die drei berechneten Werte zum Verschieben der VOI werden dann an das Modul zur Adaptierung von VOIs weitergegeben. Desweiteren werden diese Werte an die Schieberegler, die zum verschieben eines VOI implementiert wurden, weitergegeben.

Modell speichern

Ein weiterer Ablauf, der zur Verwaltung der Organmodelle gehört, ist die Möglichkeit, Organmodelle abzuspeichern. Nachdem der Benutzer entschieden hat, welche VOIs er aus dem aktuellen Bilddatensatz abspeichern möchte und unter welcher Organkategorie er diese abspeichern möchte, werden diese Daten an den OrgHandler übergeben.

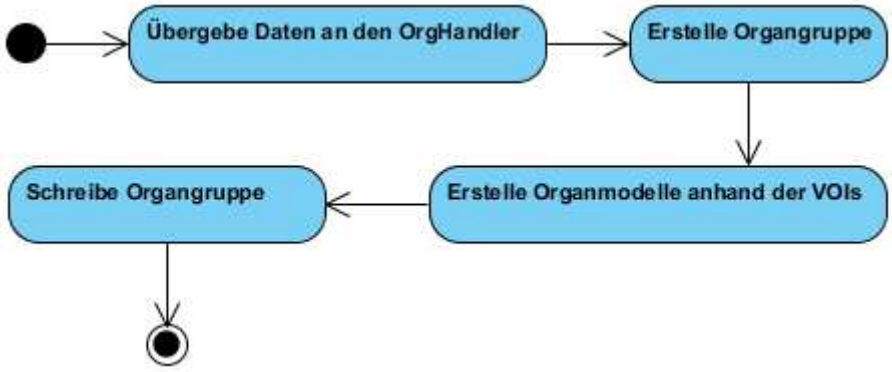

Abbildung 18: *Schematische Darstellung des Ablaufes zum Speichern eines Organmodells. Die abgerundeten Rechtecke stellen Aktivitäten und die eckigen Rechtecke stellen Objekte dar.*

Der OrgHandler erstellt aus diesen übergebenen Daten zunächst eine neue Organgruppe und fügt die Beschreibung sowie den Namen des Bilddatensatzes an. Danach wird für jedes VOI, das der Benutzer ausgewählt, hat ein neues Organmodell erstellt und der Organgruppe hinzugefügt. In jedem neu erstellten Organmodell wird das VOI gleich mit angehängt. Nun sind die Organmodelle bereits beim OrgHandler hinterlegt aber noch nicht auf der Festplatte gespeichert. Aus diesem Grunde wird im Anschluss direkt die komplette Organgruppe geschrieben.

Dazu werden aus allen Organmodellen, die der Organgruppe untergeordnet sind, die VOIs herausgeholt und wiederum in ein neues VoiModel gespeichert. Zusätzlich werden Beschreibungen und der Name des Bilddatensatzes im VoiModel hinterlegt. Dieses VoiModel wird dann mit Methoden aus dem VoiManSTL Modul in Form einer VDX-Datei im entsprechenden Ordner der Organkategorie abgespeichert. Damit ist das Abspeichern eines Organmodells bzw. mehrerer Organmodelle abgeschlossen.

3.4.2 Adaption von VOIs

Der nächste wichtige Schritt war die Implementierung eines Moduls, das die Oberflächendaten eines Organmodells - also VOIs - verschiebt, rotiert und skaliert. Wie bereits erwähnt, konnte hierbei auf bereits vorher implementierte Teile eines Moduls, welches ähnliche Aufgabe erfüllt, zurückgegriffen werden. Diese Teile wurden in der Octopus-Version von VIRTUOS implementiert (siehe 2.4.5). Allerdings mussten diese Implementierungen zum größten Teil überarbeitet werden, da diese teilweise zu statisch waren und in der Handhabung umständlich. Entscheidend für diese Implementierung ist, dass alle Punkte in homogenen Koordinaten ausgedrückt werden (siehe 2.4.3), was bedeutet, dass alle Transformationen von VOIs durch Matrixmultiplikationen realisiert werden können. Dies hat ebenfalls den Vorteil, dass komplexe Transformationen durch das Multiplizieren von einfachen Transformationen realisiert werden können.

Hat der Benutzer einen Wert der Schieberegler zur Transformation oder Translation verändert, so werden alle Werte der Schieberegler an das VoiManSTL Modul übergeben und dort verarbeitet. Zunächst wird der Mittelpunkt der Rotation und der Skalierung festgelegt. Aus diesem Grunde wird wieder der Schwerpunkt berechnet und als Mittelpunkt der Transformation festgelegt. Anschließend wird dann aus den drei Verschiebungswerten die Translationsmatrix erstellt. Gehen wir von den drei Translationswerten t_x, t_y, t_z aus, so sieht die Translationsmatrix wie folgt aus:

$$T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Nun muss die Matrix zur Rotation berechnet werden. Dies erfolgt in mehreren Einzelschritten. Zunächst werden die einzelnen Matrizen zur Rotation um die Koordinatenachsen berechnet. Dies erfolgt für die Rotation um die X-Achse mit Einbezug des Mittelpunktes auf folgende Weise. Rotationen erfolgen um den Mittelpunkt der VOI, deshalb muss der Mittelpunkt der VOI in den Ursprung des Koordinatensystems verschoben, die Rotationen durchgeführt und die VOI wieder an die ursprüngliche Position zurückverschoben werden:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -M_x & -M_y & -M_z & 1 \end{pmatrix}, B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ M_x & M_y & M_z & 1 \end{pmatrix}$$

Diese beiden Matrizen werden benötigt, um die Rotation um den Mittelpunkt des VOI berechnen zu können. Die eigentliche Rotationsmatrix mit dem Rotationswinkel α schaut wie folgt aus:

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

Diese Matrix berechnet die Rotation um die X-Achse. Um nun den VOI Mittelpunkt einzubeziehen, müssen zwei Matrixmultiplikationen erfolgen.

$$R_x = A * Rx$$

$$R_x = R_x * B$$

Die Berechnung der Rotationsmatrizen für die Rotation um die Y- und Z-Achse erfolgt auf die gleiche Weise. Lediglich die eigentliche Rotationsmatrix wird dafür leicht verändert (siehe 2.4.3).

