# Endbericht zum Berufspraktikum

Christian Schafleitner

B A K K A L A U R E A T S A R B E I T

Nr. 03-1-0238-058-B

eingereicht am
Fachhochschul-Bakkalaureatsstudiengang

Medientechnik und -design

in Hagenberg

im September 2006

Praktikumsstelle:

Siemens Corporate Research, Inc.
Imaging and Visualization Dept.

755 College Road East
Princeton, NJ 08540, USA

1-609-732-6500
www.scr.siemens.com

Kontaktperson:

Dr. Jens Gühring
Projektleiter

# Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Hagenberg, am 31. August 2006

Christian Schafleitner

# Inhaltsverzeichnis

# Kurzfassung

Berufspraktikum bei Siemens Corporate Research, Inc. in Princeton, New Jersey, USA von Christian Schafleitner, Jahrgang 2003 MTD

*Siemens Corporate Research* ist ein Forschungsunternehmen der Siemens AG. Ich arbeitete an Projekten für *Siemens Medical Solutions* zur Visualisierung und Verarbeitung medizinischer Bilddaten. Im Besonderen beschäftigte ich mich mit Applikationen zur Darstellung von Herzkranzgefäßen aus Magnetresonanztomografien.

Bei meiner Arbeit setzte ich Wissen aus der 3D Computergrafik und der digitalen Bildverarbeitung um. In den Projekten galt es unter anderem die Benutzerfreundlichkeit und Bedienbarkeit medizinischer Prototypen zu erweitern und zu verbessern. Auch das Testen und die Entwicklung einer Testumgebung war eine meiner sehr abwechslungsreichen Aufgaben bei Siemens Corporate Research.

Ich konnte bei diesen Praktikum nicht nur in meinem Studium erworbenes Wissen praxisnah umsetzen, sondern vor allem sehr viele Erfahrungen bei meiner Arbeit in diesem internationalen Unternehmen sammeln.

# Kapitel 1

# Das Unternehmen

Siemens Corporate Research, Inc. (nachfolgend SCR genannt) wurde 1977 in Princeton, New Jersey, USA gegründet und ist ein Forschungsunternehmen der Siemens AG mit Hauptsitz in München, Deutschland. SCR gehört innerhalb der Firmenhierachie von Siemens zu Siemens Corporate Technology[1], der Forschungs- und Entwicklungs-Zentralabteilung der Siemens AG. Siemens Corporate Technology beschäftigt weltweit über 2.500 Forscher auf 3 Kontinenten. In Abbildung 1.2 wird die Platzierung von SCR innerhalb des Corporate Technology Netzwerkes verdeutlicht. SCR beschäftigt derzeit ca. 200 fixe Mitarbeiter, sowie jährlich an die 200 Praktikanten aus aller Welt. Bei SCR werden innovative Lösungen und Patente, welche in zukünftigen Produkten ihren Einsatz finden werden, entwickelt.

SCR hat viele Verbindungen und Kooperationen mit Universitäten und Forschungseinrichtungen weltweit.

[1] www.ct.siemens.com

**Abbildung 1.1:** Siemens Corporate Research Gebäude in Princeton, NJ.

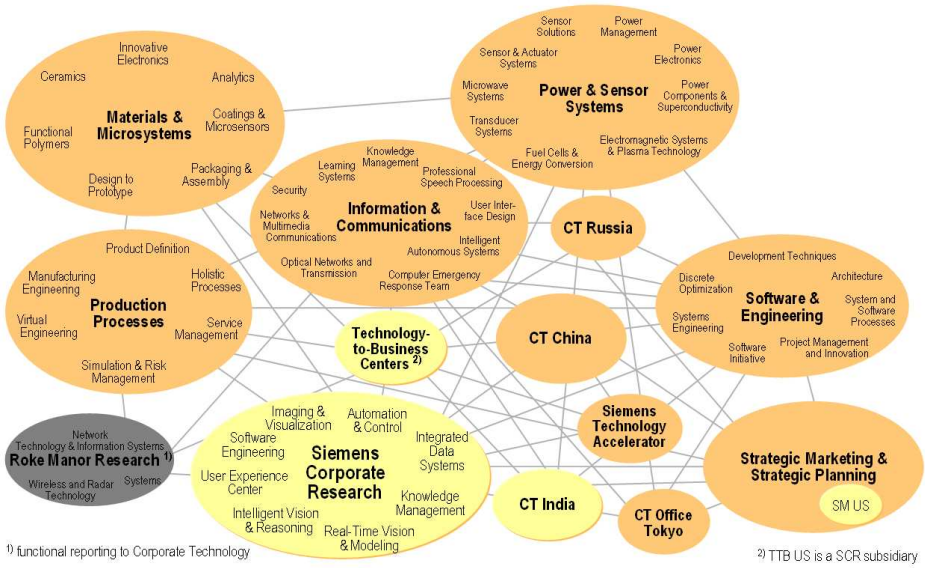



**Abbildung 1.2:** Unternehmenstruktur von Siemens Corporate Technology. Aus [SCR06].

## 1.1 Abteilungen bei Siemens Corporate Research

SCR ist also einer der großen Pfeiler im Corporate Technology Netzwerk der Siemens AG. Innerhalb des Standortes in Princeton ist SCR aber wiederum in folgende Abteilungen gegliedert:

- Automation & Control
- Imaging & Visualization
- Integrated Data Systems
- Intelligent Vision & Reasoning
- Real-Time Vision & Modeling
- Multimedia & Video Technology
- Software Engineering
- User Experience Center

## 1.2 *Imaging & Visualization*

Ich absolvierte mein Praktikum in der *Imaging & Visualization* Abteilung, welche wiederum in kleinere Gruppen und Teams unterteilt ist. Die größeren dieser Untergruppen sind zum Beispiel die *Imaging Architecture Group*,


**Abbildung 1.3:** Siemens MAGNETOM Magnet-Resonanz Scanner. Aus [Sie06].

die an einer Software-Entwicklungsplattform (siehe auch Kapitel 2.1) für medizinische Applikationen arbeiten. Eine andere Gruppe beschäftigt sich zum Beispiel mit *Interventional Imaging*, das computerunterstützte operative Eingriffe ermöglichen soll.

Ich arbeitete im MR-CV[2] Team, welches Applikationen für Magnetresonanz (MR-) Bilder des menschlichen Herzens entwickelt. Themen, wie die Segmentierung einzelner Herzkammern bis zum ganzen Herzen, die Visualisierung einzelner Gefäße bis zu deren automatischen Vermessung und Analyse werden innerhalb dieses Teams behandelt und erforscht.

## 1.3 *Siemens Medical Solutions*

Die bei Siemens Corporate Research entwickelten Algorithmen und Programme werden meist nicht direkt an Kunden geliefert, sondern werden erst in anderen Siemens Abteilungen zu Endprodukten.

Der „Haupkunde“ des *Imaging Departements* ist *Siemens Medical* mit zahlreichen Standorten weltweit. Der Prototyp, welcher in Abschnitt 3.4 beschrieben wird, kommt daher zum Beispiel zu *Siemens Medical Solutions* nach Erlangen, Deutschland. Die dortige Abteilung beschäftigt sich mit Softwarelösungen für MR-Scanner (siehe Abb. 1.3). [Sie06]

Neben General Electric und Philips ist Siemens einer der drei großen *Global Player* am medizinischen Weltmarkt.

