Entwicklung einer Plattform zur 3D-Visualisierung und -Segmentierung medizinischer Daten

YaDiV - **Y**et **a**nother **Di**com **V**iewer

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Dissertation

von Dipl.-Math. Karl-Ingo Friese
geboren am 2. Oktober 1971 in Hildesheim

Hannover, Juli 2010

Referent:
Prof. Franz-Erich Wolter
Institut für Mensch-Maschine-Kommunikation
Gottfried Wilhelm Leibniz Universität Hannover

Koreferentin:
Prof. Dr. Nadia Magnenat-Thalmann
MIRALab
Université de Genève

Tag der Promotion:
2. Februar 2010

Inhaltsverzeichnis

Kurzdarstellung

In dieser Arbeit wird der Aufbau und die Entwicklung einer Visualisierungs- und Segmentierungsoftware für medizinische 3D-Daten beschrieben. Ziel war es, eine breite Basis für die schnelle Erprobung neuer Ideen zu erhalten und dabei möglichst unabhängig von Hardware und Betriebssystem zu bleiben, um insbesondere interdisziplinäre Forschungsprojekte mit vielen verschiedenen Partnern zu erleichtern.

Um die Entwicklung dokumentieren zu können, wird zusätzlich ein Überblick über die verschiedenen Gebiete der medizinischen 3D-Datenverarbeitung gegeben, insbesondere den bildgebende Verfahren, der Visualisierung, der Segmentierung und der Verwendung von haptischen Eingabegeräten.

Ein besonderer Schwerpunkt stellt die schnelle, interaktive Volumenvisualisierung und eine Untersuchung zum Einzug von Virtueller Realität (VR) sowie insbesondere Haptik (Kraftrückkopplung) in medizinische Anwendungen dar.

Schlagworte: Medizin, Visualisierung, Segmentierung, Virtuelle Realität

Abstract

This thesis presents the conceptional design and implementation of a new software to visualize and segment medical 3D data. The main goal was to create a platform that would allow to try out new approaches and ideas while staying independent from hardware and operation system, especially allowing interdisciplinary research groups to work together on one project without having to switch the software.

To understand how the program was designed, this work will also give an overview over the different topics of medical volume data processing, e.g. medical imaging, visualization, segmentation and the the use of haptic input devices.

A special focus will be fast and interactive volume visualization and a survey on the use of Virtual Reality (VR) and especially haptic/force feedback in medical applications.

Keywords: Medicin, Visualization, Segmentation, Virtual Reality

1. Vorwort

„As a tool for applying computers to science, visualization offers a way to see the unseen. As a technology, Visualization in Scientific Computing promises radical improvements in the human/computer interface and may make human-in-the-loop problems approachable.“ [47]

Visualisierung wird in der Wissenschaft disziplinübergreifend als Werkzeug für die Datenanalyse verwendet. Eine Tabelle führt präzise jedes Detail auf, die Tendenz wird jedoch in der Darstellung durch einen Graphen leichter erkennbar. Bei mehrdimensionalen Daten führt oft erst ein Multiplotverfahren zum erhofften Erkenntnisgewinn. Die zu visualisierenden Daten sind dabei meist abstrakt. So wird bspw. die Druck- oder Temperaturverteilung in einem Werkstück in einer Falschfarbendarstellung visualisiert oder die regionale Preisentwicklung bei Lebensmitteln in einer Kombination aus Landkarte und Balkendiagramm skizziert.

Insbesondere durch nichtinvasive 3D-Bildgewinnungsmethoden (CT, MRT, Ultraschall, etc.) fallen in der modernen Medizin große Datenmengen an. Solche Aufnahmen werden zur Diagnose einer Krankheit oder einer Verletzung eingesetzt, zur konkreten Planung einer komplizierten Operation und zunehmend auch für die anschließende Qualitätskontrolle. Für einige Aufgaben ist es wichtig, eine möglichst schnelle, für andere eine möglichst hochwertige Darstellung der gewonnenen Daten zu erzeugen. Besonders für die 3D-Segmentierung ist eine angemessene, intuitiv verständliche, exakte und vor allem interaktive Visualisierung der vorhandenen Volumendaten bedeutsam.

„Volume visualization is a method of extracting information from volumetric datasets through interactive graphics and imaging, and is concerned with the representation, manipulation, and rendering of these datasets.“ [29]

Aus Sicht der Computergraphik ist die Visualisierung medizinischer Daten ein reizvolles und spannendes Gebiet. Zum einen müssen aufgrund der Datenmengen, die auch heutige Computer an ihre Grenzen bringen, besonders raffinierte Algorithmen entwickelt und implementiert werden. Zum anderen handelt es sich bei diesen Daten bereits um ganz konkrete, physische 3D-Daten, die keine abstrakte Größen beinhalten, sondern die Aufnahme eines Gehirns, eines Thorax oder einer menschlichen Hand. Wo es uns bei der Visualisierung abstrakterer Daten oft schwer fällt, einen Darstellungsansatz zu finden,

haben wir bei der Darstellung eines Körperteils, auch wenn sie schematisch ist, eine sehr konkrete Vorstellung – und oft auch höhere Ansprüche.

„Intuition ist die Fähigkeit, Einsichten in Sachverhalte, Sichtweisen, Gesetzmäßigkeiten oder die subjektive Stimmigkeit von Entscheidungen ohne diskursiven Gebrauch des Verstandes, also etwa ohne Schlussfolgerungen, zu erlangen. [...] Intuition steht letztlich hinter aller Kreativität. Der danach einsetzende Intellekt führt nur noch aus oder prüft bewusst die Ergebnisse, die aus dem Unbewussten kommen.“ [Wikipedia]

Im Vergleich zu technischen Anwendungen, wie der Aufnahme eines Werkstücks, sind biomedizinische Daten oft wesentlich komplexer und nicht durch einfache geometrische Grundformen beschreibbar. Daher ist es oft schwer, eine angemessene Form der Visualisierung zu finden, die sowohl intuitiv verständlich als auch hinreichend exakt ist. Als ein vergleichsweise neues Instrument zum Verständnis komplexer dreidimensionaler Strukturen hat sich inzwischen die stereographische Darstellung entwickelt, die auch mit computerunterstützten Techniken vertrauten Medizinern einen ganz neuen Zugang zu den Bilddaten ermöglicht.

Eine weitere Verbesserung der Darstellung erfordert oft eine vorangehende Segmentierung. Ein Segment ist eine Struktur, durch die einzelne Elemente der Volumendaten mit einer Metainformation versehen werden können, bspw. einer Gewebeeigenschaft („Knochen“) oder der Zuordnung zu einer bekannten organischen Struktur („Nucleus Caudatus“). Um nicht jedem Volumendatenelement einzeln diese Metainformation geben zu müssen, werden automatisierte Segmentierungsverfahren verwendet. Die stereographische Visualisierung der Volumendaten unterstützt das Erkennen einer solchen Struktur, die durch die Information über die Segmentzugehörigkeit zusätzlich hervorgehoben werden kann.

Ziele dieser Arbeit

Ziel dieser Arbeit war es, eine Visualisierungs- und Segmentierungsplattform für medizinische 3D-Daten zu entwickeln. Dabei richtet sich das Programm in erster Linie an Wissenschaftler aus den Gebieten der medizinischen Visualisierung und Segmentierung, die neue Methoden – wie z.B. ein anspruchsvolles Segmentierungsverfahren oder die Erprobung neuer, haptischer Eingabemöglichkeiten – entwickeln wollen. Das Resultat dieser Arbeit ist das Programmpaket YaDiV, das als Open-Source ab sofort der Wissenschaftsgemeinde zur Verfügung steht.

Vor der Entwicklung von YaDiV wurden verschiedene quelloffene Softwarepakete aus diesem Bereich studiert (Kapitel 7.1, S. 121). Gab es im 2D-Visualisierungsbereich noch einige vielversprechende Kandidaten, sah es für 3D-Anwendungen schon sehr viel schlechter aus. Nur eines der getesteten Programme bot Möglichkeiten zur Segmentierung, war

aber sehr langsam und durch ein besonderes Lizenzmodell nur eingeschränkt nutzbar. Ein weiteres, sehr ausgereift wirkendes Programm wird ausschließlich für eine einzige Plattform (Mac OS X) entwickelt.

Neben der leichten Erweiterbarkeit um neue Module, war Plattformunabhängigkeit eine der Hauptanforderungen, um möglichst vielen Wissenschaftlern die Möglichkeit zu geben, YaDiV als Grundlage für ihre Arbeit zu nutzen. Ein weiteres Kriterium war die konsequente Multi-Thread-Unterstützung, um die zum Teil sehr aufwendigen Methoden (z.B. im Bereich Direct Volume Rendering oder Moving-Contour Segmentierung) möglichst optimal an die sich stetig weiter verbreitenden Mehrprozessor- (oder zumindest Mehrkern-) Systeme anpassen zu können. Die zu Beginn der Entwicklung getroffene Entscheidung, Java als Programmiersprache zu verwenden, wurde besonders durch diese Anforderungen bestärkt. Auch wenn es um die Verfügbarkeit freier Software im Bereich medizinische Visualisierung heute deutlich besser aussieht als zu Beginn der Entwicklung, ist der Bedarf an einer erweiterbaren und plattformunabhängigen Software immer noch gegeben. Dies wird u.a. an den vielen elektronischen Anfragen zu YaDiV noch vor der Open Source Veröffentlichung ersichtlich.

Überblick und Aufbau

Die Entwicklung einer sehr breit angelegten Plattform, wie sie in dieser Arbeit beschrieben wird, ähnelt oft dem klassischen Henne-Ei-Problem. Um zum Beispiel die Güte einer Segmentierungsmethode evaluieren zu können, ist eine angemessene Segmentvisualisierung erforderlich, für deren Entwicklung wiederum ein Segment vorliegen muss. Ein ähnliches Problem entstand auch bei der Konzeption der Ausarbeitung. An einigen Stellen werden Begriffe – zumindest als Motivation — benötigt, die erst in einem späteren Kapitel ausführlich erklärt werden können. Um trotzdem eine gute und strukturierte Darstellung zu erhalten ist diese Dissertation wie folgt aufgebaut:

Das nächste Kapitel gibt einen Einblick über die **bildgebenden Verfahren**, die die Datengrundlage für die medizinische Datenverarbeitung erfassen und für die digitale Weiterverarbeitung abspeichern. Im dritten Kapitel wird auf einige wichtige **Grundlagen** wie das DICOM-Format oder der für diese Arbeit sehr zentrale Begriff der Grid-Daten eingegangen. Da die Darstellung der für die Entwicklung von YaDiV benötigten Grundbegriffe zusammen mit einem angemessenem Stand der Literatur ein einziges Kapitel sprengen würde, wurde diese Diskussion in die Kapitel **Segmentierung**, **Visualisierung** und **Haptik** aufgeteilt.

Kapitel 7 beschäftigt sich mit der Plattform **YaDiV**, die während dieser Arbeit entwickelt wurde. Hier wird weniger im Detail auf die einzelnen Knöpfe der Benutzeroberfläche, als auf bleibende Konzepte und die Philosophie hinter deren Verwendung eingegangen.

Ebenso werden einige technische Details erläutert, wenn sie wichtig für das weitere Verständnis sind oder eine besonders effiziente Umsetzung einer Methode bzw. Strategie ermöglichen.

Die Ergebnisse werden anschließend in Kapitel 8 **zusammengefasst**. Im letzten Kapitel soll dann ein **Ausblick** auf zukünftige Arbeiten, die die hier vorgestellte Plattform nutzen oder erweitern, gegeben werden.

Danksagungen

Mein Dank gilt dem Welfenlab, das mir die Möglichkeit zur Promotion gegeben hat, insbesondere meinem Doktorvater Prof. Dr. Franz-Erich Wolter für die wissenschaftliche Betreuung der vorliegenden Arbeit. Für ihre Mühen bei der Begutachtung der Dissertation sei Prof. Nadia Magnenat-Thalmann vielmals gedankt. Ihre Besuche an unserem Lehrstuhl und insbesondere ihre Vorträge, auch auf den gemeinsam organisierten Konferenzen, waren immer eine Quelle der Inspiration.

Diese Arbeit wäre ohne die Mithilfe vieler talentierter Studierender nicht möglich gewesen, die entweder als wissenschaftliche Hilfskraft oder in Form einer Abschlussarbeit an der Entwicklung von YaDiV mitgewirkt haben. Daher geht ein besonderer Dank an:

- Benjamin Berger (Licht Editor, 2D-Rekonstruktionsfilter)
- Richard Guercke (höhenlinienbasierte Segmenttriangulierung)
- Robert Meyer (atlasbasierte Segmentierungsverfahren)
- Maximilian Müller (High Quality Visualisierung mit Raycasting-Verfahren)
- Dominik Sarnow (Analyse unterschiedlicher Direct Volume Rendering-Verfahren)
- Björn Scheuermann (Level-Set basierte Segmentierungsverfahren)
- Lara Toma (Experimente zur Grauwertdarstellung auf LCD Displays)
- Marc Christoph Vollmer (Entwicklung einer haptischen Schnittstelle)
- Johanens Wahle (Lit Sphere Maps)
- Yifan Yu (Vergleich mehrerer Marching Cube- und Marching Tetraeder-Varianten)

Zuletzt möchte ich mich bei meinen Freunden bedanken, die mir in der letzten Phase dieser Arbeit durch Korrekturlesen und Spenden seelischen Beistands geholfen haben: Jasper Berndt-Gerdes, Philipp Blanke, Antje Gerdes, Marion Kühl und Nadine Weber. Da es in den letzten Wochen nicht immer einfach mit mir war, möchte ich mich dabei auch gleichzeitig nachträglich entschuldigen. Ihr habt etwas bei mir gut!

2. Bildgebende Verfahren

Eines der zentralen Themen dieser Arbeit ist die Visualisierung und die Segmentierung von zuvor gewonnenen medizinischen Daten. In diesem Kapitel soll ein kurzer Überblick über die unterschiedlichen Arten der Datengewinnung gegeben werden. Der Schwerpunkt wird dabei auf **nicht-invasiven Gewinnungsmethoden** wie der Computertomographie (CT) und der Magnet-Resonanz-Tomographie (MRT) liegen, die beide zu den sogenannten „bildgebenden Verfahren“ gehören. Als bildgebendes Verfahren bezeichnet man eine Methode, mit der vor allem medizinische Befunde, aber auch andere physikalische und chemische Phänomene visualisiert werden. Allen Verfahren ist gemeinsam, dass Messungen eines von einem Objekt ausgehenden physikalischen Effekts in ein Bild umgewandelt werden. Dieser Effekt wird z.B. durch eine Exposition mit ionisierender oder nicht ionisierender Strahlung hervorgerufen, z.B. mit Ultraschall, Röntgenstrahlen oder Radioisotopen.

Bildgebende Verfahren werden in nahezu allen naturwissenschaftlichen Disziplinen eingesetzt. Sie haben besonders in den verschiedenen Fachgebieten der Medizin eine große Bedeutung bei der Darstellung und der darauf aufbauenden Diagnose pathologischer Gewebeveränderungen erlangt. Sowohl in der Forschung, als auch in der klinischen Praxis kommen eine Vielzahl unterschiedlicher Verfahren zum Einsatz: von der konventionellen Röntgendiagonstik über die Sono- bzw. Echographie (Ultraschalluntersuchung) bis hin zur Computer-Tomographie[1] und Kernspin-Tomographie, oft auch als Magnet-Resonanz-Tomographie bezeichnet.

Da sich diese Arbeit vor allem mit der Interpretation und Manipulation der gewonnenen Daten beschäftigt, sollen die nun folgenden Bildgewinnungsmethoden kurz, aber verständlich zusammengefasst und die interessierten Leser an die entsprechende Fachliteratur verwiesen werden. Eine sehr gute Einführung in die „Bildgebenden Verfahren in der Medizin“ wird u.a. im gleichnamigen Buch [16] von Olaf Dössel gegeben. Nur wenig älter, aber ebenfalls sehr empfehlenswert ist [54], das durch eine große Zahl an Autoren ein breiteres Spektrum abdeckt. Neben der Vorstellung der Technik hinter den bildgebenden Verfahren werden auch andere Aspekte dieser Thematik beschrieben. So wird z.B. eine Einführung in die Physiologie des menschlichen Sehens gegeben und eine Diskussion über den Begriff der Bildgüte angestoßen.

[1] aus dem Griechischen τομή „Schnitt“ und γράφειν „schreiben“

Weil die Mehrzahl der in dieser Arbeit verwendeten Volumendatensätze aus CT und MRT Scannern stammen, soll auf diese beiden Verfahren genauer eingegangen werden. Für einige Anwendungen ist es sinnvoll, zumindest rudimentär mit den physikalischen Grundlagen der Datengewinnung vertraut zu sein. Für das Verständnis der weiteren Kapitel dieser Arbeit reicht es jedoch auch aus, sich die gewonnenen Daten als diskrete Messwerte vorzustellen, die auf einem regulären, orthogonalen Gitter (Grid) (siehe Kapitel 3.3) vorliegen, also als (äquidistante) Abtastung des Volumens, bei der je nach Aufnahmemethode unterschiedlich zu interpretierende (Intensitäts-)Werte gespeichert werden, die bspw. als Graustufen visualisiert werden können.

2.1. Computer-Tomographie (CT)

Bei einer konventionellen Röntgenaufnahme wird das abzubildende Objekt von einer Röntgenquelle durchstrahlt und auf einem Röntgenfilm abgebildet. Es entsteht eine Art Schattenbild, also eine Projektion des Objekts auf eine Fläche. Bereiche mit hoher Schwächung der Röntgenstrahlung erscheinen heller, Bereiche mit niedriger Schwächung dunkler. Durch die 2D-Projektion gehen jedoch die Informationen über die dritte Dimension des durchleuchteten Körpers weitgehend verloren. Es kann nicht mehr unterschieden werden, ob die im Röntgenbild sichtbare Schwächung durch ein Material höherer Dichte (bzw. stärkerer Absorption) oder durch eine größere Schichtdicke hervorgerufen worden ist.

Abbildung 2.1.: 16-Zeilen-Multidetektor-CT *Quelle: Wikipedia*

In der Computertomographie bleibt diese dritte Dimension erhalten. Vereinfacht gesagt wird eine Vielzahl von Röntgenbildern des Objekts aus unterschiedlichen transversalen Richtungen[1] erstellt und nachträglich rechnerunterstützt die verlorenen Volumeninformationen rekonstruiert. In der Regel setzen sich die so erzeugten 3D-Rekonstruktionen aus Einzelschnitten bzw. Schichtbildern zusammen.

[1] Rotation um die Längsachse.

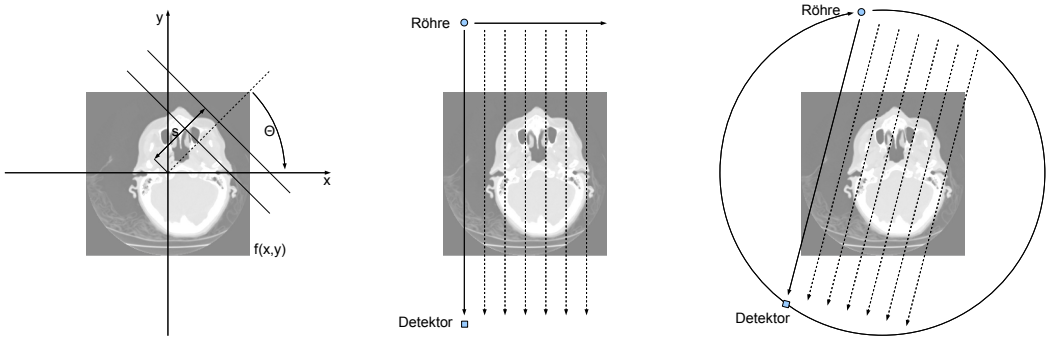
2.1.1. Prinzip

Die Mathematik, auf der die Computertomographie basiert, geht zurück auf den österreichischen Mathematiker Johann Radon, der sich im Jahr 1917 für die „Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten“ [64] interessierte. Nach ihm wurde die sogenannte Radon-Transformation benannt, die eine integrierbare Funktion $f(x, y)$ durch alle geraden Linienintegrale über ihrem Definitionsgebiet beschreibt:

$$R(f)(m, b) = \int_{-\infty}^{\infty} f(x, mx + b)dx$$

Dabei ist f eine stetige Funktion in der Ebene, die außerhalb eines Kreises mit endlichem Radius identisch Null ist. Die Umkehrung dieser Transformation wird in der Computertomographie dazu benutzt, um aus den gemessenen Projektionen ein einzelnes, zweidimensionales Bild zu rekonstruieren.

Als Ordnungsschema für die Linien bietet es sich an, diese nach ihrem Abstand zum Nullpunkt s und dem Winkel ihrer Normalen Θ anzuordnen (Abbildung 2.2 a). Der prinzipielle Aufbau eines CT-Scanners erinnert stark an diese Situation: Betrachtet man einen einzelnen Röntgenstrahl, der von einer Röhre emittiert und nach Durchdringung des Scan-Objekts von einem Detektor aufgefangen wird, entspricht die gemessene Schwächung der Intensität gegenüber dem emittierten Wert gerade dem Integral über den Röntgenschwächungskoeffizient über alle Punkte des Strahls. Werden mehrere parallele Strahlen (Translation) verwendet und die Messung aus unterschiedlichen Winkeln (Rotation) wiederholt, lässt sich der Röntgenschwächungskoeffizient für einzelne, diskrete 2D Volumenelemente rekonstruieren (siehe Abbildung 2.2 b und c).



(a) Radon-Transformation (b) CT-Translation (c) CT-Rotation

Abbildung 2.2.: CT Scanner und Radon-Transofmation

2.1.2. Scanner Generationen

Im Jahr 1970 erhielt Godfrey Newbold Hounsfield das Patent für die erste CT Aufnahme. Die Aufnahme selbst dauerte 9 Tage und die Bildrekonstruktion im Rechenzentrum über 2 Stunden. Für seinen Beitrag zur Entwicklung des ersten CT Scanners erhielt Hounsfield 1979 zusammen mit A.M. Cormack den Nobelpreis für Medizin. Der Aufbau dieses ersten Scanners war verhältnismäßig einfach: Eine einfache Röhre emittiert einen Röntgenstrahl, der das zu scannende Objekt durchdringt und anschließend von einem einzelnen Detektor erfasst wird. Dabei müssen Röhre und Detektor sowohl horizontal entlang des Objekts bewegt werden (Translation), als auch eine vollständige Rotation durchführen, d.h. nach jeder vollständigen Translationsreihe wird rotiert.

Bei Scannern der 2. Generation werden gleichzeitig mehrere Detektoren nebeneinander eingesetzt (Detektor-Array). Anstelle eines einzelnen Strahls wird ein ganzer Fächer emittiert (1975: ca. 30 Detektoren, 10° Fächerbreite). Durch diese Technik konnte die Aufnahmedauer bereits auf 20 Sekunden reduziert werden. Auch hier sind noch mehrere Translations- und Rotationsschritte notwendig, durch die fächerförmigen Messungen ändert sich die Berechnung der Rekonstruktion. CT Scanner der 3. Generation lehnen sich vom Aufbau an die Scanner der 2. Generation an: Auch hier wird ein Strahlenfächer von einem Reflektorarray aufgefangen; durch die Verbreiterung des Fächers auf 40° bis 60° kann jedoch das gesamte Volumen auf einmal erfasst werden, wodurch die Translation beim Scan-Vorgang entfällt. Zudem konnte durch das „springende Fokusverfahren“ die Auflösung erhöht werden ([16], S. 114).

Die wesentliche Neuerung von Scannern der 4. Generation ist ein stehender Detektorring (360°), so dass lediglich die Röhre selbst rotiert werden muss. Auch bei diesen Geräten entfällt natürlich die Röhrentranslation. Heutzutage werden in der Humanmedizin meist Scanner aus der 3. und 4. Generation eingesetzt. Standard sind mittlerweile die sogenannten Spiral-CT, bei denen der Patient auf einer sich kontinuierlich vorwärts bewegenden Unterlage liegt. Dadurch verlaufen die von Röhre und Detektor aufgenommenen Werte spiralförmig; um daraus ebene Schichtbilder zu erzeugen, wird zwischen den einzelnen Spiralschlaufen interpoliert.

Die Strahlenbelastung bei CT-Aufnahmen ist relativ hoch und abhängig von verschiedenen Parametern (Anzahl und Dicke der Schichtaufnahmen, Umfang des zu untersuchenden Bereich, etc.). Sie ist bis zu 100 mal größer als bei konventionellen 2D Röntgenbildern. Insbesondere in der Forschung ist daher ihre Anwendung gesetzlich geregelt [40]. Basierend auf der novellierten Röntgenverordnung (RöV) vom 1. Juli 2002, in der die Anwendung radioaktiver Stoffe und ionisierender Strahlung am Menschen zum Zwecke der medizinischen Forschung (§§ 23 und 24) reglementiert wird, ist grundsätzlich eine Genehmigung durch das Bundesamt für Strahlenschutz erforderlich. Zur Genehmigung eines Forschungsvorhabens, für das neue CT-Aufnahmen erhoben werden, wird dort u.a. ein

Studienplan und die Stellungnahme einer durch das BfS registrierten Ethikkommission gefordert.

2.2. Magnet-Resonanz-Tomographie (MRT)

Eine Alternative zur CT stellt die Magnetresonanztomographie (MRT) dar. Die beiden Hauptvorteile dieses Verfahrens gegenüber der CT sind der Verzicht auf schädliche Röntgenstrahlung und die Möglichkeit, Organe und Gewebe auch ohne Kontrastmittel mit hohem Weichteilkontrast abzubilden. Nachteile sind unter anderem der höhere Anschaffungspreis und die ebenfalls höheren Betriebskosten der MRT-Geräte sowie längere Untersuchungszeiten.

Abbildung 2.3.: 3T Achieva MRI Scanner, Philips *Quelle: Wikipedia*

Die Magnet-Resonanz-Tomographie basiert auf der sogenannten Kernspinresonanz. Der Spin bezeichnet den Eigendrehimpuls von Protonen und Neutronen und den dadurch entstehenden magnetischen Moment des Atomkerns. Durch ein Magnetfeld werden bestimmte Atomkerne (meistens Wasserstoff) dazu angeregt, ihren Spin der Magnetfeldrichtung anzugleichen. Wird nun ein zweites Magnetfeld angelegt, führt dies zu einer andauernden Präzession, die durch einen Radiowellenimpuls gemessen werden kann. Dieser Impuls bringt die Kerne dazu, selbst Radioenergie auszustrahlen.

Als Grundlage für den Bildkontrast werden die unterschiedliche Relaxationszeiten verschiedener Gewebearten (z. B. Muskel, Knochen) verwendet, die durch eine sensible Messtechnik aufgenommen werden können. Dabei bezeichnet T1 den Wert für Zeit, die ein Kern für die Ausrichtung benötigt und T2 den Zeitpunkt, an dem 63% der Radioenergie abgestrahlt wurde. Beide lassen sich als Intensitätswert interpretieren und damit bspw. als Grauwertbild visualisieren. Bilder mit stärkerer T1-Gewichtung haben dabei üblicherweise eine höhere Ortsauflösung, während Bilder mit stärkerem T2-Anteil einen höheren Gewebekontrast besitzen. Im Gegensatz zum CT lassen sich die aufgenommenen (Relaxations-) Werte nicht eindeutig physikalisch interpretieren bzw. einem bestimmten

Gewebe zuordnen. Die Bildinterpretation stützt sich daher auf den Gesamtkontrast und das Erkennen bekannter Strukturen.

Da keine Belastung durch ionisierende Strahlung entsteht, werden MRT-Aufnahmen routinemäßig eingesetzt. Allerdings muss wegen der starken Magnetfelder besonders auf Implantate oder andere metallische Teile, wie z.B. Schrauben, im Körper des Patienten geachtet werden. Auch im Untersuchungsraum selbst ist besondere Vorsicht geboten, oft gibt es eine Schleuse mit Metall-Detektoren, die Patienten und Personal vor dem Betreten durchqueren müssen. Dies gilt insbesondere beim Einsatz von sogenannten offenen MRT-Scannern (siehe Abbildung 2.4), die teilweise auch vertikal aufgebaut sein können, um z.B. Belastungen bei einem stehenden oder sitzenden Patienten aufzunehmen.

Abbildung 2.4.: Offener MRT Scanner *Quelle: MHH*

Synonym zur Bezeichnung Magnetresonanztomographie wird auch der Begriff Kernspintomographie verwendet (umgangssprachlich gelegentlich zu Kernspin verkürzt). Die ebenfalls zu findende Abkürzung MRI stammt von der englischen Bezeichnung Magnetic Resonance Imaging.

2.3. Weitere Verfahren

Neben MRT und CT gibt es eine Vielzahl weiterer bildgebender Verfahren, auf die im Detail einzugehen den Umfang dieser Arbeit bei weitem sprengen würde. Dennoch soll ein kurzer Überblick über weitere gebräuchliche Verfahren versucht werden, der sich jedoch auf 3D bildgebende Geräte beschränkt.

Die **Sono- oder Echographie** verwendet Ultraschall zur Bildgebung. Ursprünglich als Methode zur Ortung von U-Booten entwickelt, wurde diese Technik seit Mitte des letzten Jahrhunderts auch auf die Medizin ausgeweitet und hat sich seitdem stark weiterentwickelt. Ein großer Vorteil der Sonografie gegenüber der Röntgenuntersuchung liegt in

der Unschädlichkeit der eingesetzten Schallwellen, weswegen sie u.a. auch bei Schwangeren eingesetzt werden kann. Zu Beginn des 21. Jahrhunderts wurden die ersten 3D-Ultraschall-Scanner eingesetzt, die ein räumliches Bild liefern. Sogenannte 4D-Ultraschall-Scanner (auch Live-3D-Scanner genannt) liefern die Bilddaten so schnell, dass sie während der Untersuchung interaktiv als Animation dargestellt werden können.

Die **Positronen-Emissions-Tomographie (PET)** gehört in den Bereich der nuklearmedizinischen Diagnostik. Dem Patienten wird eine schwach radioaktiv markierte Substanz injiziert und anschließend deren Verteilung gemessen. Die Aufnahmegeräte selbst arbeiten daher komplett passiv. Die Halbwertzeiten der Markersubstanzen sind sehr kurzlebig (10-100 Minuten), was u.a. dazu führt, dass sie in unmittelbarer Nähe hergestellt werden müssen.

Ein weiteres nuklearmedizinisches Verfahren ist die **Single Photon Emission Computed Tomography (SPECT)**. Das Verfahren arbeitet mit Gammastrahlung: Auch hier wird ein radioaktiver Marker injiziert, dessen Strahlung von mehreren rotierenden Gamma-Kameras gemessen wird. Verglichen mit der PET ist SPECT weniger aufwendig, da keine kurzlebigen Radionuklide verwendet werden und auch die Scanner selbst wesentlich billiger sind. Das rechnerische Verfahren zur Rekonstruktion der Bilddaten setzt – wie beim CT – die inverse Radon-Transformation ein.

In den letzten Jahren kommen auch in der Medizin verstärkt **3D-Bodyscanner** zum Einsatz, welche z.B. über Lasertechnik eine reine Oberflächendarstellung des Patienten liefern. Die so gewonnenen Daten können in Kombination mit einer digitalen Fotografie und einem tomographischen Datensatz verwendet werden. Einsatzgebiete sind neben der plastischen Chirurgie vor allem die Rechtsmedizin, insbesondere die postmortale Forensik.

3. Grundbegriffe

Das Ergebnis dieser Arbeit ist ein modulares Programm zur Visualisierung und Segmentierung medizinischer 3D-Daten, welches als Plattform für die schnelle prototypische Erprobung neuer Ideen in diesem Bereich dienen soll. Das Programm und damit auch diese Arbeit richtet sich damit eher an Leser aus dem Gebiet der Computergraphik bzw. der Bilderkennung, das Anwendungsgebiet ist jedoch die Medizin und die Anwender sind in der Regel Ärzte.

In diesem Kapitel wird deshalb ein besonderes Augenmerk auf Begriffe und Konzepte aus dem medizinischen Bereich gelegt, soweit sie für das Verständnis der weiteren Arbeit von Bedeutung sind. Da die Computergraphikgemeinde mit Begriffen wie „Voxel" oder „RGB-Daten" vertraut ist, soll auf die Grundlagen in diesem Bereich nur kurz eingegangen werden.

3.1. DICOM-Format

DICOM ist die Abkürzung für *Digital Imaging and Communications in Medicine*. Es handelt sich um einen offenen Standard zum Austausch von Informationen in der Medizin, der von fast allen Herstellern bildgebender oder -verarbeitender Systeme unterstützt wird. Im DICOM-Standard wird sowohl das Format zur Speicherung der Daten als auch das Kommunikationsprotokoll zum Datenaustausch beschrieben.

Der DICOM-Standard ist die Grundlage für die digitale Bildarchivierung in Praxen und Krankenhäusern (Picture Archiving and Communication System, PACS). Da das im Rahmen dieser Arbeit entwickelte Programm DICOM-Daten einliest und verarbeitet, soll hier kurz auf die Entwicklung des Standards und den (vereinfachten) Aufbau von DICOM-Daten eingegangen werden.

3.1.1. Geschichtliches

Entwickelt wird der Standard seit 1983, ursprünglich vom ACR (American College of Radiology) und der NEMA (National Electrical Manufacturers Association). Ziel war es, die

Interoperabilität zwischen bildgebenden und -verarbeitenden Systemen von verschiedenen Herstellern zu gewährleisten. 1985 wurde die erste Version als ACR/NEMA-Standard veröffentlicht. Seit 1993 wird der Name DICOM-Standard verwendet und bis heute von mehreren Arbeitsgruppen gepflegt und weiterentwickelt. Die unterschiedlichen Revisionen werden anhand ihres Erscheinungsjahres gekennzeichnet. Derzeit aktuell ist die Version 2008, die unter [57] erhältlich ist.

3.1.2. Terminologie

In diesem Abschnitt werden einige grundlegende Begriffe und Konzepte erläutert, auf denen der DICOM Standard aufgebaut ist und die auch im weiteren Verlauf dieser Arbeit immer wieder benötigt werden.

Der Standard beschreibt den Aufbau einer Binärdatei für den medizinischen Gebrauch und unterstützt eine Vielzahl von Bildformaten und Erweiterungen. Ein Datensatz dient als Container und enthält außer den Bilddaten auch Metainformationen wie Patientenname, Aufnahmedatum, Geräteparameter oder den Namen des behandelnden Arztes. Diesen Metadaten wird das sogenannte **Real World Information Model** zugrunde gelegt, das in die Stufen Patient, Studie, Serie und Instanz unterteilt ist.

Eine **Studie** (DICOM-Standard: Study) ist eine Sammlung von medizinischen Untersuchungen eines bestimmten Patienten. Im Verlauf einer Studie können mehrere unterschiedliche bildgebende Verfahren zum Einsatz kommen, bspw. eine MRT und eine CT-Aufnahme. Alle Bilder, die im Verlauf einer Studie aufgenommen werden, erhalten bereits bei der Aufnahme die selbe *Study Instance UID* (UID = Unique Identifier), um sie auch nachträglich der zugrundeliegenden Studie zuordnen zu können.

Unter einer **Serie** (DICOM: Series) wird eine Sequenz von Bildern verstanden, die im Verlauf einer Untersuchung durch ein einzelnes bildgebendes Verfahren zustande gekommen ist. Alle Bilder dieser Serie haben die selbe *Series Instance UID* und natürlich auch die selbe *Study Instance UID* und wurden mit einem einzigen Gerät erzeugt.

Objekte werden durch eine sogenannte **Information Object Definition**, kurz IOD, definiert. Diese besteht aus mehreren Modulen, die wiederum einzelne Attribute bzw. Sequenzen von Attributen enthalten. Ein Schichtbild hat z.B. die Attribute „Breite" und „Höhe". Die Kombination aus einem Informationsobjekt und einer Aktion bildet ein **Service-Object-Paar** (abgekürzt SOP, z. B. „CT-Bild speichern", „Ultraschallbild drucken", etc.). SOPs bilden die funktionellen Grundeinheiten des Standards. Jedes Bild einer Serie bekommt so neben der *Study Instance UID* und der *Series Instance UID* noch eine eindeutige *SOP Instance UID*.

Als Datengrundlage dieser Arbeit spielen DICOM-Dateien eine wichtige Rolle. Genau genommen handelt es sich dabei lediglich um einen Container für medizinisch relevante

Informationen. Diese Informationen können sehr unterschiedlicher Natur sein. Im Folgenden werden wir uns aber der Einfachheit halber auf den sicherlich häufigsten und für diese Arbeit relevanten Fall beschränken, dass die gekapselte Information ein Schichtbild aus einer Sequenz von Aufnahmen ist.

3.1.3. Aufbau

Jede DICOM-Datei beschreibt – vereinfacht dargestellt – ein einzelnes Bild aus einer Serie von Schnittbildaufnahmen. Zusätzlich zu den reinen Bilddaten sind auch Aufnahmeparameter wie der x-, y- und z-Abstand zwischen zwei Gridpositionen, die Aufnahmeschichtbreite oder die Orientierung des Patienten sowie Informationen über die Umstände der Aufnahme enthalten. Da jede einzelne Datei personalisiert ist, ist die Weitergabe nicht-anonymisierter Dateien auch für rein wissenschaftliche Untersuchungen ein sensibeles Thema.

Da der Aufbau einer DICOM-Datei sehr komplex sein kann, soll hier eine kurze, vereinfachende Darstellung gegeben werden, um zumindest eine grobe Vorstellung des Binärformats zu vermitteln. Eine DICOM-Datei besteht immer aus zwei Teilen: Dem Kopf (Header) und den eigentlichen Daten (Data Set), wie in Abbildung 3.1 skizziert.

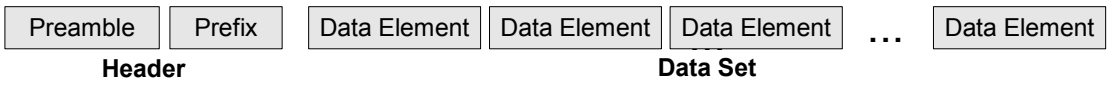


Abbildung 3.1.: Aufbau einer DICOM-Datei

Der **Header** einer DICOM Datei besteht immer aus 132 Bytes (0-131). Die ersten 128 Bytes bilden die sogenannte *Preamble* und sind nicht vorgegeben. Sie können je nach Anwendung unterschiedliche Daten aufnehmen. Byte 128-131 werden als *Prefix* bezeichnet und enthalten den ASCII Code der Buchstaben 'D', 'C', 'I' bzw. 'M'. Da das DICOM Format keine etablierte Dateiendung (wie bspw. .jpg oder .png) besitzt, wird der Prefix oft für einen schnellen (aber natürlich nicht ausreichenden) Test auf Formatzugehörigkeit einer Datei verwendet.

Nach dem Header folgen im **Data Set** die eigentlichen Inhalte der DICOM Datei. Das Data Set besteht dabei aus einer Aneinanderreihung von Datenelementen, deren Syntax etwas komplexer ist. Ein **Data Element** setzt sich aus mehreren Bestandteilen zusammen: Dem *Data Element Tag* (Kennung), der *Value Representation* (kurz VR, Format der Daten, nur bei expliziter VR-Darstellung direkt angegeben), der *Value Length* (kurz VL, Datenlänge) und dem *Value Field*, also den eigentlichen Daten dieses Data Elements. Das Data Set kann in impliziter oder expliziter-VR Darstellung (siehe weiter unten) vorliegen und sowohl im little-, als auch im big-endian Format angegeben werden.

Die ersten vier Bytes eines Data Element enthalten das sogenannte **Data Element Tag** und damit die (eindeutige) Kennung des Elements. Die ersten beiden Tag-Bytes werden als *Group ID*, die zweiten als *Element ID* bezeichnet. PS 3.6 des DICOM-Standards enthält das Data Dictionary, welches alle standardisierten Tags, deren VR und eine textuelle Beschreibung des Inhalts aufführt. Alle Standard-Tags besitzen eine gerade Group ID[1]. Ungerade Gruppenzahlen[2] sind proprietären Datenelementen vorbehalten, deren Bedeutung je nach Gerät oder Anwendung variieren kann.

Das Data Element Tag wird auch in der Dokumentation immer in hexadezimaler Darstellung angegeben. So ist z.B. bei Tag (0028,0011) 0028 die Group ID und 0011 die Element ID. In diesem Fall würde es sich um das Data Element handeln, das die Bildbreite beschreibt.

Nach dem Data Element Tag folgt entweder die **Value Representation** (explizites VR Format) oder bereits die Value Length (implizites VR Format). Ob eine Datei die explizite oder implizite VR Darstellung benutzt, kann über die Transfer Syntax UID feststellt werden ([57], PS 3.10, Kapitel 7.1). Bis zum Ende der Meta-Group (Group ID 0002) ist das Format immer little endian mit expliziter VR. In der Praxis werden beim Einlesen eines DataElement auch oft die ersten beiden Bytes nach dem Data Element Tag eingelesen und dann überprüft, ob es sich um eine gültige VR Angabe (siehe unten) handelt. In diesem Fall würde eine explizite VR Darstellung angenommen. Im impliziten Fall wird die VR aus dem Data Dictionary geholt, andernfalls kann das Data Dictionary zur Kontrolle benutzt werden, ob eine explizit angegebene VR vom Standard abweicht.