Haben wir alle drei Rotationsmatrizen R_x, R_y, R_z berechnet, folgt nun die Berechnung der Gesamtrotationsmatrix:

$$R = R_x * R_y * R_z$$

Als letztes folgt nun die Berechnung der Skalierungsmatrix. Auch bei der Skalierung muss der VOI Mittelpunkt mit einbezogen werden, da sich ansonsten die Skalierung auf den Koordinatenursprung bezieht und dementsprechend falsch wäre. Deshalb brauchen wir hier auch wieder die beiden Matrizen A und B. Danach wird die normale Skalierungsmatrix, welche noch keinen Mittelpunkt berücksichtigt, mit den Skalierungsfaktoren S_x, S_y und S_z aufgestellt:

$$S = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Wie bei der Rotation wird auch hier durch Matrixmultiplikation die Gesamtskalierungsmatrix berechnet:

$$S = A * (S * B)$$

Dank der Verwendung der homogenen Koordinaten, kann nun eine Gesamttransformationsmatrix aus den drei Matrizen T, R und S berechnet werden:

$$G = (S * R) * T$$

G entspricht nun der Gesamttransformationsmatrix und wird deshalb in das VOI, das adaptiert werden soll, gesetzt. Da es allerdings möglich ist, dass dieses VOI untergeordnete VOIs enthält, müssen auch diese adaptiert werden. Deshalb werden alle untergeordneten VOIs geholt und deren Transformation mit der neuen Transformation des übergeordneten VOI multipliziert. Diese neue Matrix wird dann in dem untergeordneten VOI als neue Transformation gesetzt. Haben die untergeordneten VOIs ebenfalls wieder Sub-Vois, so wird mit diesen ebenso Verfahren, bis die niedrigste Hierarchieebene erreicht ist.

Nach all diesen Schritten muss nur noch die Benutzeroberfläche mit den verschiedenen Ansichten der CT-Bilder aktualisiert werden, damit die neue Position des VOI angezeigt wird.

3.5 Implementierung

Die Implementierung der Programmkomponente „OrgHandler“, die zur Verwaltung von Organmodellen dient und die direkt in VIRTUOS integrierte Funktion zum adaptieren von VOIs war die Hauptaufgabe dieser Arbeit. Dieses Kapitel beschäftigt sich mit der Entwicklung des ersten Moduls. Dabei konnten zum Teil bereits vorhandene Methoden von früheren Arbeiten genutzt werden. Zum einen wurde das Modul VoiManSTL in den OrgHandler integriert, da dieses Methoden und Klassen zur Verwaltung von VOIs und VoiModels beinhaltet (siehe 2.4.3). Außerdem wurde in der Arbeit das Modul XMLIOHandler verwendet. Dieses Modul stellt Methoden zur Verfügung um XML-Dateien zu laden und zu speichern. Beide Teile der Arbeit wurden in den Programmiersprachen C und C++ (siehe 2.5.1) geschrieben und eignen sich sowohl zum Einsatz auf Windows als auch auf dem Einsatz auf Linux. Zusätzlich wurde das MVC-Entwurfsmuster für die Implementierung genutzt (siehe 2.5.2).

Um eine effiziente Verwaltung der Organmodelle zu ermöglichen, enthält der OrgHandler mehrere Methoden zum Laden, Speichern und zur Rückgabe von Organmodellen. Folgende Abbildung erläutert, wie dabei der Zugriff auf den OrgHandler funktioniert.

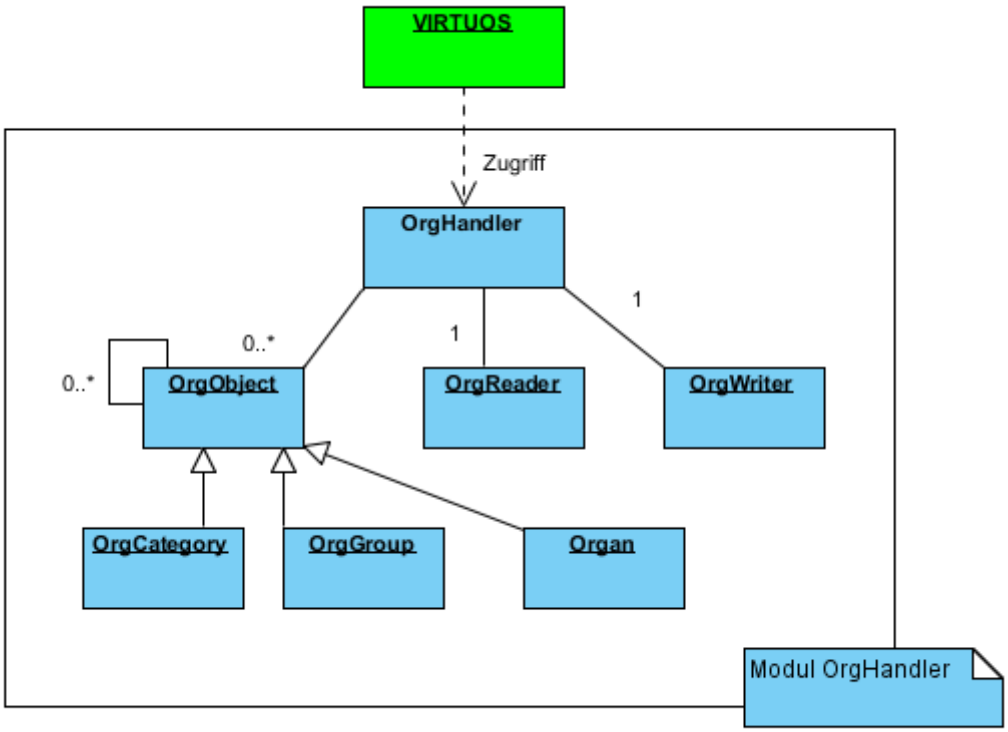

Abbildung 19: *Schematische Darstellung des Zugriffes auf das Modul OrgHandler.*

Der Zugriff auf das Modul OrgHandler kann nur über die Klasse OrgHandler geschehen. In dieser Klasse sind alle Methoden implementiert, die zur Verwaltung der Organmodelle nötig sind. Die Klassen OrgCategory, OrgGroup und Organ repräsentieren die oben erläuterten Objekte Organkategorie, Organgruppe und das Organmodell. Die Klassen OrgReader und OrgWriter wurden zum Laden und Speichern von Organmodellen implementiert.

3.5.1 OrgObject

Die Klasse OrgObject stellt die Basisklasse zu allen Modellklassen dar. Das heißt OrgObject ist die Basisklasse zu OrgCategory, OrgGroup und Organ. Da alle drei Modellklassen zum großen Teil über die gleichen Funktionalitäten verfügen, bietet sich dies an, da so sehr viel Quellcode gespart werden kann. Weiterhin bietet es den Vorteil, dass die Änderung an den Basisfunktionalitäten, die von allen Modellklassen unterstützt werden sollen, nur in der Basisklasse vollzogen werden müssen. Wird hier ohne Basisklasse gearbeitet, führt dies zu Mehraufwand und häufig zu Fehlern, da die Änderungen in jeder Modellklassen gemacht werden müssten. Folgendes Klassendiagramm zeigt alle Methoden und Attribute die diese Klasse unterstützt:

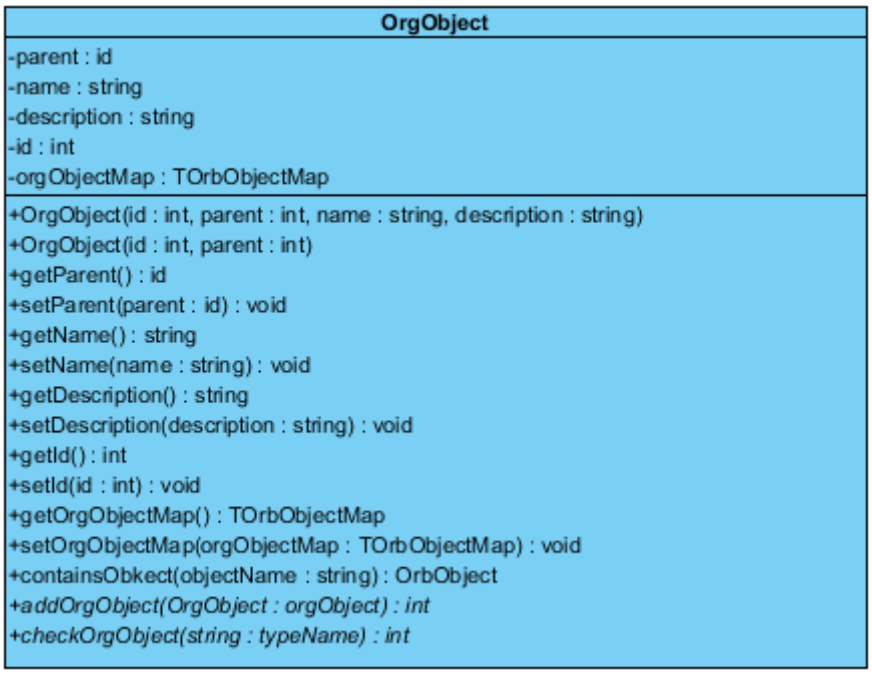

Abbildung 20: *UML-Klassendiagramm der Basisklasse OrgObjective. Die Methoden addOrgObjective und checkOrgObject sind abstrakte Methoden die in jeder Subklasse definiert seien müssen.*

Um jede Instanz einer Modellklasse unterscheiden zu können, gibt es das Attribut id. Die Generierung dieses Integers erfolgt beim Erstellen einer neuen InstanzDie id muss immer positiv und eindeutig sein. Auf die Eindeutigkeit der id wird beim Erstellen geachtet. Kann keine id erzeugt werden, so kann auch keine neue Instanz erzeugt werden. Da ein OrgObject häufig ein Vaterobjekt besitzt, ist es von großem Nutzen, wenn die id des Vaterobjekts im untergeordneten Objekt gespeichert wird. Dies ermöglicht ein schnelles Referenzieren des Vaterobjektes und vermeidet unnötige und teure Suchläufe.

Da, wie in Abbildung 12 erläutert, jede Instanz einer Modellklasse ein oder mehrere untergeordnete Instanzen von Modellklassen besitzen darf, wurde das Attribut orgObjectMap hinzugefügt. Dieses Attribut ist vom Typ TOrgObjectMap, welches als map mit dem Schlüssel id und dem Wert OrgObject definiert ist. Da nicht jedem Modellobjekt jedes Modellobjekt hinzugefügt werden darf, muss das Hinzufügen eingeschränkt werden. Zu diesem Zweck wurde die Basisklasse OrgObject um die abstrakte Methode addOrgObject() erweitert. Diese Methode muss von allen abgeleiteten Klassen implementiert werden. Mit dieser Methode ist es möglich einer OrgCategory, OrgGroup oder einem Organ ein OrgObject hinzuzufügen. Es können allerdings nur OrgObjects hinzugefügt werden, die

vom erlaubten Typ sind. Um nun überprüfen zu können, von welchem Typ ein bestimmtes OrgObject ist, wurde die abstrakte Methode checkOrgObject() eingeführt. Diese überprüft, ob der übergebene string mit dem Typ des Objektes übereinstimmt. Auch diese Methode muss deshalb von jeder Subklasse implementiert sein. Das Hinzufügen eines OrgObject wird nun über den Aufruf von addOrgObject() realisiert. Diese überprüft, ob das übergebene OrgObject vom richtigen Typ ist und fügt dieses dann der map hinzu.

Da sowohl Organkategorie, Organgruppe als auch das Organmodell selber einen Namen und auch eine Beschreibung besitzen, wurden in der Basisklasse die äquivalenten Attribute name und description hinzugefügt. Passende Getter und Setter Methoden wurden ebenfalls hinzugefügt.

OrgCategory

Die Modellklasse OrgCategory, die die Organkategorie definiert, erbt direkt von der Klasse OrgObject. Das heißt diese Klasse muss die beiden Methoden addOrgObject() und checkOrgObject() definieren. Da, wie in Abbildung 12 erläutert, eine Organkategorie sowohl ein oder mehrere Organkategorien als auch Organgruppen besitzen darf, wurde die Methode addOrgObject() so definiert, dass Instanzen der Typen OrgGroup und OrgCategory hinzugefügt werden können.

Die Methode checkOrgObject() wurde so implementiert, dass sie übergebene Parameter mit dem Wert „OrgCategory“ als korrekt erkennt.

Da die Organkategorie nicht mehr Anforderungen stellt, benötigt die erläuterte Klasse keine weiteren Erweiterungen.

OrgGroup

Die Klasse OrgGroup, die die Organgruppe definiert, erbt ebenfalls direkt von der Klasse OrgObject. Die Methode addOrgObject() wurde so definiert, dass nur Instanzen vom Typ Organ hinzugefügt werden können. Analog dazu wurde die Implementierung der Methode checkOrgObject() so realisiert, dass diese den Wert „OrgGroup“ als korrekt zurückgibt.

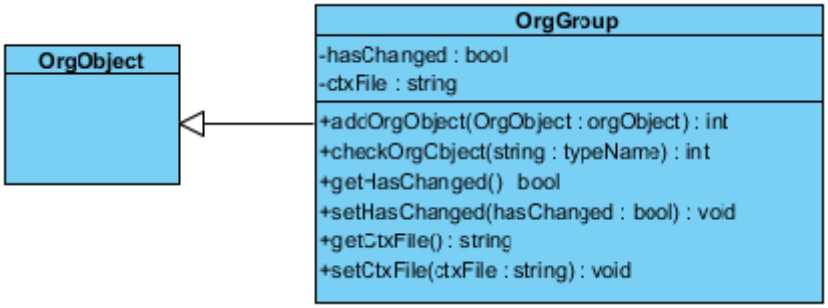

Abbildung 21: *UML-Klasendiagramm der Modellklasse OrgGroup. Die geerbten Methoden addOrgObject() und checkOrgObject mussten implementiert werden. Die Oberklasse OrgObject wird hier vereinfacht dargestellt.*

Dieser Klasse mussten noch zwei weitere Attribute mit entsprechenden Getter und Setter Methoden angehängt werden. Da die Oberfläche der Organmodelle durch eine VOI gegeben ist und die

Organgruppe ein oder mehrere Organmodelle hält, sollte der gemeinsame Bilddatensatz auf dem die Organmodelle beruhen, hinterlegt sein (siehe 3.2). Dafür wurde das Attribut ctxFile hinzugefügt.