[2]Magnetic Resonance Tomography for Cardiovascular Applications

# Kapitel 2

# Die Entwicklungs- und Arbeitsumgebung

## 2.1 RadBuilder

RadBuilder[1] ist eine visuelle Entwicklungsumgebung, um möglichst schnell und effizient Applikationen zur Verarbeitung und Visualisierung medizinischer Bilddaten zu erstellen. Es stehen bereits viele Module zum Laden von Patienten und Bilddaten und zur Verarbeitung sowie zur Visualisierung dieser, zur Verfügung.

Entwickelt wird der RadBuilder von der *Imaging Architecture Group*, einer Untergruppe der *Imaging & Visualization* Abteilung von *Siemens Corporate Research*, in Princeton.

Die RadBuilder Entwicklungsumgebung basiert auf OpenInventor. OpenInventor ist eine objektorientierte Grafik API[2], welche auf OpenGL basiert. Bereits OpenInventor beinhaltet eine Vielzahl an schon vorhandenen Knoten und Modulen, mit denen einfache Netzwerke gebaut werden können.

In Abbildung 2.1 ist die Programmoberfläche dieser Entwicklungsplattform erkennbar. Es wurde ein einfaches Netzwerk erstellt. Der Wurzelseparator wurde geöffnet, somit ist die gerenderte Ansicht des Szenengraphen sichtbar.

Für den Entwickler ist er sehr einfach möglich, eigene Module für den RadBuilder bzw. OpenInventor in C++ zu schreiben.

### 2.1.1 OpenInventor Konzept

Wie schon angesprochen, werden mithilfe der vorhandenen, bzw. auch selbsgeschriebenen Module, Netzwerke aufgebaut, welche in einer baumartigen Struktur organisiert sind.

[1] RAD, Rapid Application Development
[2] Programmierschnittstelle (Application Programming Interface)

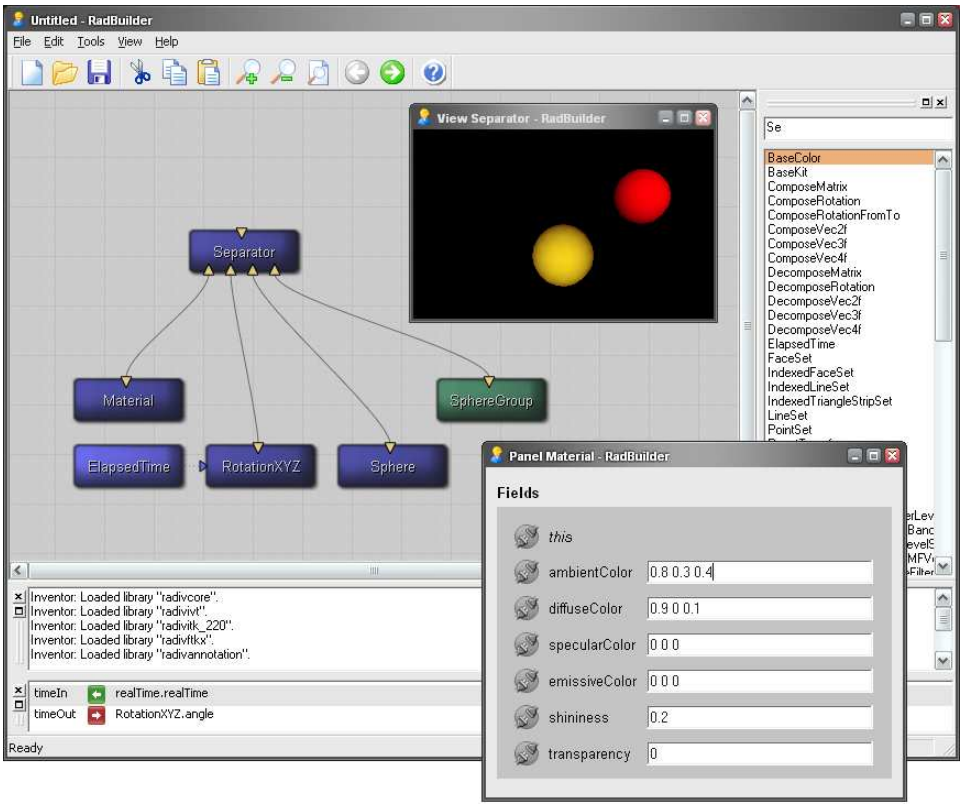



**Abbildung 2.1:** Die RadBuilder Programmoberfläche mit einem einfachem Netzwerk und geöffneter RenderView.

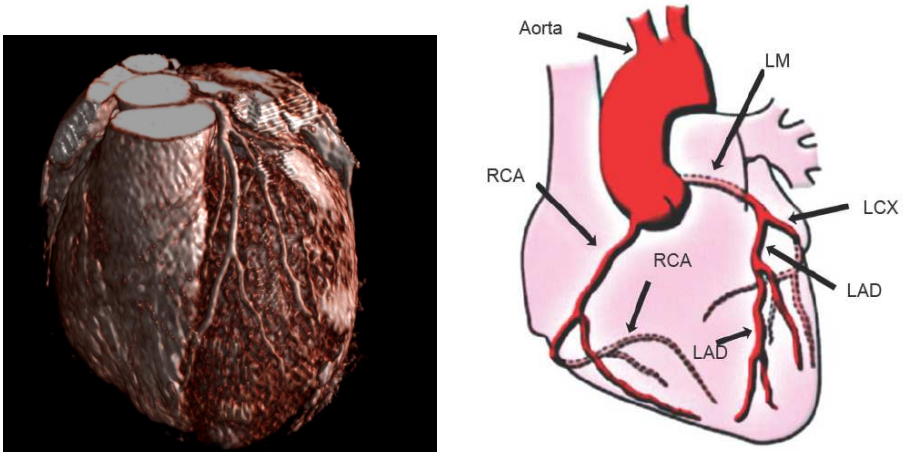
Diese Knoten können für geometrische Objekte, für Eigenschaften, für Transformationen oder andere Dinge stehen. Der Baum wird systematisch abgearbeitet, Transformationen und Materialeigenschaften bleiben bei der Abarbeitung erhalten. Einzelne Transformationen bzw. Eigenschaften kann man in sogenannten Seperatoren[3] kapseln.

Spezielle Arten von Knoten sind *Events* (reagieren auf Events aus dem Windowing-System), *Actions* (arbeiten Szenenbaum ab), *Sensors* (reagieren auf Veränderungen) und *Manipulators* (erlauben einfaches Verändern von Objekten).

Einmal konstruierte Objekte können mehrfach wiederverwendet werden (normalerweise über Transformationen an verschiedenen Stellen im Raum). Bestimmte Objekte können dann über Pfade eindeutig identifiziert werden.

Die beiden Bücher *The Inventor Mentor* [Wer94a], sowie *The Inventor Toolmaker* [Wer94b] von Josie Wernecke halfen mir bei der neuen Arbeit mit OpenInventor.

[3]Ein Seperator arbeitet wie ein `push/pop` Block in OpenGL.




(a) (b)

**Abbildung 2.2:** Anatomie des Herzens: (a) Volumenrendering eines Herzens. (b) Schematische Darstellung. (© Texas Heart Institute) Aus [Tuc04].