Die beiden VR Bytes werden als einzelne ASCII Zeichen interpretiert und können die folgenden Werte annehmen: AE, AS, AT, CS, DA, DS, DT, FD, FL, IS, LO, LT, PN, SH, SL, SS, ST, TM, UI, UL, US, UT, OB, OF, OW, SQ, UN, QQ sowie OX, DL und XX. Diese werden auch als Data Types bezeichnet. Dabei steht z.B. IS für die Value Representation „Integer String“, also eine Ganzzahl, deren einzelne Dezimalstellen jeweils als ein ASCII Byte abgelegt wurden.

Eine vollständige Erklärung aller VR Typen steht im PS 3.5 des DICOM-Standards, Kapitel 6.2. Eine besondere Erwähnung soll lediglich der VR Typ SQ finden. SQ steht in diesem Fall für „Sequence“ und bedeutet, dass das Value Field dieses Datensatzes eine eigene Sequenz von Data Elements enthält, was eine eigene Sonderbehandlung erfordert, da die VL in diesem Fall nicht angegeben sein muss (*Undefined Length Sequence*[3]). Innerhalb einer Sequenz können auch Tags auftreten, die bereits außerhalb angegeben wurden. Diese sind daher strikt zu trennen. Komplizierter für die Entwicklung eines Parsers wird

[1] Ausnahmen: (0000,eeee), (0002,eeee), (0004,eeee), und (0006,eeee)

[2] Ausnahmen: (0001,eeee), (0003,eeee), (0005,eeee), (0007,eeee), und (FFFF,eeee)

[3] In diesem Fall definieren zwei spezielle Data Elemente (fffe,e000) und (fffe, e00d) den Anfang bzw. das Ende der Sequenz.

die Situation dadurch, das auch eine Sequenz selbst wiederum eine Sequenz enthalten kann.

Nach der VR (im expliziten Format, sonst direkt nach dem Data Element Tag) folgt die **Value Length** (VL). Die VL gibt die Anzahl der Bytes im nachfolgenden Value Field an und besteht in der expliziten VR Darstellung aus

- 4 Bytes für die VR Data Types OB, OW, SQ, UT, UN bzw.
- 2 Bytes für alle anderen VRs

und wird entweder als Integer (4 Byte) oder Short (2 Byte) aufgefasst. Im Fall von 4 Bytes müssen die ersten beiden Bytes nach der VR übersprungen werden und die VL steht in den darauf folgenden 4 Byte, andernfalls folgt sie sofort auf die VR. Bei impliziter VR Darstellung besteht die VL immer aus 4 Byte. Abbildung 3.2 zeigt eine schematische Darstellung eines DataElements..

Explizite VR-Darstellung für OB, OW, SQ, UT, UN

Group ID	Element ID	VR	--	VL	Value Field
2 Bytes	2 Bytes	2 Bytes	2 Bytes	4 Bytes	„VL“ Bytes

Explizite VR-Darstellung für alle anderen VR Typen

Group ID	Element ID	VR	VL	Value Field
2 Bytes	2 Bytes	2 Bytes	2 Bytes	„VL“ Bytes

Implizite VR-Darstellung

Group ID	Element ID	VL	Value Field
2 Bytes	2 Bytes	4 Bytes	„VL“ Bytes

Abbildung 3.2.: Aufbau eines Datenelements

Das **Value Field** enthält die eigentlichen Daten. Es besteht aus der durch die VL angegebenen Anzahl Bytes und wird mittels der VR interpretiert.

3.1.4. Unterstützte Bildformate

Der DICOM-Standard erlaubt mehrere unterschiedliche Bildformate ([57], PS 3.5, Kapitel 8). Das verwendete Format lässt sich anhand der Transfer Syntax UID ([57], PS 3.5, Kapitel 10) feststellen. Die Pixeldaten selbst stehen immer im Value Field Bereich des Datenelements mit der Kennung (7FE0,0010). Die Interpretation der dort gespeicherten Byte-Folge hängt vom gewählten Bildformat, der Pixel Representation (Unsigned Integer oder 2-Komplementdarstellung), vom Binärformat (Little- bzw. Big-Endian) und der Value Representation von (7FE0,0010) (OB bzw. OW) ab.

Das Default-Format, welches in der Praxis am häufigsten verwendet wird, ist die **Raw Data** Kodierung ([57], PS 3.5, Kapitel 8.1.1). Bei diesem Format werden die Pixeldaten als unkomprimierter Byte-Array abgelegt. Dabei wird vorher festgelegt, wie viele Bits pro Pixel gespeichert (Bits Allocated) und welche davon tatsächlich verwendet werden (Bits Stored). Das High-Bit gibt an, welcher Teil der allozierten Bits mit Bild-Daten belegt ist. Die Byte-Reihenfolge (Little- oder Big-Endian) wird durch die Transfer Syntax festgelegt. Abbildung 3.3 zeigt ein Beispiel mit Bits Allocated = 16, Bits Stored = 12 und High Bit = 11.

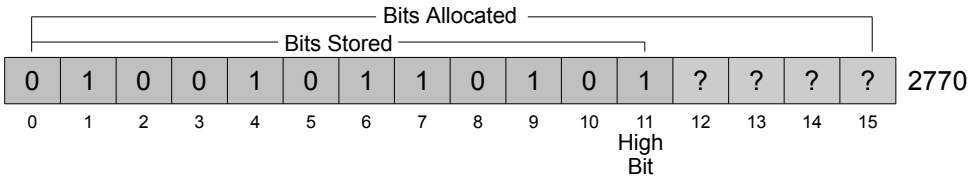


Abbildung 3.3.: Beispiel für Raw Data Pixelcodierung

Die Interpretation der einzelnen Pixel im Raw-Data-Format wird durch die *Photometric Interpretation* bestimmt (siehe [57], PS 3.3, Anhang 7.6.3.1.2). Die gültigen Bezeichner lauten MONOCHROME1, MONOCHROME2, PALETTE COLOR, YBR_FULL, YBR_PARTIAL_422, RGB, YBR_ICT, YBR_RCT und YBR_PARTIAL_420. In CT- und MRT-Aufnahmen wird üblicherweise die MONOCHROME2-Interpretation verwendet.

Neben dem Raw-Data-Format wird auch eine Auswahl von gekapselten Standardformaten unterstützt. Der prominentesten Vertreter ist hierbei die JPEG Kompression, die sowohl in verlustfreien, als auch verlustbehafteten Schema unterstützt wird ([57], PS 3.5, Kapitel 8.2.1). Ein weiterer verlustfreier Vertreter ist eine Variante der Run Length Encoding (RLE) Kompression, ([57], PS 3.5, Kapitel 8.2.2) die auch in der TIFF 6.0 Spezifikation als „PackBits" vorkommt. Weitere unterstützte Formate sind u.a. JPEG-LS, JPEG 2000 und spezielle Formate für Multi-Frame-DICOM Dateien, die eine Sequenz von Bildern enthalten können (bspw. MPEG2 MP@ML). Bei allen gekapselten Formaten schreibt der Standard vor, dass die Angaben im gekapselten Bildformat (Bildhöhe, Bildbreite, etc.) nicht von den zugehörigen DICOM-Datenelementen abweichen dürfen.

Anmerkung: Der DICOM-Standard verzichtet als rein technischer Standard explizit darauf, eine Empfehlung abzugeben, für welchen Einsatzzweck eine verlustbehaftete Kompression von Bilddaten ethisch vertretbar ist. So diskutieren unterschiedliche Gruppen darüber, ab welchem Grad einer verlustbehafteten Kompression der dadurch entstehende Fehler minimale, aber möglicherweise kritische Details verändert (siehe [60], S. 1368 und [3], S. 2204). Als allgemeine Empfehlung gilt, in klinischen Anwendungen nur verlustfreie Kompressionsverfahren zu verwenden. Verlustbehaftete Verfahren sind danach nur in nicht-klinischen Anwendungen, sowie in der Lehre zulässig.

3.1.5. Wichtige Datenelemente

Zum abschließendem Verständnis folgt hier noch noch einmal eine Auflistung der wichtigsten Datenelemente, die für das Einlesen von DICOM Dateien und insbesondere deren Visualisierung notwendig sind.

(xxxx,0000) **Group Length** ist das erste Data Element einer Gruppe und beinhaltet die Länge der Gruppe (in Bytes). Dies kann z.B. benutzt werden, um beim Einlesen ganze Gruppen zu überspringen.

(0002,0002) **Media Stored SOP Class Unique Identification**, kurz Media Stored SOP Class UID, enthält eine eindeutige Kennung zur Art der vorliegenden Daten. Mit der SOP Media Class lässt sich bspw. feststellen, ob es sich um ein mit CT oder MRT gewonnenes Bild handelt.

(0002,0010) **Transfer Syntax UID** ist eine eindeutige Kennung über das Format der folgenden Daten. Die Transfer Syntax UID 1.2.840.10008.1.2 steht z.B. für „Raw data, Implicit VR, Little Endian". Weitere Informationen hierzu gibt es in [57], Part 5, Section 10.

(0018,0050) **Slice Thickness** enthält die Schichtdicke (Abstand zwischen zwei Bildern) in mm. Da diese Angabe in der Praxis geräteabhängig unterschiedlich verstanden wird, werden zusätzlich meist „Image Orientation Patient" (Bildnormale) und „Image Position" (Bildabstand) verwendet, um den Wert zu überprüfen und gegebenenfalls korrigieren zu können.

(0018,5100) **Patient Position** beschreibt die Position des Patienten relativ zum Gerät, dies ist u.a. für CT- und MRT-Aufnahmen wichtig. Mögliche Werte sind z.B. HFP für „head first-prone" (Patient wird in Bauchlage mit dem Kopf zuerst in den Scanner geschoben), HFS für „head first-supine" (Rückenlage), FFS „feet first-supine" etc. Eine gute Einführung hierzu findet man in [22].

(0020,000d) **Study Instance UID** enthält die eindeutige Kennung der Studie.

(0020,000e) **Series Instance UID** enthält die eindeutige Kennung der Aufnahmeserie (innerhalb einer Studie).

(0020,0013) **Image Number** entspricht der (fortlaufenden) Nummer des Bildes in der Aufnahmesequenz der aktuellen Serie. Je nach Aufnahmegerät beginnt diese bei 0 oder bei 1, bei manchen Herstellern gehören auch vorgeschaltete Messbilder zu einer Sequenz.

(0020,0032) **Image Position Patient** definiert die x-, y- und z-Koordinaten der linken oberen Ecke des Bildes (Mittelpunkt des ersten übertragenen Voxels) in mm.

(0020,0037) **Image Orientation Patient** gibt die Richtungsvektoren der ersten Zeile und der ersten Spalte des übertragenen Bildes bezogen auf den Patienten an und definiert damit das Patientenkoordinatensystem.

(0029,0004) **Photometric Interpretation** enthält eine Stringdarstellung der photometrischen Darstellung. So steht MONOCHROME1 bspw. für eine Grauwertdarstellung, in der niedrige Werte für weiß stehen. In der öfter verwendeten MONOCHROME2-Interpretation ist dies genau umgekehrt.

(0028,0010) **Rows** gibt die Bildhöhe in Pixeln an.

(0028,0011) **Columns** enthält die Bildbreite in Pixeln.

(0028,0030) **Pixel Spacing** entspricht dem Abstand zwischen zwei Pixeln in (Bild-) x- und y-Richtung, angegeben in mm.

(0028,0100) **Bits Allocated** ist die Anzahl Bits, die pro Pixelwert reserviert werden.

(0028,0101) **Bits Stored** gibt die Anzahl der reservierten Bits an, die tatsächlich für Pixeldaten verwendet werden.

(0028,0102) **High Bit** ist das höchstwertigste Bit.

(0028,0103) **Pixel Representation** gibt an, ob es sich bei den Pixeldaten um eine Unsigned-Integer- (0) oder um eine 2-Komplement-Darstellung handelt (1).

(0028,0120) **Pixel Padding Value** ist ein definierter Wert für Pixel, die außerhalb des Scanbereichs liegen. Übliche CT-Scanner nehmen bspw. ein kreisförmiges Schnittbild auf, alle Pixel außerhalb erhalten den hier angegebenen Wert.

(0028,1050) **Window Center** und (0028,1051) **Window Width** definieren die Transferfunktion für die Umrechnung der Intensitätswerte in Grauwerte für die Darstellung auf einem Computermonitor.

(0028,1052) **Rescale Intercept** und (0028,1053) **Rescale Slope** sind bei CT Daten die Parameter für die Umrechnung der Intensitätswerte in die Hounsfield Skala.

(0028,1053) **Rescale Type** gibt die Einheit der Werte an, nachdem diese mit Rescale Intercept und Slope modifiziert wurden. Nach dem Standard gültige Einheiten sind OD (Optical Density), HU (Hounsfield Units, nur bei CT) und US (unspecified). Weitere Angaben sind möglich, werden aber nicht durch den Standard spezifiziert (siehe [57], Part 3, Anhang C.11.1.1.2).

(7FE0,0010) **Pixel Data** enthält die eigentlichen Pixeldaten.

Diese Aufzählung ist nicht vollständig, sollte aber zumindest für die Implementation eines einfachen Parsers und Bilddateninterpreters ausreichen.

3.1.6. DICOM in der Praxis

Der DICOM Standard ist sehr detailliert und dadurch auch sehr umfangreich. Er besteht in der aktuellen Version aus 16 Teilen[1] mit insgesamt 3592 Seiten.

In der Praxis hat das oft zur Folge, dass PACS-Entwickler, die mit DICOM Daten arbeiten müssen, den Standard nur teilweise implementieren. So gilt es beispielsweise als gängiges Verfahren, die Transfer Syntax UID unberücksichtigt zu lassen und anhand der ersten beiden Byte nach der Kennung abzuschätzen, ob es sich um eine explizite oder implizite VR-Darstellung handelt. Eine weitere, häufig auftretende Verletzung des Standards findet sich z.B. in vielen anonymisierten Datensätzen, bei denen die patientenbezogenen Angaben entfernt oder mit Zufallsangaben überschrieben werden, ohne die Group Length der zugehörigen Gruppe anzupassen.

Auch von den Herstellern bildgebender Geräte wird der Standard an vielen Stellen unterschiedlich interpretiert, was zu zahlreichen Fallunterscheidungen bei der Entwicklung eines Parsers von DICOM-Daten führt. Zwar könnte man sich hier auf den Standpunkt stellen, dass ein Parser nur standardkonforme Daten verarbeiten soll. Bei vielen Anwendern in der Praxis würde dies aber nur schwerlich auf breite Zustimmung stoßen, da diese in der ersten Linie ein Interesse daran haben, „ihre" Datensätze öffnen zu können. Da ein zu toleranter Parser wiederum die Aufweichung des Standards unterstützt, ist jede Entwicklung hier zwangsläufig eine Gratwanderung.

Zusammengefasst lässt sich festhalten, das jeder neu entwickelte Parser mit einer Vielzahl von Datensätzen aus unterschiedlichen Geräten getestet werden muss, um als praxistauglich zu gelten. Selbst dann gibt es oft Überraschungen, bspw. gibt es einige MRT-Scanner, die ein zum restlichen Datensatz orthogonales Kalibrierungsbild (sogenannte *scout images*) mit gleicher Series- und Study-UID aufnehmen. Da dies oft das erste Bild einer Serie ist, muss gerade dieser spezielle Fall gesondert behandelt werden, um das zur Datenhaltung benötigte Volumen nicht mit falschen Dimensionen zu reservieren.

3.2. Hounsfield Skala

Die Hounsfield-Skala wurde als Normierung für die Unterscheidung von Geweben anhand ihrer Absorptionseigenschaft von Röntgenstrahlung – insbesondere in der Computertomographie – eingeführt (siehe z.B. [9], S. 404). Bei einer CT-Aufnahme wird der Patient in einer Röntgenröhre liegend von einem schmalen Röntgenstrahlenfächer durchstrahlt, wie in Kapitel 2.1 beschrieben. Gegenüberliegende Detektoren messen die Schwächung der

[1] Die Bezeichnung der einzelnen Teile mit Part 1-18 kann hier täuschen, da die Teile 9 und 13 nicht mehr im Standard enthalten sind.

Strahlung. Aus den Schwächungen in vielen verschiedenen Richtungen lassen sich Schwächungswerte für einzelne Raumpunkte berechnen, die oft fälschlicherweise als Dichtewerte bezeichnet werden.

3.2.1. Definition

Die Medizin interessiert sich im Rahmen der Computertomografie besonders dafür, die Schwächung der Röntgenstrahlung bestimmten Geweben zuzuordnen. Der englische Nobelpreisträger Godfrey Hounsfield – der als einer der Väter der Computertomographie gilt – hat eine Skala auf der Basis des Schwächungswertes von Wasser vorgeschlagen, um unabhängig von der verwendeten Röntgenenergie zu sein. Da die Absorptionswerte der meisten Organe sich nur wenig von dem des Wassers unterscheiden, wird die Abweichung in der Hounsfield Skala in Promille angegeben.

Der lineare Absorptionskoeffizient μ beschreibt, wie stark die monochromatische Röntgenstrahlung beim Durchdringen von Materie entlang des durchstrahlten Wegs abgeschwächt wird. Für ein bestimmtes Material bzw. Gewebe erfolgt dann die Umrechnung in die Hounsfield Skala wie folgt:

$$H(\mu_{Gewebe}) := \frac{\mu_{Gewebe} - \mu_{Wasser}}{\mu_{Wasser}} * 1000$$

Die Einheiten der Skala werden als Hounsfield Units (HU) bzw. als Hounsfield Einheiten (HE) bezeichnet. Einige Beispiele für HU Zahlen ausgewählter Gewebe zeigt Tabelle 3.1. So absorbiert bspw. Luft Röntgenstrahlung nahezu gar nicht und hat eine CT-Zahl von -1000. Wasser hat gemäß der Definition eine HU von 0. Die Skala ist theoretisch nach oben offen. In der (medizinischen) Praxis trifft man jedoch auf Werte im Bereich von -1024 HU bis 3071 HU, die sich als 12-Bit-Zahlen darstellen lassen. Andere Materialien wie z.B. Metalle können jedoch eine deutlich stärkere Absorption bewirken.

Gewebe	**HU**
Luft	-1000
Fettgewebe	ca. -100
Wasser	0
Muskel	ca. 40
Knochen	ca. 500 bis 1500

Tabelle 3.1.: Hounsfield-Werte ausgewählter Gewebe und Stoffe

Da Computertomographen in der Realität keine monochromatische Strahlung erzeugen, ist die Definition der Hounsfield-Skala zum Teil umstritten. Da die Gewebeübergänge allerdings nicht exakt, sondern in Intervallen angegeben werden, ist sie jedoch praktisch von Nutzen, um die unterschiedlichen Gewebe zumindest grob einteilen zu können.

3.2.2. Anwendung für CT Daten im DICOM Format

Für die Umrechnung der Intensitätswerte aus Bilddaten - nicht nur bei CT - werden die Datenelemente Rescale Slope m und Rescale Intercept b verwendet. Nach [57], PS 3, Anhang C.11, Seite 876 gilt dabei die Formel:

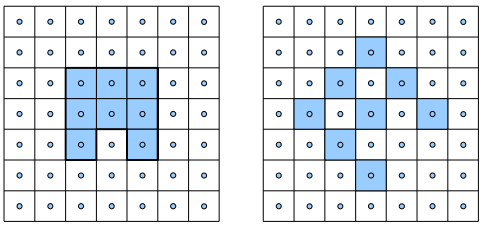
$$OutputUnit = m \cdot SV + b$$

SV steht für „stored value", also dem in der Datei gespeicherten Pixelwert. Das Ergebnis, als „Output Unit" bezeichnet, kann je nach Art der Daten unterschiedlich interpretiert werden. Im Fall von CT-Daten ergibt sich nach der Umrechnung der Hounsfield Wert.

3.3. Grid Daten

Die bildgebenden Verfahren der Medizin liefern in der Regel äquidistant abgetastete Intensitätswerte, die sich in einer Matrixform speichern lassen. Die Verallgemeinerung dieser Struktur wird als Grid-Daten bezeichnet.

Eine fundamentale Abhandlung des theoretischen Hintergrunds von (ein- und zweidimensionalen) Grid-Daten findet man in [74]. Dort wird zunächst der Begriff des Grids selbst definiert, das aus einer Teilmenge Y (Punkte) des $\mathbb{Z}^1$ oder $\mathbb{Z}^2$ und einer Teilmenge $D = Y \times Y$ (Kanten) besteht. D beschreibt somit die Nachbarschaftsbeziehung bzw. Konnektivität im Grid, so dass dieser Begriff als Spezialfall eines Graph aufgefasst werden kann.



(a) Einfaches Gebiet (b) Rotation um 45°

Abbildung 3.4.: Morphologische Probleme bei einfachen Operationen *Quelle: [74], S. 173*

Durch diese Bedingungen werden gewisse morphologische Probleme verhindert, die entstehen, wenn ein diskretes Bild als 2D-Matrix ohne explizit angegebene Konnektivität aufgefasst wird. Abbildung 3.4 demonstriert dies an einem einfachen, zweidimensionalem Beispiel. Betrachtet man das Gebiet aus a), erscheinen Zusammenhang als auch Kontur

intuitiv eindeutig zu sein. Nach einer Rotation von 45° um den Mittelpunkt ist die Situation bereits unklar; enthält das rotierte Gebiet b) zwei eingeschlossene Löcher – oder nicht? Zusätzlich wird daher gefordert, dass es

1. eine Untergruppe von Translationen in mindestens zwei Richtungen geben muss, für die Y und D invariant sind (t-Invarianz-Bedingung) und
2. für je zwei Kanten (y_1, y_2) und (y_3, y_4) gilt, das diese sich nicht überschneiden (non-crossing Bedingung).

In dieser Arbeit wird diese strenge Grid Definition selten gebraucht. Stattdessen soll hier eine etwas einfachere Definition genügen. Dennoch ist es hilfreich, sich mit der etwas abstrakteren Theorie auseinanderzusetzen, da sie in vielen praktische Anwendungsfällen ein tieferes Problemverständnis ermöglicht.

Unter der Bezeichnung „Grid-Daten" soll im Folgenden jede Form von Daten, die sich durch eine d-dimensionale Wertematrix darstellen lassen, verstanden werden. Der Nachbarschaftsbegriff ist dabei nicht Teil der Grid-Daten-Definition selbst, sondern kann je nach Situation anders ausgelegt werden. Damit sind Grid-Daten eine Verallgemeinerung des Konzeptes von Pixel-Daten (2D-Matrix) oder Voxel-Daten (3D-Matrix). Grid-Daten sind eine der fundamentalen Datenstrukturen für diese Arbeit. Dabei kann davon ausgegangen werden, dass der Abstand zwischen zwei Punkten des Grids in jeder Koordinatenrichtung gleich bleibt. In der Literatur wird hier von einem regulärem Grid gesprochen.

Mathematisch gesehen wird ein Grid-Daten Objekt repräsentiert durch eine Funktion

$$G : V \to W$$

wobei V ein endliches Intervall aus $\mathbb{N}^d$ mit $d \in \mathbb{N}_{>0}$ ist. d wird als Dimension des Grids bezeichnet, W repräsentiert den Werteraum der betrachteten Daten, also z.B. Farb-, Dichte- oder Absorptionswerte. Die entsprechende Matrix wird bei der Implementierung üblicherweise durch ein mehrdimensionales Array repräsentiert. Damit sind Grid-Daten auch eine Verallgemeinerung des Begriffs des Skalar-Feldes, bei dem der Werteraum (wie der Name schon andeutet) aus Skalaren besteht.

Die populärsten Vertreter von Grid-Daten sind von den Dimensionen 2 und 3. In der Computergraphik werden zweidimensionale Grid Daten oft als Rastergrafiken oder Bilder bezeichnet. Der Werteraum ist hier meist eine Farbe, üblicherweise in RGB-Repräsentation. Die Einträge dieser zweidimensionalen Matrix werden als **Pixel** (**pi**cture matri**x** **el**ement) bezeichnet. Ein einzelnes Schichtbild aus einer Serie von Schnittbildaufnahmen lässt sich auch als Rastergrafik mit Intensitätswerten anstelle von Farben auffassen.

Das dreidimensionale Analogon eines Pixels wird oft als **Voxel** (von **vo**lumetric pi**xel** oder Volumenpixel) bezeichnet. Ein Voxel beschreibt eine Einheit in einem räumlichen

Datensatz, der in diskreter Form in karthesichen Koordinaten vorliegt, so dass in O(1) der Wert des Grids an der Stelle (x, y, z) mit $x, y, z \in \mathbb{N}$ abgerufen werden kann. Eine Sequenz von Schnittbildaufnahmen lässt sich auch als 3D-Grid-Datenobjekt auffassen, bei der jedes Schnittbild genau einer Schicht im Grid entspricht. Ein weiteres Beispiel für den Einsatz von 3D-Griddaten ist die schnelle Kollisionserkennung: Soll eine große Dreiecksmenge auf Kollision mit einem oder mehreren Testpunkten überprüft werden, wird in einigen Verfahren einmalig eine Voxelstruktur angelegt, die eine gleichmäßige Unterteilung der Bounding Box der Dreiecksfläche repräsentiert. Der Werteraum ist in diesem Fall die Menge der Dreiecke, d.h. in jedem Gridpunkt werden Referenzen auf die in der Box liegenden Dreiecke abgelegt.

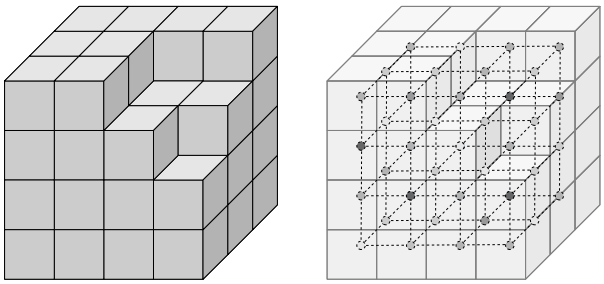


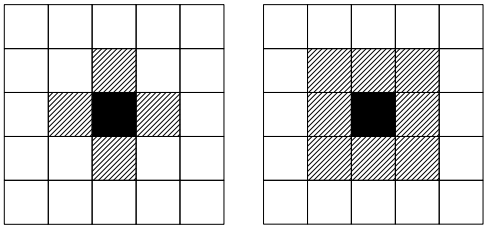
Abbildung 3.5.: Voxel - Kleine Würfel oder Wert des Mittelpunkts einer Zelle?

Konkret: Ein Pixel (x, y, w) bzw. Voxel (x, y, z, w) mit $(x, y, z \in N)$ bezeichnet in dieser Arbeit nur den (verorteten) Eintrag in der internen Wertematrix. Wie dieser Wert räumlich zu interpretieren ist, kann von Fall zu Fall unterschiedlich sein. Für viele Anwendungen reicht die gebräuchliche Vorstellung, ein Pixel entspräche einem kleinen Rechteck und ein Voxel einem kleinen Quader, bspw. beim texturbasierten Direct Volume Rendering. In anderen Fällen wird darunter nur der Wert an einem einzelnen Raumpunkt (dem Mittelpunkt des Rechtecks bzw. Quaders) verstanden[1], wie z.B. bei der Interpolation innerhalb einer Zelle im Volume Raycasting. Abbildung 3.5 skizziert diese beiden gängigen Vorstellungen.

3.3.1. Nachbarschaft

Vor allem im Bereich der Segmentierung wird von der Nachbarschaft (oder Konnektivität nach Serra [74]) eines Pixels bzw. Voxels gesprochen. In der klassischen 2D-Bildanalyse gibt es zwei unterschiedliche Definitionen für diesen Begriff, wie in Abbildung 3.6 skizziert:

[1] Über die grundsätzliche Frage, was ein Pixel eigentlich genau ist, gibt es tatsächlich einige sehr interessante (und zum Teil emotional geführte) Diskussionen. Einen lesenswerten Einstieg in diese Thematik bietet z.B. der Artikel „What is a Pixel" von Jim Blinn [4] und das Microsoft Technical Memo von Alvy Ray Smith mit dem schönen Titel „A Pixel Is Not A Little Square, A Pixel Is Not A Little Square, A Pixel Is Not A Little Square! (And a Voxel is Not a Little Cube)" [77]



(a) 4er Nachbarschaft (b) 8er Nachbarschaft

Abbildung 3.6.: Nachbarschaftsbegriff für diskrete 2D-Grid-Daten

- **4er Nachbarschaft**
  Als direkte Nachbarn eines Pixels werden nur die horizontal bzw. vertikal angrenzenden Pixel betrachtet (Abbildung 3.6 a).
- **8er Nachbarschaft**
  Zusätzlich zur 4er Nachbarschaft werden auch die diagonal angrenzenden Pixel als direkte Nachbarn gezählt (Abbildung 3.6 b).

Überträgt man diese Begriffe von einem 2D-Pixel-Bild auf einen 3D-Voxel Cube, lassen sich diese Begriffe auf die **6er** bzw. **26er Nachbarschaft** erweitern. Im Folgenden bezeichnen $N_6(v)$ bzw. $N_{26}(v)$ die entsprechenden Nachbarschaftsmengen des Voxels v.

3.3.2. Voxel Cube

Eine für die Entwicklung eines 3D-DICOM Visualisierungsprogramms zentrale Variante einer Voxeldatenstruktur ist der sogenannte Voxel Cube. Der Werteraum ist hier ganzzahlig (integer). Der Voxel Cube wird dafür verwendet, die Werte der Pixel aus den einzelnen Schichtbildern abzuspeichern und in einer räumlichen Datenstruktur abzulegen. Diese Pixelwerte sind im MONOCHROME2-Format immer ganzzahlig und müssen für die Visualisierung zuerst in einen RGB-Wert umgewandelt werden.

In der Terminologie der Informatik ist ein Voxel Cube ein dreidimensionales Array mit fester Größe. Der Zugriff auf einen Voxel ist $O(1)$. Aus Effizienzgründen kann die interne Repräsentation durch ein eindimensionales Feld erfolgen, in dem die Werte aufsteigend nach Schicht und Zeile angeordnet werden.

3.3.3. Bit Cube

Wie beim Voxel Cube handelt es sich auch beim Bit Cube um eine dreidimensionale Matrix, allerdings ist der Werteraum in diesem Fall die Menge $\{0, 1\}$. Verwendet wird ein

solcher Bit Cube beispielsweise in einer Instanz eines Segments, die für jeden Voxel die Segmentzugehörigkeit abspeichert.

Durch diese Form der Datenhaltung kann die Segmentzugehörigkeit eines Voxels ebenfalls in $O(1)$ festgestellt werden. Durch die hohe Speichereffizienz bei gleichzeitigem schnellen Zugriff eignet sich die Struktur jedoch auch als temporärer Markierungsspeicher, bspw. um während eines Region Grow-Verfahrens festzuhalten, welche Voxel bereits in der Queue der noch zu bearbeitenden Nachbarn enthalten sind.

3.3.4. Interpolation und Rekonstruktion

Für viele Anwendungen in dieser Arbeit, bspw. die hochqualitative Visualisierung, ist es notwendig, die vorliegenden diskreten (2D- oder 3D-) Daten in eine kontinuierliche Form zu bringen. Dabei wird davon ausgegangen, dass die der Aufnahme zugrundeliegenden Strukturen selbst von kontinuierlicher Natur sind und die diskrete Aufnahme nur eine (meist äquidistante) Abtastung darstellt. Um die ursprüngliche Form zu erhalten, muss zwischen den diskreten Messpunkten interpoliert werden. Dafür können unterschiedliche Interpolationsmethoden, in der Literatur auch als **Rekonstruktionsfilter** bezeichnet, verwendet werden. Man spricht daher auch oft von der sogenannten Sub-Voxelauflösung.

Die an den Gridpunkten gespeicherten Daten müssen dabei nicht notwendigerweise Intensitätswerte sein, bspw. werden bei Volume Raycasting Verfahren (siehe Kapitel 5.3.1.3) auch Oberflächennormalen, d.h. dreidimensionale Vektoren, mit den gleichen Methoden interpoliert.

In [44] werden von Marschner und Lobb verschiedene Filterverfahren auf ihre Eignung zur Signalrekonstruktion von dreidimensionalen Volumen aus äquidistant aufgenommenen diskreten Daten untersucht. Die folgenden Filter sind jeweils für einen dreidimensionalen Voxeldatensatz angegeben, d.h. die zu interpolierenden Zellen sind Würfel mit jeweils 8 Eckpunkten (bis auf die Randvoxel). Alle Verfahren lassen sich auch auf zweidimensionale Daten (z.B. einzelne Schichtbilder) anwenden.

3.3.4.1. Nearest Neighbour-Interpolation

Die **Nearest Neighbour-Interpolation**[1] ist bezogen auf äquidistante Grid-Daten ein sehr einfaches Interpolations-Verfahren, bei dem jeder Voxel als ein gefüllter Würfel aufgefasst wird. Entsprechend bekommt jeder Punkt innerhalb einer Zelle den Intensitätswert seines nächsten Nachbarn. Als Vergrößerungsfilter eingesetzt, würde im 2D-Fall jeder Pixel eines Bildes durch ein Quadrat mit dem Vergrößerungsfaktor als Kantenlänge ersetzt

[1] Sehr selten auf deutsch als Nächste-Nachbar-Methode bezeichnet, der englische Begriff hat sich hier etabliert.

werden. Der Nearest Neighbour-Filter ist damit eine (triviale) Variante des euklidischen Voronoi Diagramms.

3.3.4.2. Trilineare Interpolation

Ein schneller und dennoch für viele Anwendungen ausreichender Rekonstruktionsfilter ist die **trilineare Interpolation**, bei der die Werte der Würfeleckpunkte linear entlang der drei Hauptachsenrichtungen interpoliert werden. Die trilineare Interpolation gehört damit zu den sogenannten **separablen Filtern**, die sich als Produkt von drei eindimensionalen Filtern aufschreiben lassen. Abbildung 3.7 zeigt dies in einer Skizze; es wird jeweils zwischen t_{11} und t_{12}, zwischen t_{21} und t_{22} und als letztes zwischen zwischen t_{31} und t_{32} linear interpoliert.

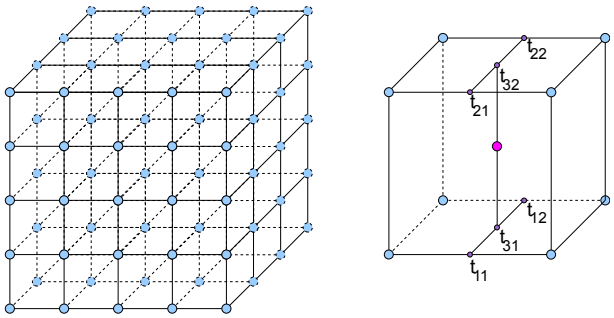


Abbildung 3.7.: Reguläres Voxelgitter, trilineare Interpolation

Eine Kenngröße für Rekonstruktionsfilter ist die Größe des Bereiches, der lokal zur Rekonstruktion verwendet wird, bzw. dessen **Filterradius** r_i. Bei der trilinearen Interpolation, bei der nur die acht umliegenden Voxel berücksichtigt werden, ist der Radius $r_i = 1$. Ein größerer Rekonstruktionsradius erlaubt eine bessere Rekonstruktion, bringt aber auch meist eine aufwendigere Berechnung mit sich.

Die trilineare Interpolation eignet sich besonders für schnelle Preview-Berechnungen. Für qualitativ hochwertige Ergebnisse kommen jedoch meist Methoden mit einem größeren Filterradius zum Einsatz.

3.3.4.3. Kubische Filter

Etwas aufwendiger als die linearen sind die sogenannten **kubischen Filter**, da dabei alle 64 umliegenden Voxel der Zelle berücksichtigt werden. Der Filterradius ist in diesem Fall 2, wodurch sich bessere Ergebnisse erzielen lassen.

Mitchell und Netravali haben in [51] durch die Einführung von Nebenbedingungen wie Stetigkeit und Differenzierbarkeit die Anzahl der freien Parameter der allgemeinen abschnittsweisen kubischen Interpolation auf zwei reduziert und den Filterkern

$$k_s = \frac{1}{6}\begin{cases} (12 - 9B - 6C) \cdot |x|^3 + (-18 + 12B + 6C) \cdot |x|^2 & \\ \quad +(6 - 2B) & \text{falls } |x| < 1 \\ (-B - 6C) \cdot |x|^3 + (6B + 30C) \cdot |x|^2 & \\ \quad +(-12B - 48C) \cdot |x| + (8B + 24C) & \text{falls } 1 \leq |x| < 2 \\ 0 & \text{sonst} \end{cases} \tag{3.1}$$

mit den Parametern B und C vorgestellt, der auch als Mitchell-Netravali-Filter bezeichnet wird. In der gleichen Arbeit wurde weiter gezeigt, dass der Kern, mit den entsprechenden Parametern, bekannten Spline-Funktionen entspricht.

In [44] wurden von Marschner und Lobb die Auswirkungen verschiedener Parametrisierungen in der Volumenrekonstruktion untersucht und optimale Kombinationen zur Vermeidung von Artefakten identifiziert. Als Nebeneffekt lässt sich ein kubischer Filter durch eine entsprechende Parametrisierung auch zum starken Glätten benutzen. Der Effekt ähnelt einem Gaußschen Weichzeichner.

3.3.4.4. sinc-Filter

Eine weitere Interpolationsmethode ist in der Signalverarbeitung als idealer Tiefpass-Filter bekannt und wird dort meist als **sinc-Filter** bezeichnet. Dieses Verfahren ist jedoch sehr rechenaufwendig.

In [44] wird gezeigt, dass mit einem Produkt aus sinc-Funktion – der inversen Fourier-Transformation eines Würfels – ein idealer Rekonstruktionsfilter erhalten werden kann. Da für eine reale Implementierung aber nur endlich große Fenster verwendet werden können, wird dort vorgeschlagen, die sinc-Funktion mit einer Fenster-Funktion zu multiplizieren, um Artefakte durch Abschneiden der sinc-Funktion am Rande des Fenster zu vermeiden. Damit ergibt sich der folgende *windowed sinc*-Filterkern:

$$k_s(x) = (1 + \cos(\pi \cdot \frac{x}{x_m}))\,\mathrm{sinc}(4 \cdot \frac{x}{x_m}) \quad \text{für } |x| < x_m \tag{3.2}$$

Im Rekonstruktionsfilterpaket von YaDiV wurde zu Vergleichszwecken auch ein sinc-Filter implementiert. Aufgrund der aufwendigen Berechnung und einer kaum wahrnehmbaren optischen Verbesserung wird dieser jedoch eher selten verwendet.

3.3.4.5. Gemeinsamkeiten und Vergleich

Ein praktisches Problem, das allen Filtermethoden gemeinsam ist, ist das Verhalten am Rand der diskreten Daten, da hier die Nachbarschaft nicht vollständig im Bereich des erfassten Volumens liegt. Hier ist neben der Einigung auf eine Randbedingung („was liegt außerhalb?“) auch eine algorithmische Sonderfallbehandlung notwendig. Da die Filter in allen damit verbundenen Anwendungen meist sehr oft aufgerufen werden, ist darauf zu achten, dass diese Sonderfallbehandlung nicht die Effizienz des (wesentlich häufiger auftretenden) Falls der vollständig im Volumen-Inneren liegenden Zelle beeinträchtigt.

(a) neighbour (b) bilinear (c) bicubic 1 (d) bicubic 2 (e) sinc

Abbildung 3.8.: Rekonstruktions-Filter Vergleich *Quelle: [52]*

Abbildung 3.8 zeigt die unterschiedlichen Filterverfahren im direkten Vergleich. Dabei wurden die für YaDiV implementierten Rekonstruktionsfilter auf eine 2D-Pixelgrafik (32x32) angewendet. Bild a) zeigt eine Vergrößerung des Ausgangsbildes mit der Nearest Neighbour Methode, die bilineare Interpolation b) wirkt bereits deutlich glatter. Bild c) und d) wurden beide mit einem bikubischem Filter erzeugt, mit den Parametern ($B = 0.4$, $C = 0.3$) bzw. ($B = 1.5$, $C = -0.25$). An Bild d) lässt sich gut erkennen, dass der Filter auch zur starken Glättung bzw. als Weichzeichner eingesetzt werden kann. Das letzte Bild e) wurde mit dem sinc-Filter (Radius 2) erzeugt.

3.3.5. Gradienten

Als Gradient bezeichnet man einen Differentialoperator, der auf ein Skalarfeld angewandt werden kann. Das Ergebnis ist ein Vektorfeld, das Änderungsrate und Richtung der größten Änderung des Skalarfeldes angibt. Der Gradient an einem Punkt im Skalarfeld ist ein Vektor, der in Richtung des größten Anstieges des Skalarfeldes zeigt. Der Betrag des Vektors entspricht dabei der Stärke des Anstieges. Der Gradient an einem lokalen Extremum oder einem Sattelpunkt entspricht dem Nullvektor (der Rand des Skalarfeldes bildet hier eine Ausnahme).