Das zweite Attribut, welches hinzugefügt wurde, heißt hasChanged und wird benötigt um zu markieren, dass sich innerhalb der Organgruppe etwas geändert hat. Eine Änderung kann bedeuten, dass zum Beispiel eine der Beschreibungen der Organmodelle, die die Organgruppe besitzt, oder die Beschreibung der Organgruppe selbst durch den Benutzer geändert wurde. Ist dies der Fall, so muss die VDX-Datei, auf dem die Organgruppe basiert, neu geschrieben werden.

Organ

Die letzte Modellklasse ist die Klasse Organ. Diese definiert ein Organmodell, welches durch ein VOI definiert wird (siehe 3.2). Da ein Organmodell untergeordnete Organmodelle besitzen darf, wurde die geerbte abstrakte Methode addOrgObject() so realisiert, dass diese nur Instanzen vom Typ Organ hinzufügt. Die Methode checkOrgObject() gibt Parameter mit Wert „Organ“ als korrekt zurück.

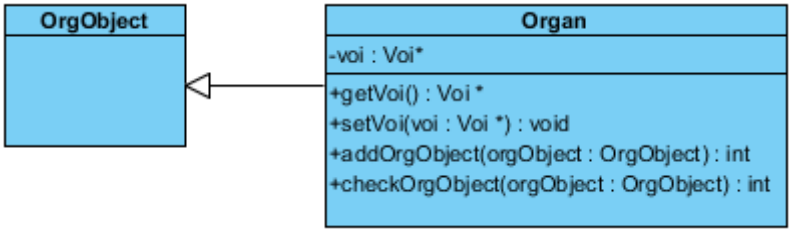

Abbildung 22: *UML-Klasendiagramm der Modellklasse Organ. Die geerbten Methoden addOrgObject und checkOrgObject mussten implementiert werden. Die Oberklasse OrgObject wird hier vereinfacht dargestellt.*Das Attribut voi enthällt einen Zeiger auf die Speicheradresse, in dem das VOI gespeichert ist, das die Oberfläche des Organmodells beschreibt.

3.5.2 OrgReader und OrgWriter

Zum Speichern und Laden von Organmodellen wurden die beiden Klassen OrgWriter und OrgReader bestimmt. Die Implementierung der Speicheralgorithmen sind in der Klasse OrgWriter und äquivalent dazu sind die Ladealgorithmen in der Klasse OrgReader zu finden. Da die Organmodelle in Organgruppen zusammengefasst sind und eine Organgruppe durch ein VoiModel definiert ist (siehe 3.2), liegt es nahe, dass die zum Schreiben und Laden von VoiModels implementierten Methoden im VoiManSTL Modul verwendet werden. Die Methoden zum laden von VDX-XML Dateien wurden ebenfalls im Laufe dieser Arbeit erstellt.

Da das Auslesen von Organmodellen die Navigation durch Verzeichnisse erfordert und diese Routinen Betriebssystem abhängig sind, musste etwas gefunden werden, dass sowohl Linux als auch Windows als Betriebssystem unterstützt. Das Modul file_util.c, dass im Hauptprogramm VIRTUOS implementiert wurde, stellt diese Funktionalität zu Verfügung und wurde deswegen ebenso eingebunden.

OrgReader

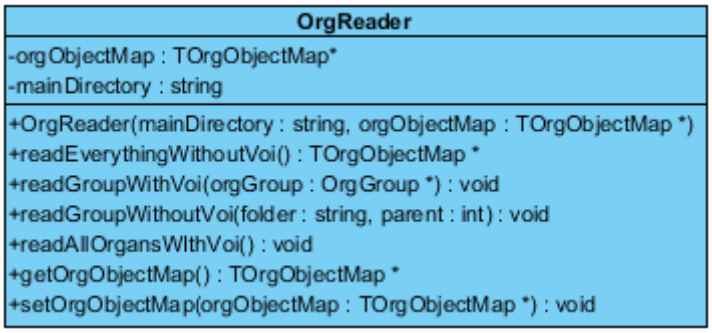

Abbildung 23: *Klassendiagramm zur Klasse OrgReader. Private Methoden wurden nicht aufgenommen.*

Der OrgReader wird mit zwei Parametern instanziiert. Der erste Parameter ist der string mainDirectory. Dieser enthält den Namen des Ordners in dem alle Organmodelle hierarchisch durch Organkategorien und Organgruppen geordnet abgespeichert sind. Der zweite Parameter ist vom Typ TOrgObjectMap und enthält einen Zeiger auf die Speicheradresse der map, die alle Organmodelle, Organgruppen und Organkategorien hält.

Zum Auslesen von Organmodellen gibt es gleich mehrere Routinen. Dies ermöglicht eine effiziente Verwaltung der Organmodelle. Zum einen gibt es die Methode readEverythingWithoutVoi(), die alle Organkategorien, Organgruppen und Organmodelle ausliest. Entscheidend in dieser Methode ist, dass nur der „Header“ (siehe 3.2.2) der einzelnen VDX-Dateien gelesen und der Bereich „Body“ (siehe 3.2.2) ignoriert wird. Dies führt dazu, dass die gelesenen VOIs keine Oberflächendaten beinhalten und der entsprechende Wert im Objekt Organ nicht gesetzt wird. readGroupWithVoi() ist die nächste Prozedur, die entwickelt wurde. Diese Methode fügt allen Organen der übergebenen Organgruppe die entsprechende VOI hinzu. In dieser Methode wird die übergebene OrgGroup erneut ausgelesenen. Dabei wird in der betroffenen VDX-Datei sowohl der „Header“- als auch der „Body“-Bereich ausgelesen. Dies hat zur Folge, dass die Organmodelle, die der Organgruppe untergeordnet sind, VOIs mit den entsprechenden Oberflächendaten halten. Die Methode readGroupWithoutVoi() liest eine Organgruppe, definiert durch den string folder, ohne Oberflächendaten aus. Der Parameter folder definiert den Ordner, der der Gruppe zugeordnet ist. Diese Methode wird maßgeblich in readEverythingWithoutVoi() verwendet. readAllOrgansWithVoi() ist die letzte wichtige realisierte Methode. Diese Routine liest alle Organkategorien, Organgruppen und Organmodelle aus. Die Organmodelle werden ebenfalls mit den entsprechenden VOIs ausgelesen. Diese verschiedenen Methoden ermöglichen es Organmodelle effizient zu verwalten. Da es sehr unwahrscheinlich ist, dass zur Laufzeit des Programms jedes Organmodell benötigt wird ist es unsinnig, zu jedem Organmodell direkt die Oberflächendaten zu laden, da dies sehr viel Zeit in Anspruch nimmt. Dafür gibt es die Methoden readGroupWithoutVoi() und readEverythingWithoutVoi(). Werden nun von einem Organ in einer Organgruppe Oberflächendaten benötigt können diese mit den Methoden readAllOrgansWithVoi() und readGroupWithVoi() geladen werden.