## 2.2 Medizinische Bilddaten

### 2.2.1 DICOM Format

Der Industriestandard für medizinische Bilddaten ist das ***D**igital **I**maging and **Co**mmunications in **M**edicine* [NEM06] Format. Fast alle Hersteller medizinisch bildgebender Systeme wie z.B. Digitales Röntgen, Magnetresonanztomographie oder Computertomographie implementieren den DICOM-Standard in diesen Geräten, siehe auch [Wik06]. In den DICOM Dateien werden nicht nur die meist 12-Bit Graustufenaufnahmen gespeichert, sondern auch Informationen über den Patienten, sowie den Aufnahmebedingungen wie zum Beispiel Position, Lage und Zeit der aufgenommenen Bilder.

Daten aus CT[4] bzw. MR[5]-Scannern bestehen meist aus einer Vielzahl an Schichten, die mit Hilfe von Software in den verschiedensten Weisen visualisiert werden können. Abbildung 2.2 (a) zeigt ein Volumenrendering eines Herzdatensatzes. Typische Größen eines solchen Herzdatensatzes sind $512 \times 512 \times 256$ Voxel. Voxel steht für *volumetric pixel*, da die einzelnen Bildpunkte bestimmten Dichtewerten der aufgenommen Regionen entsprechen.

### 2.2.2 Das menschliche Herz

Der Großteil meiner Arbeit beschäftigte sich in Zusammenhang mit dem menschlichen Herzen, insbesondere der Herzkranzgefäße.

[4]Computertomografie
[5]Magnetresonanztomografie

Die Herzkranzgefäße versorgen das Herz mit sauerstoffreichem Blut. Die beiden Hauptäste der Herzkoronarien, die rechte (RCA) und die linke koronare Arterie entspringen aus der Aorta kurz nach den Herzklappen. Die linke Hauptarterie (LM) teilt sich weiters in eine *Linke Anterior Descending* (LAD) und eine *Linke Circumflex* (LCX) Arterie. Diese Gefäße verteilen sich weiter in kleinere Äste, die den gesamten Herzmuskel mit sauerstoffreichen Blut versorgen. Die kleinere rechte Seite ist für den Lungenkreislauf verantwortlich, die linke muskelreiche Seite des Herzens pumpt das Blut durch den gesamten Körper. Abbildung 2.2 (b) zeigt eine schematische Darstellung der Koronararterien.

Eine Verstopfung dieser Herzkranzgefäße, zum Beispiel durch Arteriosklerose[6], führt zu Sauerstoffengpässen in der Herzmuskulatur. Der daraus resultierende Herzinfarkt ist in den westlichen Industrieländern die häufigste Todesursache.

[6] Ablagerungen von Blutfetten, Thromben, Bindegewebe und Kalk.

# Kapitel 3

# Projekte und Tätigkeiten im Praktikum

## 3.1 Einarbeitungszeit

Am Beginn meines Praktikums in den USA musste zuerst einmal sehr viel Papierkram erledigt werden. Ich musste zu einer Orientierungsveranstaltung des VISA Sponsors, sowie dauerte es auch einige Tage, bis ich vollen Zugriff auf das Siemens Netzwerk, die e-Mail Konten, sowie auf das Versionskontrollsystem hatte.

Neben diesen organisatorischen Dinge musste ich mich auch in diesen ersten Tagen in die in meinem Team verwendete Entwicklungsumgebung einarbeiten. Ich bekam ein paar Papers und Bücher zu lesen, die mir den Einstieg in die RadBuilder Entwicklungsumgebung erleichterten.

## 3.2 RadBuilder Module

**Projektzeitraum:** Ende Februar bis Mitte April, Erweiterungen und Bugfixes im Juni/Juli 2006.

Am Beginn meines Praktikums habe ich an diversen Erweiterungsmodulen für den RadBuilder gearbeitet. Dabei ging es meist darum, die Bedienbarkeit des in Abschnitt 3.4 behandelten Prototypen zur Visualisierung der Herzkranzgefäße zu verbessern. Um mich nicht mit der Komplexität des schon sehr unüberschaubaren Netzwerkes dieser Applikation zu überfordern, arbeitete ich immer auf einer eigenen Programmplattform und hatte anfangs nur wenig mit dem Endprodukt zu tun.

Bei dieser Arbeit lernte ich nun erstmals das Arbeiten mit medizinischen Bilddaten kennen, was sich anfangs als gar nicht so einfach erwies, mit etwas Übung fand ich aber bald die wichtigsten Teile des menschlichen Herzens.

(a) (b)

**Abbildung 3.1:** *FindBestAlignment* Modul: (a) schlechte Ausrichtung, (b) gewünschtes Ergebnis.

Die wichtigsten Module, an denen ich gearbeitet habe, werden in diesem Abschnitt kurz erläutert.

### 3.2.1 *FindBestAlignment* Modul

#### Ausgangssituation

Um eine reformatierte Ansicht[1] der Koronarien zu generieren, muß zuerst eine Ausrichtung gefunden werden, in der sich die Gefäße nicht überschneiden, sowie möglichst weit „ausgestreckt“ sind. Abbildung 3.1 (a) zeigt eine schlechte Ausrichtung, (b) zeigt das gewünschte Ergebnis.

#### Umsetzung

Die Umsetzung erfolgte als OpenInventor Engine, welche als Eingabeparameter die Koordinaten der einzelnen Punkte der Koronarien erhält. Ausgabe dieser Engine ist eine Ebene, bzw. der Sichtvektor auf die 3D Darstellung der Koronarienbäume.

Klingt anfangs recht einfach, doch um ein wirklich zufriedenstellendes Ergebnis zu erreichen, bedurfte es einiger Zeit. Zu Beginn versuchte ich eine Ebene durch die Punktwolke zu legen. Problem dabei war allerdings, dass es zu Überschneidungen zwischen den beiden (linken und rechten) Koronarienbäumen, als auch innerhalb eines Baumes (siehe Abb. 3.1(a)) kam.

Somit entschied ich mich im Endeffekt zuerst alle Ausrichtungen, in denen Überschneidungen vorkamen, auszuschließen. Eine Überschneidung

[1] auch *Soapbubble*-Ansicht genannt, darunter versteht man die Visualisierung eines Teilabschnittes oder des gesamten Koronarienbaumes in nur einem Bild. Bei herkömmlicher Ansicht der MR-Bilddaten kann nur der Quer- oder ein Längsschnitt des Gefäßes betrachtet werden, mit dieser Methode ist es aber möglich den gesamten Baum zu sehen.