3.3.5.1. Definition und Anwendungsgebiete

In der Mathematik ist der Gradient klassischerweise für reellwertige Funktionen definiert. Für eine kontinuierliche, partiell differenzierbare Funktion $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$ liefert die bekannte Form

$$f'(x) = \lim_{|\alpha| \rightarrow 0} \frac{f(x+\alpha) - f(x)}{|\alpha|}$$

deren lokale Ableitung in Richtung α. Setzt man bei einem gegebenen Skalarfeld eine solche Funktion als zugrunde liegend voraus, lässt sich die lokale Ableitung abschätzen. Die Anwendung des Nabla-Operators ∇ auf ein Skalarfeld φ liefert dessen Gradienten. Für den $\mathbb{R}^3$ ergibt sich laut Bronstein die Definition ([6] S. 669ff.):

$$\text{grad } \varphi = \nabla\varphi = \left(\frac{\delta\varphi}{\delta x}, \frac{\delta\varphi}{\delta y}, \frac{\delta\varphi}{\delta z}\right) \tag{3.3}$$

Der Gradient beschreibt damit einen dreidimensionalen Vektor, dessen Komponenten gerade aus den partiellen Ableitungen bestehen. Geometrisch interpretiert entspricht die Richtung von grad φ im Punkt P der Richtung des Normalenvektors $\vec{n}$ der Niveaufläche E in P mit $\varphi(P) = \varphi(E) = const$ und ist in Richtung steigender Werte von φ orientiert. Auch für die Gradienten lässt sich eine Subvoxelauflösung (die Interpolation des Gradienten für einen Punkt im Inneren einer durch das Skalarfeld definierten Zelle) durch Verwendung eines Rekonstruktionsfilters (siehe vorheriges Kapitel) erzielen.

In der klassischen Bildverarbeitung wird der Gradient meist zur Kantendetektion verwendet. In der medizinischen 3D-Volumendatenverarbeitung findet der Gradient sowohl in der Visualisierung als auch in der Segmentierung Anwendung, z.B. für die Erzeugung von Normalen für eine schattierte Darstellung oder die Optimierung von Kontrollpunkten in einem nichtlinearen Registrierungsschritt (Kapitel 4.5.3.2. S.74). Neben der Bildverarbeitung finden Gradienten vor allem in den Ingenieurswissenschaften Anwendung, bspw. in der Strömungsmechanik, dort wird der Operator von Skalar- auch auf Vektorfelder erweitert.

3.3.5.2. Finite Differenzen

Um den Gradient eines Skalarfeldes abzuschätzen, gibt es unterschiedliche Verfahren, die sich in Rechenaufwand und Genauigkeit unterscheiden. Die einfachste, aber auch ungenaueste Methodik ist die Verwendung der sogenannten **Vorwärts-** bzw. **Rückwärts-Differenzen**, die den Skalarwert $\varphi(P) = \varphi(p_0, \ldots, p_n)$ eines n-dimensionalen Skalarfeld φ mit seinem direkten Vorgänger bzw. Nachfolger vergleicht:

$$(\text{grad } P)_{vorwärts} \approx \begin{pmatrix} \varphi(p_0+1,p_1,p_2,\ldots,p_n) - \varphi(p_0,p_1,p_2,\ldots,p_n) \\ \varphi(p_0,p_1+1,p_2,\ldots,p_n) - \varphi(p_0,p_1,p_2,\ldots,p_n) \\ \vdots \\ \varphi(p_0,p_1,p_2,\ldots,p_n+1) - \varphi(p_0,p_1,p_2,\ldots,p_n) \end{pmatrix} \tag{3.4}$$

bzw.

$$(\text{grad } P)_{rückwärts} \approx \begin{pmatrix} \varphi(p_0,p_1,p_2,\ldots,p_n) - \varphi(p_0-1,p_1,p_2,\ldots,p_n) \\ \varphi(p_0,p_1,p_2,\ldots,p_n) - \varphi(p_0,p_1-1,p_2,\ldots,p_n) \\ \vdots \\ \varphi(p_0,p_1,p_2,\ldots,p_n) - \varphi(p_0,p_1,p_2,\ldots,p_n-1) \end{pmatrix} \tag{3.5}$$

Die vermutlich am häufigsten verwendete Methode zur Berechnung des Gradienten eines Skalarfeldes ist jedoch die Verwendung der **zentrale Differenzen**, die den Gradient an einem Punkt durch die Differenz seiner nächsten Nachbarn in den Hauptrichtungen berechnet:

$$(\text{grad } P)_{zentral} \approx \begin{pmatrix} \frac{1}{2}\Big(\varphi(p_0+1,p_1,p_2,\ldots,p_n) - \varphi(p_0-1,p_1,p_2,\ldots,p_n)\Big) \\ \frac{1}{2}\Big(\varphi(p_0,p_1+1,p_2,\ldots,p_n) - \varphi(p_0,p_1-1,p_2,\ldots,p_n)\Big) \\ \vdots \\ \frac{1}{2}\Big(\varphi(p_0,p_1,p_2,\ldots,p_n+1) - \varphi(p_0,p_1,p_2,\ldots,p_n-1)\Big) \end{pmatrix} \tag{3.6}$$

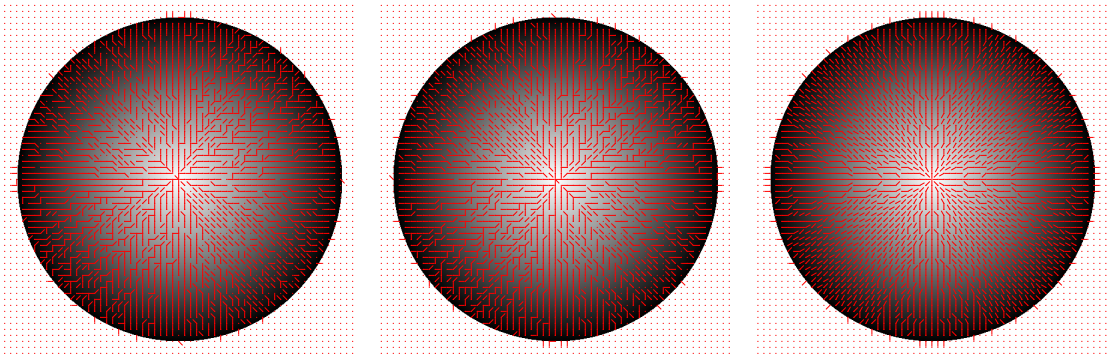
Aufgrund des guten Kompromisses zwischen Berechnungskosten und Qualität der Abschätzung wird auch in YaDiV meist die zentrale Differenzenmethode verwendet, bspw. im Raycasting-Modul.

3.3.5.3. Vergleich

Abbildung 3.9 zeigt einen optischen Vergleich der Approximationsverfahren, zum besseren Verständnis angewendet auf ein 2D-Skalarfeld; hohe Intensitäten werden im Bild durch einen dunkleren Grauwert visualisiert. Die (normierten!) Gradienten sind als rote Pfeile dargestellt. Besonders im Bereich mit starken Intensitätswechsel wirkt das zentrale Differenzenverfahren zumindest optisch überzeugender als die Vor- bzw. Rückwärtdifferenzenmethoden.

Vom Aufwand her sind Vorwärts- und Rückwärtsdifferenzen gleich; Für jeden Punkt im Skalarfeld werden zwei Lookup-Operationen pro Dimension (also sechs bei einem 3D-Volumendatensatz) und eine Differenz-Operation benötigt. Sollen alle Gradienten eines

diskreten Volumendatensatzes berechnet werden, lassen sich die Lookup Operationen in beiden Fällen um die Hälfte reduzieren. Die Approximationsgüte ist allerdings begrenzt. Ein etwas besseres Ergebnis erhält man durch die Verwendung der zentralen Differenzen, hier wird pro Punkt eine zusätzliche Division nötig.



(a) Vorwärts (b) Rückwärts (c) Zentral

Abbildung 3.9.: Vergleich der unterschiedlichen Gradientenverfahren

3.3.5.4. Lineare Regression

Die bisher vorgestellten Verfahren arbeiten einerseits sehr schnell, liefern jedoch andererseits eine relativ grobe Approximation des Gradienten. Eine genauere, aber auch aufwendigere Alternative wurde von Neumann et al. in [58] vorgestellt. Das dort vorgeschlagene Verfahren approximiert die Intensitätsfunktion selbst durch eine dreidimensionale Hyperebene mittels linearer Regression, bei der die 26er-Nachbarschaft jedes Punktes berücksichtigt wird. Das Verfahren erhöht die Qualität der Gradienten, ist jedoch erheblich rechenaufwändiger. Diese Methode wurde im Rahmen des Ray Casting-Moduls (siehe auch Kapitel 5.3.1.3 bzw. 7.5.10) ebenfalls implementiert. Es hat sich jedoch in der Praxis gezeigt, dass der Einfluss der Sampling-Rate zumindest im Ray Casting-Verfahren einen wesentlich höheren Einfluss auf die Qualität der Darstellung hat als die Wahl des Gradientenverfahrens, daher wurde dieser Ansatz vorerst nicht weiterführend untersucht.

3.3.6. Ableitung höherer Ordnung

In vielen Anwendungen werden auch die höherwertigen Ableitungen der reellwertigen Intensitätsfunktion φ benötigt. Aufbauend auf der Definition des Gradienten, lässt sich z.B. die Krümmung eines Punktes x_0 als

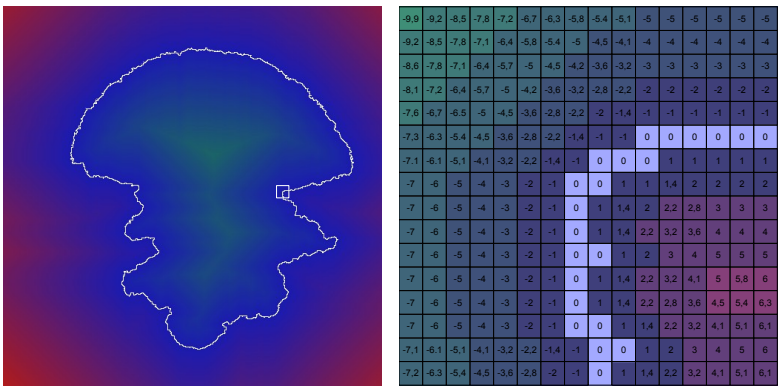
$$k(x_0) = div\left(\frac{\nabla\varphi(x_0)}{|\nabla\varphi(x_0)|}\right) \tag{3.7}$$

definieren. Die dafür benötigten zweiten partiellen Ableitungen werden im diskreten Fall oft durch die Hintereinanderausführung der diskreten Differenzen approximiert. So ergibt sich bspw. für die zentrale Differenzenmethode

$$\begin{aligned} \nabla^{xx}\varphi_{x,y,z} &= \frac{1}{4}\left(\varphi_{x+2,y,z} - \varphi_{x-2,y,z} - 2\varphi_{x,y,z}\right) \text{ und} \\ \nabla^{xy}\varphi_{x,y,z} &= \frac{1}{4}\left(\varphi_{x+1,y+1,z} - \varphi_{x+1,y-1,z} - \varphi_{x-1,y+1,z} + \varphi_{x-1,y-1,z}\right) \end{aligned}$$

3.3.7. Distanzfelder

Distanzfelder sind eine diskrete Darstellung einer Distanzfunktion. Je nach Anwendung kann diese die klassische euklidische Distanz, deren Quadrat (schneller und für Vergleiche ausreichend) oder eine vorzeichenbehaftete Funktion sein, bei der z.B. Voxel im Segmentinneren eine negative Distanz zur Kontur erhalten. Abbildung 3.10 zeigt ein zweidimensionales Beispiel für eine vorzeichenbehaftete Distanzfunktion eines Segments. Positive Distanzen zur Kontur werden durch eine rote, negative durch eine grüne Färbung dargestellt, eine blaue Färbung entspricht einer Distanz nahe bei 0.



(a) Distanzfunktion (b) Ausschnitt

Abbildung 3.10.: Vorzeichenbehaftete Distanzfunktion

Allgemein formuliert gibt ein Distanzfeld für einen diskreten Volumensatz V für jedes Element die Distanz zu einer Menge M an, die je nach Anwendung eine Kontur, ein Punkt oder ein Volumenkörper sein kann. Dabei muss beachtet werden, dass bei aufgenommenen Daten die realen x-, y- bzw. z-Abstände zwischen zwei Punkten unterschiedlich sein können (im DICOM-Standard bspw. durch *Pixel Spacing* und *Slice Thickness/Patient Position* definiert, siehe Kapitel 3.1.5, S. 27).

Der Aufwand zur Berechnung eines solchen Distanzfeldes ist damit im Allgemeinen $O(|M| * |V|)$, da für jeden Voxel aus V der nächstliegende Punkt aus M gesucht wird. Je nach An-

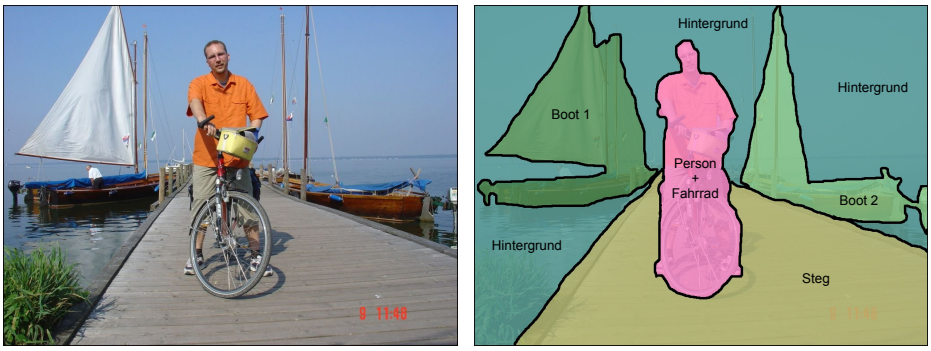
wendung lässt sich der Aufwand reduzieren, bspw. durch ein **Narrow-Band-Verfahren**. Diese Technik wird eingesetzt, wenn man nur an den Distanzen in einer gewissen Umgebung zur Referenzmenge (hier: der Kontur) interessiert ist. Dabei wird um jeden Voxel aus M eine Kugel mit Radius r (oft auch Würfel mit Kantenlänge $2r$) betrachtet und nur für diesen Narrow-Band-Bereich das Distanzfeld initialisiert. Nachteil des Verfahrens ist, dass auch der Aufbau des Narrow-Bandes Zeit kostet und bei sehr großen, komplexen Referenzmengen letztlich mehr Zeit verbraucht, als die direkte Berechnung.

Distanzfelder bilden ein wichtiges und universelles Werkzeug in der medizinischen 3D-Datenverarbeitung. Anwendung finden sie in sehr unterschiedlichen Teilgebieten, z.B. Moving-Contour-Verfahren (Kapitel 4.4), bei denen der Abstand eines jeden Voxels zur Kontur eines Segments benötigt wird, aber ebenso in der schnellen Kollisionserkennung für die haptische Rückkopplung (Kapitel 6.4.3).

Die interne Repräsentation eines Distanzfeldes der des Voxel Cube sehr ähnlich; auch hier handelt es sich um dreidimensionale Grid-Daten, deren Wertebereich allerdings üblicherweise nicht mehr ganzzahlig ist.

4. Segmentierung

In der medizinischen (3D-)Bildverarbeitung wird häufig der Begriff der Segmentierung verwendet. Darunter versteht man die Identifikation von Regionen aus Pixeln bzw. Voxeln, die bestimmte organische oder anatomische Strukturen abbilden. Eine solche (nicht notwendigerweise geometrisch zusammenhängende) Region wird als **Segment** bezeichnet. Abbildung 4.1 zeigt ein Beispiel für Segmente in der klassischen 2D-Bildverarbeitung: Während das Bild selbst aus reinen Farbinformationen besteht, werden durch die Segmentierung zusätzliche Meta-Informationen wie „Boot" oder „Hintergrund" erfasst.



(a) Bild (b) Mögliche Segmente

Abbildung 4.1.: Segmentierung in der Bildverarbeitung

Im medizinischen Bereich kann ein Segment zum Beispiel aus einem (gesunden oder pathologisch veränderten) Organ bestehen, dessen Form oder Volumen für die Diagnose benötigt wird. Ein anderes Beispiel wäre ein komplizierter Trümmerbruch, dessen einzelne Bruchstücke zuerst segmentiert und dann reponiert werden sollen.

4.1. Begriffe und Konzepte

Je nach Kontext hat das Wort Segmentierung in der Literatur zwei unterschiedliche Bedeutungen. Der Begriff bezeichnet sowohl den Prozess der Segmenterzeugung, als auch eine aus mehreren Segmenten bestehende Menge. Wenn unter dem Begriff Segmentierung eine Menge von Segmenten verstanden wird, so lässt sich diese Menge gemäß ihrer Eigenschaften weiter klassifizieren.

- Eine Segmentierung wird als **vollständig** bezeichnet, wenn jedem Pixel mindestens ein Segment zugeordnet wurde.
- Sie heißt **überschneidungsfrei**, wenn jeder Pixel in höchstens einem Segment liegt.
- Ein Segment heißt **zusammenhängend**, wenn die Menge der zum Segment gehörenden Voxel ein zusammenhängendes Gebiet definiert. Eine Segmentierung heißt **zusammenhängend** wenn diese Eigenschaft auf jedes zur Segmentierung gehörende Segment zutrifft.

Abbildung 4.1 b) ist somit ein Beispiel für eine vollständige, überschneidungsfreie, aber nicht zusammenhängende Segmentierung, da das Segment „Hintergrund" in mehrere Zusammenhangskomponenten zerfällt.

Historisch gesehen ist die Segmentierung ein Teilgebiet der digitalen Bildverarbeitung und des maschinellen Sehens. Segmentierung steht dort für die Erzeugung von inhaltlich zusammenhängenden Regionen in einer aufgenommenen Szene. Traditionell für 2D-Daten entwickelt, lassen sich die meisten Begriffe und Konzepte auch auf den 3D-Fall übertragen. In der Prozesskette des maschinellen Sehens erfolgt die Segmentierung an dritter Stelle:

(Szene) → Bildaufnahme → Bildvorverarbeitung → **Segmentierung**
→ Merkmalsextraktion → Klassifizierung → Aussage

Fasst man „Segmentierung" als Prozess der Erzeugung eines Segments[1] auf, lässt sich die Menge der verwendeten Verfahren in unterschiedliche Klassen einteilen.

1. **Voxelorientierte Verfahren**
   Die Entscheidung, ob ein Voxel zu einem Segment gehört, wird für jeden Voxel unabhängig von seiner Umgebung entschieden, basiert also rein auf Intensitätswert und Position.
2. **Kantenorientierte Verfahren**
   Vor der Segmenterzeugung wird in der Voxelmenge nach Kanten bzw. Objektübergängen gesucht, die z.B. als Silhouetten der zu findenden Segmente interpretiert werden können.
3. **Regionenbasierte Verfahren**
   Regionenbasierte Verfahren erzeugen zusammenhängende Gebiete. Im Kern steht meist wieder ein voxelorientiertes Verfahren, welches aber zusätzlich Nachbarschaften berücksichtigt.
4. **Modellbasierte Verfahren**
   Für spezielle Anwendungen können zusätzliche Informationen über das zu segmentierende Objekt (z.B. Form, typische Materialdichte) ausgenutzt werden.

[1] Um Irritationen zu vermeiden, wird in dieser Arbeit meist der Begriff des Segmentierungsverfahren verwendet.

Diese Einteilung ist nicht scharf abgrenzbar, da aufwendigere Verfahren sich nicht eindeutig zuordnen lassen und in mehrere Kategorien fallen. So gibt es z.B. einige Vertreter der aktiven Konturverfahren, die eigentlich in den Region Grow-Bereich fallen, deren Ausbreitungsgeschwindigkeit jedoch von harten Objektgrenzen (= Kanten) gebremst wird. Während die ersten drei Kategorien allgemeine Segmentierungsverfahren beschreiben, bilden die modellbasierten Verfahren eine Ausnahme, da diese speziell für einen gesonderten Anwendungsfall (bspw. für die Segmentierung eines bestimmten Organs) konzipiert wurden. Modellbasierte Verfahren sind vergleichsweise eingeschränkt in ihrer Verwendung und meist deutlich aufwendiger in der Entwicklung, liefern aber in der Regel die besten Ergebnisse.

In den folgenden Kapiteln sollen die Verfahren vorgestellt und analysiert werden, die auch in YaDiV implementiert wurden. Ein breiter Überblick über Segmentierungsmethoden wird bspw. von Heinz Handels in [26] gegeben (Kapitel 5, S. 96).

4.2. Min-Max-Verfahren

Ein sehr einfacher, aber oft verwendeter Vertreter aus dem Bereich der voxelorientierten Verfahren ist die Segmentierung anhand einer unteren bzw. oberen Intensitäts-Schranke. Es werden alle Voxel selektiert, deren (Original- oder auf die VOI skalierte) Intensitätswerte zwischen einer unteren Schranke *min* und einer oberen Schranke *max* liegen (siehe Abbildung 4.2).

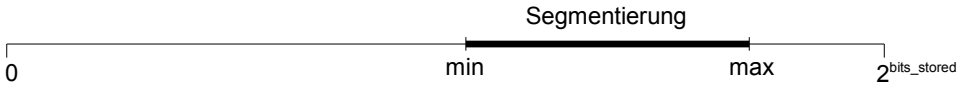


Abbildung 4.2.: Min-Max Segmentierung

Das Verfahren ist sehr einfach zu implementieren (siehe Algorithmus 4.1): Wie man leicht sieht, ist der Aufwand linear zur Größe des Voxeldatensatzes. In der Praxis wird es oft bei CT-Daten eingesetzt, da hier der Intensitätswert eine Materialzuordnung erlaubt. So lassen sich knöcherne Strukturen vergleichsweise einfach durch ein Min-Max-Verfahren segmentieren. Um jedoch einen einzelnen Knochen zu erhalten, ist hier oft ein Postprozessing notwendig, bspw. eine Auftrennung des Segments in Zusammenhangskomponenten.

4.3. Region Grow-Verfahren

Eine ebenfalls häufig verwendete Segmentierungs-Methode ist das sogenannte Region Grow-Verfahren. Diese Klasse von Verfahren breitet sich von einem Startpunkt, häufig

```
void build_range_segmentation(Segment seg, VoxelCube vc, int min, int max) {
  for (int z=0; z<dim_z; z++) {
    for (int y=0; y<dim_y; y++) {
      for (int x=0; x<dim_x; x++) {
        if (vc.get_xyz(x,y,z)>=min && vc.get_xyz(x,y,z)<=max) {
          seg.set_xyz(x,y,z, true);
        } else {
          seg.set_xyz(x,y,z, false);
        }
      }
    }
  }
}
```

Listing 4.1: Pseudocode für Min-Max Segmentierung

als *seed* bezeichnet, aus. Die Entscheidung, ob zwei benachbarte Voxel zusammenhängen und damit zum selben Gebiet gehören, wird dabei wie beim Min-Max Verfahren über den Intensitätswert getroffen, der mit dem des *seed* verglichen wird. Anders formuliert handelt es sich um eine Min-Max-Verfahren, bei dem zusätzlich die Nachbarschaft (bzw. der Zusammenhang mit dem Seed) berücksichtigt wird. Abbildung 4.3 skizziert die Ausbreitung eines 2D-Region Grow-Verfahrens mit einem 4er Nachbarschaftsmodell. Die Region Grow-Methode gehört zu den regionenbasierten Segmentierungsverfahren.

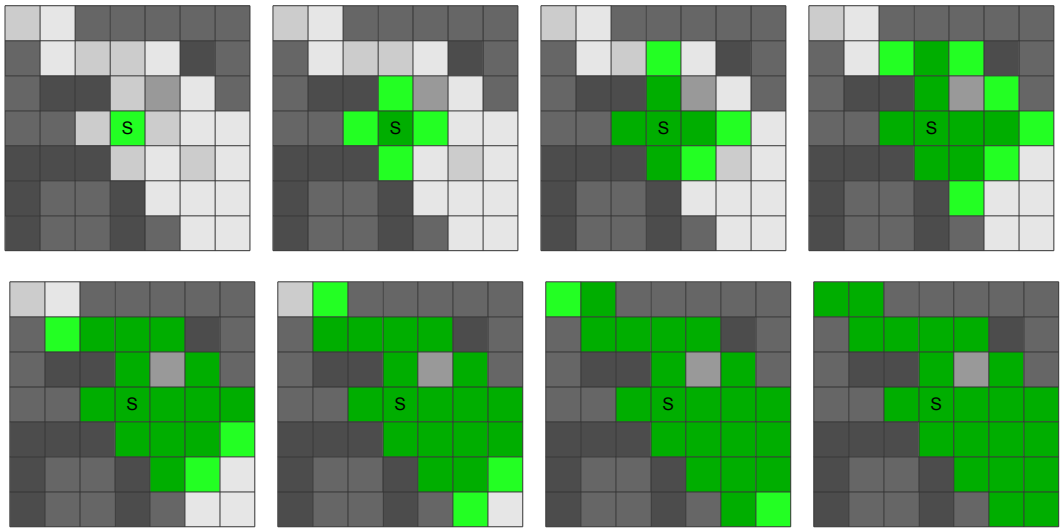


Abbildung 4.3.: Ausbreitungsschritte beim Region-Grow Algorithmus

Der Algorithmus benötigt zwei Parameter: Den Startpunkt s und die erlaubte (relative oder absolute) Abweichung (Variance) v des zugehörigen Intensitätswertes. Einige

Varianten dieser Methode erlauben auch mehrere Startpunkte. Als einfache und leicht verständliche Form der Implementierung wird oft ein rekursiver Ansatz gewählt. Es bezeichne $g(p)$ den Intensitätswert eines Voxels p. Für jeden Nachbarn (siehe unten) n von s wird nun überprüft, ob sich $g(n)$ innerhalb der Varianz von $g(s)$ befindet:

1. $g(n) \in [(g(s) - v \cdot g(s), g(s) + v \cdot g(s)]$: n wird zum Segment hinzugefügt
2. $g(n) \notin [(g(s) - v \cdot g(s), g(s) + v \cdot g(s)]]$ n wird nicht zum Segment hinzugefügt

Für jeden so hinzugefügten Voxel werden nun wiederum die Nachbarn überprüft. In einem rekursiven Verfahren können so alle indirekten Nachbarn von s bestimmt und auf ihre Segmentzugehörigkeit überprüft werden. Eine mögliche Variante des Verfahrens ist es, nicht die Abweichung des Intensitätswertes von der des Startpunkts zu betrachten, sondern die Intensitätsdifferenz zum aktuell überprüften Nachbarn. Dies hat jedoch meist den unerwünschten Effekt, dass abhängig vom Datensatz auch von der Anfangsintensität stark abweichende Intensitätswerte erreicht werden können, wenn eine Region mit einem fließenden Intensitätsübergang enthalten ist.

Obwohl das Verfahren auf den ersten Blick einfach erscheint, muss aufgrund der Größe der zu segmentierenden Daten ein besonderes Augenmerk auf eine effiziente Implementierung des Verfahrens gerichtet werden. Während im 2D-Fall der rekursive Ansatz auch bei größeren Regionen nicht mehr an derzeit übliche Hauptspeichergrenzen stößt, ist er in der 3D-Praxis aufgrund des hohen Stack-Speicherverbrauchs durch die rekursiven Funktionsaufrufe unbrauchbar. Deshalb muss hier die Rekursion in ein iteratives Verfahren umgewandelt werden. Ein erster, einfacher Ansatz wäre z.B. die folgende Vorgehensweise:

Benötigt: Voxel-Container TODO, Möglichkeit zur Markierung einzelner Voxel

1. Füge den Seed als erstes Element in den TODO-Container ein.
2. Voxel-Abarbeitung
   a) Entferne Element p aus dem Voxel-Container.
   b) Überprüfe bei allen Nachbarn von p die Regionszugehörigkeit mittels Grauwert und Varianz.
   c) Füge alle Nachbarpunkte, bei denen die Regionszugehörigkeit gilt und die noch nicht als fertig markiert wurden, in den TODO-Container ein.
3. Markiere p als fertig.
4. Fahre fort bei Schritt 2 bis die TODO leer ist.

Dieser erste Entwurf funktioniert, ist aber noch nicht optimal. Bspw. würden Voxel, die Nachbarn mehrere, bereits in der Queue befindlicher Elemente sind, ebenfalls mehrfach in die Queue eingefügt, im ungünstigsten Fall also 26 mal. Eine relativ einfache Umdeutung des Markierungsschrittes behebt dieses Problem: Fasst man die Markierung nicht als „bereits bearbeitete“ sondern als „bereits bearbeitete oder demnächst zu bearbeitende“ Kennzeichnung auf, behält das Verfahren seine volle Gültigkeit bei drastisch reduziertem Speicher- und Laufzeitverbrauch.

Dennoch finden weiterhin überflüssige Regionszugehörigkeitsvergleiche statt, da nicht mehr zum Segment gehörende Randvoxel (also Voxel, die über das gewählte Nachbarschaftsmodell mit einem zum Segment gehörenden Voxel benachbart sind) immer noch von jedem benachbarten Segmentvoxel aus einmal abgefragt werden. Eine Umdeutung der Markierung von „bereits bearbeitete oder demnächst zu bearbeitende“ zu „bereits bearbeitete, bereits als außerhalb des Segments liegend festgestellte oder demnächst zu bearbeitende“ Voxel verbessert zwar nicht den theoretischen Aufwand, jedoch die praktische Laufzeit des Algorithmus. Es können so viele Varianzvergleiche pro nicht zum Segment gehörendem Randvoxel eingespart werden, wie benachbarte Segmentvoxel vorliegen.

Die zweite Fragestellung betrifft die Wahl eines geeigneten Voxel Containers. Hier bietet sich eine Queue an. Durch die FIFO-Eigenschaft („First In First Out“) werden die nächsten Nachbarn des Seeds zuerst bearbeitet. Besonders bei großen, kompakten Segmenten führt dies i.A. zu einer drastischen Reduzierung der durchschnittlichen TODO-Containergröße, da die ersten Elemente der Queue statistisch signifikant weniger Nachbarn besitzen, die noch nicht bereits in der Queue sind. Dies lässt sich durch reale Versuche belegen, wie Tabelle 4.1 demonstriert. In diesem Beispiel (CT-Scan eines Kopfes), bei dem ein einzelner Start-Seed im Bereich des Schädelknochens gesetzt wurde, wurden neben der Laufzeit auch die maximalen und durchschnittlichen TODO-Containergrößen gezählt. v steht für die jeweilige Varianz der jeweiligen Segmentierung. Wie man leicht sieht, bleiben die Laufzeiten nahezu konstant, jedoch sinken die Containergrößen in der Queue-Implementation drastisch; die maximale Containergröße wird in diesem Beispiel um den Faktor 60-90 reduziert. An der Gesamtzahl (total) der segmentierten Voxel lässt sich gut erkennen, dass bei einer Varianz von 0.5 ein Punkt erreicht wurde, der es dem Verfahren ermöglicht hat, sich in einen bis dahin unerreichten Teil des Knochens auszubreiten.

		Stack			**Queue**		
v	total	average	max	time	average	max	time
0.1	142856	29535	52522	0s 422ms	672	945	0s 390ms
0.2	246829	57288	105363	0s 437ms	1141	1613	0s 406ms
0.3	330882	80269	143445	0s 579ms	1400	2210	0s 547ms
0.4	415582	104486	189010	0s 671ms	2041	3650	0s 672ms
0.5	1595850	422576	812637	2s 500ms	8874	12303	2s 625ms

Tabelle 4.1.: Region Grow: Stack vs. Queue

Damit erhält man das modifizierten Verfahren:

Benötigt: Voxel-Queue TODO, BitCube zur Markierung einzelner Voxel

1. Füge den Seed in den TODO-Container ein und markiere ihn als fertig.
2. Voxel-Abarbeitung
   a) Entferne Element p aus dem Voxel-Container.
   b) Überprüfe bei allen Nachbarn von p die Regionszugehörigkeit mittels Intensitätswert und Varianz.
   c) Füge alle Nachbarpunkte, bei denen die Regionszugehörigkeit gilt und die noch nicht als fertig markiert wurden, in den TODO-Container ein.
   d) Markiere alle überprüften Nachbarn als fertig.
3. Fahre fort bei Schritt 2 bis TODO leer ist.

Algorithmus 4.2 zeigt die Pseudocode Darstellung dieses Verfahrens. Auch hier ist der Laufzeitaufwand linear zur Volumenegröße, da im Worst-Case jeder Voxel betrachtet werden muss.

```
void build_region_grow_segmentation(Segment seg, VoxelCube vc,
    Voxel seed, double v) {
  int min = vc.get(seed) - v*vc.get(seed);
  int max = vc.get(seed) + v*vc.get(seed);

  Queue todo = new Queue();
  todo.push(seed);
  BitCube gotcha = new BitCube(vc._dimx, vc._dimy, vc._dimz);
  gotcha.set(seed, true);

  do {
    Voxel next = todo.pop();
    seg.set(next, true);
    forall (neighours n of next) {
      if (!gotcha.get(n)) {
        gotcha.set(n, true);
        if (min<=vc.get(next) && max>=vc.get(next)) todo.add(next);
      }
    }
  } while (!todo.isEmpty());
}
```

Listing 4.2: Pseudocode für Region-Grow Segmentierung

In der Praxis trifft man häufig auf die Situation, in der die Ausbreitung sich über eine enge Verbindung in ein unerwünschtes Gebiet bewegt. Eine Möglichkeit, dies zu verhindern, ist die Einführung sogenannter „Stopping"-Segmente, die explizit Voxelregionen vorgeben, die für die Ausbreitung des Region Grow-Algorithmus gesperrt sind.

4.4. Aktive Konturen

Aktiv-Kontur-Verfahren verwenden eine vorgegebene Startkontur, die sich im optimalen Fall schrittweise der Kontur des zu segmentierenden Gebiets annähert. Die sich in jedem Teilschritt verändernde Kontur wird dabei auch oft als *Snake* bezeichnet. Ein Anwendungsgebiet bildet hier die medizinische Bildverarbeitung, wo aktive Konturen beispielsweise zur computergestützten Objektverfolgung in Ultraschallaufnahmen verwendet werden. Schon 1988 wurde in [80] beschrieben, wie mit aktiven Snake-Verfahren Kanten und Objektübergänge identifiziert werden können.

Auch im Bereich der Segmentierung haben sich aktive Konturverfahren etabliert. Ziel ist es, die Segmentkontur durch eine parametrische Kurve zu beschreiben. Diese wird meist manuell initialisiert und in Abhängigkeit von den sogenannten internen und externen Energien iterativ entwickelt bis die Kurve der gewünschten Segmentkontur entspricht. Im Kern steht dabei das Minimierungsproblem

$$E_{snake}(\alpha(s)) = E_{ext}(\alpha(s)) + E_{int}(\alpha(s)) \tag{4.1}$$

In einem 3D-Volumendatensatz wird die **externe Energie** dabei aus den Intensitätswerten im Bezug zur Position der Kontur berechnet. Oft ist z.B. gewünscht, das sich die Kontur auf den im Datensatz vorhandenen Kanten verläuft, die sich über den Gradienten bestimmen lassen. Der Gradient zeigt in die Richtung der größten lokalen Veränderung, seine Länge entspricht dabei der Stärke des Anstieges. Eine mögliches Energiefunktional für die externe Energie könnte also durch

$$E_{ext}(\alpha(s)) = \int -\omega_1 \left|\nabla\alpha(s)\right| ds, \tag{4.2}$$

beschrieben werden. Die externe Energie entspricht in einem diskreten Beispiel dem Integral (und damit im diskreten Fall der endlichen Summe) über den negativen Betrag des Gradienten für alle Punkte auf der Kontur. Verläuft die Kontur weitgehend auf den Kanten des Volumendatensatzes, sind die integrierten Gradientenbeträge relativ groß und das externe Energiefunktional folglich relativ klein. Für die Verwendung in der Minimierungsgleichung (4.1) wird oft noch ein zusätzlicher Faktor ω_1 eingeführt, mit dem das Verhältnis von externer zu interner Energie gewichtet werden kann.

Die **interne Energie** der aktiven Kontur wird im Gegensatz zur externen aus der Form der Kontur bestimmt. Dabei gehen z.B. Kantenlänge und Krümmung der aktiven Kontur ein. In [80] wurde dafür der Term

$$E_{int}(\alpha(s)) = \frac{1}{2}\int \omega_2 \left|\frac{\partial\alpha(s)}{\partial s}\right|^2 + \int \omega_3 \left|\frac{\partial^2\alpha(s)}{\partial s^2}\right|^2 ds \tag{4.3}$$

vorgeschlagen. Die Parameter ω_2 bzw. ω_3 kontrollieren dabei das Verhältnis von Kantenlänge und Krümmung; wird bswp. ω_2 sehr viel kleiner als ω_3 gewählt erhält man eine Kurve mit minimaler Krümmung, also eine sehr glatte Kontur.

Setzt man (4.2) und (4.3) in die Ausgangsgleichung (4.1) ein, erhält man den Term

$$\begin{aligned} E_{snake}(\alpha(s)) &= E_{ext}(\alpha(s)) + E_{int}(\alpha(s)) \\ &= \int -\omega_1 \left|\nabla\alpha(s)\right| ds + \frac{1}{2}\int \omega_2 \left|\frac{\partial\alpha(s)}{\partial s}\right|^2 + \int \omega_3 \left|\frac{\partial^2\alpha(s)}{\partial s^2}\right|^2 ds \end{aligned} \tag{4.4}$$

der bzgl. $\alpha(s)$ – also der Kontur selbst – minimiert werden muss. Die richtige Gewichtung der Kontrollparameter ω_1, ω_2 und ω_3 hängt stark von der Geometrie des zu segmentierenden Gebiets ab. Die für die Minimierung notwendigen Techniken kommen dabei aus der Variationsrechnung.

Ein offensichtliches Problem, das in [80] nicht gelöst wurde, ist, dass die Initialisierung der Startkontur der finalen Segmentierung entsprechen muss, da die Kurve durch die explizite Darstellung als parametrisierte Kurve während der Optimierung nicht den Zusammenhang verändern und sich aufteilen kann. Dies ist in vielen Anwendungsfällen hinderlich, bspw. bei einem Trümmerbruch, bei dem die Zahl der Bruchstücke nicht im Voraus bekannt sein muss. Weiterhin war es nicht möglich, das Verfahren ohne eine erhebliche Erhöhung des Rechenaufwandes auf höhere Dimensionen zu verallgemeinern. Beide Probleme wurden 1988 von Osher und Sethian in [61] mit der sogenannten Level Set Methode gelöst.

4.4.1. Level-Set Verfahren

Bei der Level-Set Methode wird die Segmentkontur als Nullstellenmenge einer höherdimensionalen Funktion dargestellt. Dadurch lässt sich die Kontur in einem festen Eulerschen Koordinatensystem implizit darstellen ohne eine explizite Parametrisierung zu verwenden. Durch diesen Trick ist es möglich, dass sich die Topologie der Segmentkontur während der Ausbreitung beliebig verändern kann, womit insbesondere auch nicht zusammenhängende Segmente erkannt werden können. Björn Scheuermann hat in [72]

zwei Level-Set basierte Verfahren untersucht und implementiert, die nach Abschluss der Arbeit in überarbeiteter Form in die Hauptversion von YaDiV integriert wurden.

Sei $\Omega \in \mathbb{R}^3$ eine offene und beschränkte Menge und $\partial\Omega$ deren Rand. Es liegen (gemessene) Volumendaten vor, durch die die Funktion $v_I : \Omega \to \mathbb{R}$ implizit definiert wurde. Sei nun $\omega \subseteq \Omega$ (in diesem Anwendungsfall das Segment) ebenfalls offen und $C = \partial\Omega$ die Kontur von ω. Sei weiterhin $d(x) = \min\{|c - x|, c \in C\}$ für alle $x \in \Omega$ der euklidische Abstand zur Kontur C. Dann bezeichne

$$\begin{aligned} \varphi : \Omega &\to \mathbb{R} \\ x &\to -d(x), \text{ falls } x \in \omega \\ &\quad\ +d(x), \text{ sonst} \end{aligned} \tag{4.5}$$

die vorzeichenbehaftete Abstandsfunktion von C (siehe Kapitel 3.3.7, S. 42). Man sieht sofort, das durch diese Definition der Funktionswert von φ auf der Kontur C Null ist. Umgekehrt lässt sich C über diese Gleichung definieren, so dass die zweidimensionale Kontur C des dreidimensionalen Segments ω als Nullstellenmenge der Funktion φ aufgefasst werden kann.

Um ein aktives Konturverfahren unter diesen Bedingungen formulieren zu können, werden noch zwei weitere geometrische Größen der Kontur benötigt: die Flächennormale und die (mittlere) Krümmung. Der Normalenvektor $N(x_0)$ eines Punktes $x_0 \in C$ zu einem konstanten Zeitpunkt t lässt sich als normierter Gradient der vorzeichenbehafteten Abstandsfunktion

$$N(x_0) = \frac{\nabla\varphi(x_0)}{|\nabla\varphi(x_0)|} \tag{4.6}$$

abschätzen, der mit der zentralen Differenzenmethode (Kapitel 3.3.5.2) berechnet wird. Ebenso erhält man die mittlere Krümmung als Ergebnis von

$$k(x_0) = \operatorname{div}\left(\frac{\nabla\varphi(x_0)}{|\nabla\varphi(x_0)|}\right) \tag{4.7}$$

$$= \frac{\begin{array}{c}\varphi_{xx}(\varphi_y^2+\varphi_z^2)+\varphi_{yy}(\varphi_x^2+\varphi_z^2)+\varphi_{zz}(\varphi_x^2+\varphi_y^2) \\ -2(\varphi_x\varphi_y\varphi_{xy}+\varphi_x\varphi_z\varphi_{xz}+\varphi_y\varphi_z\varphi_{yz})\end{array}}{(\varphi_x^2+\varphi_y^2+\varphi_z^2)^{\frac{3}{2}}} \tag{4.8}$$

Um die Bewegung der Kontur zu beschreiben, muss φ noch um eine zeitliche Komponente erweitert werden:

$$\varphi : \Omega \times \mathbb{R} \to \mathbb{R} \tag{4.9}$$

Grundidee aller aktiven Konturverfahren ist es, dass sich die Kontur mit einer gewissen Geschwindigkeit ausbreitet. Dabei gehen verschiedene Größen ein; die Konturnormalen, die Intensitätswerte des Volumendatensatzes oder die mittlere Krümmung, mit der sich die Elastizität der Kontur beeinflussen lässt. Einige dieser Verfahren erlauben, dass die Kontur an manchen Stellen schrumpft (negative Konturgeschwindigkeit in einem Punkt), bei anderen Varianten kann die Konturbewegung lediglich (lokal) verlangsamt werden. Im Kern aller Verfahren steht die Evolutionsgleichung (in [55] für den 2D-Fall beschrieben),

$$\frac{\partial \varphi}{\partial t} + F\,|\nabla \varphi| = 0 \tag{4.10}$$

mit der Ausbreitungsgeschwindigkeit F. Ziel ist es, die Funktion φ iterativ zu entwickeln, bis am Ende die Grenzen des gewünschten Segments erreicht sind. Dies kann durch Vorgabe einer gewissen Grenze (z.B. das die Ausbreitungsgeschwindigkeit in jedem Konturpunkt unter eine vorgegebene Schranke fällt) oder einen manuellen Eingriff in die Evolution erfolgen. Auch hier steht erneut ein Minimierungsproblem im Zentrum der Berechnung, das unter Zuhilfenahme der Variationsrechnung gelöst werden kann (siehe [72], Seite 20f).