OrgWriter

Die Klasse OrgWriter basiert maßgeblich auf drei Methoden, auf die von außen zugegriffen werden kann. Ebenso wie der OrgReader hält der OrgWriter den Zeiger auf die Speicheradresse, in der die map vom Typ TOrgObjectMap liegt, welche alle OrgObjects enthält. Weiterhin enthält die Klasse auch den Namen des Hauptordners, in dem alle Organmodelle gespeichert sind. Eine Aufteilung der Lese-

und Schreibalgorithmen wurde hier durchgeführt, um eine bessere Strukturierung der Programmstruktur zu erhalten.

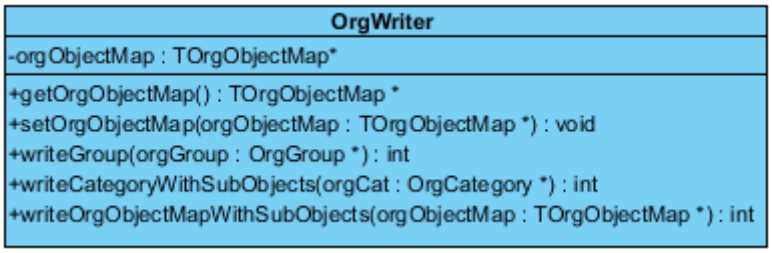

Abbildung 24: *Klassendiagramm zur Klasse OrgWriter. Private Methoden wurden nicht aufgenommen.*

Die erste der drei Hauptmethoden heißt writeGroup() und wird verwendet um eine Organgruppe mit allen Organmodellen, die sie besitzt, zu schreiben. Sollte noch kein Ordner vorhanden sein, der dem Namen der Organgruppe entspricht, so wird dieser erstellt. Weiterhin wird in diesen Ordner der in der Organgruppe angegebene Bilddatensatz kopiert, sofern er vorhanden ist. Danach wird aus der Organgruppe und allen Organmodellen, die der Gruppe untergeordnet sind, ein neues VoiModel erstellt, welches dann mit Hilfe einer Prozedur aus dem Modul VoiManSTL (siehe 2.4.3) eine neue VDX-Datei im XML-Format in den Gruppenordner schreibt. Dies ist nötig da eine Organgruppe samt ihrer Organe in einem Ordner mit den Konturdaten der VOIs und der entsprechenden Bildserie in einem Ordner abgespeichert werden muss. Die Methode writeCategoryWithSubObjects() schreibt eine Organkategorie und alle die Objekte, die ihr untergeordnet sind. Sollte auch hier kein Ordner vorhanden sein, der die Organkategorie definiert, und daher vorläufigen gleichen Namen wie die Organkategorie besitzt, so wird ein neuer erstellt. Die letzte Methode, die zum abspeichern von Informationen dient, ist die Methode writeOrgObjectMapWithSubObjects(). Diese Methode iteriert über die übergebene map vom Typ TOrgObjectMap und schreibt, je nachdem ob eine Organgruppe oder eine Organkategorie vorliegt, diese.

3.5.3 OrgHandler

Das Bindeglied zwischen dem Hauptprogramm VIRTUOS und dem entwickelten Modul OrgHandler stellt die gleichnamige Klasse OrgHandler dar. Diese Klasse stellt alle Methoden zur Verfügung, um Organmodelle, Organgruppen und Organkategorien zu verwalten und leitet diese an die weiteren Klassen des Moduls weiter.

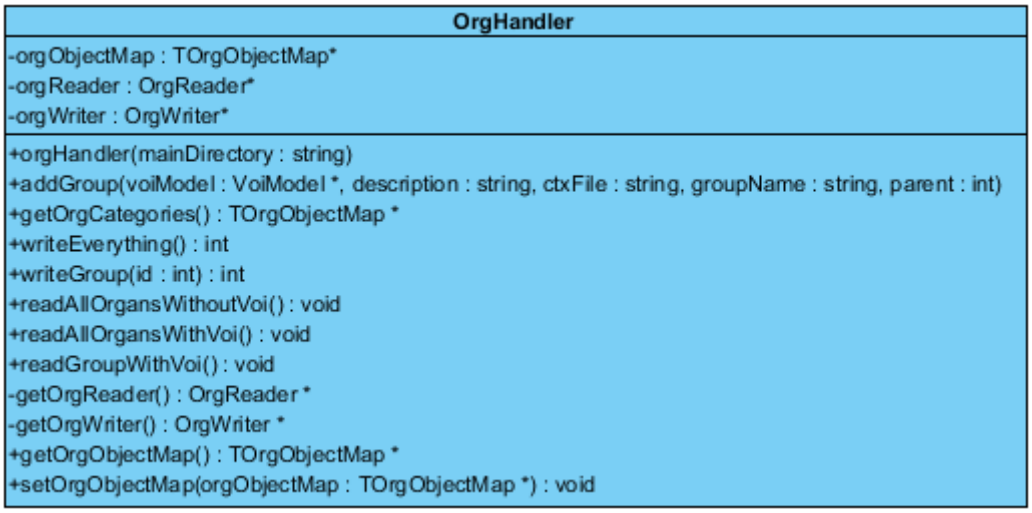

Abbildung 25: *Klassendiagramm von der Klasse OrgHandler. Nur die relevanten Methoden wurden aufgenommen.*Die Klasse OrgHandler wird von VIRTUOS mit dem Namen des Hauptordners, in dem alle Daten liegen, instanziiert. Sollte der OrgHandler die Klassen OrgReader oder OrgWriter enthalten, so werden diese einmal initiiert und dann solange gehalten, wie der OrgHandler existiert. Dieses Entwurfsmuster nennt sich Singleton. Die Methode addGroup() erstellt aus den übergebenen Parametern eine Organgruppe und schreibt diese auch direkt in das Ordnersystem. Das heißt, dass beim Aufruf von addGroup() die Verzeichnisstruktur auf der Festplatte direkt verändert wird. Alle VOIs die in dem übergebenen VoiModel enthalten sind, werden dabei zu einem Organmodell. Die Methode writeEverything() schreibt alle neuen Daten neu oder überschreibt falls nötig die alten Daten. Die Methode writeGroup() führt die gleichnamige im OrgWriter aus, zuvor wird allerdings über die id die richtige Organgruppe geholt. Die Methoden readAllOrgansWithoutVoi() und readAllOrgansWithVoi() lesen alle Daten ohne oder mit Oberflächendaten. readGroupWithVoi() führt die gleichnamige Methode in der Klasse OrgReader aus.

Die Methode getOrgCategories() ist für den Zugriff von außen die bedeutendste, da diese eine map mit dem Typ TOrgObjectMap zurückgibt. Diese map enthält alle Organkategorien, die auf der höchsten Ebene liegen. Das heißt, in dieser map sind alle Organkategorien, die keine übergeordnete Organkategorie besitzen.