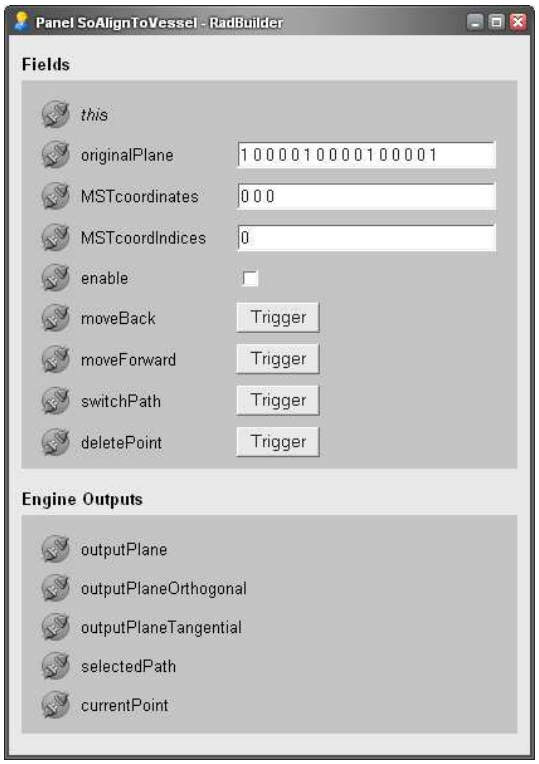



**Abbildung 3.2:** Eigenschaftenfenster/Entwicklungsinterface der *AlignTo-Vessel* Engine.

bringt nämlich grobe Artefakte im Ergebnisbild. Bei den verbleibenden Sichtvektoren wird die maximale Distanz der beiden Bäume, sowie die maximale aufspannende Fläche der einzelnen Äste zur Auswahl des besten Sichtvektors herangezogen.

#### Erfahrung

Bei der Implementierung dieses Modules beschäftigte ich mich sehr viel mit Abbildungsmatrizen, bzw. Projektionen vom 3D auf einen 2D Raum. Wissen aus den Mathematik sowie den Computergrafik Vorlesungen aus der Fachhochschule musste hier angewandt werden.

### 3.2.2 *AlignToVessel* Modul

#### Ausgangssituation

Bei diesem Modul ging es darum eine intelligente Möglichkeit zu finden durch einen Koronarienbaum zu navigieren, sowie den weiteren Verlauf anhand des schon vorhandenen Baumes vorherzusagen. Es sollten verschiedene

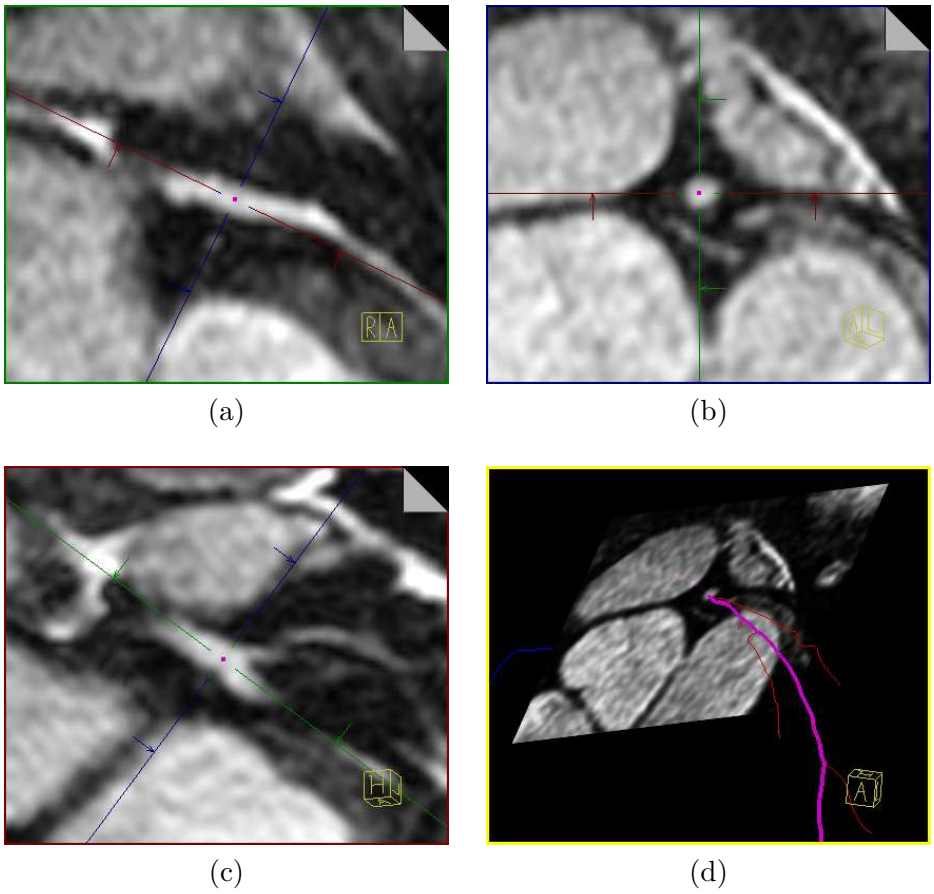



**Abbildung 3.3:** *AlignToVessel* Modul: (a)/(c) zeigt Längsschnitte, (b) zeigt einen orthogonalen Schnitt durch das Gefäß, (d) eine 3D Ansicht mit eingeblendeter Ebene (b).

Ansichten/Schnitte durch des Gefäß dargestellt werden. Für den Benutzer sollte das Navigieren durch einen Baum erleichtert werden.

### Umsetzung

Die Umsetzung erfolgte wieder als OpenInventor Engine, welche die Sichtebenen (als Matrizen dargestellt) für die verschiedenen Schnitte/Ansichten ausgibt. Als Eingabeparameter werden die Koronarienbäume, sowie der errechnete *Minimum-Spanning-Tree*[2] übergeben.

Beim Aktivieren des Modules wird ein Pfad ausgewählt, dieser wird lila dargestellt – siehe Abb. 3.3 (d). Nun kann durch diesen Pfad hindurch-

[2]Der minimale Spannbaum dient dazu, das metrische Problem des Handlungsreisenden zu approximieren. In diesem Falle werden die gegebenen Punkte mit Hilfe dieses Algorithmus zum Gefäßnetzwerk verbunden.

navigiert werden, die Ansichten werden automatisch aktualisiert. Mit dem `switchPath` Button kann ein anderer Pfadabschnitt ausgewählt werden. Ansicht 3.3 (b) zeigt einen orthogonalen Schnitt durch das Gefäß. Der Sichtvektor wird hierbei mit Hilfe der schon vorhandenen Punkte errechnet. Die beiden anderen Ansichten (siehe Abb. 3.3 (a) und (c) zeigen jeweils um 90° gedrehte Längsschnitte.

Abbildung 3.2 zeigt die Entwicklungsbenutzeroberfläche, welche für jede Engine vom RadBuilder automatisch generiert wird. Mit Hilfe dieses können Verbindungen mit anderen Modulen erstellt werden (durch Drag&Drop der einzelnen Felder auf das Feld des Zielfeldes).

#### *VesselEditor* Erweiterung

Am Ende meines Praktikums erweiterte ich dieses Modul noch um die Möglichkeit neue Punkte in den schon vohandenen Baum einzufügen, falsche Punkte zu löschen, oder schon vorhandene Punkte zu verschieben.