Im Folgenden sollen zwei spezielle aktive Konturverfahren vorgestellt werden, die in dieser Form auch als Segmentierungswerkzeuge in YaDiV zur Verfügung stehen.

4.4.2. Edge-Stopping Methode

Basis für die Implementation eines Edge-Stopping-basierten aktiven Konturverfahrens war die Arbeit von S. Osher und J.A. Sethian [61] aus dem Jahr 1988, das in der Umsetzung jedoch um einige neue Ideen erweitert und abgeändert wurde. Wie der Name bereits andeutet, wird die Ausbreitungsgeschwindigkeit durch im Volumendatensatz vorhandene Objektgrenzen (im 2D-Fall: Kanten) gebremst, welche über den Gradient der Intensitätswerte bestimmt werden. Das Verfahren fällt damit sowohl in die Kategorie der regionen- als auch der kantenbasierten Verfahren.

4.4.2.1. Theorie

Um die Theorie der Edge-Stopping Methode zu verstehen, muss zunächst etwas ausgeholt werden. Es beschreibe $w_x(t)$ mit $t \in [0, \infty]$ den Weg, den ein Punkt $x \in C$ aus der aktiven Kontur mit der Zeit zurücklegt. Da der Punkt dabei immer auf der Kontur liegt, gilt

$$\varphi(w_x(t), t) = 0 \tag{4.11}$$

Die grundsätzliche Ausbreitungsrichtung der Kontur erfolgt zunächst einmal in Richtung ihrer (Einheits-)Normalen, d.h.

$$\begin{aligned} \frac{\partial}{\partial t}\left(w_x(t)\right) &= F \cdot N \\ &= F \cdot \frac{\nabla\varphi}{|\nabla\varphi|} \end{aligned} \tag{4.12}$$

mit der Evolutionsgeschwindigkeit F, die auch als Ausbreitungsgeschwindigkeit oder „Speed Funktion" bezeichnet wird. Diese wird in [43] als

$$F = F_A + F_G \tag{4.13}$$

definiert, mit einem konstanten Term F_A, der die Kontur je nach Vorzeichen wachsen oder schrumpfen lässt und einem (kontur-)formabhängigen Term F_G der für eine Glättung der Kontur in Bereichen mit hoher mittlerer Krümmung verantwortlich ist. In dieser Arbeit wird vorgeschlagen, $F_A = 1$ zu setzen, d.h. eine konstant wachsende Kontur zu verwenden. $F_G = -\epsilon k, \epsilon \in \mathbb{R}$ wird über die mittlere Krümmung mit einem kleinen und negativen konstanten Faktor definiert.

In dieser Form ist die Ausbreitungsgeschwindigkeit nur von der Geometrie der aktiven Kontur abhängig, die Wachstumsrate soll jedoch zusätzlich durch die Intensitätswerte der Volumendaten beeinflusst werden. In der Edge-Stopping Methode wird dafür eine sogenannte *stopping*-Funktion

$$g : \mathbb{R}_{>0} \to \mathbb{R}_{>0} \tag{4.14}$$

eingeführt, die die Geschwindigkeit verringert, sobald die Kontur auf eine Kante in den Volumendaten trifft, also einen starken Intensitätswertanstieg erreicht. Deshalb wird die *stopping*-Funktion auch über den Gradienten definiert, dessen Betrag (wie auf Seite 38 beschrieben) der Stärke des größten Intensitätsanstiegs entspricht. Es können unterschiedliche *stopping*-Funktionen definiert und verwendet werden. Allen gemeinsam ist, dass sie im Intervall $[0, \infty)$ positiv, regulär und monoton fallend sein sollten, d.h. je kleiner der Gradient, desto kleiner der Geschwindigkeitsverlust durch die *stopping*-Funktion. Weiterhin sollte sich die Funktion bei großen Gradientlängen im Bereich von Null, bei kleinen im Bereich von Eins bewegen, so dass sich die Kontur in einem monotonen Bereich ohne Intensitätsunterschiede mit ungebremster Geschwindigkeit bewegen kann. Die modifizierte Evolutionsgleichung 4.10 lautet dann

$$\begin{aligned} & \frac{\partial \varphi}{\partial t} + g(|\nabla v|)(F_A + F_G)|\nabla \varphi| = 0 \\ & \frac{\partial \varphi}{\partial t} + g(|\nabla v|)(1 - \epsilon k)|\nabla \varphi| = 0 \\ \Leftrightarrow \quad & \frac{\partial \varphi}{\partial t} = -g(|\nabla v|)(1 - \epsilon k)|\nabla \varphi| \end{aligned} \tag{4.15}$$

Die Gleichung beschreibt die Veränderung der aktiven Kontur über die Zeit, für ihre Lösung muss jedoch der Term F_G erweitert werden, da dieser bisher auf der Krümmung von C basiert und damit nur auf der Kontur wohldefiniert ist. In [43] wird vorgeschlagen, für nicht auf der Kontur liegende Punkte hierfür den Wert des nächstliegenden Konturpunktes zu verwenden. Sei $q \in C$ und $p \in C = \min_{r \in C} |r - q|$ der nächste Konturpunkt zu q. Dann gilt für die erweiterte Geschwindigkeitsfunktion $F_G(q) = F_G(p)$ und man erhält die diskretisierte Evolutionsgleichung:

$$\varphi^{n+1} = \varphi^n - \Delta t \left(g(|\nabla v|)(F_A + F_G)|\nabla \varphi| \right) \tag{4.16}$$

4.4.2.2. Stopping-Funktionen

In [72] wurde u.a. die Eignung verschiedener *stopping*-Funktionen untersucht. Dazu wurden die Funktionen

$$g_1^I = \frac{1}{(1 + \sqrt{|\nabla v_I|})^p} \tag{4.17}$$

$$g_1^S = \frac{1}{(1 + \sqrt{|\nabla v_S|})^p} \tag{4.18}$$

$$g_2^I = e^{-|\nabla v_I|} \tag{4.19}$$

$$g_2^S = e^{-|\nabla v_S|} \tag{4.20}$$

implementiert und analysiert. ∇v_I steht dabei für den auf den Intensitätswert basierten Gradienten, ∇v_S bezieht sich hingegen auf den (visuellen) Gradienten der durch das VOI-Fenster[1] skalierten Werte. Während die original Intensitätswerte eine höhere Auflösung des Wertebereichs besitzen (üblicherweise zwischen 2^{12} und 2^{16}), sind die VOI-Werte auf ein 2^8 breites Wertefenster skaliert. Beide Ansätze haben daher ihre Berechtigung; die Verwendung von ∇v_I erreicht eine höhere Genauigkeit, ist aber auch wesentlich anfälliger für Störungen (kleine Intensitätsunterschiede die fälschlicherweise als Kante behandelt

[1] Region-of-Interest, siehe Kapitel 5.2.1, S. 83

werden). ∇v_S lässt viele Informationen ungenutzt, wird aber auf das u.a. zur Visualisierung genutzte Wertefenster angewendet, das der Betrachter üblicherweise so eingestellt hat, dass er die von ihm gesuchte Struktur möglichst gut erkennen kann.

Ein Problem, welches alle bisher gezeigten *stopping*-Funktionen gemeinsam haben, ist dass die Geschwindigkeit zwar sehr klein, aber nie exakt Null wird. Dies hat zur Folge, dass bei hinreichend vielen Iterationen auch starke Kanten übersprungen werden können. In g_1^I bzw. g_1^S lässt sich dieser Effekt durch die Verwendung eines höheren Exponenten reduzieren, zugleich wird dadurch die Störanfälligkeit gegenüber kleinen Intensitätswert-Schwankungen verstärkt. Deshalb wurde zusätzlich mit

$$g_3 = \frac{1}{1 + \sqrt{|\nabla v_I|^p} + \sqrt{|v_I - i_u}} \tag{4.21}$$

eine eigene Erweiterung der *stopping*-Funktionen untersucht. Grundidee war dabei die Einführung eines vorgegebenen Intensitätswertes i_u.

4.4.2.3. Algorithmus

Basierend auf 4.16 und nach der Wahl einer geeigneten *stopping*-Funktion lässt sich der Edge-Stopping-Algorithmus wie folgt definieren: In jeder Iteration wird zuerst die Geschwindigkeit der Kontur berechnet, auf den Rest der diskreten Volumendaten erweitert und anschließend die neue Distanzfunktion bestimmt.

Listing 4.3 zeigt das Verfahren im (stark vereinfachten) Pseudo-Code. Hier wird noch einmal deutlich, dass sich bei der Level-Set Methode nicht die Kontur selbst bewegt, sondern stattdessen iterativ die Distanzfunktion entwickelt wird, die dadurch implizit die Kontur definiert und somit auch Topologiewechsel ermöglicht. Der Parameter Δt gibt dabei die Approximationsgenauigkeit an; je kleiner Δt, desto genauer die Evolution.

Ein durch die Diskretisierung entstandenes Problem ist, dass die Distanzfunktion numerisch bedingt nur selten direkt Null ergibt, die Erzeugung durch einen Vergleich $df(v) = 0$ daher Lücken in der Kontur zur Folge hätte. Als Lösung wird in [84] vorgeschlagen, alle Voxel in deren (6er oder 26er) Nachbarschaft[1] ein Vorzeichenwechsel der Distanzfunktion φ stattfindet, als zur Kontur C zugehörig zu bezeichnen. Dies führt jedoch zu einer Verdickung der Kontur, da somit auch Voxel mit positiver Distanz als Kontur gezählt werden. Daher wird in dieser Arbeit die Bedingung

$$v \in C \Leftrightarrow \varphi(v) \leq 0 \text{ und } \exists n \in N_{26}(v) : \varphi(n) > 0 \tag{4.22}$$

[1] siehe Kapitel 3.3.1, S. 33

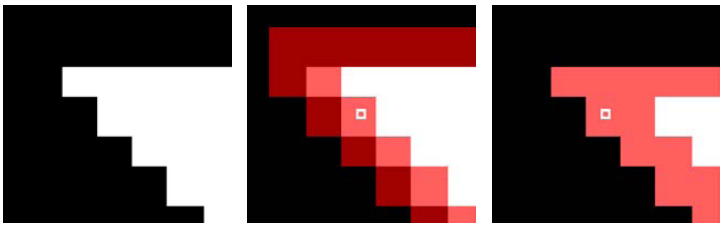
```
void build_edge_stopping_segmentation(Segment seg, VoxelCube vc, Contour c) {
  DistanceFunction dist_f = new DistanceFunction(vc, c);
  SpeedFunction speed_f = new SpeedFunction(vc);
  StopingFunction stop_f = new StopingFunction(vc);

  for (int i=0; i<number_of_iterations; i++) {
    // compute contour speed
    for (each voxel v in c) {
      F = F_A(v) + F_G(v);
      speed_f(v) = F * stop_f(v);
    }

    // extend contour speed
    for (each voxel v in vc) {
      if (!c.contains(v)) {
        p = c.find_closest(v); // could be more then one
        F = F_A(p) + F_G(p);
        speed_f(v) = F * g(v);
      }
    }

        // compute new distances
    for (each voxel v in vc) {
      dist_f(v) = dist_f(v) - Δt * speed_f(v);
    }

    // get new contour
    c = dist_f.get_contour();
  }
}
```

Listing 4.3: Pseudocode für Edge-Stopping-Verfahren

für die diskrete Konturzugehörigkeit verwendet. Abbildung 4.4 verdeutlicht diese Definition in einem (2D) Beispiel: Die klassische Definition b) gegenüber der in dieser Arbeit verwendeten c).

Bei der Erweiterung der Geschwindigkeit kann der Sonderfall auftreten, dass ein diskreter Ort v aus der Bandmenge zwei oder mehr kürzeste Entfernungen zur Kontur besitzt, also zum (diskreten) *cut locus* der Kontur C gehört. In diesem Fall wird die Geschwindigkeit aller zu v kürzesten Konturpunkte gemittelt.

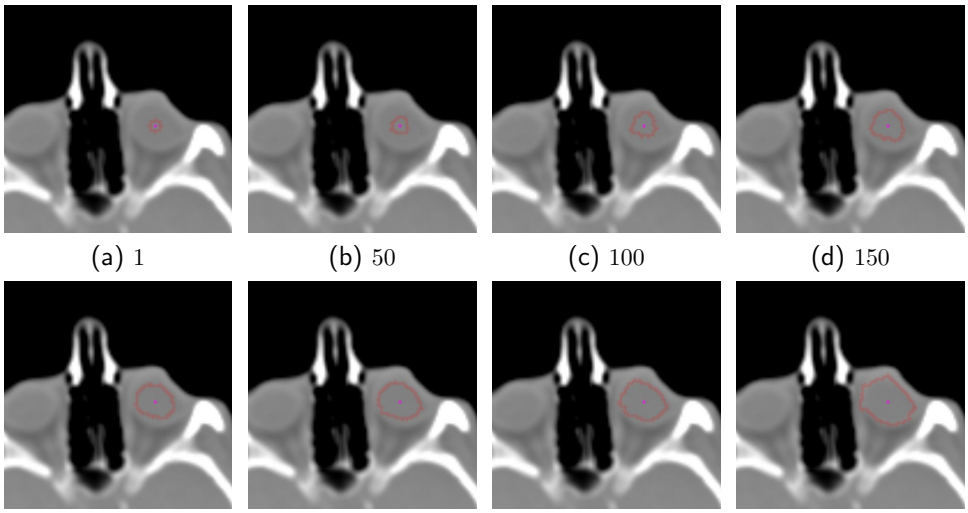
Abbildung 4.5 zeigt ein Beispiel einer Segmentierung der Augenhöhle, die mit $\epsilon = 0.025$ und $\Delta t = 0.05$ gestartet wurde. Als *stopping*-Funktion wurde g_2^S verwendet. Vor dem Start des Algorithmus wurden die Daten mit einem Gauß-Filter geglättet, um durch Rauschen entstandene Kanten zu unterdrücken. Man sieht, wie sich die als kleiner Kreis initialisierte



(a) Segment (b) Kontur 1 (c) Kontur 2

Abbildung 4.4.: Unterschiedliche Konturdefinitionen über die Distanzfunktion *Quelle: [72]*

Kontur rasch ausbreitet und dann ab ca. Iteration 300 nur noch sehr langsam wächst, da hier die *stopping*-Funktion die Ausbreitung an den sichtbaren Konturen abbremst.



(a) 1 (b) 50 (c) 100 (d) 150

(e) 200 (f) 250 (g) 300 (h) 1000

Abbildung 4.5.: Segmentierung mit der Edge-Stopping Level-Set Methode

Am Ende der Schleife lässt sich das neue Segment direkt aus der Distanzfunktion erzeugen. Da Distanzfunktion, VoxelCube und BitCube die selben Dimensionen haben, kann jedes Bit an der Stelle v im Segment-BitCube mit $\varphi(v) <= 0$ überprüft wird.

4.4.2.4. Erweiterung und Optimierung

Ein großes Problem aller Level-Set Verfahren sind Laufzeit und Speicherverbrauch. Daher wurden in YaDiV sowohl eine 2D- als auch eine 3D-Variante implementiert. Während in der 2D Variante auch auf größeren Schicht-Bildern (bis 512x512) die Kontur flüssig expandiert, ist das Verfahren auch bei kleinen Volumendatensätze (256x256x113) extrem langsam.

Wie man an Listing 4.3 leicht sieht, wird in jeder Iteration die Distanzfunktion für die Kontur aufgebaut und anschließend auf das gesamte Volumen erweitert. Ein erster Optimierungsansatz ist es, hier eine Narrow Band Strategie zu verwenden (vergl. S. 43). Da die Kontur sich in jeder Iteration nur in einer gewissen Umgebung zur vorherigen bewegen wird[1], müssen Geschwindigkeit und Distanzfunktion nur im Bandbereich neu berechnet werden. In der zur Verfügung stehenden Literatur wird das Band üblicherweise dadurch definiert, dass ein Würfelbereich um jeden Konturvoxel zur Bandmenge hinzugefügt wird. Da auch der Aufbau des Narrow Bands selbst Zeit kostet, wird in der Praxis üblicherweise das selbe Band für eine festgelegte Anzahl Iterationsschritte verwendet, in der die u.U. expandierende Kontur selbstverständlich das Band nicht verlassen darf.

Auch unter Verwendung eines Narrow Band Verfahrens bleibt der Algorithmus aufwendig, da für jeden Punkt im Bandbereich die Entfernung zu allen Konturpunkten berechnet werden muss. In [72] wurde gemeinsam ein Verfahren entwickelt und erprobt, um diesen Prozess zu beschleunigen. Im ersten Schritt werden alle N_{26} Nachbarn jedes Konturpunktes v betrachtet und sowohl die Entfernung als auch v selbst eingetragen, so dass eine Art *Zwiebelschale* um die Kontur entsteht. Im nächsten Schritt wird dieser Bereich erweitert, indem wiederum die Nachbarn der ersten Schaleniteration betrachtet werden. Das gleiche Verfahren eignet sich natürlich auch, um das Narrow Band selbst zu bestimmen und die Distanzfunktion abzuschätzen. Durch diese sogenannte *Zwiebelschalenmethode* konnte die Laufzeit drastisch reduziert werden.

Ein weiteres Problem betrifft den Speicheraufwand des Verfahrens, da neben den diskreten Volumendaten auch die diskreten Werte der Distanzfunktion φ, der Ausbreitungsgeschwindigkeit F und der *stopping*-Function G zur Laufzeit abrufbar sein müssen, was drei zusätzlichen *float* bzw. *double* pro Voxel entspricht. Laufzeit und Speicherverbrauch lassen sich gleichermaßen durch die Vorgabe einer Bounding Box reduzieren, da der Benutzer in der Regel die ungefähre Lage des gesuchten Segments (bspw. einer Orbita) vorgeben und so den Such- bzw. Evolutionsbereich eingrenzen kann.

Durch die diskrete numerische Iteration ist es notwendig, die Distanzfunktion in vorgegebenen Abständen mit der aktuellen Kontur-Iteration neu zu initialisieren. Alle Edge-Stopping Varianten müssen jedoch zwangsläufig scheitern, wenn das gesuchte Segment selbst keine klaren Konturen besitzt.

4.4.3. Energy Minimization Methode

Ein anderer Ansatz für Formulierung eines Level-Set Verfahrens ist die sogenannte Energy Minimization Methode. Bei dieser Methode wird keine gradientenbasierte *stopping*-Funktion verwendet, sondern ein Energiefunktional definiert und das Segment durch des-

[1] Andernfalls wurde Δt zu groß gewählt.

sen Minimierung entwickelt. Während bei der Edge-Stopping Methode vor allem die Kontur betrachtet wurde, wird bei der Energy Minimization Methode das Funktional über das Innere und Äußere des gesuchten Segments gebildet. Dabei gehen vor allem die durchschnittlichen Intensitätswerte aber auch das Volumen des Segments und der Flächeninhalt des Rands in die Entwicklung der aktiven Kontur ein. Eine der grundlegenden Arbeiten [56] zu energieminimierenden Segmentierungsverfahren für 2D-Bilddaten stammt von Mumford und Shah aus dem Jahr 1989.

4.4.3.1. Theorie

Wie im letzten Kapitel sei $\Omega \in \mathbb{R}^3$ offen und beschränkt mit dem Rand $\partial\Omega$ und der Intensitätsfunktion $v_i : \Omega \to \mathbb{R}$. Sei weiterhin $\omega \subseteq \Omega$ und $C = \partial\omega$ die (aktive) Kontur von C. Weiterhin sei $I(C) = \omega$ das Innere und $O(C) = \Omega \backslash \omega$ das Äußere von C und c_i der Mittelwert der Intensitäten in $I(C)$ und c_o der Mittelwert der Intensitäten $O(C)$. Auch hier soll wieder eine Startkontur vorgegeben und iterativ entwickelt werden.

Um das Prinzip zu verstehen, bietet sich ein einfaches Beispiel an: v_i sei gerade so definiert, das sie über Ω zwei verschiedene Bereiche definiert und dabei nur zwei Intensitätswerte c_1 und c_2 enthält. Das gesuchte Segment hat die Kontur C, die zum Segment gehörenden Volumendaten haben den Intensitätswert c_1. Das Minimum des Terms

$$F_I(C) + F_O(C) = \int\limits_{I(C)} |v_i(x,y,z) - c_i|^2 dxdydz + \int\limits_{O(C)} |v_i(x,y,z) - c_o|^2 dxdydz \quad (4.23)$$

bzgl. C wird ganz offensichtlich dann erreicht, wenn $F_I = F_O = 0$, also gerade dann wenn C dem Rand der beiden Bereiche entspricht. Im Jahr 2001 haben Tony Chan und Luminita Vese in [12] diesen Ansatz um den Flächeninhalt der Kontur $Area(C)$ und das Volumen des Segments $Volume(I(C))$ erweitert und die Energiefunktion

$$\begin{aligned} E_{CV}(C, c_i, c_o) &= \mu Area(C) + \nu Volume(I(C)) + \lambda_1 \int\limits_{I(C)} |v_i(x,y,z) - c_i|^2 dxdydz \\ &\quad + \lambda_2 \int\limits_{O(C)} |v_i(x,y,z) - c_o|^2 dxdydz \end{aligned} \quad (4.24)$$

mit den Steuerparametern $\mu \geq 0, \nu \geq 0, \lambda_1 \geq 0$ und $\lambda_2 \geq 0$ formuliert, mit denen die unterschiedlichen Anteile des Funktionals gewichtet werden können.

Auch bei der Energy Minimization Methode wird die Kontur selbst nun wieder über die Distanzfunktion φ definiert. Um darüber die Fläche und die Kontur bestimmen zu können wird zusätzlich eine sogenannte Heaviside-Funktion[1] verwendet:

$$\begin{aligned} H : \mathbb{R} &\rightarrow \mathbb{R} \\ x &\rightarrow \begin{cases} 1, \text{ falls } x > 0 \\ 0, \text{ falls } x \leq 0 \end{cases} \end{aligned} \tag{4.25}$$

Diese Funktion ist sehr nützlich, da sie die elegante Neuformulierung der bisher unvollständig oder gar nicht (Flächeninhalt und Volumen) angegebenen Terme aus 4.24 erlaubt. Wendet man die Heaviside-Funktion auf die Distanzfunktion φ an, so gilt $H(\varphi(v)) = 0$, falls v im Inneren von C liegt und $H(\varphi(v)) = 1$ für $v \in O(C)$. Die Kontur selbst wird durch den Gradienten $\nabla H(\varphi(v)) =: h$ beschrieben. Damit lassen sich der Flächeninhalt von C und das Volumen von $I(C)$ wie folgt definieren:

$$Area(C) = \int_{\Omega} h(\varphi(x,y,z))|\nabla\varphi(x,y,z)|dxdydz \tag{4.26}$$

$$Volume(I(C)) = Volume(\varphi(x,y,z) \leq 0) = \int_{\Omega} (1 - H(\varphi(x,y,z)))dxdydz \tag{4.27}$$

Eingesetzt in 4.24 erhält man

$$\begin{aligned} E(\varphi(x,y,z) = 0, c_i, c_o) &= \mu \int_{\Omega} h(\varphi(x,y,z))|\nabla\varphi(x,y,z)|dxdydz \\ &+ \nu \int_{\Omega} (1 - H(\varphi(x,y,z)))dxdydz \\ &+ \lambda_1 \int_{\Omega} |v_i(x,y,z) - c_i|^2(1 - H(\varphi(x,y,z))dxdydz \\ &+ \lambda_2 \int_{\Omega} |v_i(x,y,z) - c_o|^2 H(\varphi(x,y,z)dxdydz \end{aligned} \tag{4.28}$$

Auch die Mittelwerte c_i und c_o lassen sich über die Heaviside Funktion formulieren

[1] Benannt nach dem englischen Mathematiker Oliver Heaviside (1850–1925), auch als Treppen-, Schwellenwert-, Stufen- oder Sprungfunktion bezeichnet.

$$c_i = \frac{\int_\Omega v_i(x,y,z)(1-H(\varphi(x,y,z)))dxdydz}{\int_\Omega (1-H(\varphi(x,y,z)))dxdydz} \tag{4.29}$$

$$c_o = \frac{\int_\Omega v_i(x,y,z)H(\varphi(x,y,z))dxdydz}{\int_\Omega H(\varphi(x,y,z))dxdydz} \tag{4.30}$$

und in Gleichung 4.28 einsetzen, worauf hier aus Platzgründen verzichtet werden soll. Um die Gleichung bezüglich der Kontur C selbst zu minimieren, muss (wie schon beim Edge-Stopping-Verfahren) die Ableitung dieser Funktion nach der Zeit t gebildet werden und es ergibt sich

$$\frac{\partial\varphi}{\partial t} = h(\varphi)\left(\mu\left(\frac{\nabla\varphi}{|\nabla\varphi|}\right) - \nu - \lambda_1(v_i - c_i)^2 + \lambda_2(v_i - c_o)^2\right) = 0 \tag{4.31}$$

und damit wieder ein (Zero-)Level-Set, das durch eine partielle Differentialgleichung lösbar ist.

4.4.3.2. Heaviside-Funktionen

Zuvor ist es jedoch notwendig, die Heaviside-Funktion $H(x)$ und damit auch die zugehörige Ableitung $h(x)$ zu glätten. 1996 wurde dazu in [90] vorgeschlagen, dafür die Funktion

$$H_{CMOZ}(x) = \begin{cases} 1 & \text{falls } x > \epsilon \\ 0 & \text{falls } x < epsilon \\ \frac{1}{2} + \frac{x}{2\epsilon} + \frac{1}{2\pi}sin(\frac{\pi z}{\epsilon}) & \text{falls } x < epsilon \end{cases} \tag{4.32}$$

zu verwenden. Da diese Approximation jedoch gewisse Nachteile mit sich bringt, schlugen Chan und Vese im Jahr 2001 in [12] vor, dafür die Funktion

$$H_{CV}(x) = \frac{1}{2} + \frac{1}{\pi}arctan\left(\frac{x}{\epsilon}\right) \tag{4.33}$$

zu verwenden. In [72] wurde zudem noch mit

$$H_S(x) = \frac{1}{2} + \frac{1}{\pi}tanh\left(\frac{x}{\epsilon}\right) \tag{4.34}$$

ein eigener Ansatz implementiert und untersucht.

In unseren Versuchen hat sich herausgestellt, das $H_{CV}(x)$ im Allgemeinen Fall die bessere Wahl für die Anwendung in der energieminimierenden Segmentierung darstellt. Anhand von Beispielen konnte gezeigt werden, dass $H_{CMOZ}(x)$ im Prinzip ebenfalls funktioniert,

aber dazu neigt ein lokales Minimum anzustreben, gleiches gilt für $H_S(x)$. Eine ausführlichere Begründung für dieses Verhalten findet man in [72] (Kapitel 3.2.4 S. 47). Wird das Verfahren jedoch mit einer guten Vorapproximation des gewünschten Segments initialisiert, führen auch diese Funktionen zu guten Ergebnissen.

4.4.3.3. Algorithmus

Verwendet man eine Approximation der Heaviside Funktion aus dem letzten Kapitel, lässt sich Gleichung 4.28 wie in [84] vorgeschlagen durch

$$\frac{\varphi_{i,j,k}^{n+1} - \varphi_{i,j,k}^{n}}{\Delta t} = h_\epsilon \left(\varphi_{i,j,k}^{n} \left(\mu k - \nu - \lambda_1 \left(v_i(i,j,k) - c_i \varphi_{i,j,k}^{n} \right)^2 + \lambda_2 \left(v_i(i,j,k) - c_o \varphi_{i,j,k}^{n} \right)^2 \right) \right) \tag{4.35}$$

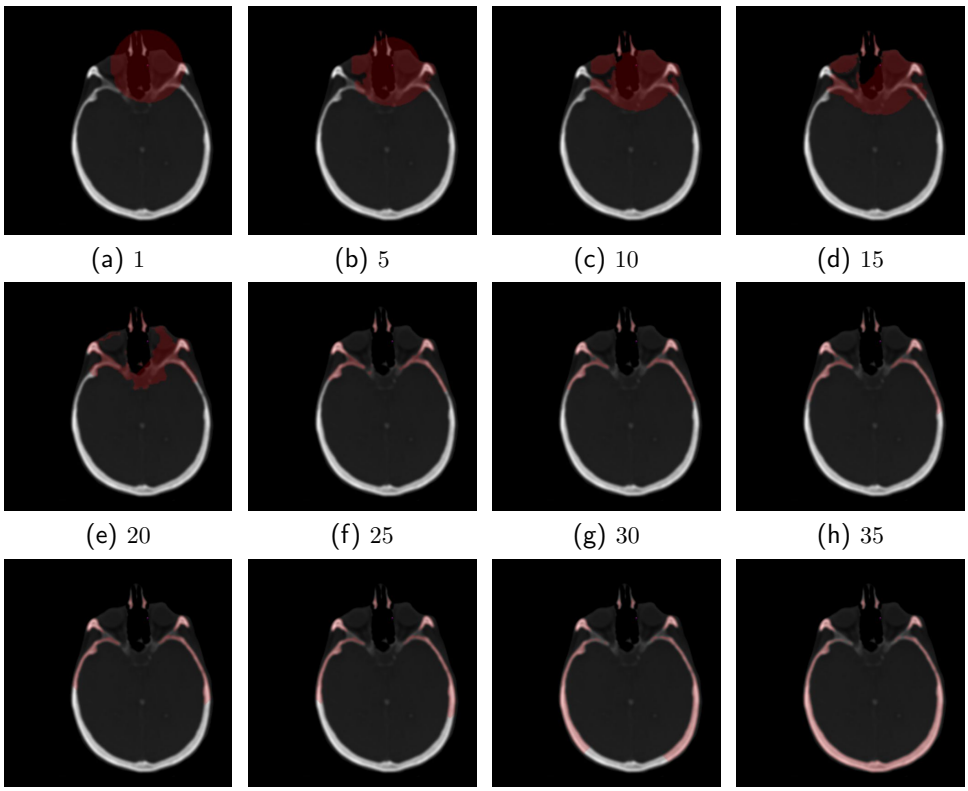
diskretisieren.

```
void build_energy_min_segmentation(Segment seg, VoxelCube vc, Contour c) {
  DistanceFunction dist_f = new DistanceFunction(vc, c);
  EnergyFunction energy_f = new EnergyFunction(vc);

  for (int i=0; i<number_of_iterations; i++) {
    for (each voxel v in vc) { // compute c1 & c2
      H = heaviside(dist_f(v));
      c_i += vc(v) * (1 - H);
      innerpoints += 1 - H;
      c_o += vc(v) * H;
      outerpoints += H;
    }
    c_i /= innerpoints;
    c_o /= outerpoints;

    for (each voxel v in vc) { // compute energy
      k = dist_f.compute_curvature(v);
      energy = heaviside.dirac(v) * (μ*k + ν + λ1*(vc(v)-c_i)^2 + λ2*(vc(v)-c_o)^2;
      energy_f(v) = energy;
    }

    for (each voxel v in vc) { // use energy to generate new distances
      dist_f(v) = dist_f(v) + Δt * energy_f(v);
    }
  }
}
```

Listing 4.4: Pseudocode für Energy-Minimization-Verfahren

Listing 4.4 zeigt den Pseudo-Code für die Energy-Minimization Methode. Der grundsätzliche Ablauf ist dabei ganz ähnlich zum Edge-Stopping Verfahren. Statt der Geschwindigkeit wird hier in jeder Iteration zuerst die Energie berechnet und damit die Distanzfunktion modifiziert. Neu ist jedoch, dass die Kontur nur noch zur Initialisierung benötigt wird und der Gradient der Intensitätswerte entfällt. Das Verfahren ist dadurch schneller und weniger speicherintensiv.

Auch bei diesem Verfahren muss jedoch die Distanzfunktion nach einer gewissen Anzahl Iterationen neu initialisiert (und dabei natürlich doch wieder die Kontur bestimmt) werden.



(a) 1 (b) 5 (c) 10 (d) 15

(e) 20 (f) 25 (g) 30 (h) 35

(i) 40 (j) 50 (k) 75 (l) 100

Abbildung 4.6.: Segmentierung mit der Energy-Minimization Level-Set Methode

Abbildung 4.6 zeigt ein Beispiel für die Ausbreitung. Dabei wurden $\mu = \nu = \lambda_1 = \lambda_2 = \epsilon = 1$ gesetzt, als Heaviside-Funktion wurde die eigene Entwicklung $H_S(x)$ verwendet. Man sieht wie sich das Segment nach der Initialisierung als etwas größere Kreisscheibe sehr schnell von der Vorgabe löst und die Regionen mit den größten Intensitätsunterschieden separiert. In diesem Beispiel wurden die durch das VOI-Fenster skalierten Werte verwendet. Nach ca. 100 Iterationen verändert sich das Segment nicht mehr.

4.4.3.4. Erweiterung und Optimierung

Genau wie das Edge-Stopping Verfahren profitiert auch die Energy-Minimization Methode stark von einer Einschränkung des betrachteten Bereichs, bspw. durch eine vom Benutzer festgelegte Bounding Box. Während bei der gradientenbasierten Evolution der Edge-Stopping Verfahren diese Bounding Box vor allem die Laufzeit positiv beeinflusst, gibt es in der energiebasierten Minimierung ein zusätzliches und starkes Argument: In vielen Anwendungen ist man an einem im Vergleich zum Datensatz kleinen Segment interessiert, das zwar lokal in den Intensitätswerten von seiner Umgebung abweicht, dessen Mittelwert (und der seiner Umgebung) nicht mit denen des restlichen Datensatzes übereinstimmen muss. So ist in vielen Datensätzen, die im Rahmen dieser Arbeit betrachtet wurden, ein großer Anteil an „Luft“ vorhanden, so dass die energiebasierten Verfahren dazu tendierten, bei einer großen Zahl an Iterationen grundsätzlich „Luft“ von „Nicht-Luft“ zu separieren. Eine Einschränkung des betrachteten Gebiets hat somit auch qualitative Vorteile.

Eine weitere Verbesserung dieser Technik, die erst nach der Integration des Energy-Minimization Verfahrens in die Hauptversion von YaDiV implementiert werden konnte, ist die optionale Auswahl einer unteren bzw. oberen Intensitätsschranke durch den Benutzer. Auch hier war das Ziel, in Bereichen, wo sich die Intensitätswerte nur minimal unterscheiden (bspw. den Kerngebieten im Gehirn), zusätzliches Benutzerwissen in die Iteration zu integrieren. Erste Versuche mit der Neuroradiologie der Medizinischen Hochschule Hannover belegen die Nützlichkeit dieser Erweiterung.

4.4.4. Zusammenfassung der Level-Set Methoden

Level-Set basierte Aktiv-Kontur-Verfahren haben einen primär mathematischen Hintergrund. Sie bieten insbesondere aufgrund der vielen Steuerparameter und der Möglichkeit, unterschiedliche Topologien zu segmentieren, großes Potential. Während die hier vorgestellten Edge-Stopping Verfahren sich zur Segmentierung von scharf kontrastierten Gebieten mit unbekannter Topologie eignen, sind mit den Energy-Minimization Verfahren auch Strukturen mit unscharfen Konturen erfassbar.

Gerade dieser mathematische Hintergrund liefert jedoch auch eine gewisse Hürde in der Anwendung:

1. Nur wenige medizinische Anwender sind bereit, sich mit unterschiedlichen Heaviside- oder *stopping*-Funktionen auseinanderzusetzen oder sich über die Gewichtung von Krümmung und Volumen des gesuchten Segments Gedanken zu machen.

2. Um gezielt zu Segmentieren, ist auch bei längerer Erfahrung eine gewisse Experimentierfreudigkeit mit den Parametern erforderlich. Das Ergebnis hängt zudem stark von der Initialisierung der Startkontur bzw. des Startsegments ab.

3. Durch Rechenaufwand und Speicherverbrauch sind die Verfahren nur bedingt für klinische Volumendaten einsetzbar.

Um die erste Hürde anzugehen ist es sicherlich erforderlich, eine angemessene Benutzerschnittstelle zu entwickeln, bei der die Gestaltparameter intuitiv vorgegeben werden können. Insbesondere diese Formparameter geben dieser Methode ihr spezifisches Potential. Hier wäre es wünschenswert, die Anzahl der versteckten Parameter weitestgehend zu reduzieren. U.a. könne die Größe des Narrow-Bandes und die Anzahl der Schritte bis zur Re-Initialisierung der Distanzfunktion automatisch an die gewählte Geschwindigkeit (u.U. auch in Abhängigkeit von den Volumendaten und der gewählten *stopping*-Funktion) angepasst und so vor dem Anwender verborgen werden. Dies steht und fällt jedoch mit der Anwendbarkeit, die derzeit vor allem durch die spürbar lange Rechenzeit beschränkt wird. Hier wäre ein interaktives „spielen“ mit den Gestaltparametern, was eine sofortige visuelle Rückantwort voraussetzt, sehr hilfreich, insbesondere, wenn es weiterhin möglich wäre, in der Iterationstiefe nicht nur vor- sondern auch zurückzugehen.

Vor allem was die Interaktivität betrifft gibt es einige vielversprechende Arbeiten die (etwas reduzierte) Ansätze auf der GPU berechnen, so z.B. in [34] durch Lefohn, Cates und Whitaker.

4.5. Atlasbasierte Segmentierung

Eine Reihe von Arbeiten beschäftigt sich mit der Verwendung von medizinischen Atlanten in der Segmentierung. Einen guten Einstieg findet man z.B. im Buch „Medizinische Bildverarbeitung“ ([26], S.149). Sowohl der Begriff des „Atlas“ als auch der „atlasbasierten Segmentierung“, manchmal auch „atlasbasierte Registrierung“, werden in der Literatur jedoch unterschiedlich definiert.

In dieser Arbeit soll darunter die Methode verstanden werden, bei der ein bekannter und bereits segmentierter Volumendatensatz $V_A : \mathbb{R}^3 \to \mathbb{R}$ auf einen unbekannten Volumendatensatz $V_D : \mathbb{R}^3 \to \mathbb{R}$ registriert (also bewegt und evtl. auch verformt) wird, um eine größtmögliche Strukturübereinstimmung zu erzielen. V_A zusammen mit der vorliegenden Segmentierung soll hierbei als **Atlas** bezeichnet werden, V_D bezeichnet die vorliegenden unsegmentierten Daten. Durch die Rücktransformation lässt sich so jedem Voxel aus V_D ein Voxel im Atlas zuordnen und dessen Segmentzugehörigkeit in V_A nach V_D übertragen. Die atlasbasierte Segmentierung gehört somit am ehesten zur Klasse der modellbasierten Verfahren, auch wenn darunter i.A. ein spezifischeres Modell verstanden wird. Bei der

atlasbasierten Segmentierung ist das Modell quasi implizit über die zum Atlas gehörende Segmentierung enthalten.

4.5.1. Literaturüberblick und Theorie

Zum Thema „atlasbasierte Segmentierung“ ist eine ganze Reihe von Veröffentlichungen erschienen. Ein guter Überblick wird u.a. in [66] gegeben. Das Prinzip des Verfahrens ist in fast allen Veröffentlichungen das selbe: grundsätzlich handelt es sich – wie auch schon bei den aktiven Konturverfahren – um ein Optimierungsproblem, bei dem die Transformation V_A auf V_D dahingehend entwickelt werden soll, dass eine möglichst große *Ähnlichkeit* erzeugt wird. Die eigentliche Herausforderung liegt in dieser Registrierung, die nach Maintz und Viergever [42] in die Klasse der voxelbasierten intrinsischen 3D/3D-Methoden fällt. Da bei sehr vielen medizinischen Anwendungen der generische Atlas lokal stark vom individuellen Patienten abweichen kann, wird der Registrierungsprozess in zwei Abschnitte unterteilt, die

1. **rigide Registrierung**, bei der der Datensatz durch eine affine Transformation skaliert, verschoben oder rotiert wird sowie die
2. **elastische Registrierung**, bei der auch nicht-lineare (lokale) Deformationen erlaubt sind.

Die affine Transformation ist in den meisten Arbeiten nahezu identisch, auch wenn sich die verwendeten Optimierungsstrategien hier leicht unterscheiden. Die größten Unterschiede gibt es jedoch in der Art der elastischen Registrierung und im Maß für die Übereinstimmung.