3.6 Evaluation

Die Evaluation der entwickelten Programmkomponente wurde ausschließlich in Bezug auf ihre Funktionalität durchgeführt. Eine Evaluation hinsichtlich der Vereinfachung der Segmentierung in Kombination mit semi-automatischen Segmentierungsalgorithmen konnte nicht durchgeführt werden, weil einerseits die Algorithmen noch nicht vollständig implementiert sind und andererseits eine solche Evaluation den Aufwand im Rahmen dieser Arbeit zu groß gestalten würde.

3.6.1 Durchführung

Zum Nachweis der entwickelten Funktionalität wurden die beschriebenen Abläufe in allen Varianten durchgeführt. Zunächst wurde das komplette Ordnersystem bereinigt, sodass die Datensätze, die zum Testen während der Entwicklung genutzt wurden, nicht mehr vorhanden waren. Nachdem dieser Schritt vollzogen wurde, wurden Organmodelle auf Basis anonymisierter Patientendaten erstellt. Die Daten wurden freundlicherweise von der Klinik für Radioonkologie und Strahlentherapie der Universität Heidelberg zur Verfügung gestellt. Es handelte sich dabei um Datensätze aus der Strahlentherapie, die für die Bestrahlungsplanung bereits segmentiert waren. Insgesamt konnten so 10 Datensätze für die Kategorien „Brust“(Thorax), „Bauch“(Körperstamm), „Hals“ und „Kopf“ importiert werden. Folgende Abbildung zeigt die Organliste und eine CT-Ansicht, auf der die gespeicherten VOIs zu sehen sind:

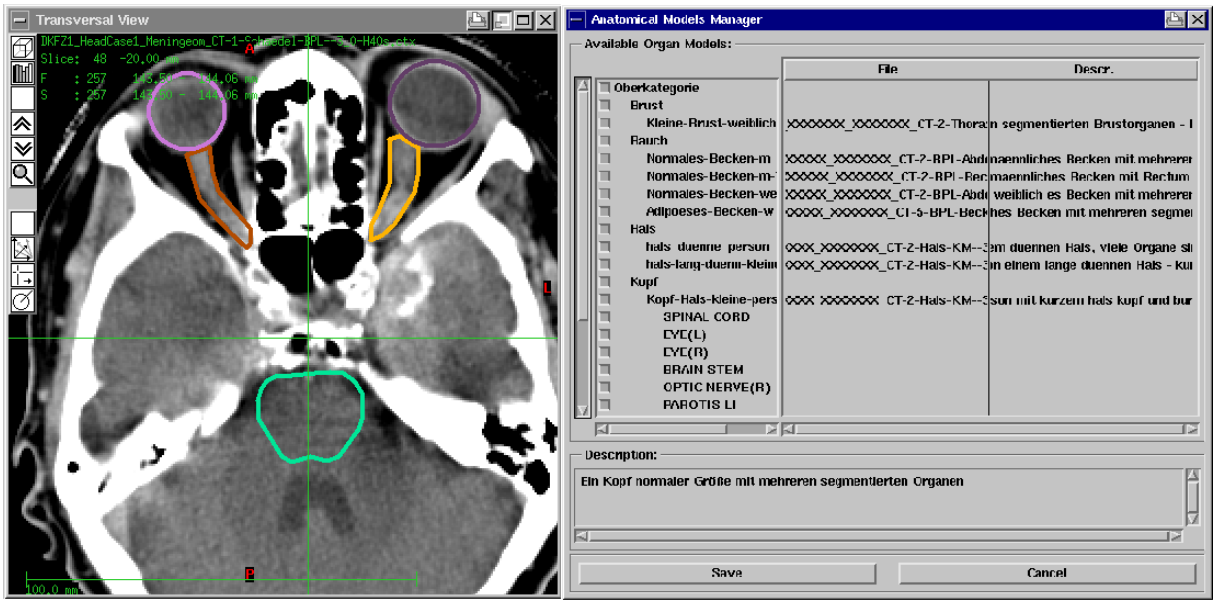

Abbildung 26: *Links sind die VOIs eines anonymisierten Datensatzes, welcher in die Organdatenbank unter der Organgruppe „Kopf-Hals-kleine-person“ aufgenommen wurde.*

Damit wurde nach gewiesen, dass der Ablauf zum Speichern von Organmodellen funktioniert.

Der nächste Ablauf, der auf seine Funktionsweise getestet wurde, ist der Ablauf zum Laden von Organmodellen und Organgruppen. Dazu wurde ein Bilddatensatz geladen, der keinerlei VOIs enthält. Danach wurde aus der Organliste ein Organ ausgewählt und geladen. Das Ergebnis ist in Abbildung 14 zu sehen, die eine Kontur des VOI (Hirnstamm) zeigt. Der Mittelpunkt des VOI wurde in die Nullebene gelegt und in der Ebene in den Mittelpunkt verlagert. Zusätzlich dazu wurde direkt das Modul zum Adaption von VOIs evaluiert. Dazu wurde zunächst das geladene Organmodell mit allen

Schiebereglern transformiert und verschoben um zu zeigen, dass alle Methoden der Transformation (Translation, Rotation und Skalierung) problemlos funktionieren. Als nächstes wurde das geladene Organmodell richtig positioniert. Das heißt es wurde dort positioniert, wo es auch im Bilddatensatz zu finden ist. Dies konnte ebenso problemlos durchgeführt werden, so dass auch hier der Nachweis erbracht werden konnte, dass das Modul seine gewünschte Funktionsweise erfüllt.

4 Diskussion und Ausblick

Ziel dieser Arbeit war es eine Programmkomponente zu entwickeln, die es ermöglicht Organmodelle in dem Bestrahlungsplanungsprogramm VIRTUOS (VIRTUal RadiOtherapy Simulator), welches am Deutschen Krebsforschungszentrum in Heidelberg entwickelt wird, effizient zu verwalten. Weiterhin soll es die Möglichkeit geben, diese in den aktuellen Patientendatensatz zu laden und diese richtig zu positionieren, auszurichten und zu skalieren, um als Vorlage für semi-automatische Segmentierungsalgorithmen, die eine Startkonfiguration benötigen, zu dienen. Um dem Benutzer die Auswahl eines Organmodells zu vereinfachen, sollten zu den Modellen auch Organgruppen und Organkategorien entwickelt werden. Dafür sollte ein Datei-/Ordnersystem geschaffen werden, welches dies ermöglicht. Die entwickelte Komponente soll daher dieses Datei-/Ordnersystem auslesen und in einer hierarchischen Form dem Benutzer präsentieren können. Weiterhin soll es dem Benutzer möglich sein durch die angezeigte Liste von Organgruppen, Organkategorien und Organmodellen zu navigieren und ein oder mehrere Organe auszuwählen und dem Patientendatensatz hinzuzufügen um ein 3D-Patientenmodell zu erstellen. Ebenfalls soll es die Möglichkeit geben Organmodelle neu zu erstellen und diese im Datei-/Ordnersystem zu hinterlegen. Durch mehrere Schieberegler soll die Möglichkeit geschaffen werden, die geladenen Organmodelle zu positionieren, auszurichten und zu skalieren.