### 3.2.3 *DetectCenterPoint* Modul

#### Ausgangssituation

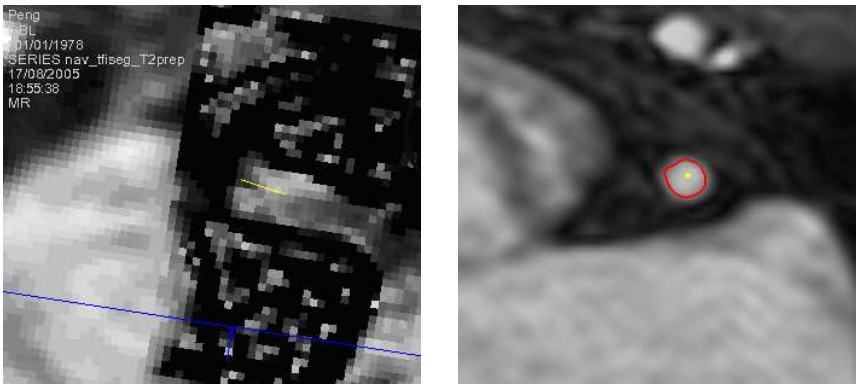
Bei den schon erwähnten Koronarienbäumen handelt sich um eine Menge an Punkten, welcher der Benutzer selbst mit Hilfe von Mausklicks bestimmt. Um ein ideales Ergebnisbild zu generieren, müssen die Punkte immer in der Mitte des Gefäßes liegen. Dies ist aber leider nicht immer ganz leicht, da die Sichtebene nicht immer orthogonal auf das Blutgefäß steht. Ein Punkt am Rande eines Gefäßes lässt dieses in der reformatierten Ansicht dann kleiner bzw. dünner aussehen, die Ansicht ist also verfälscht.

Um dieses negative Verhalten ausschließen zu können, sollte der Benutzer nur mehr in die Nähe des Gefäßes klicken müssen, und die Engine findet Gefäß und Mittelpunkt automatisch.

#### *Vesselness Measurement*

Für deratige Anwendungen werden Filter eingesetzt, die den zu untersuchenden Bereich nach tubularen Strukturen absuchen. Für jeden Pixel, bzw. in diesem Fall Voxel, wird die sogenannte *Vesselness Measure* errechnet. Umso höher deren Wert ist, umso sicherer handelt es sich um ein Blutgefäß, bzw. ein schlauchartiges Gebilde. Der verwendete Algorithmus stammte aus [FNVV98].

Nachteile der angewendeten Methode werden bei Abzweigungen erkennbar, da es hier keine eindeutige Tubusform mehr vorkommt. Auch wenn das Gefäß sehr nahe an anderen Gefäßen oder Bereichen mit ähnlicher Dichte (zum Beispiel die Herzkammer) liegt, wird das Ergebnis verfälscht.




(a) (b)

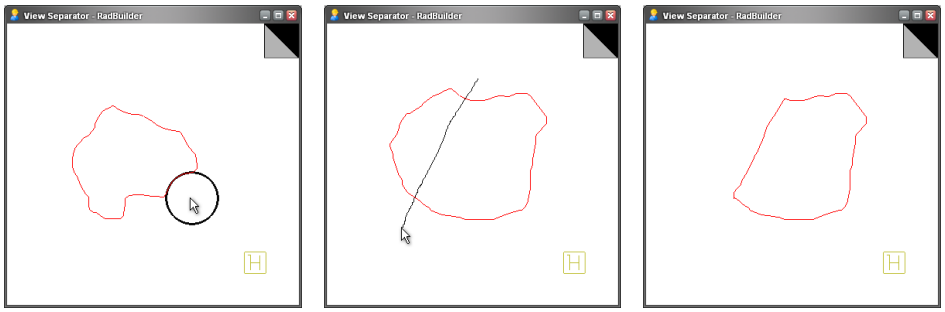
**Abbildung 3.4:** *DetectCenterPoint* Module: (a) temporäres Bild nach regionaler Anwendung des *Vesselness Measurements* Filter, (b) Ergebnis mit gefundener Kontur des Gefäßes.

Abbildung 3.4 (a) zeigt die visualisierten *Vesselness Measure* Werte. Wie hier zu erkennen ist, werden auch lokale Maxima außerhalb des Gefäßes, durch Bildrauschen verursacht, gefunden. Durch weitere Filter werden diese störenden Strukturen noch eliminiert, dabei muss beachtet werden, dass feine Gefäße nicht gelöscht werden. In diesem Falle wird nun angenommen, dass der höchste *Vesselness Measure* Wert den Mittelpunkt darstellt.

Wie sich allerdings später herausstellte, war diese Methode zu ungenau, und so wurde das Modul mit einem weiteren Modul verknüpft, das einer meiner Kollegen implementiert hatte. Dieses findet die Kontur des Gefäßes (siehe Abb. 3.4 (b)). Zuerst wird nun also das Gefäß mit Hilfe der *Vesselness Measure* gefunden, dann die Kontur gebildet, und zuletzt der Mittelpunkt bestimmt.

### Erfahrung

Leider war aber auch diese Methode noch nicht 100%ig sicher, und so wurde vorerst einmal von einer Verwendung dieses Modules im Endprodukt abgeraten. Bei der Umsetzung dieser Engine bekam ich allerdings einen guten Einblick in die digitale Bildverarbeitung und deren mathematischen Hintergründen. Ich musste mich anfangs selbst mit Hilfe von wissenschaftlichen Abhandlungen über die hier angewandten *Shape Filters* informieren, bekam aber dann bei der Umsetzung Hilfe von Kollegen, bzw. arbeitete mit anderen an der Verbesserung des Ergebnisses.




(a) (b) (c)

**Abbildung 3.5:** *ContourEditor*: (a) *Nudge Tool*, (b) *Cut Tool*, (c) Ergebnis des *Cut Tools*.

### 3.2.4 *ContourEditor*

#### Ausgangssituation

Der *ContourEditor* ist ein kleines Tool, welches es ermöglicht eine Kontur (zum Beispiel die einer Segmentierung eines Organes) zu laden und zu editieren. Dabei sollte eine solche Kontur so natürlich wie möglich zu editieren sein. Daher konnte kein „Standardtool“ wie in Photoshop oder Freehand implementiert werden, mit dem man zum Beispiel die Kontrollpunkte, die eine Linie/Kurve beschreiben, verändert.

In anderen medizinischen Applikationen ist ein sogenanntes *Nudge*[3] *Tool*, sowie ein *Cut Tool* im Einsatz. Meine Aufgabe war es dieses als RadBuilder Modul neu zu implementieren.

#### Umsetzung und Erfahrung

In Abbildung 3.5 (a) ist das *Nudge Tool* erkennbar, der Radius des Kreises, der zum Deformieren verwendet wird, bestimmt sich beim Drücken der Maustaste, in dem der minimalste Abstand zur Kontur bestimmt wird. Mit dem *Cut Tool* kann man eine Linie durch die schon vorhandene Kontur ziehen. Die so entstandene längere Kontur bleibt erhalten (siehe Abb. 3.5 (b)).

Bisher hatte ich nur OpenInventor Engines implementiert. Bei dieser Aufgabenstellung reichte eine Engine allerdings nicht mehr aus, da diese nur aus Eingangs- und Ausgangsparameter bestand, und normalerweise als Blackbox fungiert. Sie hat im Regelfalle keinen direkten Einfluss auf den zu rendernden Szenengraphen. Hierbei mussten aber sowohl Mausevents abgefangen werden, als auch die Tools, mit dem die Kontur editiert wird, gezeichnet werden. Ich implementierte dieses Modul also als *ShapeNode*. Weiteres

[3] to nudge (engl.) - stupsen, stoßen.

OpenInventor Wissen wurde hierbei gelernt.