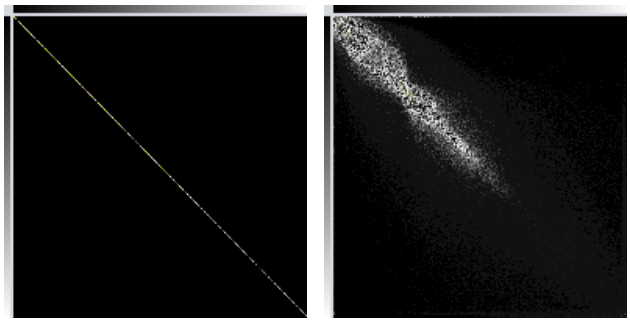
Ein weiteres wichtiges Thema ist die Auswahl bzw. die Erzeugung des Atlas. Je nach Anwendungsgebiet kann dieser ein generisch erzeugter Durchschnittsdatensatz sein oder aus einem bereits vorliegenden patientenspezifischem Modell bestehen. In bestimmten Anwendungen kann es auch sinnvoll sein, aus mehreren Atlanten auswählen zu können, die bspw. im Fall von medizinischen Daten nach Alter, Geschlecht oder einem spezifischen Krankheitsbild klassifiziert wurden.

4.5.2. Ähnlichkeit

Bevor eine iterative Registrierung entworfen werden kann, muss zunächst einmal ein Maß für die Ähnlichkeit zweier (u.U. unterschiedlich dimensionierter) Volumendatensätze definiert werden. In der Bildverarbeitung existieren eine Vielzahl unterschiedlicher Ähnlichkeitsmaße. Einen guten Einstieg findet man im Überblicksartikel [71] von Sarrut und Miguet oder etwas aktueller von Tarel und Boughorbel in [79]. Speziell in der Registrierung

von diskreten Volumendaten ist ein solches Maß nicht einfach zu finden: Im ungünstigsten Fall muss eine Aussage über die Ähnlichkeit von zwei Datensätzen mit unterschiedlichem Volumen, unterschiedlicher Auflösung, unterschiedlichen Intensitätswertebereichen und -verteilungen getroffen werden, die aber dennoch beide bspw. eine MRT Aufnahme des menschlichen Schädels enthalten.

Eine oft verwendete Form der Visualisierung der Ähnlichkeit von (n-dimensionalen) Bilddaten B_1 und B_2 ist das sogenannte **Verbund-Histogramm**. Dabei handelt es sich um ein dreidimensionales Diagramm, wobei die dritte Dimension über die Farbe bzw. Helligkeit angegeben wird. Auf der x-Achse stehen die (möglichen!) Intensitäten von V_1, auf der y-Achse die von V_2, der Farbwert an der Stelle steht für die Häufigkeit der Kombination der Wertepaare (schwarz = tritt selten auf, weiß = tritt oft auf).



(a) identisch (b) ähnlich

Abbildung 4.7.: Verbund-Histogramme *Quelle: [50]*

Abbildung 4.7 demonstriert dies anhand von zwei Beispielen. Das Verbund-Histogramm a) zeigt zwei identische Bilder, es liegen nur Wertepaare der Form (i, i) vor. Das zweite Beispiel zeigt zwei gegeneinander verschobene (und daher immer noch ähnliche) Bilder, die Werteverteilung im Verbund-Histogramm weist hier eine deutlich Streuung auf.

Mathematisch gesehen ist ein solches Ähnlichkeitsmaß eine Funktion

$$S : B \times B \rightarrow [0..m] \tag{4.36}$$

wobei B in unserem Anwendungsfall die Menge aller diskreten 3D-Bilddaten darstellt. Ein größer Wert von S ist mit einer höheren Ähnlichkeit gleichzusetzen, zwei identische Objekte haben die Ähnlichkeit m. Gilt $m = 1$ spricht man auch von einem normierten Ähnlichkeitsmaß. Als eine Art Standard für die Registrierung von (2- oder 3D) Bilddaten hat sich vor allem die sogenannte *mutual information* entwickelt, die üblicherweise über die *Entropie* des Verbunds von $B_1, B_2 \in B$ definiert wird.

Beide Begriffe stammen aus der Informationstheorie. Während die Entropie, ursprünglich ein Begriff aus der Thermodynamik, von Claude Elwood Shannon (1916–2001) als Maß

für den Informationsgehalt einer Nachricht definiert wurde, soll uns hier eine angewandte Definition genügen, auch wenn der theoretische Hintergrund der gleiche ist. Für einen Volumendatensatz V mit dem Intensitätswertebereich I bezeichne

$$H(V) = \sum_{i \in I} -p(i)log(p(i)) \tag{4.37}$$

die **Entropie** von V, wobei $p(i)$ die Wahrscheinlichkeit angibt, mit der i in V vorkommt. Da V_1 und V_2 als diskrete Volumendaten vorliegen, lässt sich $p(i)$ einfach durch die Häufigkeit von i in V geteilt durch die Anzahl Voxel in V berechnen. Seien nun V_1 und V_2 Volumendatensätze gleicher Dimension mit den Intensitätswertebereichen I_1 und I_2. Dann heißt

$$H(V_1, V_2) = \sum_{i_1 \in I_1, i_2 \in I_2} -p(i_1, i_2)log(p(i_1, i_2)) \tag{4.38}$$

die **Verbundentropie** von V_1 und V_2. $p(i_1, i_2)$ entspricht dabei der Wahrscheinlichkeit das ein Intensitätswert $i_1 \in I_1$ an der entsprechenden Stelle in V_2 auf $i_2 \in I_2$ abgebildet wird und lässt sich über die Summe aller Paarungen geteilt durch die Anzahl Voxel in V_1 bzw. V_2 berechnen.

Die **mutual information**[1] beschreibt, wie das Wissen über eine der Variablen die Information über die andere verändert. Die Definition baut auf der Verbundhomotopie auf und wurde von A. Collignon und P. Viola unabhängig voneinander formuliert. Angewendet auf die Registrierung von V_1 und V_2 durch T erhält man die (normierte) Definition:

$$I(V_1, V_2) = \frac{H(V_1) + H(V_2)}{H(V_1, V_2)} \tag{4.39}$$

Je gleichmäßiger die Verteilung der Wertpaare (V_1, V_2), desto größer wird die *mutual information*. Im Extremfall $V_1 = V_2$ ist $I(V_1, V_2) = 2$ maximal. Neben wünschenswerten mathematischen Eigenschaften, wie Symmetrie $I(V_1, V_2) = I(V_2, V_1)$, besitzt dieses Maß viele praktische Vorteile, insbesondere eine Robustheit gegen Bildrauschen sowie Helligkeitsunterschiede. Zusätzlich lässt sich das Verfahren sehr einfach implementieren, da letztlich nur das Vorkommen der Intensitätswerte und Wertepaare gezählt wird. Die Berechnung bei großen multidimensionalen Bilddaten ist aufwendig, lässt sich aber gleichzeitig gut parallelisieren.

[1] auf deutsch auch als Transinformation oder gegenseitige Information bezeichnet

4.5.3. Registrierung

Der Prozess der Registrierung von zwei Datensätzen V_1 und V_2 entspricht dem Finden einer Transformation T, so dass $I(T(V_1), V_2)$ möglichst groß wird. Wie schon zu Anfang des Kapitels beschrieben, setzt sich der Registrierungsprozess aus einer rigiden Transformation T_r und einer elastischen Teil T_e zusammen. Dabei wird zunächst die rigide (Grob-)Anpassung durchgeführt und anschließend lokale Differenzen durch die elastische Deformation ausgeglichen:

$$T = T_r \circ T_e \text{ bzw. } T(v) = T_e(T_r(v)) \tag{4.40}$$

Es handelt sich also um ein Optimierungsproblem bzgl. der Parameter von T, so dass sich die Registrierung mathematisch wie folgt definieren lässt: Sei Ω der Parameterraum der Transformation T und

$$\begin{array}{rcl} f : \Omega & \rightarrow & \mathbb{R} \\ f(p) & := & I(T(V_1), V_2) \end{array} \tag{4.41}$$

die Funktion, die einen Parametervektor $p \in \Omega$ auf das Ähnlichkeitsmaß der mit p durchgeführten Transformation abbildet. Dann entspricht das Registrierungsproblem dem Finden des (globalen) Maximums von f.

4.5.3.1. Rigide Registrierung

Die rigide oder auch affine, manchmal auch lineare Registrierung versucht die Abbildung, mit der V_1 in V_2 überführt als eine affine Abbildung der Form

$$T_r(v) = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} v + \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} \tag{4.42}$$

zu optimieren. Theoretisch besitzt die Optimierung daher 12 Freiheitsgrade, praktisch sind jedoch nur bestimmte Operationen interessant: Translation in drei Richtungen, Rotation um alle drei Koordinatenachsen und die Skalierung, ebenfalls in alle drei Richtungen, wodurch die Anzahl der Freiheitsgrade auf 9 reduziert wird.

Wollny und Kruggel geben in [88] einen Überblick über unterschiedliche Optimierungsstrategien und vergleichen diese insbesondere hinsichtlich ihrer Komplexität. Eine weit verbreitete Vorgehensweise sind Multiskalen-Verfahren, die zunächst versuchen mit relativ groben Schritten die Transformation zu verbessern (siehe z.B. [15]). In jeder Iteration

werden alle möglichen Translationen um $t \in \mathbb{R}$, Rotationen um den Winkel $\alpha \in \mathbb{R}$ und Skalierungen um den Faktor $s \in \mathbb{R}$ betrachtet. Dabei wird jeweils die *mutual information* verglichen und die Operation mit der größtmöglichen Verbesserung der Ähnlichkeit ausgewählt. Erst wenn sich mit den vorgegebenen Werten für Skalierung, Rotation und Translation keine Verbesserung mehr erzielen lässt, werden diese halbiert. Das Verfahren endet, wenn t, α, s oder das Ähnlichkeitsmaß eine vorgegebene Schranke erreichen. Die Vorgehensweise ist damit eine Variante der Best-Neighbour Methode, da Verschiebung, Rotation und Skalierung jeweils in zwei (benachbarte) Richtungen erfolgen können.

```
Transformation rigid_registration(VoxelCube V1, VocelCube V2,
      double δ, double α, double s) {
  Transformation trans = new Transformation();
  TransformationSet t_set = new TransformationSet();
  t_set.add(translationen um δ);
  t_set.add(rotationen um α);
  t_set.add(skalierungen um s);

  for {i=0; i<Anzahl iterationen; i++ {
    double mi = mutual_information(V1, V2, trans);
    double mi_new = mi;
    Transformation trans_new;

    do {
      foreach (t in t_set) {
        if (mutual_information(V1, V2, t*trans)<mi_new) {
          mi_new = mutual_information(V1, V2, t*trans);
          trans_new = t;
        }
      }
      if (mi_new < mi) {
        trans = trans_new*trans;
        mi = mi_new;
      }
    } while (verbesserung erzielt);
    δ /= 2; α /= 2; s /= 2;
  };

  return trans;
}
```

Listing 4.5: Pseudocode für rigide Registrierung

Listing 4.5 zeigt das Verfahren im Pseudo-Code. Wie man sofort sieht, liegt der Hauptteil der Laufzeitkosten in der Berechnung der *mutual information*, die für jede mögliche Transformation durchgeführt werden muss.

Die Menge an zu überprüfenden Operationen ist dabei nicht zwangsläufig fest vorgegeben.

Eine mögliche Erweiterung kann z.B. darin bestehen, eine globale Skalierung um einen festen Faktor in allen drei Hauptachsen als eigene Transformation zu betrachten, um zusätzliche Optimierungsmöglichkeiten (bspw. bei zwei unterschiedlich dimensionierten Datensätzen) zu erhalten.

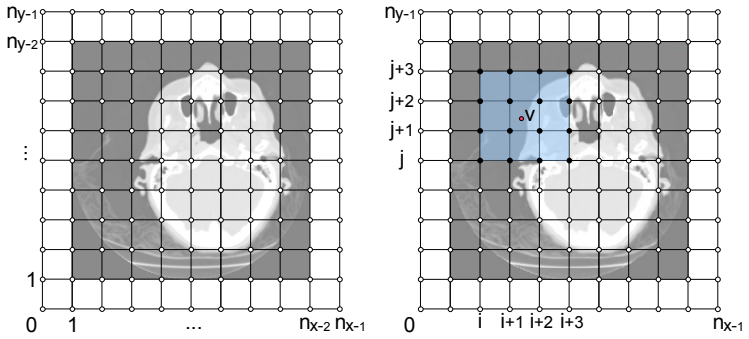
Das Verfahren endet, wenn keine weitere Verbesserung mehr erreicht werden konnte oder alternativ eine vorgegebene Iterationstiefe erreicht wurde. Dieses Verfahren ist grundsätzlich anfällig gegenüber lokalen Extrema, daher ist es empfehlenswert bei stark abweichenden Datensätzen eine grobe manuelle Vorregistrierung vorzunehmen. Das Abbruchkriterium kann auch dahingehend verändert werden, dass nach dem Erreichen der vorgegebenen Iterationstiefe ein weiterer Durchlauf gestartet wird, um so dem Algorithmus die Möglichkeit zu geben das lokale Maximum zu verlassen.

4.5.3.2. Elastische Registrierung

Während affine Transformationen immer das gesamte Bild verändern, sind in der elastischen Registrierung lokale Deformationen erlaubt und erwünscht. Gefordert wird jedoch üblicherweise der Erhalt der lokalen Nachbarschaft, um Zusammenhangskomponenten als solche zu belassen. Anschaulich dargestellt, kann man sich vorstellen das V_1 aus einem gummiartigen Material besteht, das sich lokal durch Strecken, Verdichten oder Verdrehen ändern lässt. Alternative Verfahren benutzen dafür auch fluidale oder diffusive Ansätze (siehe z.B. [26], Kapitel 4.5).

Das hier verwendete Verfahren wurde erstmalig in der Arbeit von Rueckert et al. [67] vorgestellt und verwendet eine auf kubischen B-Splines basierende Transformation. Dabei wird zunächst ein grobes Gitter von Kontrollpunkten über das Volumen gelegt. Die Kontrollpunkte werden so lange verschoben, bis keine weitere Verbesserung mehr erzielt werden kann. Das Gitter wird schrittweise verfeinert, bis eine vorgegebene Grenze erreicht wurde.

Diese Vorgehensweise erscheint auf den ersten Blick sinnvoll und zugleich rechenaufwendig zu sein. Auch ein Mensch würde bei dieser Aufgabe vermutlich zuerst grobe Abweichungen ausgleichen und dann feinere Differenzen beseitigen. Für den Algorithmus bedeutet dies, dass in jedem Iterationsschritt die Auswirkung der Translation jedes Kontrollpunktes in $x-$, $y-$ und $z-$Richtung auf eine mögliche Verbesserung geprüft werden muss. Selbst bei einem relativ groben Gitter von 10*10*10 Kontrollpunkten wäre dies ja bereits ein Optimierungsverfahren mit 3000 Freiheitsgraden, wo zusätzlich in jedem Schritt die durch die Veränderung resultierende *mutual information* neu berechnet werden müsste. Da dies nicht in vertretbarer Zeit zu lösen ist, wird stattdessen ein gradientenbasiertes Verfahren zur Verschiebung der Kontrollpunkte gewählt.



(a) Kontrollpunkt Gitter (b) Einflussbereich

Abbildung 4.8.: Kontrollpunktgitter bei der elastische Registrierung

Um dieses Verfahren zu verstehen, muss zuerst die Art und Weise definiert werden, wie die Kontrollpunkte auf den Atlas einwirken. Sei

$$\Phi = \{\Phi_{i,j,k} \in \mathbb{R}^3, 0 \leq i \leq n_x, 0 \leq j \leq n_y, 0 \leq k \leq n_z\}$$

ein Kontrollpunktgitter mit den Dimensionen n_x, n_y und n_z so über den Volumendatensatz platziert, dass in jeder Dimension jeweils der zweite und vorletzte Kontrollpunkt gerade mit dem Datensatz abschließen, wie in Abbildung 4.8 a) für eine transversale Schicht gezeigt wird. Damit ergeben sich die Abstände zwischen zwei Kontrollpunkten eines diskreten Volumendatensatzes V mit den Dimensionen dim_x, dim_y und dim_z als

$$\delta_x = \frac{dim_x}{n_x - 3}, \delta_y = \frac{dim_y}{n_y - 3} \text{ und } \delta_z = \frac{dim_z}{n_z - 3}$$

Die Position eines jeden Voxels im Volumendatensatz wird durch die jeweils 2 nächsten Kontrollpunkte in jeder Richtung beeinflusst (Abbildung 4.8 b). Sei $v = (v_x, v_y, v_z)$ eine Position aus V, dann gilt für die elastische Transformation

$$T_e(v) = \sum_{l=0}^{3} \sum_{m=0}^{3} \sum_{n=0}^{3} B_l(i_r) B_m(j_r) B_n(k_r) \Phi_{i+l,j+m,k+n} \tag{4.43}$$

In dieser Gleichung stehen $B_0, \ldots, B_3$ für die B-Spline Polynome dritter Ordnung mit dem Trägerintervall $[0, 1)$:

$$\begin{aligned} B_0(t) &= (-t^3 + 3t^2 - 3t + 1)/6 \\ B_1(t) &= (3t^3 - 6t^2 + 4)/6 \\ B_2(t) &= (-3t^3 + 3t^2 + 3t + 1)/6 \\ B_3(t) &= t^3/6 \end{aligned} \tag{4.44}$$

i, j und k stehen für die Indizes der v enthaltenden Zelle mit den relativen Koordinaten i_r, j_r und k_r mit

$$i = \left\lfloor \frac{x}{\delta_x} \right\rfloor,\ j = \left\lfloor \frac{y}{\delta_y} \right\rfloor,\ k = \left\lfloor \frac{z}{\delta_z} \right\rfloor \text{ und } i_r = \frac{x}{\delta_x} - \left\lfloor \frac{x}{\delta_x} \right\rfloor,\ j_r = \frac{y}{\delta_y} - \left\lfloor \frac{y}{\delta_y} \right\rfloor,\ k_r = \frac{z}{\delta_z} - \left\lfloor \frac{z}{\delta_z} \right\rfloor$$

Da jeder Kontrollpunkt nur ein begrenztes Gebiet beeinflusst, ist es nicht nötig, in jedem Schritt das gesamte Verbundhistogramm (auf dem die *mutual information* aufbaut) neu zu bestimmen, sondern nur die Veränderung im Kontrollbereich zu bewerten. Dennoch wäre eine „Best Neighbour"-Strategie, wie sie im rigiden Teil der Registrierung angewendet wurde, aufgrund der großen Zahl an Freiheitsgraden immer noch zu aufwendig, da in jedem Optimierungsschritt nur *ein* Kontrollpunkt in jeweils eine Richtung verschoben würde. Stattdessen wird für jeden Kontrollpunkt die Veränderung der Ähnlichkeit in allen drei Achsenrichtungen bestimmt und die Translationsrichtung über den Gradient

$$\nabla f = \frac{\partial f}{\partial \Phi} \tag{4.45}$$

$$= \frac{\partial I(T_e(V_1), V_2)}{\partial \Phi} \tag{4.46}$$

bestimmt. Wie in Kapitel 3.3.5 beschrieben, zeigt der Gradient ∇f in die Richtung der größten (positiven) Ähnlichkeitsänderung, seine Länge $|\nabla f|$ ist ein Maß für die Änderungsgröße. Dies lässt sich ausnutzen, um die Kontrollpunkte Φ in jedem Iterationsschritt um ein wählbares μ in Richtung des Gradienten zu verschieben, d.h. in jeder Iteration verändern sich *alle* Kontrollpunkte. Auch hier wird der Gradient wieder durch die Verwendung der zentralen Differenzen abgeschätzt. Da es sich bei f um eine kontinuierliche Funktion handelt, kommt dafür die kontinuierliche Form nach Kiefer/Wolfowitz aus [30] zum Einsatz.

Der Algorithmus wird abgebrochen, wenn das Maß für die Änderungsgröße $|\nabla f|$ unter eine vorgegebene Schranke ϵ fällt und so die nächste Transformationsiteration nur noch eine geringe Verbesserung des Ähnlichkeitsmaßes zur Folge hätte. Nach dem Abbruch wird das Gitter verfeinert und die Gradientenoptimierung erneut durchlaufen, bis eine ebenfalls vorgegebene Verfeinerungstiefe erreicht wurde. Für die Verfeinerung wird zwischen je zwei bereits existierenden Kontrollpunkten ein weiterer Punkt (in allen drei Achsenrichtungen) eingefügt. Dabei muss darauf geachtet werden, dass die bisherige Deformation des Gitters nach der Verfeinerung erhalten bleibt (siehe [50], S.36).

Listing 4.6 fasst das Verfahren noch einmal im Pseudo-Code zusammen. Der größte Rechenaufwand steckt in diesem Verfahren eindeutig in der Berechnung des Gradienten, lässt sich jedoch durch die im folgenden Kapitel beschriebenen Optimierungsstrategien reduzieren.

```
ControlPoints elastic_registration(VoxelCube V1, VocelCube V2,
      int grid_dim, double μ, double ε, int refinements) {
  ControlPoints phi = new ControlPoints(grid_dim);
  Zielfunktion f = new Zielfunktion(V1, V2, phi);

  for (int i=0; i<refinements; i++) {
    Gradient ∇f = f.calc_gradient();
    while (nabla.length() < ε) {
      phi.add(μ * ∇f/|∇f|);
      ∇f = f.calc_gradient();
    }

    if (i<refinements-1) phi.increase_resolution();
  }

  return phi;
}
```

Listing 4.6: Pseudocode für elastische Registrierung

4.5.4. Optimierung, Auswertung und Ausblick

Das atlasbasierte Segmentierungsmodul wurde als letztes in die YaDiV-Plattform integriert. Dennoch lassen sich bereits einige Aussagen über Verhalten und Effizienz der eingesetzten Verfahren treffen.

Sowohl die rigide, als auch elastische Registrierung arbeiten auf aktuellen Systemen hinreichend schnell, um klinisch relevante Datensätze zu bearbeiten. Dies wurde vor allem dadurch erreicht, dass für die Berechnung des Verbund-Histogramms nur ein wählbarer Prozentsatz der transformierten Voxel verwendet wurde. Erste Versuche haben gezeigt, dass sich bereits mit 2% der Voxel optimale Ergebnisse erzielen lassen ([50], S.58). Auch hier wurde außerdem konsequentes Multi-Threading eingesetzt, so dass die parallelisierbaren Anteile – wie die Berechnung der Verbundhistogramme – von der auf aktuellen Mehrkernsystemen vorhandenen Hardware profitieren können.

Erste Tests, in denen ein verformter Datensatz als sein eigener Atlas verwendet wurde, haben ergeben, dass die Umsetzung der atlasbasierten Registrierung die Inverse der zur Verformung benutzten Transformation (und damit eine größtmögliche Ähnlichkeit) anstrebt, was die prinzipielle Praxistauglichkeit der Methode demonstriert.

Dies gilt jedoch nur, wenn Atlas und reale Aufnahme bereits hinreichend ähnlich sind. Wenn hingegen Atlas und Datensatz stark voneinander abweichen, dauert die Optimierung sehr lange oder schlägt komplett fehl, da die verwendeten Optimierungsstrategien anfällig für das Finden lokaler Maxima sind. Dies lässt sich umgehen, indem vom Benutzer

eine manuelle (rigide) Transformation vorgegeben wird. Die zu diesem Zweck implementierte Schnittstelle ist jedoch derzeit sehr technisch und verwendet bspw. traditionelle Texteingabefelder für die Angabe von Winkeln oder Skalierungsfaktoren.

Hier wäre eine intuitive, z.B. durch Maus oder Haptik unterstützte Schnittstelle wünschenswert. Gerade für eine schnelle manuelle Vor-Registrierung könnte eine durch Kraftrückkopplung unterstützte stereographische Umgebung eine neue Dimension in der Ergonomie der Mensch-Maschine-Schnittstelle einleiten.

4.6. Kombination verschiedener Verfahren

In klinischen Anwendungsfällen reicht meist ein einzelnes Verfahren nicht aus, um das gewünschte Ergebnis zu erzielen. Oft ist es erforderlich, unterschiedliche Techniken zu kombinieren. Ein nützliches Hilfsmittel ist es daher, die Verknüpfung unterschiedlicher Segmente durch die traditionellen boolschen Operationen zu ermöglichend. Ein Segment kann

1. zu einem anderen Segment hinzugefügt werden (+ bzw. oder Verknüpfung),
2. von einem anderen Segment abgezogen werden (− bzw. negiertes und),
3. invertiert bzw.
4. komplett durch ein anderes Segment ersetzt werden.

Abbildung 4.9 demonstriert die boolsche Addition bzw. Subtraktion anhand eines Beispiels, in dem zum besseren Verständnis zwei einfache, geometrische Segmente erzeugt wurden. Im ersten Fall wird das blaue Segment (Kugel) zum grünen Segment (Würfel) hinzugefügt (oder), im zweiten Fall abgezogen (negiertes und).

4.7. Identifikation von Zusammenhangskomponenten

Ebenfalls oft erforderlich ist die Zerlegung eines Segments in seine einzelnen Zusammenhangskomponenten, die nur der Vollständigkeit halber kurz beschrieben werden soll. Der traditionelle Ansatz, um dies zu erreichen ist ein Durchlauf des BitCubes, bei dem jede neu gefundene Zusammenhangskomponente durch einen Flächenbrand- oder Floodfill-Algorithmus (siehe z.B. [65]) markiert wird, bevor mit dem Durchlauf fortgefahren wird. Der BitCube eines Segments kann dabei als Schwarz-Weiß Bild aufgefasst werden.

Das Verfahren ist sehr einfach, muss aber aus praktischen Gründen (da es sich um Volumendaten handelt) analog zur Region Grow Methode (Kapitel 4.3, S. 49) als iteratives

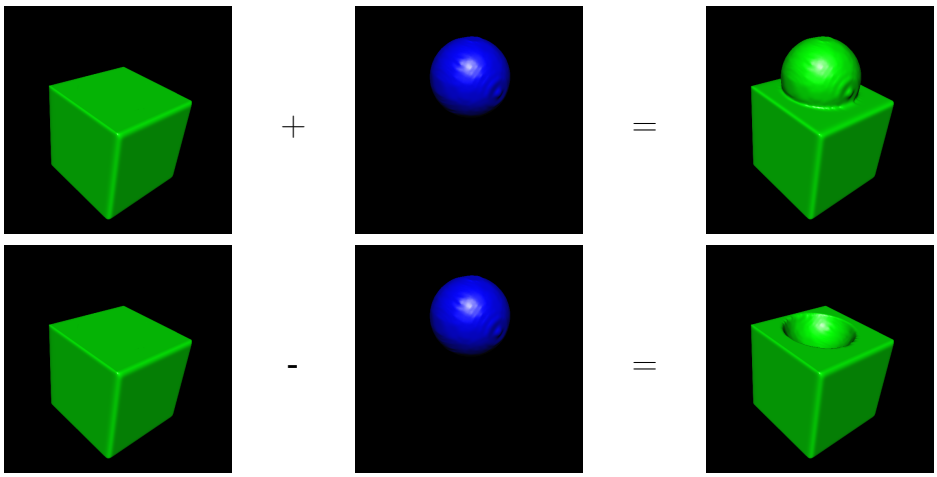


Abbildung 4.9.: Verknüpfung von Segmenten durch boolsche Operationen

Verfahren implementiert werden. Eine alternative Methode zur Findung von Zusammenhangskomponenten wurde u.a. in der Bachelorarbeit von Sebastian Intas am Lehrstuhl für Graphische Datenverarbeitung vorgestellt.

5. Visualisierung

Die angemessene Visualisierung der durch die bildgebenden Verfahren gewonnenen Daten ist eine der elementarsten und zugleich komplexesten Aufgaben medizinischer Bildverarbeitender Systeme. Sie bildet die Grundlage sowohl für Analyse und Diagnostik als auch für die Segmentierung, wo sie zusätzlich die Voraussetzung für eine schnelle Plausibilitätskontrolle bildet.

Viele Mediziner sind es aus der klassischen Röntendiagnostik weiterhin gewohnt, mit 2D-Schichtbildern zu arbeiten. Dennoch erhalten 3D-Visualisierungen und die sich daraus ergebenden neuen Diagnostikmöglichkeiten eine zunehmende Bedeutung.

In diesem Kapitel sollen die Grundlagen für die verschiedenen Visualisierungstechniken vermittelt und einen Einstieg in die entsprechende Literatur gegeben werden.

5.1. Datenvisualisierung vs. Segmentvisualisierung

Unabhängig von 2D- oder 3D-Darstellung stellt sich die Frage, was genau eigentlich visualisiert werden soll. Die Antwort auf diese Frage wird natürlich im wesentlichen durch die Anwendung bestimmt. In der Diagnostik werden zunächst oft die reinen Rohdaten dargestellt, d.h. ein Visualisierungsverfahren angewandt, das nur die direkt durch das bildgebende Gerät aufgenommenen Daten auswertet. Sind im aufgenommenen Bereich bereits Segmente identifiziert worden, können diese ebenfalls visualisiert werden, bspw. indem deren Oberfläche als Dreiecksfläche approximiert wird. Letzteres wird sowohl in der Diagnostik als auch in CAD Anwendungen benutzt (bspw. für die Entwicklung von Prothesen).

In komplexeren Visualisierungen werden sowohl Rohdaten als auch deren Segmentzugehörigkeit für die Darstellung verwendet. Mag dies im 2D-Fall noch einfach sein, bspw. indem der segmentierte Bereich eingefärbt wird, kommen hier in der 3D-Darstellung meist aufwendigere Verfahren (bspw. Raycasting) zum Einsatz. Dort kann die Segmentinformation über ein (künstliches) Material und eine konfigurierbare Gewichtung in die Farb- und Schattierungsberechnung eingehen, um so Regionen hervorzuheben, die sich nicht durch einfache Intensitätswertbereiche identifizieren lassen.

5.2. 2D-Darstellung

In der klassischen 2D-Visualisierung wird ein einzelnes Schichtbild aus einer Serie dargestellt. Die einfachste und natürlichste Form der Projektion ist es, ein Bild aus einer einzelnen DICOM-Datei darzustellen. Die Projektionsebene der Visualisierung stimmt dabei mit der Aufnahmeebene überein. Die gängige Alternative sind die dazu orthogonalen Ebenen. Diese orthogonalen Projektionsebenen für 2D-Visualisierung werden in der Medizin mit transversal, sagittal und frontal bezeichnet, deren Lage in Abbildung 5.1 skizziert wird.

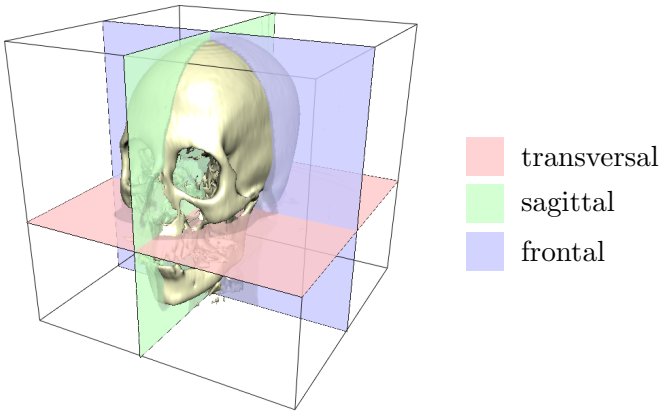


Abbildung 5.1.: Projektionsebenen für die 2D-Visualisierung

Motivation für die Benennung der Projektionsebene ist deren Orientierung gegenüber dem Patienten. Als transversal bezeichnet man eine Ebene, die senkrecht zur Längsachse eines Menschen steht und so den stehenden Körper in einen oberen und einen unteren Bereich teilt. CT und MRT Geräte liefern meist transversale Schichtbilder. Ein Schnittbild aus der sagittalen Ebene (lat.: sagittal = „pfeilwärts“) zeigt eine seitliche Aufnahme des Körpers während ein frontales Bild quasi vor der Person stehend aufgenommen wurde (und die Projektionsebene damit den Körper in „vorne“ und „hinten“ unterteilt). Abbildung 5.2 zeigt einen CT-Datensatz in transversaler, sagittaler und frontaler Ansicht.

5.2.1. Grauwertdarstellung

Ein gängiges Verfahren, um die aufgenommenen Daten zu visualisieren, ist die Darstellung als Grauwertbild, bei der „schwarz“ üblicherweise für Gewebe mit niedriger und „weiß“ für Materialien mit hoher Dichte steht. Bilder in dieser Darstellungsform lehnen sich an die konventionelle Röntgentechnik an und gelten als intuitiv verständlich.

Der Wertebereich eines Schichtbildes variiert mit der Aufnahmetechnik und liegt bspw. bei CT-Bildern üblicherweise zwischen 0 und 2^{12} (siehe auch Seite 29). Mit einem handels-

transversal sagittal frontal

Abbildung 5.2.: 2D Visualisierung in transversaler, sagittaler und frontaler Ansicht

üblichen PC und Monitor können im RGB-Modell jedoch nur 2^8 Graustufen dargestellt werden, d.h. bei der Visualisierung gehen Informationen verloren. Je nach Anwendung, bspw. in der Tumordiagnostik, kann dies schwerwiegende Folgen haben.

Für dieses Problem gibt es unterschiedliche Lösungen. Zum einen gibt es spezielle Befundungsmonitore, die neben einer hohen Auflösung und Blickwinkelunabhängigkeit auch mehr Graustufen als handelsübliche Monitore darstellen können. Experimente haben gezeigt, dass das menschliche Auge im Luminanzspektrum eines Monitors theoretisch bis zu 1023 Graustufen unterscheiden könnte (siehe [17] und [45]), daher gibt es Befundungsmonitore, die $2^{10} = 1024$ Graustufen darstellen können und die darüber hinaus über eine spezielle Grafikkarte angesteuert werden müssen. Leider wurden die meisten Untersuchungen in diesem Bereich von Monitorherstellern durchgeführt und lediglich als Technical- bzw. White-Paper veröffentlicht.

Aber auch bei 2^{10} darstellbaren Graustufen auf dem Monitor ist die Zahl der aufgenommenen Intensitätsstufen meist höher. Um diese für die Visualisierung aufzubereiten, wird üblicherweise mit einem Intensitätsfenster gearbeitet, das die **Values of Interest" (VOI)** überdeckt (siehe auch [57], PS 3.3, Anhang C.11.2.1.2). Dabei wird über einen Fenstermittelpunkt (Window Center) und die Fensterbreite (Window Width) eine Region definiert, die für den Betrachter von besonderem Interesse ist, bspw. 1000 HU als Fenstermittelpunkt mit Fensterbreite 1000 HU für einen Knochenchirurgen (siehe Tabelle 3.1, Seite 30) oder ein Weichgewebefenster für die Tumordiagnostik.

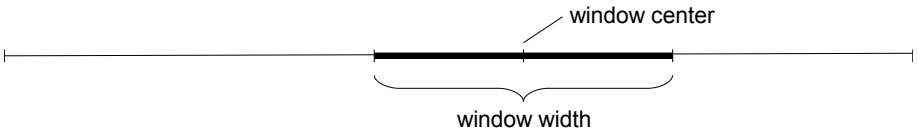


Abbildung 5.3.: Fenster im Intensitäts-Werteraum

Innerhalb des so definierten Fensters können die Intensitätswerte linear auf Grauwerte abgebildet werden. Alle Intensitätswerte unterhalb des Fensters werden auf schwarz, oberhalb des Fensters auf weiß abgebildet. Mathematisch ausgedrückt: Sei $G := [0 \dots n]$ die

Menge darstellbarer Grauwerte und c das Zentrum und w die Breite des VOI-Fensters. Dann ergibt sich der zu einem Intensitätswert i gehörende Grauwert $f(i)$ wie folgt:

$$f(i) = \begin{cases} i < c - 0.5 - (w-1)/2: & 0 \\ i > c - 0.5 + (w-1)/2: & n \\ sonst: & \frac{i-(c-0.5)}{(w-1)+0.5} * n \end{cases}$$

Abbildung 5.4 demonstriert die Auswirkung des VOI-Fensters auf das CT Bild eines Brustkorbes. Je nach Wahl des Fensters lassen sich die Details in unterschiedlichen Bildbereichen besser bzw. schlechter unterscheiden. Unter den einzelnen Aufnahmen ist das jeweilige Fenster (blau eingefärbt) über dem Voxel-Histogram eingeblendet.

schmales Gewebefenster schmales Lungenfenster breites Gesamtfenster

Abbildung 5.4.: Graustufendarstellung mit VOI-Fenster im Intensitätsraum

Wenn die Breite des VOI-Fensters größer als die Anzahl darstellbarer Graustufen ist, lässt sich ein Informationsverlust nur durch die Wahl einer anderen Darstellungsform vermeiden.

5.2.2. Falschfarbendarstellung

Eine weitere Methode, um Intensitätswerte abzubilden, ist die Falschfarbendarstellung (auch als Pseudo-Farben-Darstellung bezeichnet). Dabei wird der Werteraum nicht mehr in eine Schattierung zwischen Schwarz und Weiß, sondern auf eine im Prinzip frei wählbare Palette von Farbübergängen abgebildet.

Auch in der Falschfarbendarstellung kann ein VOI-Fenster verwendet werden. Voraussetzung für die Farbpalette ist eine Kodierung der Intensitätswerte auf eine indizierte Farbmenge, die linear aufsteigend definiert wird. Daher lässt sich die Funktion $f(i)$ aus dem vorigen Abschnitt übertragen.

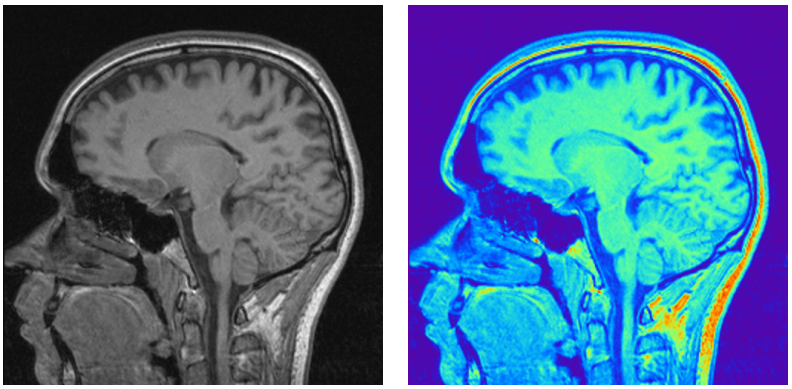


Abbildung 5.5.: Graustufen- und Falschfarbendarstellung bei identischem VOI-Fenster

Der Vorteil dieser Methode liegt darin, dass in der Darstellung mehr Intensitätswerte unterschieden werden können, wie in Abbildung 5.5 am Beispiel eines MRT-Bildes zu sehen ist. Dies geht jedoch auf Kosten der Intuitivität; der Betrachter muss erst lernen, welche Farb- bzw. Helligkeitskombination eine größere / niedrigere Intensität bedeutet („war dunkel-lila mehr oder weniger als blass-grün?“). Zudem wird es schwieriger, Teile des Bildes, wie z.B. Segmente, gesondert hervorzuheben, wenn die gewählte Farbtabelle bereits den gesamten darstellbaren Farbraum ausfüllt.

5.2.3. Skalierung

Die Skalierung der 2D-Bilddaten auf den Darstellungsbereich scheint auf den ersten Blick trivial zu sein. Nach der Anwendung des VOI-Fensters und der Umwandlung in eine Graustufen- bzw. Falschfarben-Darstellung, liegen die Schichtbilddaten als diskrete, zweidimensionale (RGB-) Daten vor und müssen nur noch in den Darstellungsbereich eingepasst werden.

Der erste Schritt hierfür ist eine Skalierung in x- bzw. y-Richtung entsprechend dem im Datensatz vorliegenden *pixel spacing* bei transversaler bzw. zusätzlich der *slice thickness* (siehe Kapitel 3.1.5) bei frontaler und sagittaler Ansicht. Durch diesen Schritt werden die Dimensionen des gerenderten Bildes der Abtastrate des bildgebenden Gerätes in x-, y- und z-Richtung angepasst.

Für einen optionalen zusätzlichen Vergrößerungs-Effekt, beispielsweise in einer „Lupe“-Funktion, können unterschiedliche Rekonstruktionsfilter (Kapitel 3.3.4) verwendet werden. Für eine optisch glatt wirkende Darstellung werden dafür meist bilineare oder bikubische Filterkerne eingesetzt. Als visuelle Schnittstelle besitzt auch die klassische Nearest Neighbour Method für viele elementare Aufgaben, wie zum Beispiel der (präzisen) Selektion eines einzelnes Voxel Vorteile, da hier die Glättungseffekte der anderen Filter störend wirken.

5.2.4. Segmentvisualisierung

In der 2D-Segmentvisualisierung wird üblicherweise jedem Segment eine eindeutige Farbe zugeordnet. In der klassischen Grauwertdarstellung kann nun ein Segment durch das Zeichnen der jeweiligen Segmentpixel in der jeweiligen Farbe hervorgehoben werden. Durch einen Transparenzeffekt, bei dem Grauwert und Segmentfarbe vermischt werden, lässt sich zusätzlich noch das ursprüngliche Grauwertbild unter dem Segment erahnen. Außerdem ist es so möglich, Segmentüberlappungen durch die entstehenden Mischfarben zu erkennen.