Das entwickelte Modul OrgHandler, welches zur Verwaltung von Organmodellen dient, wurde so aufgebaut, dass es mit Organkategorien, Organgruppen und Organmodellen umgehen kann. Dazu wurde eine Datei-/Ordnerstruktur geschaffen, die dies ermöglicht. Organkategorien und Organgruppen werden durch Ordner dargestellt. Eine Organkategorie kann dabei entweder weitere Kategorien oder Gruppen enthalten. Eine Organgruppe enthält ein oder mehrere, Organe die gemeinsam in einer Datei auf der Festplatte hinterlegt sind. Dadurch, dass diese Datei eine XML-Datei ist, können Organe Sub-Organe besitzen, die hierarchisch in dieser Datei abgespeichert werden. Diese Ordnerstruktur wird in der Benutzeroberfläche in einer separaten Dialogbox angezeigt. Dadurch ist es dem Benutzer möglich durch die Kategorien, Gruppen und Modelle zu navigieren und gegebenenfalls zu markieren. So ist es gewährleistet, dass der Benutzer schnell und einfache die gewünschte Gruppe oder das gewünschte Modell findet. Das entwickelte Modul verfügt nun über Methoden um die Kategorien, Gruppen und Modelle auszulesen, zu verwalten und zu speichern. Um die Ladezeiten gering zu halten, gibt es mehrere Abläufe zum Laden. Organmodelle können ohne ihre Oberflächendaten, die im Regelfall sehr viel Platz wegnehmen geladen werden. Bei Bedarf, wenn der Benutzer das Organ in den aktuellen Patientendatensatz hinzufügen möchte, werden die Oberflächendaten des Organs nachgeladen. Möglich ist dies durch die neu entwickelte VDX-Datei. Diese basiert auf XML und besitzt einen Dokumentenkopf, in dem alle Daten zum VoiModel und den Namen der einzelnen VOIs hierarchisch sind, sowie einem Bereich der alle Oberflächendaten der einzelnen VOIs beinhaltet. Der Dokumentenkopf kann getrennt vom Rest der Datei ausgelesen werden, sodass keine Oberflächendaten ausgelesen werden müssen. Dies ermöglicht ein effizientes Verwalten der Modelle, ohne dass Benutzer besonders beim Start des Programms durch lange Ladezeiten gehemmt wird. Alle gestellten Anforderungen zum Verwalten von Organmodellen können damit erfüllt werden.

Teilweise kann in diesem Modul der Ablauf - bedingt durch bereits implementierte Methoden - noch optimiert werden. Möchte der Benutzer lediglich ein Organmodell in einen Bilddatensatz laden, so muss die gesamte übergeordnete Organgruppe ausgelesen werden, da alle Organmodelle, die zu einer Organgruppe gehören, in einer gemeinsamen Datei gespeichert werden. Dies kann dazu führen, dass der Ablauf zum Laden eines Organmodells langsam verläuft, da gerade in solchen Dateien sehr viele Informationen stehen. Dadurch können die Lesezeiten solcher Dateien in den Sekunden- und Minutenbereich gelangen. Schon solche geringen aber bemerkbaren Vorgangseiten, die bei dem Laden eines einzelnen Organmodells nicht erwartet werden, führen dazu, dass sich der Benutzer unwohl beim Benutzen dieser Programmkomponente fühlen könnte. Hier eröffnet sich für diese Programmkomponente weiteres Verbesserungspotential zur Effizienzsteigerung. Dies ließe sich dadurch lösen, dass jedes Organmodell in einer VDX-Datei gespeichert wird, bringt aber auch den Nachteil mit sich, dass die Modelle in der aktuellen Anatomie nicht mehr zusammen angepasst werden können. Die Ladezeit wäre mit dieser Lösung kürzer.

Weiteres Verbesserungspotential zeigt sich bei der Positionierung während des Ladens eines Organmodells. Da der Schwerpunkt des VOI in Z-Richtung in die Nullebene gelegt wird, also in die Ebene, in der die Koordinate 0 durchlaufen wird, befindet sich diese beim Laden nicht immer an der selben Position. Dies führt dazu, dass die Nullebene in Z-Richtung pro Bilddatensatz unterschiedlich liegt und nicht immer in der Mitte. Auch hier wäre eine Positionierung in die aktuelle CT-Schicht oder in die tatsächliche Mitte des Datensatzes sinnvoller.

Um geladene Organmodelle zu transformieren, wurden in die Benutzeroberfläche von VIRTUOS mehrere Schieberegler eingebaut um Organmodelle zu verschieben, zu skalieren und zu rotieren. Die Rotation wird über die Angabe eines Winkels, die Translation über die Angabe des Verschiebungswertes und die Skalierung über die Angabe eines Skalierungsfaktors berechnet. Die Berechnung der Transformation kann so schnell berechnet werden, dass die Bearbeitung flüssig und ohne Wartezeiten durchgeführt werden kann. Auch hier erfüllt das Modul die angestrebten Anforderungen.

Die Zukunft bietet mehrere Aktionsfelder mit Optimierungspotential, die die Nutzbarkeit des Systems erhöhen werden. Ein ganz zentraler Punkt stellt hier die Verbindung mit einem Algorithmus zur elastischen Registrierung dar. Gerade im Bauch/Becken Bereich des Körpers ist es so, dass durch verschiedene Füllstände von Darm und Blase die umliegenden Organe deformiert werden. Selbiges kann auch durch Atembewegung passieren. Genauso ist es möglich, dass im Gehirn z.B. ein Tumor den Hirnstamm deformiert und wegdrückt. Eine Möglichkeit wäre, das in dieser Arbeit entwickelte System mit einem Algorithmus zur elastischen Registrierung zu verbinden. Damit könnte diese Deformation berechnet und direkt auf das Organmodell angewandt werden. Dies würde die Effizienz dieser Komponente um einiges erhöhen und manuelle Nachbearbeitung der Konturen verringern. Dies könnte so weit geführt werden, dass eine komplette Atlasbasierte Segmentierung - wie in Kapitel 2.3.4 erläutert- durchgeführt werden kann. Die Arbeit [MS09] zeigt dazu einen Ansatz. Dort kann anhand eines Atlasdatensatz in Kombination mit einem elastischen Registrierungsalgorithmus ein kompletter Patient segmentiert werden.

Weiterhin von Nutzen könnte die Verbindung des entwickelten Moduls mit einem „Active-Contours“ Algorithmus – vorgestellt in Kapitel 2.3.4 – sein, da diese Art der Algorithmen eine Startkontur benötigen um die gewünschte Kontur zu berechnen. Dort könnte das entwickelte Modul die Startkonturen liefern und der Algorithmus berechnet die endgültige Kontur.