## 3.3 Python-OpenInventor Testframework

**Projektzeitraum:** Mitte April bis Ende Mai 2006.

### 3.3.1 Ausgangssituation

Mit Hilfe des RadBuilders werden auch Algorithmen entwickelt und getestet, die auf einem Datensatz ausgeführt werden. Ein Beispiel hierzu wäre zum Beispiel ein Algorithmus zur Segmentierung des menschlichen Herzens aus einem Volumen, das noch weitere Strukturen enthält.

Meist werden während der Entwicklung nur ein oder wenige Datensätze zum Testen verwendet. Bevor ein Algorithmus aber als wirklich „gut funktionierend“ bezeichnet werden kann, muß er auf eine größere Zahl von Datensätzen angewandt werden. Vor allem auch wenn Änderungen an einem schon vorhandenen Algorithmus/Programm vorgenommen werden, ist es interessant, inwiefern sich die Ergebnisse verbessert oder verschlechtert haben.

Da das Laden eines Datensatzes sowie das Setzen von Einstellungen, die vom Datensatz abhängig sind, sehr viel Zeit in Anspruch nimmt, war es meine Aufgabe dieses Testverfahren zu automatisieren, sowie sollte es möglich sein Berichte und Ergebnisbilder zu erstellen.

Als Ausgangsmaterial für einen Test dient ein RadBuilder Netzwerk (vgl. Abschnitt 2.1), welches in eine OpenInventor Datei (.iv) abgespeichert wird. Dieses kann dann von der Python Applikation geladen werden.

### 3.3.2 Umsetzung

Die Umsetzung des Testframeworks besteht aus 3 Teilen: der Python Applikation, von der aus Netzwerke und Datensätze geladen und Berichte erstellt werden können, eine Python-OpenInventor Wrapper Klasse, welche es ermöglicht aus Python OpenInventor Netzwerke zu laden, zu verändern und zu rendern, sowie einem *ReportViewer* mit dem die unterschiedlichen Berichte angezeigt, verglichen und gedruckt werden können.

#### Die Python Applikation

Das Testframework wurde in Python implementiert. Hier wirft sich die Frage auf, *warum verwendet man eine Skriptsprache für so ein Projekt?*

Python wurde aus dem Grund gewählt, da es sehr viele zusätzliche Module (die zum Großteil open-source sind) für Python gibt, wie zum Beispiel einen XML Parser, einen OpenGL Wrapper, und so weiter. Auch selbst einen Wrapper zu schreiben, ist mit Python relativ einfach. So war es für dieses

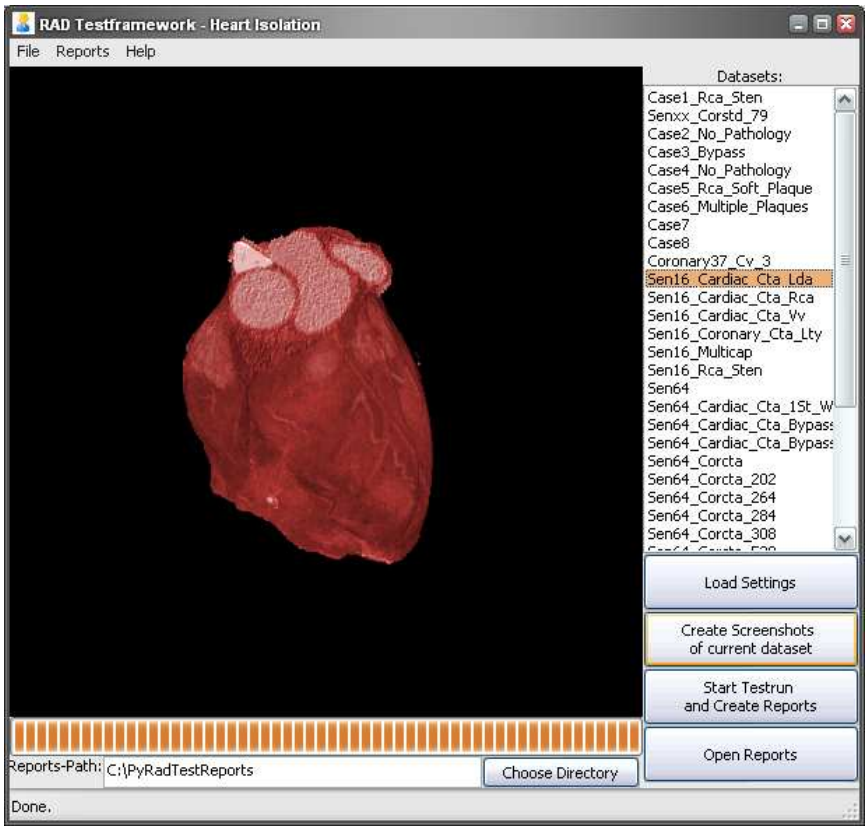



**Abbildung 3.6:** Programmoberfläche der Testapplikation.

Projekt nötig, einen OpenInventor Wrapper zu schreiben, der das Laden eines Netzwerkes, sowie das Ändern und Auslesen von Werten, als auch das Rendern des Szenengraphen ermöglichen musste.

Doch die Hauptanforderung war es, dass es auch für den Endbenutzer, in diesem Falle meist ein anderer Programmierer, möglich sein sollte, das Testframework, bzw. Teile davon (zum Beispiel die zu erstellenden Berichte), für eigene Ansprüche modifizieren zu können.

Um dennoch das Testframework ohne Eingriffe in den Programmcode möglichst flexibel zu gestalten, werden die grundlegenden Test- und Datensatz spezifischen Daten in XML Dateien gespeichert.

In Abbildung 3.6 wird die Programmoberfläche des Testprogrammes dargestellt. Eine Änderung im Netzwerk ist zu diesem Zeitpunkt nicht mehr möglich. Auch die Ausgabefelder, sowie zu rendernde Ansichten müssen bereits in der XML Datei definiert sein. Man kann nun einzelne Datensätze auswählen und das Ergebnis anzeigen oder einen kompletten Bericht erstellen lassen.