Abbildung 5.6 zeigt ein transversales Schichtbild von einem menschlichen Knie; als Segmente sind der Oberschenkelknochen (Femur), die Kniescheibe (Patella) und eine Arterie farblich hervorgehoben.

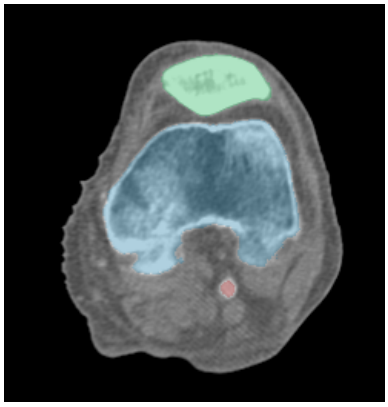


Abbildung 5.6.: Segmentvisualisierung in der 2D-Projektion

5.3. 3D-Darstellung

Bei der 3D-Visualisierung von Volumendaten unterscheidet man bei der Darstellung von Volumenkörpern zwischen indirekten und direkten Verfahren. Während bei den indirekten Verfahren der eigentliche Aufwand im Finden einer darzustellenden Oberfläche liegt, erzeugen die direkten Verfahren das Bild direkt aus den Volumendaten.

In den folgenden Kapiteln sollen Vertreter beider Kategorien vorgestellt und verglichen werden. Dabei werden die Verfahren besonders gründlich behandelt, die auch als Modul in die derzeit aktuelle YaDiV-Version integriert wurden. Auch die anderen der im Folgenden vorgestellten Algorithmen wurden (zumindest zu Vergleichszwecken) im Rahmen dieser Arbeit untersucht und implementiert.

5.3.1. Direct Volume Rendering

Direct Volume Rendering bezeichnet Verfahren für die 3D-Visualisierung von Volumendaten, die direkt auf dem Voxeldatensatz arbeiten, um daraus ein Bild zu erzeugen. Eine zentrale Rolle spielt dabei die sogenannte Transferfunktion, die jeder möglichen Voxelintensität eine Farbe sowie eine Deckkraft bzw. Transparenz zuordnet.

Das grundlegende Problem beim Direct Volume Rendering ist die Lösung des Volumen Rendering Integrals (VRI). Das VRI beschreibt, wie sich die Farbe eines Lichtstrahls beim Durchlaufen eines Mediums verändert. Eine Beschreibung des VRI für Lichtinteraktion von Medien mit verschiedenen Eigenschaften wie Absorption, Emission, Reflektion oder Streuung findet sich z.B. in [46]. Beim Rendern des Bildes muss das VRI abschnittsweise für jeden Pixel berechnet werden. Das direkte Lösen des Integrals ist im Allgemeinen nicht möglich, jedoch haben sich unterschiedliche Verfahren etabliert, die dies realisieren. Diese Verfahren sollen in den folgenden Kapiteln vorgestellt werden.

Ein ausführlicherer Überblick über die verschiedenen Techniken steht u.a. in [70]. In dieser Arbeit hat Dominik Sarnow verschiedene Direct Volume Rendering Verfahren implementiert und verglichen, die hier etwas verkürzt und zusammengefasst beschrieben werden sollen.

5.3.1.1. Emissions-Absorptionsmodell

Die meisten Direct Volume Render Verfahren basieren auf einem Emissions-Absorptionsmodell. Die Lichtintensität an dem Punkt x mit der Wellenlänge h in Richtung n lässt sich als Funktion $I(x, n, h)$ darstellen, die es zu bestimmen gilt. Die Lichtabsorption an diesem Punkt

$$A(x, n, h) = \alpha(x, n, h) + \sigma(x, n, h) \tag{5.1}$$

setzt sich zusammen aus der tatsächlichen Absorption α und der Streuung σ. Die Licht-Emission E ist definiert als Summe des gerichteten Lichts aus einer Quelle q und einem zusätzlichen Lichtanteil durch Streuung r:

$$E(x, n, h) = q(x, n, h) + r(x, n, h)$$

Die Übertragung der Strahlungsenergie entlang eines Sichtstrahls s lässt sich damit als Differentialgleichung schreiben:

$$\frac{\partial}{\partial s} I(x, n, h) = -A(x, n, h) * I(x, n, h) + E(x, n, h) \tag{5.2}$$

Bei der Streuung des Lichts handelt es sich um einen komplexen Vorgang der sowohl die Richtung n als auch die Wellenlänge h ändern kann. Im Absorption-Emissionmodell wird diese Streuung ignoriert und als Konsequenz daraus auch h. Damit erhält man die Differentialgleichung

$$I(s) = I(s_0) * e^{-\tau(s_0,s)} + \int_{s_0}^{s} q(\tilde{s}) * e^{-\tau(\tilde{s},s)} d\tilde{s} \tag{5.3}$$

mit der optischen Tiefe

$$\tau(s_0, s_1) = \int_{s_0}^{s_1} \alpha(s) ds \tag{5.4}$$

Für eine numerische Lösung muss eine Diskretisierung entlang des Strahls stattfinden. Das Intervall wird in n Teile zerlegt. Die Intensität I kann dann an den diskreten Punkten s_k iterativ mittels

$$I(s_k) = I(s_{k-1}) e^{-\tau(s_{k-1},s_k)} + \int_{s_{k-1}}^{s_k} q(s) * e^{-\tau(s,s_k)} ds, \tag{5.5}$$

berechnet werden. Der Term

$$t_k = e^{-\tau(s_{k-1},s_k)} \tag{5.6}$$

entspricht der Transparenz. Der Emissionsterm

$$b_k = \int_{s_{k-1}}^{s_k} q(s) * e^{-\tau(s,s_k)} ds, \tag{5.7}$$

beschreibt den Zuwachs der Energie, verursacht durch Emissionen im Intervall $[s_{k-1}, s_k]$ des Strahls.

Damit hat man den theoretischen Hintergrund für das Dichte-Emitter-Modell [68], nachdem ein Volumenobjekt als ein Raum mit Licht emittierenden Partikeln interpretiert wird, welche durch eine Dichtefunktion $\rho(s)$ beschrieben werden. Die Terme q und α sind dann proportional zur Partikeldichte

$$\alpha(s) = \alpha_0 * \rho(s) \text{ und } q(s) = q_0 * \rho(s) \tag{5.8}$$

mit α_0 und q_0 konstant. Eingebracht in den Emissionsterm entspricht dies

$$b_k = \frac{q_0}{\alpha_0} * (1 - t_k). \tag{5.9}$$

Zusammengefasst ergibt sich

$$I(s_k) = I(s_{k-1}) * t_k + b_k = I(s_{k-1}) * t_k + \frac{q_0}{\alpha_0}(1 - t_k). \tag{5.10}$$

Dies führt zu einer Funktion, die Voxelintensitätswerten D im Datensatz Emissions- und Absorptionswerte zuordnet.

$$I_{emit} = T_{emission}(D) \text{ und } t = T_{absorption}(D) \tag{5.11}$$

5.3.1.2. Transferfunktion

Die Funktion, die jedem Voxelwert einen bestimmten Emissions-Absorptionswert zuordnet, heißt **Transferfunktion**. Theoretisch kann die Transferfunktion eine beliebige Anzahl von verschiedenen Variablen als Argumente haben, jedoch sollten diese, um dem Begriff „Direct Volume Rendering“ gerecht zu bleiben, direkt aus den Volumendaten berechnet werden können. Den Prozess der Berechnung dieser Werte bezeichnet man als **Klassifikation**. Je nachdem, ob die Transferfunktion vor bzw. nach der Interpolationsphase angewendet wird, spricht man von **Pre-** oder **Post-Klassifikation**. Beide Verfahren führen zu unterschiedlichen Ergebnissen.

Das Finden einer geeigneten Transferfunktion ist, trotz einiger Versuche, diesen Schritt zu automatisieren, immer noch ein manueller und zeitaufwendiger Prozess, der nicht nur genaues Wissen über die vorhandenen Strukturen im Datensatz erfordert, sondern auch von der Visualisierungsanwendung abhängt, bspw. der Analyse eines konkreten medizinischen Datensatzes oder der Erzeugung einer Lehrbuchgrafik im illustrativen Stil. Deshalb ist ein Mechanismus wünschenswert, der es erlaubt, die Transferfunktion interaktiv während der Render Phase zu modifizieren.

In der Praxis werden Transferfunktionen meist als Look-Up-Tabellen mit fester Größe realisiert, auf die mit konstantem Aufwand zugegriffen werden kann.

5.3.1.3. Volume Ray Casting

Bildlich beschrieben lautet die Grundidee der Ray Casting Verfahren, einen imaginären Strahl durch jeden Pixel des zu berechnenden Bildes zu verfolgen, der ausgehend vom Betrachter den Volumendatensatz durchläuft und dabei einen Farbeindruck erzeugt (Abbildung 5.7). Das Verfahren ist somit - nicht nur namentlich - eng verwandt mit Konzepten aus dem Bereich Ray Tracing. Während die Begriffe Ray Casting und Ray Tracing in der früheren Fachliteratur [18] zum Teil synonym verwendet wurden, wird in neueren Arbeiten versucht, beide Verfahren gegeneinander abzugrenzen (z.B. in [5]). Dabei scheint ein Konsens zu bestehen, Ray Casting als die Teilmenge der Ray Tracing-Verfahren zu verstehen, bei der keine Reflektionen berücksichtigt werden, sprich der Strahl nach der ersten Interaktion mit der Oberfläche einer Struktur nicht weiter verfolgt wird.

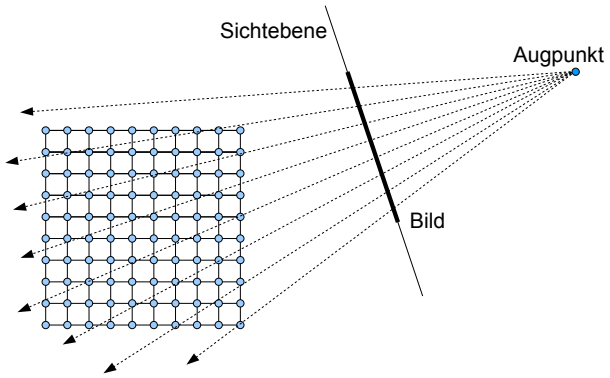


Abbildung 5.7.: Prinzip Ray Casting

Volume Ray Casting-Verfahren sind historisch gesehen die ersten Vertreter aus der Klasse der Direct Volume Renderer. Eines der ersten Paper war die Arbeit von Kajiya und von Herzen. In ihrem 1984 veröffentlichten Paper [28] heißt es im Abstract: *„This paper presents new algorithms to trace objects represented by densities within a volume grid, e.g. clouds, fog, flames, dust, particle systems. We develop the light scattering equations, discuss previous methods of solution, and present a new approximate solution to the full three-dimensional radiative scattering problem suitable for use in computer graphics.”*. Es wurde ein Verfahren dargestellt, um Volumendaten, die als Dichtewerte in einem regulären Gitter vorliegen, zu visualisieren. Die Autoren zeigten dies am Anwendungsbeispiel von Gaswolken, hoben aber den allgemeinen Charakter des Verfahrens hervor, welches benutzt werden könne, um allgemein mathematische Modelle und Funktionen im dreidimensionalen Raum zu visualisieren. Etwas später folgte die Anwendung auf medizinische Daten, mit der Darstellung von Knochen und Gewebeoberflächen [35].

Bei der Verfolgung des Strahls durch den Volumendatensatz wird beim High-Quality Raycasting der Datensatz als kontinuierliche Intensitätsfunktion angenommen, die an regulären Gitterstellen ausgewertet wurde. Dabei spannen je acht Gitterpunkte einen Würfel

auf, an dessen Eckpunkten die Werte der Intensitätsfunktion bekannt sind. Um das Volumen Rendering Integral näherungsweise lösen zu können, muss die Intensitätsfunktion im Inneren des Würfels interpoliert werden. Dabei können verschiedene Interpolationsmethoden verwendet werden, die auch als **Rekonstruktionsfilter** bezeichnet werden (siehe Kapitel 3.3.4, Seite 35).

Neben der verwendeten Filtermethode ist vor allem die **Abtastrate** entscheidend für die Qualität der Visualisierung. Da das Volume Rendering Integral numerisch approximiert wird, entspricht die Abtastrate entlang des Sichtstrahls der Approximationsgüte. Eine höhere Abtastrate erhöht dabei sowohl die Bildqualität als auch die Berechnungszeit, daher muss hier ein in der Praxis sinnvolles Mittel gefunden werden.

Um den Rendervorgang zu beschleunigen, haben sich verschiedene Optimierungsstrategien etabliert. Eine einfache, aber sehr effiziente Methode ist die **Early Ray Termination**. Da bei der numerischen Integration des DVI Farb- und Alphakanäle getrennt integriert werden, kann die Berechnung entlang eines Sichtstrahls ab einem vorgegebenem Transparenz-Schwellwert abgebrochen werden, da weiter hinten liegende Zellen keinen nennenswerten Einfluß mehr auf den zu berechnenden Bildpunkt nehmen können. Dies ist auch der Grund, warum die schrittweise Berechnung des Renderintegrals in **Front-To-Back-Order** geschieht, also das Abtasten mit Punkten nah am Augpunkt beginnt.

Etwas aufwendiger, aber ebenfalls sehr verbreitet ist die Strategie des **Empty Space Skipping**. Da die meisten medizinischen Volumendatensätze ein Objekt in der Mitte des Scanbereiches aufnehmen, gibt es am Rand meist große, leere Räume, die bei der Strahlenverfolgung übersprungen werden können. Dafür wird zunächst in einem Präprozessing-Schritt eine Datenstruktur angelegt, in der diese *empty spaces* zugriffsoptimiert abgelegt werden können. Eine ausführlichere Erklärung dieser Optimierungsstrategie findet sich z.B. in [52].

Aktuell beschäftigen sich viele Arbeiten mit der Implementierung von Ray Casting Verfahren auf der GPU.

5.3.1.4. Splatting

Eine weiterer Vertreter aus der Klasse der Direct Volume Rendering Verfahren ist das sogenannte Splatting. Idee und Begriff gehen zurück auf die Arbeiten von Lee Westover ([85], [86]) aus dem Jahren 1990/1991, mit dem Ziel, eine effiziente Volumendarstellung mit einem sparsamen Speicherverbrauch zu kombinieren. Inzwischen sind Splatting Verfahren erheblich weiterentwickelt worden und werden auch in der High-Quality Volumenvisualisierung verwendet.

Das Prinzip des Splatting (dargestellt in Abbildung 5.8) ist auf den ersten Blick ähnlich zur Idee des Volume Ray Casting. Auch beim Splatting werden Strahlen betrachtet,

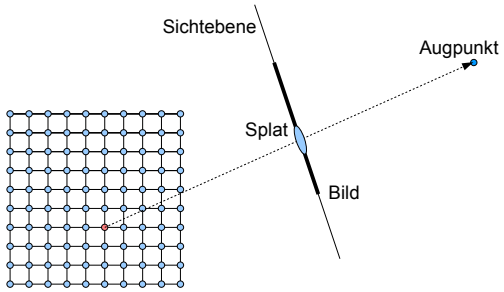


Abbildung 5.8.: Splatting-Prinzip

die den Augpunkt des Betrachters mit einem Voxel verbinden und dabei die Bildprojektionsebene durchdringen. Im Gegensatz zum Volume Ray Casting wird jedoch nicht vom Augpunkt ausgehend das Volume Rendering Integral gelöst, sondern, bildlich gesprochen, ein Farbtropfen vom Voxel in Richtung Augpunkt geschleudert und dessen „Splat“ auf dem in der Sichtebene liegendem Pixelbild berechnet. Auch hier werden Farb- und Transparenzwert separat berechnet: je mehr *Splats* sich in einem Bereich überlagern, desto stärker wird der entstehende Farbeindruck. Damit dabei die vom Augpunkt (bzw. der Sichtebene bei Parallelprojektion) aus weiter hinten liegenden Voxel nicht die vorderen verdecken, wird beim Splatting eine **Back-To-Front-Order** verwendet.

5.3.1.5. Shear Warp

Die Idee der Shear Warp Verfahren wurde erstmals auf der SIGGRAPH 1994 im Beitrag „Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation” von Philippe Lacroute and Marc Levoy vorgestellt [33].

Auch bei diesem Direct Volume Rendering-Verfahren ist die Idee namensgebend: um das Direct Volume Integral effizienter berechnen zu können, wird das Voxelgitter selbst zuerst einer Scherung (shear) unterworfen und anschließend perspektivisch verzerrt (warp), dargestellt in Abbildung 5.9. Wird eine Zentralprojektion verwendet, kommt noch ein zusätzlicher Skalierungsschritt hinzu (scale), wie in Abbildung 5.10 skizziert.

Als Zwischenergebnis erhält man ein neues Koordinatensystem (in [33] als „sheared object space“ bezeichnet), in dem alle Sichtstrahlen parallel zur z-Achse verlaufen. Anschließend wird analog zum Volume Ray Casting für jeden Bildpixel ein Sehstrahl durch das Volumen geschickt. Aufgrund der vorherigen Scherung und Verzerrung des Gitters läuft der Sehstrahl im sheared object space parallel zu den Achsen, was den Aufwand für die numerische Integration stark verringert.

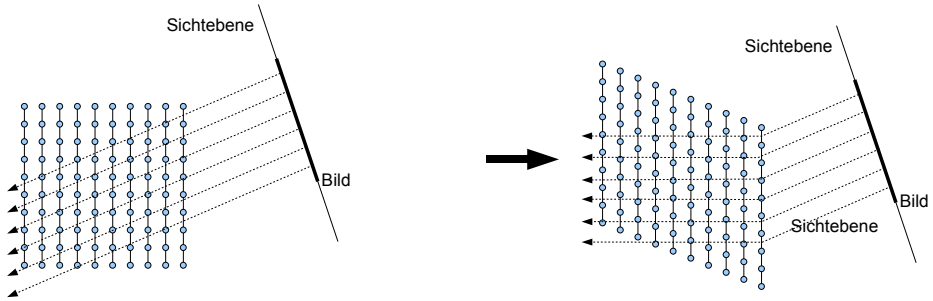


Abbildung 5.9.: Shear Warp (Parallelprojektion)

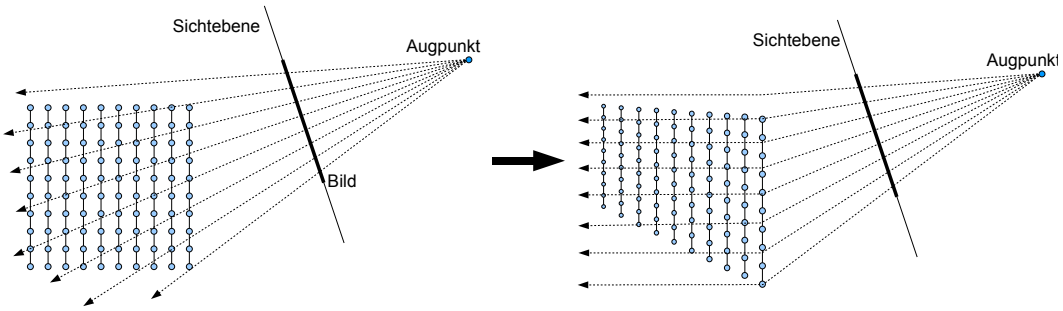


Abbildung 5.10.: Shear Warp (Zentralprojektion)

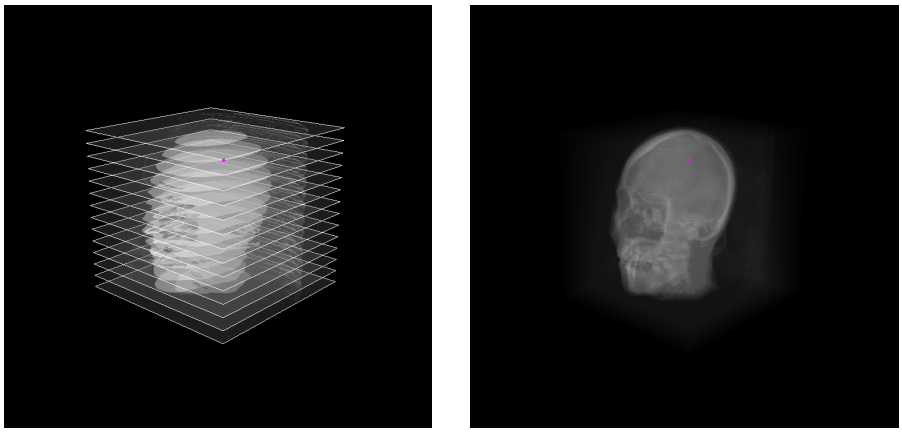
5.3.1.6. 2D-Texture Volume Rendering

Volume Raycasting, Splatting und auch Shear Warp eignen sich (ohne GPU Optimierung) nur bedingt für interaktive 3D-Visualisierung. Eines der ersten Verfahren, das dies leisten konnte, nutzte die 1994 sehr spezielle und teure Hardware der Silicion Graphics Onyx Workstation zur Beschleunigung von 2D-Texture-Mapping aus [10]. Da vergleichbare Hardware inzwischen in jeder handelsüblichen Grafikkarte verfügbar ist, haben auch die texturbasierten Direct Volume Rendering Verfahren ein großes Interesse erfahren.

Selbst günstigste Grafikkarten sind heutzutage in der Lage, ebene 3D-Polygone mit einer 2D-Textur zu zeichnen[1], die auch einen Alpha-Wert für die Transparenz besitzen kann. Daher liegt der Gedanke nahe, die Schichtbilddaten jeweils als einzelne 2D-Textur zu interpretieren. Die Texturen werden dann auf plane Quadrate (Quads) aufgetragen, wie in Abbildung 5.11 skizziert – hier wurden transversale Textur-Ebenen mit einem zum illustrativen Zweck sehr groß gewähltem Abstand verwendet.

Der Intensitätswert kann bei diesem Verfahren als Farb- und oder als Transparenzwert interpretiert werden. Um die Renderzeit und den Grafikkartenspeicherverbrauch zu reduzieren, lässt sich zudem die Anzahl der texturierten Ebenen reduzieren, bspw., indem

[1] Unterschiede zwischen den Modellen bestehen nur noch darin, welche Texturgrößen unterstützt werden bzw. ob auch Texturdimensionen verwendet werden können, die keinen 2er-Potenzen entsprechen.



(a) Projektions-Quads (b) Ergebnis

Abbildung 5.11.: Prinzip Texture2d Volume Rendering (transversale Projektionsebenen)

nur jedes zweite Schichtbild verwendet wird oder die Texturbilder in einer niedrigeren Auflösung gerendert werden.

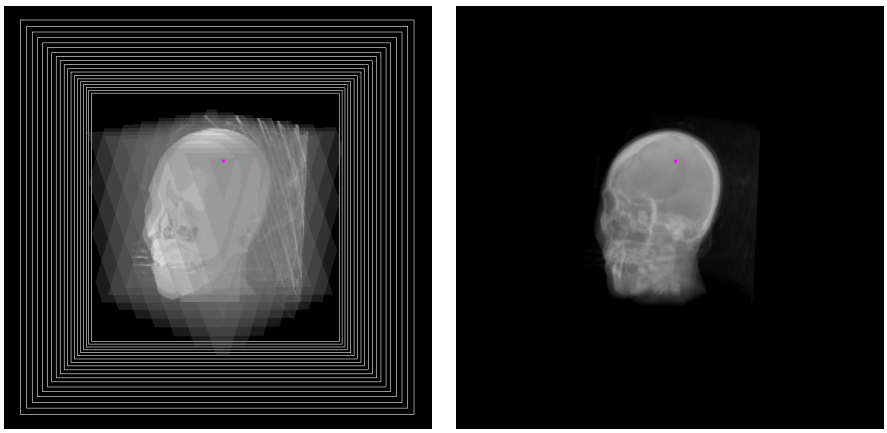
Volume Rendering mit 2D-Texturen arbeitet schnell und zuverlässig. Der visuelle Eindruck ist ausreichend; sind Texturen und Geometrie einmal im Grafikspeicher, ist ein flüssiges Navigieren möglich. Parameter wie Helligkeit, globale Ebenentransparenz etc. lassen sich zur Laufzeit interaktiv verändern. Durch die gute Unterstützung selbst auf älteren Grafikkarten ist das Verfahren sehr universell einsetzbar. In YaDiV wird es neben der Datenvisualisierung u.a. als schnelle Vorschau für die Auswirkung der Transferfunktion auf das zu erwartende Ergebnis einer hochqualitativen Volume Raycasting Berechnung genutzt.

Ein Nachteil sind die je nach Blickwinkel deutlich sichtbaren Lücken zwischen den Schichtebenen, die durch weitere Texture2D-Schichtobjekte in den sagittalen und frontalen Ebenen verhindert werden können, was die Anzahl an Texturen und damit den Grafikspeicherverbrauch verdreifacht. In der Vergrößerung eines Ausschnitts sind zudem meist störende Artefakte erkennbar.

5.3.1.7. 3D-Texture Volume Rendering

Neben den klassischen 2D-Texturen werden von allen aktuellen Grafikkarten auch 3D-Volumentexturen unterstützt, die intern wiederum aus Schichtbildern aufgebaut sind [87]. Auch hier wird eine festgelegte Anzahl an Projektions-Quads verwendet, diese sind jedoch immer parallel zur Sichtebene. Dies führt dazu, dass die auch hier zwischen den Quads vorhandenen Lücken für den Betrachter unsichtbar bleiben. Abbildung 5.12 zeigt dies an einem Beispiel, auch hier wurde der Abstand zwischen den orthogonalen Ebenen zum besseren Verständnis sehr groß gewählt. Durch eine optionale trilineare Filterung, die von

modernen Grafikkarten erst zum Teil unterstützt wird, können auch die Rekonstruktions-Artefakte deutlich reduziert werden.



(a) Projektions-Quads (b) Ergebnis

Abbildung 5.12.: Prinzip Texture3d Volume Rendering

Ein weiterer Vorteil der Anordnung der Projektionsebenen beim 3D-Textur-Verfahren hängt mit der Darstellung von (semi-)transparenten Objekten zusammen. Zwar wird Transparenz auch von 3D-Grafikkarten unterstützt, dabei muss jedoch eine bestimmte Zeichenreihenfolge eingehalten werden: Zuerst werden alle nicht-transparenten Primitiven (Dreiecke, Quads, etc.) ganz normal gezeichnet. Dann wird der Z-Buffer in den Nur-Lesen-Modus versetzt und die transparenten Primitiven aufsteigend zu ihrer Entfernung vom Augpunkt gezeichnet. Da die Grafikhardware dies nicht unterstützt, muss die Sortierung durch den Hauptprozessor erfolgen. Ein generelles Problem tritt auf, wenn sich zwei transparente Primitiven überschneiden, was unvermeidbar zu Darstellungsfehlern führt. In der klassischen 2D-Textur-Methode treten diese Überschneidungen immer dann auf, wenn mehrere (orthogonale) Projektionsebenen verwendet werden. Durch die parallele Anordnung der Ebenen in der 3D-Textur-Variante wird diese Situation vermieden. Um die Darstellungsqualität und insbesondere das Tiefenverständnis weiter zu verbessern, lassen sich auch Shading-Effekte simulieren (siehe z.B. in [21]).

Die maximal sinnvolle Anzahl der Projektionsgeometrien, die in der Darstellung mit 2D-Texturen durch die Anzahl der Schichtbilder in transversaler, sagittaler bzw. frontaler Ansicht gegeben wird, ist in der 3D-Texture-Variante durch den Durchmesser der Bounding Box des zu visualisierenden VoxelCubes begrenzt ([87], Abschnit 2.2.4). Dieser Extremfall tritt ein, wenn der Augpunkt des Betrachters auf der Bounding Box Diagonalen liegt.

Auch die Größe der Quads hängt direkt mit der Diagonalen der Bounding Box zusammen – und damit auch die Skalierung der Texturkoordinaten. Während bei 2D-Textur-Rendering Verfahren die Projektions-Quads (also deren Geometrie selbst) rotiert wer-

den, wenn der Augpunkt des Betrachters verändert wird, bleiben beim 3D-Texturing die Quad-Geometrien unverändert. Stattdessen wird eine zur Bewegung des Betrachters inverse Rotation auf den 3D-Textur-Koordinaten durchgeführt.

Bezüglich des Speicherverbrauchs haben beide Verfahren unterschiedliche Vor- und Nachteile. Für eine lückenfreie Darstellung müssen beim 2D-Textur-Verfahren sagittale, frontale und transversale Quads gezeichnet und mit Texturen versehen werden, was bei maximaler Darstellungsqualität (= maximaler Schichtbildanzahl) einer Verdreifachung des Grafikkartenspeicherbrauchs entspricht, da jeder Voxel in drei unterschiedlichen Texturen enthalten ist. Die 3D-Textur-Variante umgeht dieses Problem, allerdings skaliert der Speicherverbrauch hier nicht mit der Anzahl der Projektions-Quads wie bei der 2D-Textur-Variante. Durch die Rotation der Textur-Koordinaten bei der Augpunktrotation würde eine Auflösungsreduzierung zu sehr starken optischen Artefakten führen. Daher ist der Speicherbedarf unabhängig von der Anzahl Projektions-Quads konstant.

5.3.2. Indirect Volume Rendering

Der Schwerpunkt der aktuellen 3D-Grafikkarten liegt derzeit auf dem Zeichnen von polygonalen Grundprimitiven wie Linien, Dreiecken oder auch Quads. Die Alternative zu den Direct Volume Rendering Verfahren ist daher, vor der eigentlichen Visualisierung – sozusagen als Zwischenschritt – zunächst eine polygonale Oberflächendarstellung zu finden. In diesem Kontext spricht man auch vom Indirect Volume Rendering, da nicht die diskreten Volumendaten selbst dargestellt werden, sondern eine daraus (re-)konstruierte Oberfläche.

5.3.2.1. Marching Cubes

Das vermutlich bekannteste Indirect Volume Rendering Verfahren ist der sogenannte Marching Cubes-Algorithmus (MC), der 1987 von William Lorensen und Harvey Cline auf der Computer Graphics Konferenz vorgestellt wurde [41]. Die Marching Cubes-Technik wurde in den USA von der General Electric Company patentiert. Das Patent lief nach 20 Jahren am 5. Juni 2005 aus und ist im Nachhinein im Zuge der allgemeinen Diskussion über Softwarepatente umstritten. Da das Marching Cube-Verfahren in YaDiV vor allem für die schnelle Segmentvisualisierung intensiv genutzt wird, sollen die Grundbegriffe an dieser Stelle etwas genauer beschrieben werden.

Der Algorithmus erhält in seiner klassischen Form als Eingabe ein regelmäßiges dreidimensionales Gitter mit Skalaren, d.h. eine endliche Menge vierdimensionaler Vektoren, bei denen drei Komponenten als Koordinaten und die letzte als Intensitätswert aufgefasst werden. In diesem Dichtefeld wird nun eine Fläche gesucht, die einen vorgegebenen Intensitätswert w hat. Eine solche Fläche wird als Isofläche bezeichnet, die ein Inneres

Volumen (Voxel mit Intensitätswert kleiner als w) von einem Äußeren trennt. Da das innere Volumen nicht zusammenhängend sein muss, kann auch die Trennfläche mehrere Zusammenhangskomponenten besitzen.

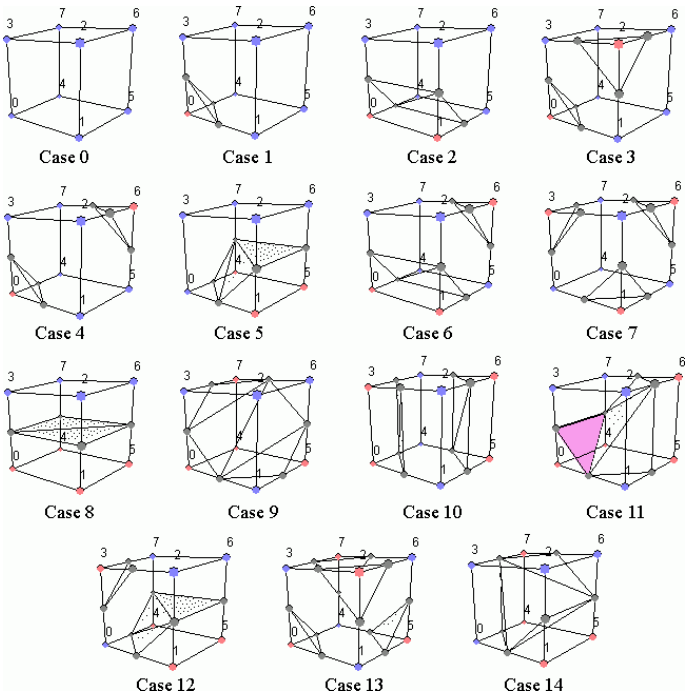


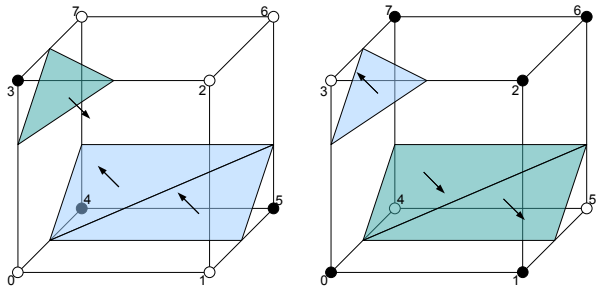
Abbildung 5.13.: Original MC Cases *Quelle: [41]*

Wie der Name schon andeutet, wird bei dem Verfahren ein durch den Volumendatensatz „wandernder Würfel" betrachtet. Kernidee des Verfahrens ist es, die Isofläche im inneren dieses Würfels losgelöst von dessen Nachbarschaft isoliert zu bestimmen und zu rekonstruieren. Dafür werden zuerst die Intensitätswerte an den Würfeleckpunkten betrachtet. Liegt ein Eckpunkt innen, muss er durch eine Trennfläche von den außen liegenden Eckpunkten separiert werden. Da der Würfel 8 Eckpunkte besitzt, die jeweils im Inneren (1) oder im Äußeren (0) liegen können, gibt es insgesamt $2^8 = 256$ unterschiedliche Variationen. Lorensen und Cline haben dazu die Dreieckskonfigurationen aus Abbildung 5.13 vorgeschlagen.

Es fällt auf, dass hier nur 15 Grundfälle[1] angeben sind. Alle anderen Fälle erhält man jedoch durch Rotation bzw. Invertierung. Bei der Rotation einer Eckpunktkonfiguration werden die zugehörigen Dreiecke ebenfalls rotiert. Bei der Invertierung ändern sich die Trennflächen nicht – die selbe Dreiecksmenge trennt sowohl innen von außen als auch außen von innen – nur die Orientierung der Dreiecke muss ebenfalls invertiert, d.h. die Reihenfolge der Eckpunkte vertauscht werden. Abbildung 5.14 zeigt dies an einem Bei-

[1]Einige Wissenschaftler sprechen auch von nur 14 echten Fällen, da der Fall 0 den trivialen Fall ohne Trennfläche im Würfelinneren darstellt.

spiel.



(a) Index 00111000 (56) (b) Index 11000111 (199)

Abbildung 5.14.: MC Index 56 und Invertierung

Die 256 möglichen Konfigurationen werden in einer Lookup-Tabelle gespeichert. Die 8-Bit-Zahl, die die Eckpunktkonfiguration kodiert, kann so gleichzeitig als Tabellenindex verwendet werden. Zur Laufzeit wird dann für jede Würfelposition der Index berechnet und die Dreiecksmenge aus der Lookup-Tabelle geholt. Die Würfelgröße kann als zusätzlicher Parameter verwendet werden: Eine größere Würfelkantenlänge führt zu einer schnelleren, aber auch gröberen Approximation.

Ein Problem, das beim klassischen MC-Verfahren auftreten kann, ist, dass durch das Triangulieren der einzelnen Würfel unabhängig voneinander Löcher an den Würfelgrenzen auftreten können. Ein Grund dafür sind die sogenannten **ambiguities** (Mehrdeutigkeiten), d.h. Würfelkonfigurationen, bei denen topologisch unterschiedliche Trennflächen im Inneren möglich sind. Abbildung 5.15 zeigt einen solchen Fall: Zwei diagonal gegenüberliegende Würfeleckpunkte, die im Inneren liegen, können entweder durch zwei Dreiecke von den restlichen Eckpunkten getrennt (a) oder durch einen „Tunnel" verbunden werden (b). Der zweite Grund für Löcher in der Dreiecksfläche besteht darin, dass die original MC-Tabelle zum Teil lückenhafte Würfelübergänge erzeugt, d.h. zwei regulär benachbarte Würfeleckpunktkonstellationen liefern zum Teil disjunkte Kanten (Tabellen-Inkonsistenz) an der gemeinsamen Würfelseite, wie in Abbildung 5.15 c) skizziert.

Damit eine Patch-Fläche überschneidungsfrei geschlossen (**wasserdicht**) ist, muss jede Kante in genau zwei Patches liegen. Liegt eine Kante nur in einem Patch, so ist die Fläche an dieser Stelle offen. Gehört eine Kante zu mehr als zwei Patches, so ist die Fläche nicht mehr überschneidungsfrei. Abbildung 5.16 zeigt ein Beispiel des klassischen MC-Verfahrens angewandt auf das Segment eines Schädels in einem realen Volumendatensatz. Die farblich hervorgehobenen Dreiecke besitzen offene Kanten.

Ist die Würfelgröße bereits minimal (und entspricht damit der Auflösung der diskreten Daten), lässt sich nur durch Rekonstruktionsfilter entscheiden, welcher Fall am ehesten der Realität entspricht. Damit diese Abschätzung nicht zu viel Laufzeit kostet, ist hierfür

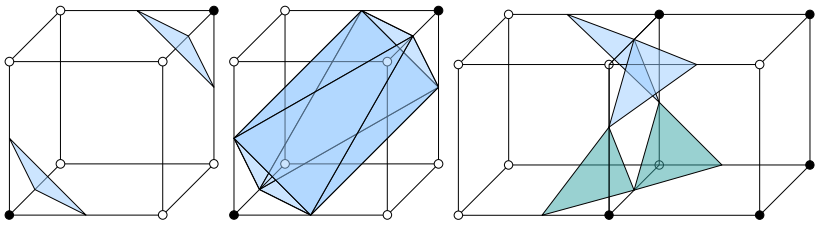


Abbildung 5.15.: Original MC Fläche: Mehrdeutigkeiten und Inkonsistenzen

eine effiziente Fallunterscheidung notwendig (siehe bspw. [14] oder [36]). Dieser Ansatz wurde auch in der Diplomarbeit von Yifan Yu [89] untersucht.

Für viele Anwendungen, bspw. für die schnelle Visualisierung, ist es jedoch bereits ausreichend, wenn die erzeugte Dreiecksfläche abgeschlossen ist, auch wenn dabei die topologische Korrektheit in Bezug auf die (kontinuierlichen) Originaldaten im Detailbereich nicht mehr gewährleistet ist. In der Masterarbeit von Richard Guercke wird gezeigt, das dies mit einer modifizierten Lookup-Tabelle möglich ist ([25], S. 39ff und [53]). Kernidee ist es, bei der Erzeugung der Lookup-Tabelle in drei kritischen Grundfällen (3, 6 und 7) neue Invertierungen zu verwenden und beim Tabellenaufbau der Rotation Präferenz vor der Invertierung zu geben.

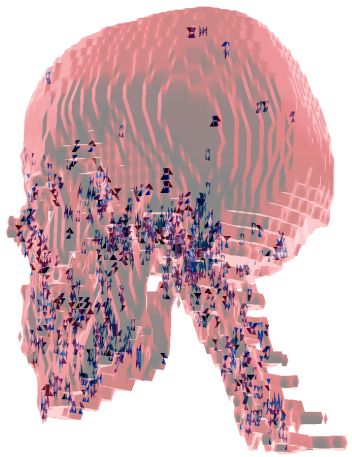


Abbildung 5.16.: Hervorgehobene Dreiecksfehler der klassischen MC Methode *Quelle: [89]*

Ein störender Faktor der Trenndreiecksmengen in der MC-Lookup-Tabelle ist die Tatsache, dass die Eckpunkte der Dreiecke immer auf die Mittelpunkte der Würfelkanten fallen. Das Ergebnis ist eine Randfläche mit unschönen Stufen und sich wiederholenden Winkeln, was besonders bei der Visualisierung störend wirkt. Um dies zu beheben und gleichzeitig die Isofläche besser zu approximieren, kann die Abweichung der Intensitätswerte an den Würfel-Eckpunkten gegenüber dem Iso-Wert verwendet werden, um die Position des Eckpunkts auf der Würfelkante linear zu interpolieren. Dabei wählt man

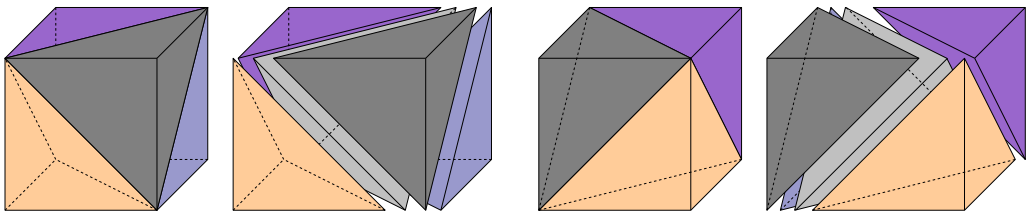
den Schnittpunkt auf der Kante gerade so, dass das Verhältnis der Entfernungen zu den Eckpunkten dem Verhältnis der zugehörigen Intensitätswerte zum Isowert w entspricht. Seien v_1, v_2 Würfeleckpunkte auf einer gemeinsamen Kante mit den Intensitätswerten i_1 und i_2 so gewählt, dass o.B.d.A $i_1 < w < i_2$ gilt. Dann erhält man den interpolierten Dreieckseckpunkt p mit $p = v_1 + t * (v_2–v_1)$ und $t = (w–i_1)/(i_2–i_1)$.