Eine weitere Verbesserung wäre die Erweiterung des Workflows zur Adaption von Organmodellen. Die Ausführung der Transformation und Translation über Schieberegler bietet zwar jegliche Funktionen, die benötigt werden, kann aber durchaus noch erweitert werden. So wäre eine Ausführung der Adaption in den CT-Ansichten eine sinnvolle Erweiterung. Der Benutzer könnte innerhalb der CT-Ansicht ein VOI direkt verschieben, rotieren oder skalieren. Da dies aus bereits ähnlichen Funktionen, die in VIRTUOS implementiert sind, bekannt ist, wäre es keine große Umstellung für den Benutzer und die Nutzung des Systems würde vereinfacht.

5 Literaturverzeichnis

[ALG06] Lisa D. Sprague, Michael Molls, Anca-Ligia Grosu: Definition of Target Volume and Organs at Risk. Biological Target Volume. *New Technologies in Radiation Oncology*, S. 167–177, Springer, 2006.

[Ben91] Rofl Bendl: *Entwicklung und Implementation einer benutzerschnittstelle zur virtuellen Strahlentherapiesimulation*. Diplomarbeit, Universität Heidelberg/Hochschule Heilbronn, 1991.

[Ben11] Prof. Dr. R. Bendl: *Therapiesysteme Teil 1: Strahlentherapie*. Skript, Juli 2011.

[Cos11] Peter H. Cossmann: Medizinische Strahlentherapie. *Medizintechnik*, S. 589–606, 2011.

[Des11] Thomas M. Deserno: Medizinische Bildverarbeitung. *Medizintechnik*, S. 825–846, 2011.

[DKF05] DKFZ: *VIRTUOS-Benutzerhandbuch-V4.6.10*, 2005.

[DKF11a] DKFZ: *Octopus – ocular tumour planning utilities*, http://www.dkfz.de/en/medphys/Therapy_planning_development/Projects/Octopus.html, August 2011.

[DKF11b] DKFZ: *VIRTUOS – virtual radiotherapy simulator*, http://www.dkfz.de/en/medphys/Therapy_planning_development/Projects/Virtuos.html, August 2011.

[Eis11] Urs Eisenmann: *Computergestützte Planung und Durchführung neurochirurgischer Interventionen*. Skript, Juli 2011.

[Fis96] Hans-Jörg Fischer: *Ein objektorientiertes Konzept zur Modellierung, Verwaltung und Manipulation geometrischer Daten und medizinisch/therapeutischem wissen von anatomischen Strukturen in der 3d-therapieplanung*. Diplomarbeit, Universität Heidelberg/Fachhochschule Heilbronn, 1996.

[MH11] Dr. Lena Maier-Hein: *Segmentierung mit Form-und Erscheinungsmodellen*. Skript, Juli 2011.

[MS07] Tobias Heimann, Hans-Peter Meinzer, Mareike Schönig: Parametrisierung geschlossener Oberflächen für die Erzeugung von 3d-formmodellen. *Bildverarbeitung für die Medizin*, S. 394–398, 2007.

[MS09] Jan Ehrhardt, Andreas Plaß, Heinz Handels, Michael Schwenke, Matthias Färber: Atlasbasierte 3d-Segmentierung medizinischer Bilddaten mit fast-marching-methoden. *Bildverarbeitung für die Medizin*, S. 172–176, 2009.

[RB95] Angelika Hoess und Wolfgang Schlegel Rolf Bendl: Virtual simulation in radiotherapy planning. *CVRMed*, S. 287–292, 1995.

[Rüp09] Jan Rüppel: *Design and implementation of a toolkit for three-dimensional manual segmentation and editing in radiotherapy simulation software.* Diplomarbeit, Universität Heidelberg/Hochschule Heilbronn, 2009.

[Sch06] Wolfgang Schlegel: New technologies in 3d conformal radiation therapy: Introduction and overview. *New Technologies in Radiation Oncology*. S. 1–6, Springer, 2006.

[Wik11a] Wikipedia: *C++*, http://de.wikipedia.org/w/index.php?title=C%2B%2B&oldid=92666670, August 2011.

[Wik11b] Wikipedia: *Extensible markup language*, http://de.wikipedia.org/w/index.php?title=Extensible_Markup_Language&oldid=92522511, August 2011.

[Wik11c] Wikipedia: *Motif*, http://de.wikipedia.org/w/index.php?title=Motif&oldid=88139146, August 2011.

[Wik11d] Wikipedia: *Mvc-konzept*, http://de.wikipedia.org/w/index.php?title=Model_View_Controller&oldid=91601675, August 2011.

[Wik11e] Wikipedia: *Strahlentherapie*, http://de.wikipedia.org/w/index.php?title=Strahlentherapie&oldid=91767075, August 2011.

[WS06] Thomas Bortfeld, Wolfgang Schlegel, Anca-Ligia Grosu: *New Technologies in Radiation Oncology*. Springer, 2006.

[WS07] Andreas Mahr Wolfgang Schlegel: *3D Conformal Radiation Therapy, CD-ROM Multimedia Introduction to Methods and Techniques*. Springer, 2007 .

6 Anhang

Abbildungsverzeichniss

Abkürzungen

CT – Computertomograph

MRT – Magnetresonanztomograph

VOI – Volume Of Interst

ROI – Region of Interest

DKFZ – Deutsches Krebsforschungszentrum

XML - Extensible Markup Language

VIRTUOS - VIRTUal RadiOtherapy Simulator

IMRT – Intensitätsmodulierte Strahlentherapie

IGRT – Image-guided Radiation Therapy

Danksagung

Während der Erstellung dieser Arbeit haben mir viele Personen geholfen denen ich hiermit meinen Dank aussprechen möchte.

Ich möchte sehr herzlich Prof. Dr. Rolf Bendl danken, der meine Bachelorarbeit beaufsichtigt hat und mir immer mit Rat und Tat, besonders bei der Erstellung der Benutzeroberfläche zur Seite stand. Auch geht dieser Dank an Dr. Kristina Giske, die sich als Ko-Korrektorin für diese Arbeit zu Verfügung gestellt hat.

Desweiteren möchte den Mitarbeitern der Arbeitsgruppe Therapieplanung – Entwicklung, die für mich immer ein offenes Ohr hatten und mit wertvollen Tipps meine Arbeit sehr erleichtert haben danken.

Auch möchte ich mich bei meinen Eltern bedanken, die dieses Studium immer unterstützt haben und auch mir auch in schwierigen Zeiten Mut gemacht haben. Besonderer Dank geht an meinen Vater der sich bereit erklärt hat, diese Arbeit zu prüfen.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eidesstatt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen(einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher weder im Inland noch Im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und ist auch noch nicht veröffentlicht.

Bernd Hemmer

Neuss, den 31.August 2011