Das User-Interface wurde mit Hilfe von *wxPython*, der Python Version

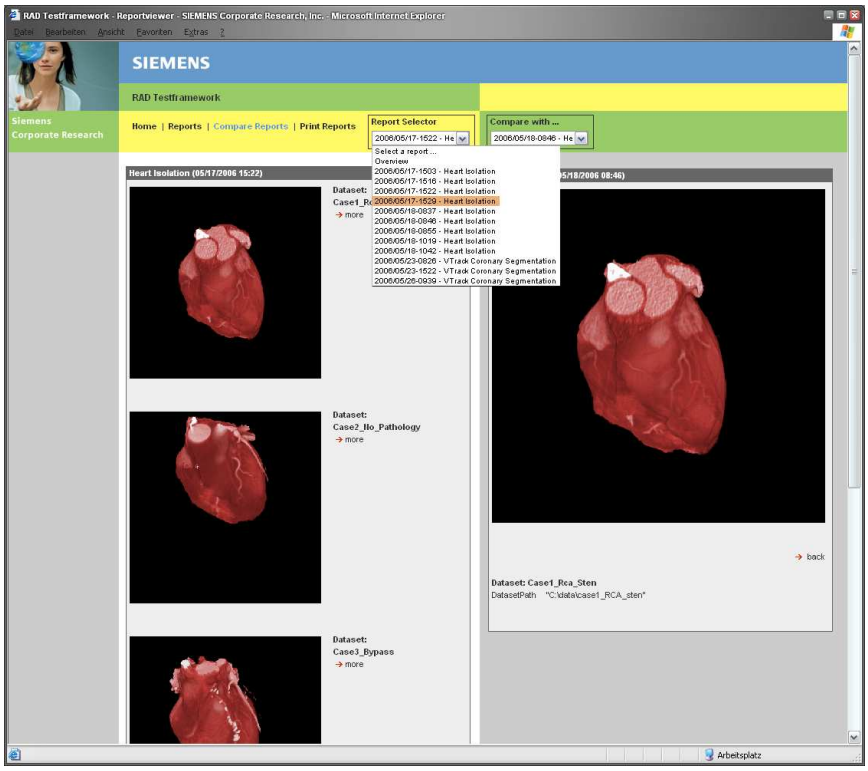



**Abbildung 3.7:** Der in HTML umgesetzte *ReportViewer*.

der plattformunabhängigen *wxWidgets* erstellt.

### Der Python-OpenInventor Wrapper

Da es zur Zeit der Entwicklung des Testframeworks nur kommerzielle OpenInventor Wrapper für Python gab, und für dieses Testframework nur ein einfacher Wrapper benötigt wurde, erstellte ich einen solchen. Mit dieser Wrapperklasse, die in jeder Pythonapplikation verwendet werden kann, ist es möglich OpenInventor-, also auch RadBuilder-Netzwerke, sowie zusätzlich benötigte Module/PlugIn zu laden, als auch Einstellungen und Verbindungen im Netzwerk zu ändern, sowie einzelne Seperatoren zu rendern.

### Der *ReportViewer*

Anforderung an den *ReportViewer* war, dass dieser möglichst plattformunabhängig und präsentierfähig sein sollte. Weiters sollte es möglich sein Reports ins Web zu stellen und zu drucken. Daher entschied ich mich dafür, die Reports im HTML Format zu erstellen. Um es zu ermöglichen, dass man unterschiedliche Berichte vergleichen kann, musste ich einen Weg finden,

dynamisch Teilseiten in die bestehende HTML Seite nachzuladen und dies ohne der zu Hilfenahme eines Servers. Das `XMLHTTPRequest`[4] Objekt war die Lösung. Somit werden in der Python Applikation die Bilder, einzelne Teilseiten, sowie eine XML Datei mit den vorhandenen Berichten erstellt. Das Betrachten ist über eine „Masterpage“ möglich, die zuerst die XML Datei einliest und somit die Ansichten generiert.

Abbildung 3.7 zeigt den *ReportViewer*. Mit den beiden Dropdownfelder können Testläufe (linke Spalte) geladen werden, ein Klick auf einen Datensatz zeigt Detailinformationen dazu (rechter Bereich).

### 3.3.3 Erfahrungen

Das interessante an diesem Projekt war, dass ich die Planung, Durchführung und das erste Testen selbst übernehmen durfte. Mein Betreuer erzählte mir von der Idee eines solchen Testframeworks, von den Funktionen, die es haben sollten, die Umsetzung war mir aber selbst überlassen.

Das Framework wurde zuletzt von einer weiteren Siemens Mitarbeiterin getestet, die ihre eigenen Algorithmen damit zu testen versuchte. Dabei zeigten sich noch einige Fehler auf, da jeder Benutzer anders an ein Programm herangeht, und somit neue Fehler gefunden wurden. Weiters brachte sie auch neue Ideen zur Verbesserungen und Erweiterungen des Frameworks ein. In einer weiteren Version setzte ich dann ihre Vorschläge um und besserte noch vorhandene Fehler aus.

## 3.4 *CoronaViz* - Coronary Visualization Prototyp

**Projektzeitraum:** Mitte Juni bis Anfang August 2006.

In den letzten beiden Monaten meines Praktikums arbeitet ich direkt an dem schon vorhandenen Prototypen zur Visualisierung von Herzkranzgefäßen. Die Besonderheit an diesem Prototyp war, dass mit Hilfe einer erstellten *reformatierten* Ansicht (siehe Abb. 3.8) das gesamte Gefäß darstellbar ist. Die dazu verwendete Methode kann in [EBvM+02] bzw. [Tuc04] nachgelesen werden. Man kann DICOM Datensätze laden, Koronarienbäume definieren, speichern und ebenfalls wieder laden und verschiedene Ansichten generieren und speichern.

### 3.4.1 Ausgangssituation und Aufgabe

Meine Aufgabe bestand darin, zunächst die Benutzeroberfläche zu überarbeiten und das Programm einfacher und intuitiver bedienbar zu machen. Es musste auf vorhandene Workflows Rücksicht genommen werden, und darauf

[4] auch unter dem Namen *Ajax* bekannt.


**Abbildung 3.8:** Reformatierte Ansicht des linken und rechten Koronarienbaumes.

geachtet werden, dass nur wenige Maus bzw. Tastenklicks zum Wechseln der einzelnen Modi bzw. Bearbeitungsschritten nötig sind. Weiters haben sich in die schon vorhandenen Module Fehler und dergleichen eingeschlichen, die es nach ausführlichen Testläufen zu eliminieren galt. Das Programm war zunächst auch äußerst instabil und hatte grobe Performanceschwierigkeiten, die durch eine Analyse und Verbesserung des zu rendernden Szenengraphen verbessert werden konnte.

Zu guter Letzt musste die gesamte Applikation aus der RadBuilder Entwicklungsumgebung in den RadViewer, einer MFC[5] Applikation, portiert werden.

### 3.4.2 Ergebnis und Erfahrungen

In Abbildung 3.9 wird die fertige Applikation samt User Interface dargestellt. Das User Interface entspricht dem *Syngo* Style, der bei allen medizinischen Applikationen von Siemens angewandt wird.

Bei diesem Projekt hatte ich mit einem äußerst komplexen OpenInventor Netzwerk zu tun. Ich musste die Erfahrungen, die ich während dem Implementieren der vorangegangen erläuterten Module gesammelt habe, hier wieder einsetzen um die Problemstellungen lösen zu können.