Als Eckpunktnormalen können die klassischen geometrischen Normalen berechnet werden (Mittelung aller an einen Punkt angrenzenden Dreiecksnormalen), meist werden hier jedoch die Gradientennormalen (siehe Kapitel 3.3.5) verwendet. In diesem Fall werden zuerst die Gradienten für die 8 Würfeleckpunkte bestimmt und dann analog zum Kantenschnittpunkt linear interpoliert.

Das MC-Verfahren kann auch zur Visualisierung eines Segments verwendet werden. In diesem speziellen Fall gibt es nur die Intensitätswerte 0 und 1, der Isowert zur Erzeugung der MC-Fläche ist 0.5. Da hier die Grauwerte nicht unbedingt zum Glätten verwendet werden können (z.B. wenn das Segment eines stark verrauschten Datensatzes unter Ausnutzung von Expertenwissen zustande gekommen ist), ist hier ein klassisches geometrisches Glättungsverfahren angebracht.

5.3.2.2. Marching Tetraeder

Eine weitere Möglichkeit, das Problem der Mehrdeutigkeit des MC-Verfahrens zu umgehen, ist die Verwendung des Marching Tetraeder-Ansatzes (kurz MT, siehe auch [89], Kapitel 2.2.3). Zu Beginn der Entwicklung von YaDiV wurden einige Verfahren ausprobiert. Obwohl das Marching Tetraeder Verfahren letztlich nicht zur Segmentvisualisierung verwendet wurde, enthält es doch einige Aspekte, die nicht nur im Kontext dieser Arbeit relevant sind.



(a) (b)

Abbildung 5.17.: Zwei Konfigurationen zur Zerlegung des MC Würfels in Tetraeder

Einer der Hauptgründe für die schnelle Verbreitung der ersten Marching Tetraeder-Verfahren [75] war die damals noch nicht ausgelaufene Patentierung der Marching Cube-Methode. Während einige MT-Varianten jeden Würfel in 6 Tetraeder durch Schnitte ent-

lang der Hauptdiagonalen aufteilten, soll hier der Ansatz von Carneiro, Silva und Kaufman vorgestellt werden [11]. Kurz zusammengefasst wird bei dieser Marching Tetraeder-Variante der ursprüngliche Würfel in zwei unterschiedliche Konfigurationen mit je 5 Tetraedern zerlegt (Abbildung 5.17), die alternierend verwendet werden: Bezeichnet (i, j, k) die Würfelposition, so wähle Konfiguration a) falls $(i + j + k) \mod 2 = 0$, sonst wähle Konfiguration b).

Betrachtet man nun die Trennfläche im Inneren des Tetraeders, vereinfacht sich die Lookup-Tabelle signifikant: statt 2^8 bleiben $2^4 = 16$ Fallunterscheidungen übrig, bei denen keine Mehrdeutigkeiten auftreten. Um das Verfahren zu beschleunigen, können auch für jeden Tetraeder einzelne Tabellen angelegt werden, was zusammen mit den zwei Zerlegungen zu 2*5 Tabellen mit insgesamt 160 Einträgen führt.

Durch die Verwendung von 5 statt 6 Tetraedern kann die Anzahl der Dreiecke um ca. 15% reduziert werden, dennoch erzeugt das Verfahren erheblich mehr Dreiecke als die klassische Marching Cube-Methode. Für die Auswahl einer geeigneten Indirect Volume Rendering-Methode für YaDiV wurden in [89] unterschiedliche Algorithmen zur Randflächenapproximation von Volumenkörpern in Voxeldarstellung speziell für die Anwendung der Segmentvisualisierung untersucht. Beim Vergleich der Anzahl der durch MC bzw. MT erzeugten Dreiecke in den dort untersuchten Beispielen (sowohl medizinische als auch klassisch geometrische) ergab sich ein recht konstanter Faktor von ca. 2.4 ([89], S. 46).

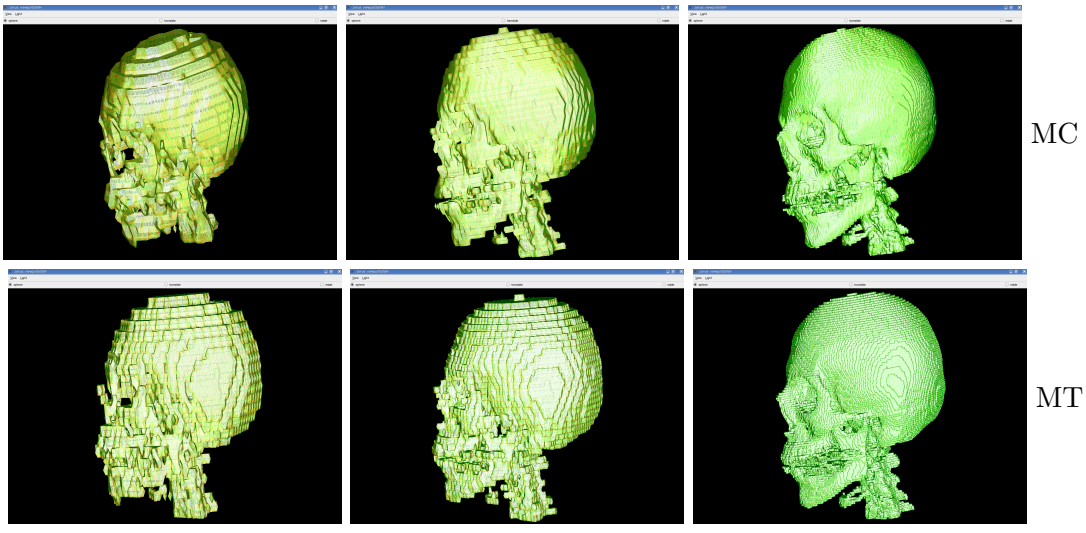


Abbildung 5.18.: Vergleich von Marching Cube und Marching Tetraeder *Quelle: Bilder: [89]*

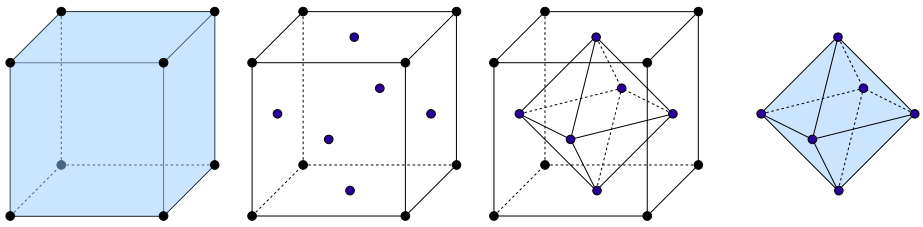
Abbildung 5.18 zeigt einen direkten Vergleich der beiden Verfahren (ohne nachfolgende Glättung) bei der Triangulierung eines Schädel-Segments. Die erzeugten Dreiecke wurden transparent gelb gezeichnet mit einem grün hervorgehobenem Wireframe Modell,

wodurch sich der „Grünstich“ in den höheren Auflösungen erklärt. Neben den Cube Size bedingten Qualitätsunterschieden kann man gut die bei der Anwendung auf Segmentdaten entstehende Treppenstruktur erkennen.

5.3.2.3. Dual Marching Cube

Eine Erweiterung des Marching Cube-Konzepts wurde 2004 von Nielson in seiner Arbeit über „Dual Marching Cubes“ [59] vorgestellt. Auf den ersten Blick scheint es, als würde Nielson lediglich Dreiecke gegen Quad-Patches austauschen, doch steckt dahinter ein interessanter geometrischer Aspekt.

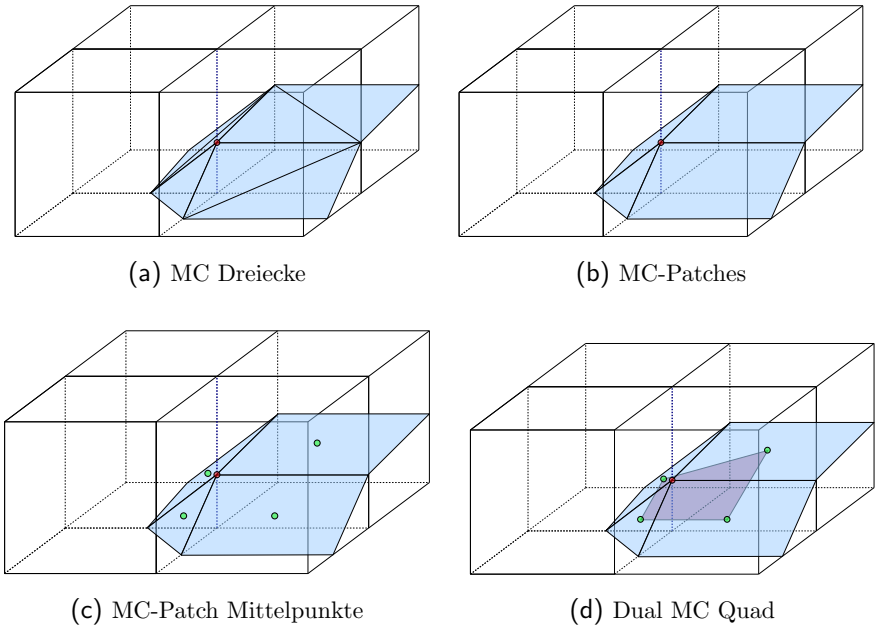
Der Begriff der Dualität ist sowohl aus der Geometrie als auch aus der Graphentheorie bekannt. Zwei Polytope P und Q heißen **kombinatorisch dual**, wenn ihre Seitenverbände (Ecken, Kanten, Flächen usw.) anti-isomorph sind. Betrachtet man z.B. einen dreidimensionalen konvexen Polyeder P, so lässt sich ein dualer Polyeder Q konstruieren, indem die Mittelpunkte der Seitenflächen von P genau dann mit einer Kante verbunden werden, wenn die entsprechenden Seitenflächen von P eine gemeinsame Kante besitzen. Die Eckenzahl von Q ist dann gleich der Flächenzahl von P und umgekehrt, die Kantenanzahl bleibt gleich. Abbildung 5.19 zeigt ein Beispiel. Mit dem gleichen Konstruktionsprinzip lässt sich auch zu einem planaren Graphen ein entsprechender dualer Graph erzeugen.



(a) E=8, F=6, K=12 (b) (c) (d) E=6, F=8, K=12

Abbildung 5.19.: Duale Polyeder in der Geometrie

Nielson stellt nun ein Verfahren vor, mit dem sich dieses Prinzip auf die vom Marching Cube Verfahren erzeugten Dreiecksflächen anwenden lässt. Dazu werden zunächst die inneren Kanten eines jeden MC-Würfels entfernt (bzw. gar nicht erst erzeugt), bis nur noch Kanten übrig bleiben, die auf den Würfelseiten liegen (im Paper als MC-Patch Surface bezeichnet). Im nächsten Schritt wird die zur Patch-Fläche duale Fläche erzeugt, indem jeweils vier benachbarte, an einer gemeinsamen Kante angrenzende MC-Würfel betrachtet werden. Wie zuvor geschildert, werden auch hier zunächst die Schwerpunkte der einzelnen Patches berechnet und genau die Punkte durch eine Kante verbunden, deren zugehörige MC-Würfel eine gemeinsame Seitenfläche besitzen. Da jede Kante immer in genau vier MC-Würfeln liegt, setzt sich die so konstruierte duale Fläche aus Quad-Patches zusammen.



(a) MC Dreiecke (b) MC-Patches

(c) MC-Patch Mittelpunkte (d) Dual MC Quad

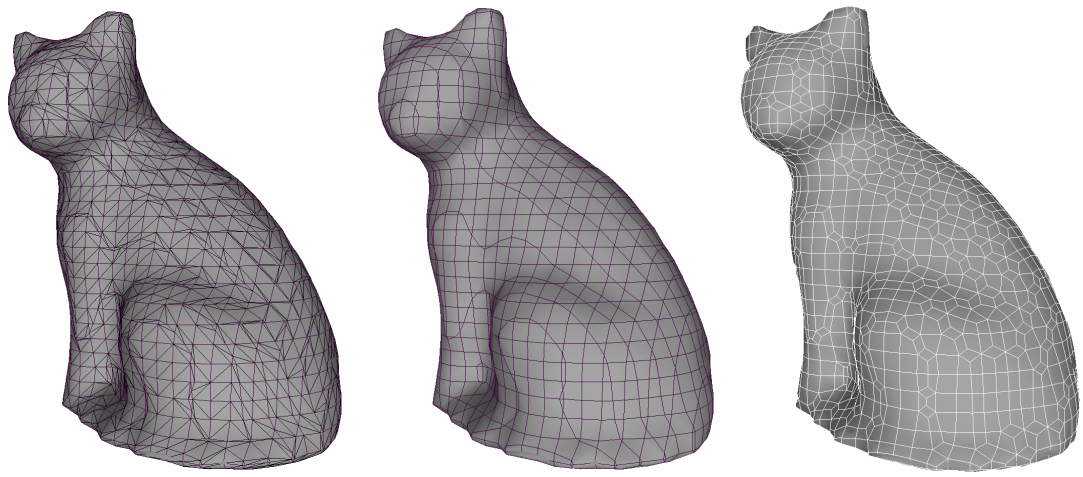
Abbildung 5.20.: Konstruktion des dualen Quad Patches

Das Prinzip wird in Abbildung 5.20 deutlich. Das Bild a) zeigt vier MC-Würfel mit einer gemeinsamen Kante. Zuerst wird die MC-Patch Fläche durch das Entfernen der Würfelinneren Kanten der MC-Dreiecke erzeugt. Danach werden die Mittelpunkte bestimmt und schließlich durch das Einfügen neuer Kanten zum neuen Quad-Patch verbunden.

Das Verfahren lässt sich beschleunigen, indem eine neue Lookup-Tabelle auf Basis des dualen Konstruktionsprinzips erstellt wird, in der die Konnektivität der Patches abgelegt wird. Abbildung 5.21 zeigt die Auswirkung auf einer Beispielfläche in Form einer Katze. Durch die Verwendung der Dual-MC-Fläche lassen sich optische Verbesserung erzielen, da durch die Elimination quasi-entarteter Dreiecke, die beim klassischen MC-Verfahren auftreten, Darstellungsfehler im Shading reduziert werden. Eine weitere Anwendung dieser Methode ist die Verwendung in CAD/CAM Applikationen, die üblicherweise mit Quad-Patches arbeiten.

5.3.3. Lit Sphere Mapping

Alle bisher beschriebenen Verfahren zur Datenvisualisierung setzen bisher implizit das klassische Gouraud Shading Modell voraus, sind jedoch prinzipiell nicht darauf festgelegt. Während auch die bekannteren Alternativen wie Phong- oder Schlick-Shading vor allem das Ziel verfolgen, möglichst natürliche und realistisch wirkende Bilder zu generieren, soll mit dem Lit Sphere Mapping ein etwas weniger verbreitetes Verfahren vorgestellt werden, das es erlaubt, andere Darstellungsformen zu erzeugen – darunter auch einen illustrativen



(a) MC (b) MC Patch (c) Dual MC

Abbildung 5.21.: Beispiel für duale MC Fläche *Quelle: [59]*

Stil, wie er zum Teil in Lehrbüchern verwendet wird. Damit gehört das Verfahren zur Familie der sogenannten Non-Photorealistic-Rendering Verfahren (NPR).

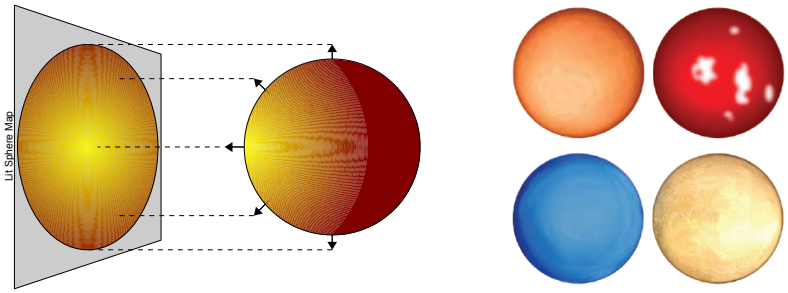
Das Konzept geht zurück auf die Arbeit [76] von Sloan et al. aus dem Jahr 2001, wo die Autoren ein Verfahren vorschlugen, mit dem sich Schattierungstechniken, wie sie in der Kunst verwendet werden, vergleichsweise einfach in Rendering Verfahren integrieren lassen. Künstler verwenden Schattierungs-Skizzen in Form einer projizierten Halbkugel, um in ihrem Bild einen bestimmten Zeichenstil zu verwenden. Dabei wird die Farbschattierung als Textur auf einer Kreisscheibe aufgetragen, die alle möglichen Normalen der Halbkugel als Stil- bzw. Farbinformation enthält. Dieses Prinzip bildet die Basis für das *Lit Sphere Mapping*.

Üblicherweise verwenden Shading-Modelle die Normalen des Objekts und berechnen die Helligkeit eines Punktes über den Winkel zur Richtung evtl. vorhandener Lichtquellen Lichtquelle und u.U. auch der Sichtrichtung des Betrachters ab. Die Farbinformation setzt sich aus Objektfarbe, Lichtfarbe(n) und (falls vorhanden) aus Texturinformationen zusammen.

Auch beim *Lit Sphere Mapping* wird die Objektnormale verwendet, jedoch sind alle anderen Informationen impliziert über eine Textur vorgegeben. Es können zudem weitere Effekte erzielt werden, da die als Shading-Palette verwendete Textur keinen Einschränkungen unterworfen ist. Die Indizierung der Textur ist dabei sehr einfach. Sei $n = (n_x, n_y, n_z)$ die normierte Normale in Sichtkoordinaten, dann erhält man die Texturkoordinaten

$$t = (t_x, t_y) \text{ mit } t_x = \frac{n_x + 1}{2} \text{ und } t_y = \frac{n_y + 1}{2} \tag{5.12}$$

Da die z-Koordinate ignoriert wird, wird bei der Farbbestimmung nicht zwischen Vorder- und Rückseite des Objekts unterschieden. Dies ist insbesondere dann hilfreich, wenn die Normalen über den Gradienten bestimmt wurden. Abbildung 5.22 a) demonstriert das Prinzip.



(a) Prinzip (b) Beispieltexturen [7]

Abbildung 5.22.: Lit Sphere Mapping

In [7] haben Bruckner et al. gezeigt, wie dieses Modell in einem Ray Casting Verfahren verwendet werden kann, um einen illustrativen Lehrbuchstil zu imitieren. Während die klassische Transferfunktion jedem Intensitätswert einen Farb- und einen Alpha-Wert zuordnet, werden in der Style-Transferfunktion anstelle des Farbwerts *Lit Sphere Maps* verwendet. Dabei wurde eine Stützpunktdarstellung mit linearer Interpolation gewählt, d.h. jeder Intensitätswert erhält eine Mischfarbe, die sich aus zwei *Lit Sphere Maps* zusammensetzt. Neben einer Auswahl an Texturen wird auch ein Verfahren zur Hervorhebung von Konturen vorgestellt.

5.3.4. Stereovisualisierung

Eine wesentliche Verbesserung der Verständlichkeit der Daten lässt sich durch eine stereographische Visualisierung erreichen. In der klassischen 3D-Ansicht wird – ähnlich einem Foto – ein einzelnes Bild vom Standpunkt des Betrachters erzeugt. Der dreidimensionale Tiefeneindruck entsteht durch die Perspektive, sowie optische Hilfsmittel wie Verdeckung, Schattierung oder Bewegung.

Die Technik, räumliche Abbilder durch Zentralprojektion auf einer ebenen Zeichenfläche zu erfassen, ist hinreichend bekannt und wurde bereits 1525 in Albrecht Dürers „Underweysung der Messung mit dem Zirckel und Richtscheyt“ umfassend beschrieben. Seine damalige „Zeichenmaschine“ (Abbildung 5.23) entspricht im Grundsatz dem, was heute eine moderne Grafikkarte leistet: Die Projektion von Punkten aus dem dreidimensionalen Raum auf eine Ebene, die dann (1525 noch manuell) durch Linien verbunden die Kontur des Objekts wiedergeben. Dieses Modell geht jedoch von einem Betrachter aus, der nur ein einziges Auge besitzt (oder das andere zuhält).

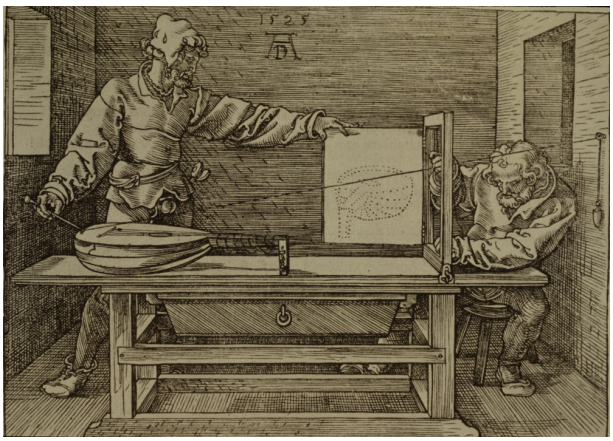
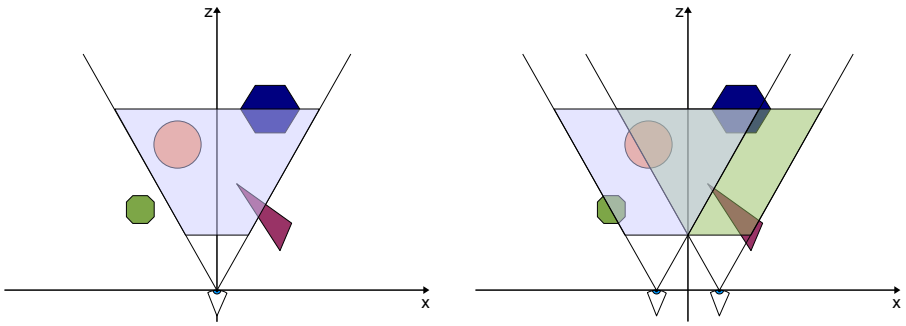


Abbildung 5.23.: Zeichenmaschine von Albrecht Dürer

In der stereographischen Visualisierung (siehe z.B. auch [18], Kapitel 18.11.5, Seite 915 ff.) wird daher die Szene zusätzlich von einem leicht in $x-$Richtung versetzten Betrachterstandpunkt aus erneut gerendert, wie in Abbildung 5.24 skizziert.



(a) Mono (b) Stereo

Abbildung 5.24.: Mono- und Stereoprojektion

Das Ergebnis sind perspektivisch korrekte Bilder für das rechte bzw. linke Auge, die mit einem entsprechenden Ausgabemedium (z.B. Stereomonitor, Stereobeamer, Head Mounted Display, siehe auch Abbildung 7.20, Seite 156) für jedes Auge getrennt dargestellt werden. Vor allem in der Visualisierung von komplexen medizinischen Volumendaten führt dies zu einem wesentlich vereinfachten Zugang mit einem schnelleren (und höherem) Erkenntnisgewinn.

6. Haptik

Im Zuge der rasanten Entwicklung der Rechen- und Speicherleistung von Computern und insbesondere Grafikkarten lassen sich heutzutage mit vergleichsweise geringen Kosten virtuelle Umgebungen schaffen. Diese basieren darauf, mittels gleichzeitiger 3D-Visualisierung, 3D-Klangerzeugung sowie taktilem und haptischem Feedback eine realitätsnahe Umgebung für den Benutzer zu erzeugen und so eine intuitive Benutzerschnittstelle zu schaffen. Virtuelle Realität (VR) hält in vielen Bereichen des täglichen Lebens Einzug.

Der Einsatz von Haptik als zentraler Bestandteil von VR-Applikationen wird zunehmend populärer. Auch in der Medizintechnik gibt es erste Versuche mit haptischen Eingabegeräten, z.B. in Trainingssimulatoren zur Operationsplanung. Die Breite der möglichen Anwendungen wird u.a. in [38] beschrieben.

6.1. Grundlagen

Unter dem Begriff Haptik wird die Lehre der haptischen[1] Wahrnehmung verstanden, die das aktive Erfühlen von Größe, Kontur, Oberflächentextur, aber auch Temperatur oder Gewicht eines Gegenstandes bezeichnet. Neben der visuellen entstehen vor allem durch haptische Wahrnehmung die wichtigsten Sinneseindrücke, um dreidimensionale Körper zu erfassen. Dabei wird zwischen

- taktiler Wahrnehmung (Erfühlen von Oberflächeneigenschaften),
- kinästhetischer Wahrnehmung (Erkennen von Trägheit, Steifigkeit, Gewicht),
- Temperaturwahrnehmung und
- Schmerzwahrnehmung

unterschieden. Während die letzten beiden Bereiche für die Benutzerinteraktion von geringem Interesse sind, gibt es intensive Versuche, taktile und kinästhetische Wahrnehmung als intuitive Schnittstelle in der Mensch-Maschine-Kommunikation zu etablieren.

[1] aus dem Griechischen: haptós („fühlbar“) bzw. haptikós (zum Berühren geeignet“)

Die Forschung im Bereich der taktilen Interaktion steht ziemlich am Anfang, da selbst die Modelle für die taktile Wahrnehmung noch entwickelt werden und die meisten Ausgabegeräte erst das Stadium eines Prototyps besitzen. Diese reizen über unterschiedliche Mechanismen die Mechanorezeptoren in der Haut, um das Erfühlen von Strukturen einer (virtuellen) Oberfläche zu simulieren. Ein guter Überblick über die Bandbreite von taktilen Eingabegeräten wird z.B. in der Dissertation [1] von Dennis Allerkamp gegeben (Kapitel 3, S. 26), die sich mit der taktilen Simulation von Textilien beschäftigt.

In dieser Arbeit geht es vor allem um die kinästhetische Wahrnehmung und der Begriff „Haptik“ wird auf den folgenden Seiten auf diesen Bereich reduziert. Die Tiefensensibilität bzw. die kinästhetische Wahrnehmung informiert über den inneren Zustand eines Körperteils und dient dadurch zum Erkennen der Trägheit, Steifigkeit und des Gewichts von Objekten. Dabei unterscheidet man zwischen dem **Lagesinn** (Lage des Gelenks), dem **Kraftsinn** (Muskelanspannung) und dem **Bewegungssinn**.

6.2. Haptische Eingabegeräte

Haptic interface devices behave like small robots that exchange mechanical energy with a user. [69]

Der Begriff des „haptisches Eingabegeräts“ ist an sich etwas irreführend, da ein solches Gerät zwar Eingaben des Benutzers erfasst, aber gleichzeitig auch mit einer Gegenkraft reagiert, so dass es sich im Prinzip um ein haptisches Ein- und Ausgabegerät handelt. Da dieser Begriff genau wie seine Alternativen (Kraftrückkopplungsgerät, Force-Feedback-Device) etwas zu sperrig klingt, wird meist diese verkürzte Variante gewählt.

Haptische Eingabegeräte ermöglichen die taktile und kinästhetische Exploration von virtuellen Objekten, d.h. ein (virtuelles) Objekt im dreidimensionalen Raum wird nicht nur visuell sichtbar, sondern real erfühlbar. Auch externe Kräfte wie Strömung oder die Bewegung einer schwingenden Textilie (z.B. in [8]) können so dargestellt werden. Die wesentlichen Merkmale für haptische Eingabegeräte sind die **Anzahl der Freiheitsgrade**, die **Dimensionen des Arbeitsraums**, die Anzahl der **haptischen Interaktionspunkte** und die **maximale Kraft**.

Bezüglich des Arbeitsraumes, auch als Workspace bezeichnet, kann grundsätzlich zwischen Desktop-Geräten wie dem „Phantom Omni“ der Firma Sensable oder dem Falcon der Firma Novint (Abbildung 6.1 a und b), die einen Arbeitsbereich von einigen Zentimetern besitzen, und Geräten wie dem „Inca 6D“ (Abbildung 6.1 c) der Firma Haption mit einem Arbeitsbereich von mehrere Metern unterschieden werden. Während Geräte mit einem sehr großen Arbeitsraum meist fest installiert sind und einen eigenen Raum benötigen, sind Desktop-Geräte hier wesentlich flexibler und eignen sich auch prinzipiell eher für andere Aufgaben. So wird eine chirurgische Simulation, bei der es vor allem auf eine

(a) Phantom Omni (b) Novint Falcon (c) Inca 6D

Abbildung 6.1.: Haptische Eingabegeräte

exakte Gewebesimulation und Feinmotorik ankommt, nur bedingt von einem großen Arbeitsvolumen profitieren. Ein großer Arbeitsbereich führt jedoch – insbesondere mit einem stereographischen Display – zu einer stärkeren Immersion, was eine intuitive Schnittstelle, bspw. für Registrierungs-Prozesse oder Erreichbarkeits-Simulationen, erlaubt.

Neben dem Arbeitsraum unterscheiden sich die Geräte vor allem durch die Anzahl der möglichen Freiheitsgrade, auch als DOF („Degree of Freedom“) abgekürzt. Während 3DOF-Geräte nur die Position im Raum messen, erfassen 6DOF-Geräte auch die Winkel der Torsion um die x-, y- bzw. z-Achse. Genau wie der Arbeitsraum sind auch die Torsionswinkel meist durch die Mechanik des Geräts limitiert. Eine feinere Unterteilung differenziert hier zusätzlich zwischen Eingabe-Freiheitsgraden und Ausgabe-Freiheitsgraden.

Generell ist ein Gerät aus **Sensoren**, **Aktuatoren** und **Endeffektoren** aufgebaut. Der Begriff des Endeffektors stammt ursprünglich aus der Robotik und bezeichnet „das letzte Glied einer kinematischen Kette“[1]. Im Bezug auf haptische Eingabegeräte hat der Begriff in der Literatur zwei verwandte, aber leicht unterschiedliche Bedeutungen. Entweder wird darunter der (gesamte) Teil der Apparatur verstanden, der die Kraft zum Benutzer überträgt (Roboterarm, Seilzug, etc.), oder nur dessen letztes Glied, das vom Benutzer tatsächlich berührt wird, bspw. ein Handgriff, ein Stift oder ein Fingerhut. Im Folgenden soll in dieser Arbeit die zweite Bedeutung gemeint sein. Die Position des Endeffektors wird dabei auch als HIP (*Haptic Interface Point*) bezeichnet.

Die Sensoren dienen zur Bestimmung der aktuellen Position des Endeffektors, abhängig von den Freiheitsgraden des jeweiligen Geräts. So werden bei drei Freiheitsgraden (3DOF) die xyz-Positionen und bei sechs Freiheitsgraden (6DOF) noch zusätzlich die Torsionswinkel bestimmt. Aufwendigere Geräte können zusätzlich die vom Benutzer aufgewandte Kraft messen. Auch die Sensorauflösung kann sich je nach Gerät stark unterscheiden. Selbst einfache Geräte erlauben jedoch bereits eine – verglichen mit klassischen Eingabegeräten wie der Computermaus – wesentlich intuitivere räumliche Navigation.

[1] Quelle: http://de.wikipedia.org/w/index.php?title=Endeffektor&oldid=48516698 (Permanenter Link)

Bei den Aktuatoren handelt es sich meist um Servo-Motoren, welche die rückwirkenden Kräfte erzeugen. In Abhängigkeit des haptischen Geräts können diese in unterschiedliche Richtungen wirken. Auch hier kann eine Unterteilung aufgrund der möglichen Freiheitsgrade vorgenommen werden.

Es gibt inzwischen eine breite Bandbreite von haptischen Eingabegeräten, die vom Benutzer üblicherweise berührt oder angezogen werden (bspw. ein Datenhandschuh mit Kraftrückkopplung). Salisbury et al. bezeichnen diese Geräte daher in [69] auch als *device-body-interface*, um die physische Verbindung mit dem Benutzer zu unterstreichen.

6.3. Medizinische Anwendungsgebiete

In der Medizin werden haptisch unterstützte Navigation, Visualisierung und Interaktion in Bereichen mit sehr geringen Fehlertoleranzen eingesetzt. Forschungsschwerpunkte sind die präoperative Planung, tele- und roboterunterstützte Chirurgie sowie Trainingssituationen während der Ausbildung.

Bereits 1998, als kommerzielle haptische Eingabegeräte noch in einem sehr frühen Stadium waren, haben Giess et al. in [23] über volumenbasiertes haptisches Rendering in chirurgischen Planungsszenarios geschrieben. Die Arbeit, die sich trotz des etwas irreführendes Titels in erster Linie mit einer Anwendung zur Segmentierung der Leber beschäftigt, verwendete dafür ein flächenbasiertes haptisches Rendering-Modell. Eine ähnliche Richtung schlägt auch die Arbeit [27] von Harders und Székely aus dem Jahr 2002 ein, in der die Autoren ein Verfahren vorstellen, um Organe mit Hilfe einer haptischen Benutzerschnittstelle zu segmentieren. Ausgehend von einer einfachen Schwellwert-Segmentierung wird ein System präsentiert, in dem ein Experte mit Hilfe eines Einfingerkontakt-Gerätes (PHANToM) die Segmentierung verbessern kann.

Eine große Zahl von Arbeiten beschäftigt sich mit dem Einsatz von Haptik bei minimal-invasiven Eingriffen. Im Gegensatz zur offenen Chirurgie, wo der zu operierende Bereich für den chirurgischen Zugang weiträumig geöffnet wird, arbeitet die minimal-invasive Chirurgie meist mit langen und schmalen Instrumenten (Endoskop), die durch sehr kleine Schnitte eingeführt werden können. Das Ziel ist ein operativer Eingriff mit einem möglichst geringen Trauma[1]. Minimal-invasive Eingriffe sind nicht unproblematisch, da durch die geringe Öffnung leicht Fehler entstehen, was besonders in kritischen Bereichen (wie etwa dem Sehnerv) schwerwiegende Folgen haben kann. Um dem operierenden Arzt möglichst viele Informationen über die Umgebung des Instruments zu geben, wird versucht, neben der visuellen auch die haptische Schnittstelle verstärkt einzusetzen.

[1] Trauma bezeichnet in der Medizin oder Biologie eine Schädigung, Verletzung oder Wunde, die durch äußere Gewalt entstanden ist.

Ein weiteres populäres Anwendungsgebiet sind Trainingsprogramme, die dem Benutzer mittels einer Simulation die Möglichkeit geben, Eingriffe zu üben und Methodiken zu evaluieren. Einen guten Einstieg in diese Literatur findet man bei Basdogan et al. [2] oder Srinivasan et al. [78]. Während Basdogan et al. sich vor allem auf das Training minimal-invasiver Operationen konzentriert, liegt der Fokus bei Srinivasan et al. auf der Analyse von Simulationsumgebungen für Endoskopie und speziell der Laryngoskopie[1]. Weitere Beispiele für Haptik im medizinischen Training sind das System von Färber et al. [19] das mit einem 6DOF-PHANTOM Lumbalpunktionen simulieren kann, oder Krüger et al. [32], die ein System zur Evaluation minimal-invasiver Operationen mittels einer haptischen Simulation der verwendeten endoskopischen Instrumente beschreiben.

Im Bereich der tele- und roboterunterstützten Chirurgie gibt es ebenfalls Bemühungen, die Benutzerschnittstelle durch Haptik zu verbessern. So wurden Systeme für roboterunterstützte Operationen entwickelt, die durch Kraftrückkopplung dem steuernden Mediziner zusätzliche Informationen geben. Ein interessantes Beispiel ist das System von Ortmaier et al. [62], bei dem der Bediener zusätzlich zur visuellen Schnittstelle die am OP-Roboter auftretende Kräfte über ein 6DOF-Phantom erfährt. Das verwendete chirurgische Instrument wurde dafür mit Kraftsensoren ausgestattet. Auch hier wird ein minimal-invasiver Eingriff angestrebt. Die Autoren konnten im abschließenden Test zeigen, dass sich durch den Einsatz der Kraftrückkopplung die Genauigkeit der Operationen signifikant erhöht hat.

Einen guten und zugleich breiten Überblick gibt das Survey-Paper [49] von Meijden und Schijven aus dem Jahr 2009, das sich allgemein mit dem Einsatz von VR und insbesondere Haptik in minimal-invasiven Operationen, roboterunterstützter Chirurgie und operativen Trainingssituationen beschäftigt. In ihrer Arbeit diskutieren die Autoren kontrovers den Nutzen von Haptik anhand der analysierten Arbeiten und kommen zu unterschiedlichen Ergebnissen. Zwar wird in den meisten Arbeiten der Einsatz haptischer Hilfsmittel in der minimal-invasiven Chirurgie positiv bewertet, jedoch erfolgt die Beurteilung meist aufgrund subjektiver Eindrücke, da objektive und anerkannte Validierungsmethoden fehlen. Im Bereich roboterunterstützter Chirurgie scheint der Einsatz von Haptik zu helfen, chirurgische Fehler zu vermeiden. In Trainingssituationen kann Kraftrückkopplung zu einem frühen Zeitpunkt der Ausbildung dabei helfen, psychomotorische Fähigkeiten[2] zu erlernen.

[1] Endoskopische Untersuchung des Kehlkopfes.

[2] Psychomotorik beschreibt die Gesamtheit der sensorischen, kognitiven, motorischen und sozial-emotionalen Dimension von Bewegung.

6.4. Haptisches Rendering

Im folgenden wird meist von einem Gerät mit nur einem *Haptic Interface Point* ausgegangen, auch wenn sich die meisten Konzepte auf mehrere erweitern lassen.

Während beim visuellen Rendering der optische Eindruck (meist durch farbige Pixel) erzeugt wird, werden beim haptischen Rendering Kräfte für die aktive Kraftkopplung berechnet. Stößt der Benutzer über den Endeffektor an ein starres virtuelles Objekt, muss die ausgegebene Kraft genauso groß wie die vom Benutzer aufgewendete sein, um den Eindruck eines festen Körpers zu vermitteln.

Um dies zu erreichen, wird genau wie beim visuellen Rendering die Kraft in regelmäßigen Zeitabständen neu berechnet und ausgegeben, jedoch mit einer wesentlich höheren Wiederholrate. Während für das Auge 25 Bilder pro Sekunde[1] ausreichen, um den Eindruck einer flüssigen Darstellung zu erhalten, sind für die kinästhetischen Sinne 1000 Krafteindrücke pro Sekunde notwendig, was einer Wiederholrate von 1 kHz entspricht. Das Auslesen der Position, die Berechnung von eventuell stattgefundenen Kollisionen und der daraus resultierenden Kräfte wird auch als **Servo-Loop** bezeichnet.

Die rückwirkenden Kräfte werden dabei vor allem durch die vom Benutzer selbst aufgewendete Kraft und durch Materialeigenschaften wie Steifigkeit oder Friktion der Oberfläche beeinflusst. So sollte es z.B. möglich sein, in einen Gummiball geringfügig einzudringen, während eine metallische Kugel eine glatte, harte Oberfläche besitzt. Zusätzlich können die ausgegebenen Kräfte durch Simulationsparameter beeinflusst werden, z.B. wenn sich HIP und/oder Kollisions-Objekt in einer zähen Flüssigkeit befinden.

Prinzipiell werden beim Haptischen Rendering zwei unterschiedliche Design Patterns verfolgt: das sogenannte **Direct Rendering** oder alternativ das **Virtual Coupling** (u.a. in [39] beschrieben). Gegeben sei eine dreidimensionale Szene, bei der ein dynamisches Objekt, bspw. ein chirurgisches Werkzeug, über das haptische Gerät bewegt werden kann. Während beim *Direct Rendering* Position und Orientierung des Endeffektors direkt auf das dynamische Objekt übertragen und die bei Kollision mit fixen Szeneobjekten entstehenden Kräfte zurückgegeben werden, wird beim *Virtual Coupling* ein virtuelles Feder-Dämpfer-System zwischengeschaltet. Der größte Unterschied zum Direct Rendering ist, dass die Position des dynamischen Objekts nicht zwangsläufig der Position des HIP entspricht, sondern mittels entsprechender Algorithmen berechnet wird. Die ausgegebene Kraft wird dabei durch die Differenz zwischen der Position und Orientierung des HIP und dem dynamischen Objekt berechnet. Da die meisten haptischen Eingabegeräte nur die Position des HIP, nicht aber die Geschwindigkeit oder die aufgewandte Kraft messen,

[1] Aktuelle Studien deuten jedoch an, dass für die flüssige Auge Hand-Koordination etwa 100 Bilder pro Sekunde erforderlich zu sein scheinen.

müssen diese durch das verwendete physikalische Modell berechnet bzw. approximiert werden.

In dieser Arbeit werden zwei grundsätzliche haptische Rendering-Modelle vorgestellt, die sich in ihrer Datenbasis (voxel- oder flächenbasiert) unterscheiden. Als Anwendung soll hier die Bewegung und die Transformation eines Segments im Hintergrund stehen, bspw. bei der Relokation eines Trümmerbruchs oder einer schnellen manuellen Vorregistrierung für ein atlasbasiertes Segmentierungsverfahren. Zuvor ist es jedoch hilfreich, etwas über die Erkennung von Kollisionen im Allgemeinen zu erfahren.