Bei diesem Projekt stand ich auch in enger Zusammenarbeit mit meinem Supervisor, der immer wieder neue Ideen und Anregungen einbrachte,

[5]Microsoft Foundation Classes

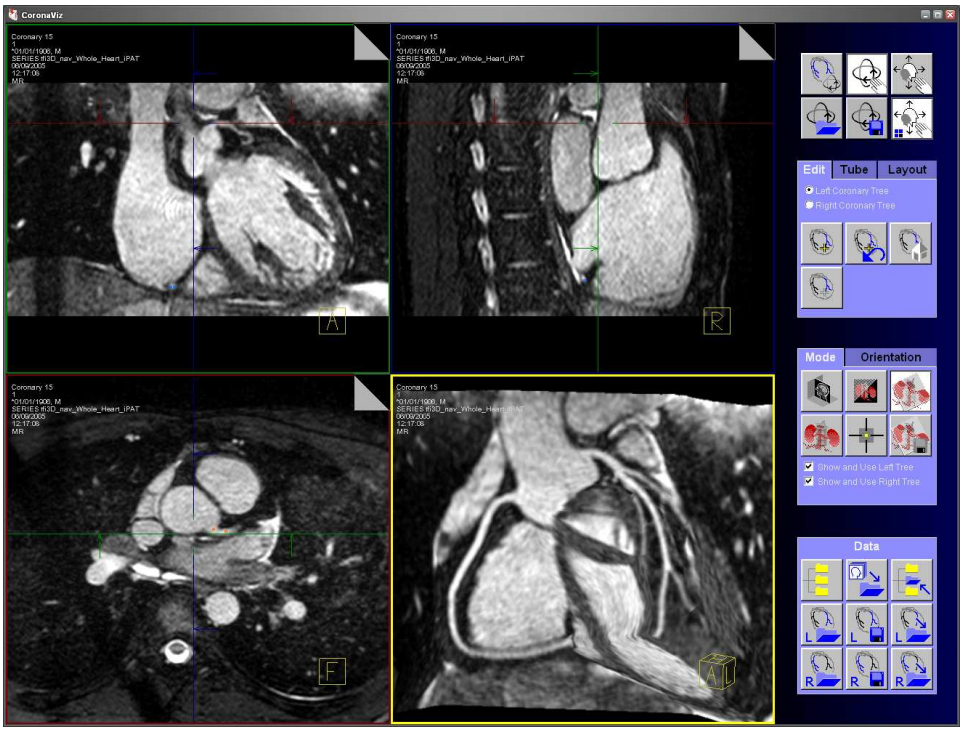



**Abbildung 3.9:** *CoronaViz* Applikation.

die ich umzusetzen versuchte. Alle ein bis zwei Wochen wurde eine *Release* Version samt Installationsprogramm erstellt, welches nach Deutschland zur MR Abteilung bei *Siemens Medical* gesandt wurde, um dort getestet zu werden. Die anfangs deprimierte Testerin konnte gegen Ende meines Praktikums mit einer sehr stabilen, sowie benutzerfreundlichen, Version erfreut werden, und noch Anfang August wurde das Produkt zu den ersten Ärzten bzw. Radiologenpraxen gebracht.

# Kapitel 4

# Erfahrungen und Zusammenfassung

Der Lernfaktor war bei diesem Berufspraktikum auf alle Fälle sehr hoch, ich musste mich mit interessanten neuen Themenbereichen, wie der medizinischen Bildverarbeitung auseinandersetzen, konnte aber auch schon erlerntes und gefestigtes Wissen in der Computergrafik, sowie in Usability Fragen erfolgreich in den übertragenen Aufgabengebieten einsetzen.

Da Siemens Corporate Research hauptsächlich eine Forschungsinstitution ist, gab es für Praktikanten kaum Zeit- oder Erfolgsdruck, was sich äußerst positiv im Betriebsklima bemerkbar machte. Die Arbeitszeiteinteilung war mir selbst überlassen, allerdings sollten 40 Stunden pro Woche gearbeitet werden. SCR betreut jährlich an die 200 Praktikanten, die nicht nur für Siemens, sondern vor allem auch für sich selbst arbeiten können. Meine Betreuer haben versucht, dass ich durch meine Arbeit immer wieder mit neuen Dingen konfrontiert wurde und somit gefordert wurde, etwas Neues lernen zu können. Auch meine Ideen und Vorschläge waren willkommen und es wurde versucht diese umzusetzen.

Ich war zwar in einem Team „eingeordnet“, die meisten Arbeiten machte ich aber alleine. Dennoch war es sehr gut zu wissen, wo man wen um was fragen konnte, und so immer Hilfe bekam, wenn dies nötig war.

Eine wichtige Herausforderung und Erfahrung war für mich auch die Arbeit in einem internationalen englischsprachigen Unternehmen. Obwohl mein Betreuer zwar aus Deutschland kam, bekam ich trotzdem genug Gelegenheiten Englisch zu sprechen, und denke auch, dass ich dieses verbessern konnte. Vor allem die Teammeetings waren sehr „international“ und es saßen Kollegen aus bis zu 10 Nationen am Tisch. Ein Großteil der Praktikanten bei SCR kommt allerdings aus Europa, besonders aus Deutschland und Frankreich, aber auch viele Chinesen und Inder findet man in Princeton. In den Sommermonaten sind auch sehr viele amerikanische Praktikanten bei Siemens beschäftigt. Generell kann man sagen, dass auch bei den Hauptberuflichen

ein reger Austausch zwischen den verschiedensten Siemens Standorten in Amerika, Europa und Asien stattfindet, was den Forschungscharakter des Unternehmens stark fördert.

Obwohl das Praktikum sehr technisch und programmierlastig war, war die Ausbildung die ich während meines Medientechnik und -design Studium erworben hatte eine ideale Vorbereitung. Ich konnte sehr viel erworbenes Wissen aus den Bereichen der Computergrafik, der digitalen Bildverarbeitung und der Mathematik praktisch umsetzen, und musste mich auch mit gestalterischen Themen wie User Interface und Grafikdesign beschäftigen.

Abschließend kann ich nur sagen, dass mir dieses Praktikum große Freude bereitet hat. Ich habe viele neue Erfahrungen in einem fremden Land, sowie in einer neuen Arbeitsumgebung gewonnen. Ich denke, dass diese Erfahrungen für mein weiteres Berufsleben äußerst positiv sind und ich daher diese nicht missen möchte.

# Literaturverzeichnis


[EBvM+02] Etienne, Alex, René M. Botnar, Arianne M.C. van Muiswinkel, Peter Boesiger, Warren J. Manning und Matthias Stuber: *„Soap-Bubble“ Visualization and Quantitative Analysis of 3D Coronary Magnetic Resonance Angiograms*. In: *Magnetic Resonance in Medicine 48*, Seiten 658–666. Wiley-Liss, Inc., 2002.

[FNVV98] Frangi, A.F., W.J. Niessen, K.L. Vincken und M.A. Viergever: *Multiscale vessel enhancement filtering*. In: *Medical Image Computing and Computer-Assisted Intervention - MICCAI'98*, Seiten 130–137, Berlin, Germany, 1998. A. Colchester and S.L. Delp (Eds.), Lecture Notes in Computer Science, vol. 1496 - Springer Verlag.

[NEM06] NEMA: *DICOM Spezifikationen und Informationen*. URL, http://medical.nema.org/, August 2006.

[SCR06] SCR: *Firmenpräsentation Siemens Corporate Research*, Princeton, NJ, August 2006.

[Sie06] Siemens, AG: *Siemens Medical Solutions Webpage*. URL, http://www.siemensmedical.de, August 2006.

[Tuc04] Tuchschmid, Stefan: *CoroViz: Visualization of 3D Whole-Heart Coronary Artery MRA Data*. Diplomarbeit, University of California, Dept. of Radiology and Swiss Federal Institute of Technology, Biomedical Engineering, San Francisco/Zürich, September 2004.

[Wer94a] Wernecke, Josie: *The Inventor Mentor*. Addison-Wesley Professional, 1994.

[Wer94b] Wernecke, Josie: *The Inventor Toolmaker: Extending Open Inventor*. Addison-Wesley Professional, 1994.

[Wik06] Wikipedia: *Lexikoneintrag DICOM*. URL, http://de.wikipedia.org/wiki/DICOM, August 2006.