6.4.1. Kollisionserkennung

Ein zentraler Bestandteil jeder *Servo-Loop* ist die Kollisionserkennung. Diese betrifft vor allem das über das Eingabegerät gesteuerte dynamische Objekt, da hier gerade die Update-Rate von 1 kHz gefordert wird. Zwar können auch andere Objekte der Szene miteinander kollidieren, was jedoch nur mit der visuellen Update-Rate berechnet werden muss.

Je nach verwendeter Geometrie können unterschiedliche Ansätze zur Kollisionserkennung verwendet werden. Beide gehen wiederum davon aus, dass durch das haptische Eingabegerät vom Benutzer ein dynamisches Objekt bewegt wird, welches mit anderen Objekten der Szene kollidieren kann.

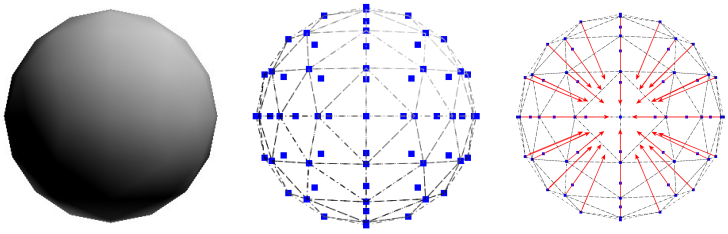
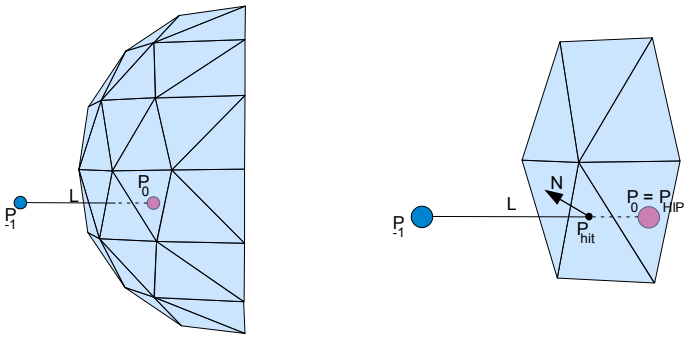


Abbildung 6.2.: Point Shell

Bei größeren dynamischen Objekten wird als Repräsentant eine sogenannte **Point Shell** verwendet, beschrieben von McNeely et al. in [48]. Wie der Name bereits andeutet, approximiert die Point Shell das dynamische Objekt durch (möglichst gleich verteilte) Oberflächenpunkte, die zusätzlich jeweils eine nach innen gerichtete Normale besitzen. Abbildung 6.2 zeigt ein Beispiel einer Point Shell. Die Normalen werden später für die Berechnung der Gegenkraft verwendet.

6.4.2. Flächenbasiertes Modell

In diesem 3DOF-Modell werden die geometrischen Objekte, statische wie dynamische, durch eine facettierte Oberfläche $S = \{s_1, s_2, \ldots, s_n\}$ beschrieben, bspw. in einer Dreieckstruktur, wie sie bei der Triangulierung einer Segmentoberfläche durch das Marching-Cube-Verfahren entsteht. Wenn man zunächst vereinfachend davon ausgeht, dass es sich beim dynamischen Objekt um einen sehr kleinen Ball handelt, kann man dessen Struktur durch einen einzelnen Punkt in seinem Zentrum vereinfachen. Die Koordinaten dieses Punktes stimmen mit der Position des HIP überein. Eine Kollision findet folglich statt, wenn die Bewegung des HIP die Oberfläche durchdringt, d.h. die Strecke L zwischen letzter und aktueller HIP-Position, P_{-1} und P_0, eine Facette der Fläche schneidet, skizziert in Abbildung 6.3 a). Bei komplexeren Objekten kann der selbe Test für die Punkte der Point Shell durchgeführt werden.



(a) (b)

Abbildung 6.3.: Flächenbasiertes Modell zur Kollisionserkennung und Kraftrückkopplung

Aufgrund der benötigten Update-Rate von 1 kHz kann nicht jedes einzelne Polygon auf eine potenzielle Kollision untersucht werden, was ein effizienteres Verfahren zur Kollisionserkennung erforderlich macht. Eine mögliche Strategie ist es, für die statischen Objekte der Szene eine sogenannte „Oriented Bounding Box"-Hierarchie (Gottschalk, Lin und Manocha, 1996 in [24]) zu erzeugen.

Im Allgemeinen ist eine Bounding Box an Hauptachsen ausgerichtet, d.h. die Seitenflächen liegen parallel zu den Koordinatenebenen (auch als Axis Aligned Bounding Box bezeichnet). Diese Beschränkung ist bei der **Oriented Bounding Box** (OBB) aufgehoben, sie ist der kleinste begrenzende Quader eines Objektes. Eine Oriented Bounding Box wird definiert durch den Mittelpunkt M, der Basis B, bestehend aus den Vektoren x, y, z sowie den Halbseitenlängen Ha, Hb und Hc. Die Berechnung der OBB für eine Menge von Polygonen ist nicht trivial und soll hier nur kurz angedeutet werden: Ziel ist es, die OBB entlang der Längsachse der Polygonmenge auszurichten. In [24] wird vorgeschlagen, die Eigenvektoren der Kovarianzmatrix, aufgebaut aus der Verteilung von geeigneten

Sample-Punkten der Polygone, als Basis der OBB zu verwenden. Da die Kovarianz-Matrix immer symmetrisch ist, sind ihre Eigenvektoren orthogonal. Dieses Verfahren wird in [82] genauer erläutert und in etwas vereinfachter Form implementiert.

Die OBB-Hierarchie (Abbildung 6.4) stellt eine Partitionierung eines aus Polygonen aufgebauten Objekts dar, meist in Form eines binären Baumes, dessen Knoten jeweils einen OBB enthalten. Der Baum kann zum Beispiel so konstruiert werden, dass das gesamte Objekt rekursiv entlang der längsten Hauptachse unterteilt wird (Top-Down Verfahren), bis ein vorgegebener Maximalwert von Dreiecken pro OBB erreicht wird. Lässt sich eine Untergruppe von Polygonen nicht an der längsten Hauptachse unterteilen, wird die zweitlängste, andernfalls die kürzeste verwendet. Falls auch dies fehlschlägt, wird die Gruppe als unteilbar markiert.

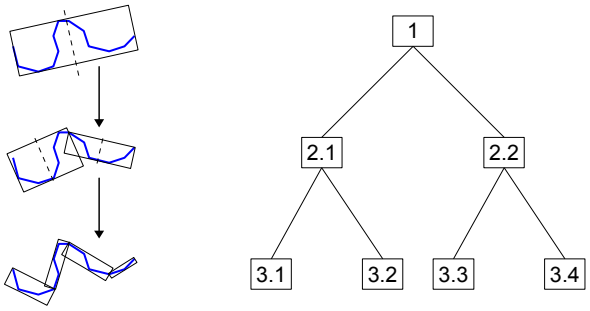


Abbildung 6.4.: Aufbau des OBB-Baumes (Top-Down Verfahren)

Der Schnitttest zwischen der Strecke L (letzte und aktuelle-HIP Position) und dem dynamischen Objekt lässt sich nun beginnend an der Wurzel des OBB-Baumes als Top-Down Verfahren implementieren, wie in Listing 6.1 dargestellt wird.

Für einen schnellen Schnitttest zwischen Linie und OBB kann das „Separating Axis Theorem“ verwendet werden, in dem jede OBB-Seite darauf geprüft wird, ob Start- und Endpunkt von L jeweils in links bzw. rechts von der durch die Seite definierte Ebene liegen. Liegen beide Punkte auf der Außen-Seite, findet keine Überlappung statt. Nach der Bestimmung der Schnitt-OBB müssen nur noch wenige Dreiecke auf einen potentiellen Schnittpunkt getestet werden.

Mit Hilfe der OBB-Hierarchie ist so eine Kollisionserkennung in $log(n)$ Schritten möglich, wobei n der Anzahl der Polygone im Objekt entspricht. Der Aufwand für die Berechnung der OBB-Hierarchie selbst ist $O(n \log(n))$ (bzw. $O(n \log^2(n))$ abhängig von der Güte der Sample-Points die für die Bestimmung der Kovarianz-Matrix der einzelnen OBB verwendet wurden, siehe [24], S.173).

In diesem Modell wird die rückwirkende Kraft F durch die Eindringtiefe $|P_{hit} - P_{HIP}|$ und die Oberflächennormale N des Schnittpunkts P_{hit} bestimmt (Abbildung 6.3 b). F ergibt sich als Ergebnis von

```
OBBNode calc_intersection(OBBNode root, Line line) {
  Queue<OBBNode> nodes;

  nodes.push(root);
  while (!nodes.empty()) {
    OBBNode current = nodes.pop();
    if (current.intersects(line)) {
      if (current.is_leaf()) {
        return current;
      } else {
        current.push(current.left_child());
        current.push(current.right_child());
      }
    }
  }

  return null;
}
```

Listing 6.1: Pseudocode für Schnitt zwischen Linie und OBB-Tree

$$F = \kappa |P_{hit} - P_{HIP}| * N \tag{6.1}$$

wobei mit κ in $\mathbb{R}_{>0}$ die Steifigkeit des Objekts modelliert werden kann.

Da jedoch im nächsten Durchlauf der *Servo-Loop* keine Kollision mehr stattfinden würde (die HIP-Position ist ja bereits im Inneren des Objekts), wird ein sogenanntes *God Object* (siehe [91]) eingeführt. Solange keine Kollision stattfindet, stimmt die Position des *God Object* mit der des HIP überein. Während jedoch der HIP in ein statisches Objekt eindringen kann, bleibt das *God Object* stets außerhalb. Wenn sich der HIP innerhalb des Objekts bewegt, folgt auch das *God Object* in diese Richtung, bleibt jedoch auf der Oberfläche. Die Differenz zwischen *God Object* und HIP wird dabei zur Kraftberechnung verwendet.

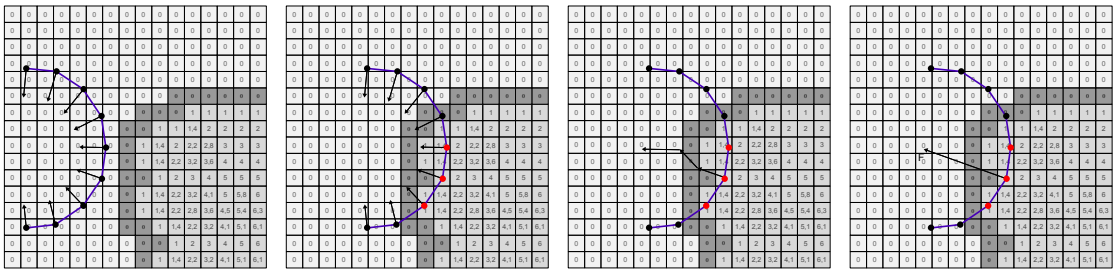
6.4.3. Voxelbasiertes Modell

Der zweite Ansatz zur Kollisionserkennung und der daraus resultierenden Kraftberechnung verwendet eine Volumendatendarstellung auf Voxelbasis. Der Vorteil dieses Modells ist die einfache und schnelle Kollisionserkennung, bei der im Gegensatz zum vorherigen Modell keine aufwendige hierarchische Datenstruktur erstellt werden muss.

Alle statischen Objekte in diesem Modell müssen dabei in Voxeldarstellung vorliegen, was bspw. bei Segmenten in der medizinischen Datenverarbeitung in der Regel bereits

der Fall ist. Im Folgenden soll davon ausgegangen werden, dass das Segment in Form einer BitCube Datenstruktur vorliegt, wie sie in Kapitel 3.3.3 beschrieben wurde. Das dynamische Objekt selbst liege in einer Point Shell-Stuktur vor.

Um während der *Servo-Loop* schnell und ohne Schnitttest die rückwirkenden Kräfte berechnen zu können, wird ein diskretes Distanzfeld φ (siehe Kapitel 3.3.7) verwendet, das die Entfernung der Voxel aus dem Segmentinneren zur Oberfläche enthält.



(a) (b) (c) (d)

Abbildung 6.5.: Voxelbasiertes Modell: Kraftrückkopplung mit Distanzfeld

Wie im flächenbasierten Modell erfolgt auch hier die Kraftberechnung über die Eindringtiefe, als Richtung wird jedoch die Normale der Point Shell-Struktur verwendet. Falls mehrere Punkte ins Innere des Objekts eingedrungen sind, werden sowohl Kraft als auch Richtung gemittelt. Sei $D = (P, N)$ die Point Shell mit den Punkten $P = \{P_1, ...P_n\}$ und den dazugehörigen Normalen $N = \{N_1, ...N_n\}$. Dann berechnet sich die Gegenkraft als

$$F = \kappa \sum_{i=1}^{n} \varphi(P_i) N_i \tag{6.2}$$

wie in Abbildung 6.5 skizziert. Auch hier kann eine Sub-Voxelauflösung durch Interpolation des Distanzfeldes erzielt werden, bspw. durch die Verwendung eines trilinearen Verfahrens (siehe Kapitel 3.3.4.2, S. 36).

6.4.4. Umsetzung und Vergleich

Neben anderen Untersuchungen wurden beide Modelle im Rahmen der Masterarbeit [82] von Christoph Vollmer implementiert und verglichen. Ziel war es, eine haptische Schnittstelle für YaDiV zu entwickeln. Eine wichtige Frage war es dabei auch, wie gut sich die implementierten Modelle in eine Java-Applikation integrieren lassen, sowohl aus Performance-Gründen als auch aufgrund der Tatsache, dass die Gerätetreiber bzw. die von den Herstellern der haptischen Geräte mitgelieferten Bibliotheken meist nur für C/C++ zur Verfügung stehen.

Das flächenbasierte Modell lässt sich relativ einfach implementieren. Aus Performance-Gesichtspunkten wurde hier jedoch eine über JNI (Java Native Interface) eingebunden Lösung in C++ verwendet. Zur Initialisierung des haptischen Renderings werden Objektgeometrie und (Sicht-) Transformationen an die in C++ programmierte *Servo-Loop* übergeben. Dort findet die eigentliche Kraftberechnung statt, während die Java-API regelmäßig die Position des HIP erhält und für die Visualisierung zuständig bleibt. Diese Umsetzung wurde mit verschiedenen Geometriegrößen getestet. Selbst bei knapp 1 Millionen Dreiecke lag die Zeit für die Berechnung der Kollision und rückwirkenden Kraft nur knapp über 0,01 ms[1], so dass die geforderte Update-Rate von 1 kHz leicht erreicht werden konnte. Ohne die vergleichsweise langsame Erzeugung der OBB-Hierarchie wäre hier auch eine rein Java basierte Variante denkbar.

Das Verfahren arbeitet sehr schnell, ist aber in der vorliegenden Form nur für 3DOF-Anwendungen geeignet. Eine Erweiterung auf 6DOF ist nur mit erheblichem Aufwand möglich. Ein weiterer Nachteil dieses Verfahrens ist die relativ aufwendige Berechnung der OBB-Hierarchie, die mit der derzeitigen Implementation als Top-Down-Verfahren bei komplexen Segmenten bis zu 15 Sekunden dauern kann. Eine alternativ implementierte native Java Lösung erwies sich um den Faktor 3-5 langsamer.

Da das voxelbasierte Modell ohne Kollisionserkennung und aufwendige Datenstruktur auskommt, wurde hier eine plattformunabhängige Form des haptischen Renderings gewählt. So wurde die gesamte Kraftberechnung in Java implementiert, die Anbindung des haptischen Geräts erfolgt dabei als Client-Server-Modell (Abbildung 6.6) über eine Socket Verbindung. Die Initialisierung des Distanzfeldes dauerte auf dem gleichen Test-System auch bei großen Segmenten nur knapp über eine Sekunde.

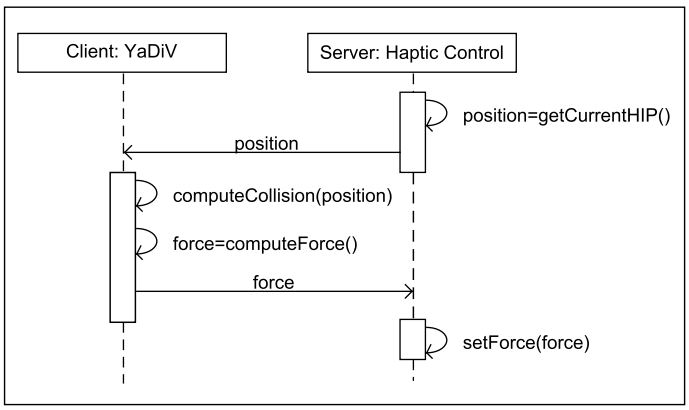


Abbildung 6.6.: Haptik: Ansteuerung über Client-Server Modell *Quelle: [82]*

Ein großer Vorteil liegt in der Möglichkeit der Erweiterung auf sechs Freiheitsgrade, da hier lediglich die Normalen der Point Shell an die Rotationen des Endeffektors angepasst

[1]Testsystem: Intel Core 2 Duo Prozessor mit 2.4GHz, 4GB Arbeitsspeicher und einer NVidia 9400m Grafikkarte.

werden müssen. Weiterhin erlaubt dieses Modell eine vergleichsweise einfache Implementation von Deformationsverfahren, z.B. indem das Voxelgitter durch ein Federmodell erweitert wird, wie es u.a. in [31] beschrieben wird.

Ein Problem dieses Verfahrens liegt in der Verwendung des Distanzfeldes begründet. Da kein echter Schnittest mit einer Oberfläche stattfindet, muss die Auflösung des Distanzfeldes sehr hoch sein, da andernfalls bei schneller HIP Bewegung eine Durchdringung dünner Objekte möglich ist. Dies könnte auch durch ein angepasstes Ray Casting-Verfahren behoben werden, bei dem eine hinreichend hoch gewählte Sampling-Rate die nötige Auflösungserhöhung liefert.

6.4.5. Anwendungsmöglichkeiten

Der Einsatz von Haptik als Ein- und Ausgabemedium steht derzeit noch ziemlich am Anfang. Es gibt zwar inzwischen stabile kommerzielle Geräte, diese beschränken sich jedoch meist auf einzigen HIP bzw. Einfingerkontakt. Viele Anwendungen erfordern jedoch zumindest eine Greifbewegung mit zwei Fingern oder der ganzen Hand, deren praktische Umsetzung zwar in Prototypen mehrfach gezeigt wurde, jedoch noch nicht zur Serienreife gebracht wurde. Auch auf Softwareseite besteht noch Entwicklungsbedarf; so fehlt es an ausgereiften, geräteunabhängigen High- und Lowlevel-Bibliotheken zur Steuerung des haptischen Renderings. Derzeit muss immer noch jede Applikation selbst ein physikalisches und geometrisches Modell für die Kraftrückkopplung implementieren; wünschenswert von Entwicklerseite wäre hier sicher eine API, mit der sich Volumenkörper zusammen mit ihren physikalischen Eigenschaften („Ball, dynamisches Objekt, 30cm Durchmesser, weiches Gummi“) an die *Servo-Loop* übergeben lassen. Erst seit kurzem gibt es offene Projekte[1], um zumindest eine Geräteunabhängigkeit zu erreichen.

Grundsätzlich wird die Einführung einer haptischen Schnittstelle in die Arbeit mit biomedizinischen 3D-Daten bestehende Arbeitsläufe vereinfachen und gänzlich neue Möglichkeiten eröffnen. Ein Bereich, der in der aktuellen Forschung über Haptik in der Medizin bisher zu kurz kommt, ist der Einsatz der haptischen Schnittstelle über die konkrete chirurgische Simulation hinaus als neues Paradigma in der Mensch-Maschine-Kommunikation.

Die Haptik selbst kann in dieser Schnittstelle sowohl der Datenexploration dienen als auch dabei helfen, die menschliche Interaktion zu erleichtern. Bereits ohne aktive Kraftrückkopplung wird durch die neuen Eingabegeräte die räumliche Navigation mit sechs Freiheitsgraden (insbesondere in Verbindung mit stereoskopischer Visualisierung) stark vereinfacht. Eine Vielzahl der anfallenden Arbeiten, wie das Markieren von Punkten, Schnittflächen oder Bereichsbegrenzung innerhalb aufgenommener Volumendaten, sind mit 2D-Eingabegeräten wie der Maus nur schichtweise durchzuführen. Diese etablierte

[1] z.B. CHAI3D oder das JTouchtoolkit (für Java Anwendungen).

Arbeitsweise ist zwar sehr präzise, aber zugleich auch unkomfortabel und zeitraubend. Hier fehlen bisher grundlegende Arbeiten, um die ergonomischen Möglichkeiten der neuen Schnittstellen besser zu verstehen. So ist es zwar – wie in einer ersten Testanwendung der haptischen Schnittstelle festzustellen war – sehr einfach, ein Segment im Raum haptisch unterstützt zu bewegen, am gewünschten Zielort lässt es sich jedoch nur sehr schwer räumlich präzise platzieren. Selbst das einfache Drücken des Knopfes am Endeffektor führt zu einer leichten Positions- oder Torsionsänderung. Für das „Festhalten“ am Zielort müssten daher bspw. eine Art Gitter aus Navigationspunkten über den Raum gelegt werden, so dass der Benutzer eine vorgegebene Kraftschwelle überwinden muss, um eine Veränderung zu erzielen. Schon dieses einfache Beispiel zeigt, dass sich neue Möglichkeiten erschließen lassen, wenn die haptische 3D-Schnittstelle durch ergonomisch bedingte Kraftrückgabe erweitert wird.

Besonders in der OP-Planung, wie z.B. bei der Reponierung eines komplizierten Trümmerbruchs, wäre die Entwicklung einer angemessenen Pseudo-Physik für die haptische Bewegung der Teilstücke wünschenswert, die nicht notwendigerweise einen realistischen Kontakt simulieren muss, sondern vielmehr ein schnelles, präzises und gleichzeitig intuitiv verständliches Arbeiten ermöglicht. Auch eine Schnittstelle für eine einfache, manuelle haptisch unterstützte Registrierung (affin oder elastisch) ist vorstellbar.

Insbesondere für die elastische Verformung müssen die bisher verwendeten Algorithmen allerdings noch weiter beschleunigt und/oder parallelisiert werden. Einen guten Einstieg in die Kollisionserkennung mit deformierbaren Objekten bietet bspw. das gleichnamige (englischsprachige) Paper [81] von Teschner et al.

7. YaDiV

Das im Rahmen dieser Arbeit entwickelte Programm YaDiV („Yet another DICOM Viewer“) ist als Plattform für die Entwicklung neuer Konzepte und Ideen aus dem Bereich der medizinischen 3D-Datenverarbeitung konzipiert. Als Datengrundlage dienen Volumendatensätze im DICOM-Format. In das Hauptprogramm wurden bereits eine Vielzahl an Modulen zur 3D-Segmentierung, -Registrierung und -Visualisierung sowie VR-Komponenten wie eine haptische Schnittstelle und die Unterstützung von Stereo-Darstellung integriert.

7.1. Vergleich mit anderen (freien) Projekten

Vor Beginn der Entwicklung von YaDiV (2006) wurde eine Evaluation der bestehenden Open Source DICOM-Visualisierungssoftware durchgeführt. Zu diesem Zeitpunkt gab es eine Vielzahl kleinerer Projekte (die meisten davon 2D basiert) in unterschiedlichen Stadien der Entwicklung. Bis auf OsiriX, das aus anderen Gründen für uns nicht in Frage kam (siehe Kapitel 7.1.3), wirkte zum damaligen Zeitpunkt keine Software besonders ausgereift. Viele der damaligen Projekte wurden in der Zwischenzeit ganz eingestellt.

Diese Situation hat sich in den letzten Jahren geändert. Mit Projekten wie Slicer oder MeVisLab sind leistungsfähige Programme mit hoher Entwicklungsdynamik entstanden. In den folgenden Abschnitten soll eine aktuelle Auswahl vielversprechender Kandidaten vorgestellt und auch eine Abgrenzung gegenüber YaDiV gegeben werden.

7.1.1. VTK und ITK

Zwar sind weder das Visualization Toolkit (VTK) noch das Insight Segmentation and Registration Toolkit (ITK) eigenständige Applikationen, da aber sehr viele der getesteten Programme auf diesen beiden Bibliotheken aufbauen, dürfen sie in dieser Übersicht nicht fehlen.

ITK wurde 1999 von der US National Library of Medicine of the National Institutes of Health in Auftrag gegeben, es handelt sich dabei um ein Open Source Toolkit für Registrierungs- und Segmentierungsverfahren, das sowohl die Algorithmen als auch die

dafür benötigten Datenstrukturen zur Verfügung stellt. Das Hauptanwendungsgebiet ist die Medizin, generell eignet es sich jedoch auch für andere Bereiche. ITK wird in C++ entwickelt und besitzt Interfaces zu vielen Programmiersprachen (Tcl, Java, Python, ...). ITK selbst bietet keine Möglichkeit zur Visualisierung.

VTK war ursprünglich Bestandteil des Buches „The Visualization Toolkit – An Object-Oriented Approach to 3D Graphics“ von Will Schroeder, Ken Martin und Bill Lorensen [73], welches sich schnell als eines der Standardwerke in der Lehre etablierte. Eine wachsende Entwicklergemeinde überarbeitete und erweiterte das Toolkit bis zu seinem heutigen Stand. Die Bibliothek stellt Funktionen aus dem Bereich 3D-Grafik, Modellierung, Bildbearbeitung, Volume Rendering und wissenschaftliche Visualisierung zur Verfügung. Wie ITK ist auch VTK in C++ implementiert. Es können sowohl polygonale als auch diskrete Daten visualisiert werden.

7.1.2. (3D) Slicer

Slicer (Aktuelle Version 3.4 von Mai 2009, auch als 3D Slicer bezeichnet) ist ein Open Source Software Projekt für Visualisierung und Bildanalyse. Das Projekt wird in C++ entwickelt und ist für verschiedene Betriebssysteme erhältlich, u.a. für Windows, Linux und MacOs X. Der Grundstein für das Programm wurde 1998 durch eine Masterarbeit in einer Kooperation zwischen dem Surgical Planning Laboratory at the Brigham and Women's Hospital und dem MIT Artificial Intelligence Laboratory gelegt. 2007 erfolgte dann eine komplette Überarbeitung mit einer moderneren Architektur die 2008 zur ersten Stable Release 3.2 führte. Als Lizenzmodell wird BSD verwendet. Eine kommerzielle Nutzung ist ebenfalls möglich.

Slicer ist vollständig modulbasiert und erlaubt die Entwicklung externer Bausteine. Bereits integriert sind Module zur Visualisierung, Segmentierung, Registrierung und diverse Filterpakete. Bei der Implementierung wurden in erster Linie etablierte Standardbibliotheken wie VTK und ITK verwendet. Grundlage für die modulare Architektur ist das NA-MIC Kit.

7.1.3. OsiriX

Eines der ältesten und seit langem sehr erfolgreichen Open Source Programme für die Visualisierung multimodaler Bilddaten ist OsiriX. Wie auch Slicer setzt OsiriX auf VTK/ITK auf, wird jedoch in Objective-C und ausschließlich für das Betriebssystem Mac OS X entwickelt. Als Lizenzmodell wird die GNU General Public License verwendet.

Aufgrund der Tatsache, dass OsiriX nur unter MacOS X lauffähig war (und ist), konnte das Programm vergleichsweise eingeschränkt getestet werden, da dem Institut nur bedingt

Apple Computer zur Verfügung stehen. Da außerdem keiner unser medizinischen Partner an der Medizinischen Hochschule Hannover (MHH), auf deren Zusammenarbeit sich unsere Bemühungen im Bereich Medical Imaging stützt, bisher Apple Computer einsetzt, war dies einer der Gründe, eine eigene, plattformunabhängige Software zu entwickeln.

7.1.4. MeVisLab

Im Gegensatz zu den anderen hier vorgestellten Programmen ist MeVisLab selbst kein Open Source Paket, sondern es handelt sich um ein Software Developement Kit (SDK), das es erlaubt, über einen graphischen Editor eigene Module zur 3D-Bildbearbeitung und -Analyse zu entwickeln.

MeVisLab wurde ursprünglich von Fraunhofer MEVIS entwickelt. Im Jahr 2008 wurde die Entwicklung in die MeVis Medical Solutions AG ausgelagert. Das Programm ist ebenfalls auf VTK/ITK aufgebaut. Das Programmpaket erlaubt die kommerzielle und nichtkommerzielle Nutzung in vier unterschiedlichen Paketen (Commercial, Non-commercial, Evaluation und Unregistered).

Durch die gelungene Umsetzung des Moduleditors eignet sich MeVisLab sehr gut für anwendungsbezogene Forschung und insbesondere für die schnelle und einfache Kombination bestehender Module. Die Entwicklung und Erprobung völlig neuer Konzepte gestaltet sich etwas schwieriger. Da MeVisLab selbst nicht quelloffen ist, ist es für unsere Bedürfnisse, die auch die Grundlagenforschung im Bereich medizinische Visualisierung und Segmentierung mit einschließt, nicht geeignet.

7.1.5. Weitere DICOM Software

Neben den 3D Programmen gibt es eine Vielzahl weiterer Programme zur Analyse oder Manipulation von DICOM Daten. Viele davon sind 2D basiert, einige arbeiten nur auf der Kommandozeile, bei einigen wurde die Entwicklung inzwischen eingestellt oder der Vertrieb kommerziell. Eine Übersicht ist nur für eine sehr begrenzte Zeit interessant, zumal es diverse Übersichtsseiten im Internet gibt. Da deren Installation und Erprobung eine mühsame Aufgabe darstellt, sollen an dieser Stelle nur verkürzt einige der Programme vorgestellt werden, die während der Entwicklung von YaDiV, u.a. während der Entwicklung des DICOM-Parsers, verwendet und getestet wurden.

- **DICOManonymizer**, Multiplattform, kommerzielle und nicht-kommerzielle Version erhältlich, nicht quelloffen.
  Java basiertes Programm zur Anonymisierung von DICOM-Dateien. Die freie Version (DICOManonymizer Light) besitzt viele Einschränkungen, u.a. kann immer

nur eine einzelne Datei anonymisiert werden (kein Batch Modus).
Anbieter: NeoLogica. Aktuelle Version: 1.1.5.

- **DICOMdumper**, Multiplattform, Freeware, nicht quelloffen.
  Einfaches Java Programm (mit GUI), das die Data Elements einer DICOM Datei als Text Datei ausgibt.
  Anbieter: NeoLogica. Aktuelle Version: 1.1.

- **DICOMscope**, Freeware, erhältlich für Windows, Quelltext in Java und C++.
  DICOMscope wurde im Rahmen einer vom „Committee for the Advancement of DICOM“ ausgeschriebenen Demonstration als Prototyp implementiert und seitdem weiterentwickelt. 2D DICOM Image Viewer für unkomprimierte, monochrome DICOM-Bilder aller Modalitäten, unterstützt Monitorkalibrierung nach Teil 14 des DICOM-Standards sowie das Supplement 33: Grayscale Softcopy Presentation State Storage. Basiert auf dem OFFIS DICOM-Toolkit DCMTK, die graphische Benutzerschnittstelle wurde in Java implementiert, kann kein komplettes Verzeichnis öffnen.
  Entwickelt von M. Eichelberg, K. Kleber, J. Riesmeier, A. Schröter und A. Thiel in einer Zusammenarbeit von Kuratorium OFFIS, Institute for Microtherapy Bochum und OTech Inc., TX, USA. Aktuelle Version: 3.5.1 (Juli 2001, Binärversion 3.6.0 ist verfügbar, Zukunft scheint unklar).

- **DICOM Works**, Windows, freie Version erhältlich, nicht quelloffen.
  Freier 2D DICOM-Viewer, Konverter und Data Element Editor (inkl. Anonymisierung und Batch Modus). Erlaubt Directory Scan, mehrere 2D-Filter, Messung im Bild.
  Entwickelt von Philippe Puech. Aktuelle Version: 1.3.5 vom 4.8.2002, Entwicklung scheint eingestellt.

- **DPTools (ACTIV 2000)**, Windows, Freeware, nicht quelloffen.
  Analyse von Diffusion und Perfusion in MR und CT, Koregistrierung, statistische Analyse, beinhaltet Activ 2000. Programmiersprache ist Delphi. Besonderheit: Erfordert persönliche Registrierung beim Autor. Entwickelt von Denis Ducreux. Aktuelle Version: 4.30.1.

- **EViewBox**, Freeware (GNU License), Multiplattform, Quelltext in Java.
  Bildbetrachter, der unter anderem DICOM-Bilder anzeigen kann. Hauptsächlich für 2D konzipiert, erlaubt aber auch multiplanare Rekonstruktion, wenn die Bilddaten in gleicher Größe vorliegen. Ausführbares Jar-File das nicht installiert werden muss. Kann nur Daten einlesen, die auf dem gleichen Laufwerk liegen.
  Entwickelt von Serge Derhy. Aktuelle Version ist unbekannt, Entwicklung scheint eingestellt.

- **ezDICOM**, Freeware, Windows, OpenSource (BSD), Quelltext in Delphi.
  2D DICOM Viewer. Erlaubt das Öffnen mehrere Bilder auf einmal sowie mit unterschiedlichen LUT. Import von DICOM, Analyze, Raw oder ECAT 6/7.
  Entwickelt von Wolfgang Krug and Chris Rorden. Aktuelle Version: 1, Rev. 24 (Dezember 2002).

- **FP Image**, Windows, Kostenlose Testversion erhältlich, nicht quelloffen.
  Image Processing Software, enthält DICOM Image Viewer und Browser. Erlaubt Scripting. Aktuelle Version 0.8.01.09 (15.1.2008), Entwicklung scheint eingestellt.

- **Jivex**, Windows + Mac OS X, kostenlose Personal Edition erhältlich, nicht quelloffen.
  2D DICOM-Viewer, erlaubt Annotaions und einfache Bildoperationen (Spiegeln, Rotieren um 90°, ...) sowie den Vergleich von Bildern und Serien durch eine Matrixansicht. Das Programm ist in Java implementiert, die Personal Edition ist im Funktionsumfang eingeschränkt.
  Anbieter: VISUS Technology Transfer GmbH. Aktuelle Version 4.3.0.4 (29.04.2009).

- **MIPAV**, Multiplattform, Freeware, nicht quelloffen.
  Medical Image Processing Tool, implementiert in Java. Quantitative Analyse und Visualisierung medizinischer Daten. Beinhalted PACS Client, Filterpakete, viele nützliche Funktionen, z.B. DICOM Anonymisierung ganzer Verzeichnisse. Erweiterbar durch Plugin Konzept, Module für Direct Volume Rendering (relativ langsam und sehr speicherintensiv).
  Entwickelt von Matthew McAuliffe, CIT Center for Information Technology, National Institutes of Health. Aktuelle Version: 4.3.0 vom 26.6.2009.

- **MRIcron**, Windows, Linux, Mac OS X, Quelltext in delphi.
  Anzeige und Analyse von NIfTI/Analyze Dateien, spezialisiert auf MRI Daten. Unterstützt Single- und Multislice View. Besonderheit: Einfacher 3D Software Renderer.
  Entwickelt von Chris Rorden. Aktuelle Version: 7 (Juli 2009).

- **Rubo Dicom Viewer**, Windows, freie Evaluations-Version erhältlich, nicht quelloffen.
  Reiner 2D DICOM-Viewer, unterstützt nahezu alle Bildformate. Erlaubt Directory Scan, kann Data Elements anzeigen. Ebenfalls erhältlich: DICOM-Parser (Textansicht + Anonymizer für einzelne Dateien).
  Entwickelt von Rubo Medical Imaging. Aktuelle Version: 2.0 build 6533.

- **SimpleDICOM**, Freeware, erhältlich für Windows, nicht quelloffen.
  2D DICOM Image Viewer, erlaubt Scannen von Directories nach DICOM Datensätzen, 2D Ansicht, Zoom, ändern von Window Width + Window Center, Export

der Data Elements als XML Datei.
Entwickelt vom UPMC Department of Radiology. Aktuelle Version: V1.1.0045.

Neben den echten DICOM-Visualisierungs- und Bearbeitungsprogrammen, gibt es auch viele klassische 2D-Bildbetrachter die DICOM-Dateien (zum Teil allerdings sehr eingeschränkt) lesen können, wie z.B. ImageJ, ImageMagick oder XNView.

7.2. Konzept

In diesem Kapitel soll ein kurzer Einblick in einige grundlegende Konzepte gegeben werden, die in YaDiV Verwendung finden. Da es sich um ein komplexes Softwaresystem handelt, werden in diesem Kapitel nur modulübergreifende Konzepte behandelt, während modulinterne Konzepte (wie die Layer- und 2D-Renderer-Verwaltung im Viewport2d) in späteren Kapiteln separat behandelt werden.

7.2.1. Java3D

Da Java3D in YaDiV für alle interaktiven 3D-Visualisierungskomponenten eingesetzt wird, soll an dieser Stelle (in der gebührenden Kürze) in diese Bibliothek eingeführt werden, um in den darauf folgenden Kapiteln einige der Konzepte verwenden zu können. Einen guten Einstieg in die Java3D API findet man z.B. in [13] oder [63].

Intel, Silicon Graphics, Apple und Sun entwickelten 1996 parallel verschiedene *scene graph* basierte 3D-Bibliotheken (englisch API = Application Program Interface). Da alle Firmen planten, auch eine Java Version herauszubringen, wurde beschlossen die Entwicklung zusammenzulegen. Die Entwicklung von Java3D wurde ursprünglich von Sun koordiniert und führte 1998 zur ersten Release von Java3D 1.0. Seit 2004 wird Java3D als Open Source Community Projekt weitergeführt und ist zur Zeit für Linux, Solaris, Windows und Mac OS X erhältlich. Die zur Zeit dieser Arbeit aktuelle Softwareversion ist 1.5.2.

Java3D ist eine High-Level API für die Darstellung von 3D Daten in Java Programmen. Intern benutzt Java3D je nach Version unterschiedliche Low-Level Bibliotheken, z.B. OpenGL oder unter Windows wahlweise auch DirectX. Es besteht eine hohe Verwandtschaft mit Konzepten aus VRML bzw. X3D, so ist z.B. auch in Java3D der *scene graph* das tragende Konstrukt für alle sichtbaren und unsichtbaren Szenenelemente. Weitere *scene graph* basierte APIs sind z.B. OpenSG oder die etwas älteren Bibliotheken PHIGS und SGI Inventor

Der **scene graph** von Java3D ist eine Baumstruktur[1], in der die Position, Geometrie und Material von geometrischen Objekten ablegt werden, aber z.B. auch Umweltelemente wie Beleuchtung, Nebel oder Sound. Der scene graph ist ein klassischer Graph bestehend aus Knoten und Verbindungen. Dabei beschreiben die Knoten konkrete oder abstrakte Objekte in der virtuellen Welt, die von den Verbindungen im Szenegraphen in Beziehung zueinander gesetzt werden. Es gibt zwei Arten einer Verbindung: Vater-Kind-Beziehung (parent-child relationship) und Referenz-Beziehung (reference relationship). Die erste wird z.B. verwendet, wenn nach einem Transformationsobjekt mehrere geometrische Objekte folgen. Die zweite beschreibt u.a. die Beziehung von einem geometrischen Objekt zu seinen Materialeigenschaften. Die Wurzel des scene graph ist das *VirtualUniverse* Objekt, der Weg von einem Blatt (= Objekt) bis zur Wurzel beschreibt die Art und Weise in der das Objekt letztendlich gezeichnet wird.

In den Szenegraphen können verschiedene Arten von Objekten (als Knoten) eingefügt werden. Sichtbare Objekte (**Visual Objects**) sind von der Vaterklasse Shape3D abgeleitet und bestehen aus einem Geometrie- (**Geometry**) und einem Material-Objekt (**Appearance**). Beide stehen in Referenz-Beziehung zum Objekt selbst, so dass z.B. mehrere visual objects das selbe Appearance Objekt verwenden können. Eine **branch group** fasst mehrere Objekte zu einer Gruppe zusammen und erlaubt deshalb mehrere Kind-Knoten. Die **transform group** beschreibt eine affine Transformation (Translation, Rotation, Skalierung) die sich auf alle direkten und indirekten Nachfolger im *scene graph* auswirkt.

Da es sich um einen Graph handelt, kann dieser auch zur Strukturvisualisierung eines Java3D Programmes benutzt werden. Abbildung 7.1 zeigt ein Beispiel für eine typische Java3D Applikation.

Java3D wird beständig weiterentwickelt und kapselt nahezu alle Features aktueller Grafikkarten. So gibt es u.a. auch eine Schnittstelle für die GPU Programmierung (Cg und GLSL). Die Möglichkeit zur Stereo-Visualisierung (QuadBuffer Stereo oder Shutter Glasses) wird bereits seit langem direkt in der Java3D API unterstützt, was die Implementierung des YaDiV Features für die Visualisierung auf Stereoprojektionssystemen stark vereinfacht hat.

7.2.2. Verwendete Design Patterns

Bei der Programmierung von YaDiV wurden an unterschiedlichen Stellen Design Patterns verwendet. Viele davon sind inzwischen so geläufig (Iterator, Container, ...), dass sie kaum mehr gesondert beschrieben werden müssen. Neben vielen neueren Büchern bietet [20] einen guten und ausführlichen Einstieg. In diesem Kapitel soll auf einige der in YaDiV verwendeten Konzepte eingegangen werden.

[1] Eigentlich ein (gerichteter) Graph, da durch die Referenz-Beziehung mehrere visuelle Objekte bspw. die selbe Appearance verwenden können. Dennoch wird er in der Literatur meist als Baum aufgefasst.

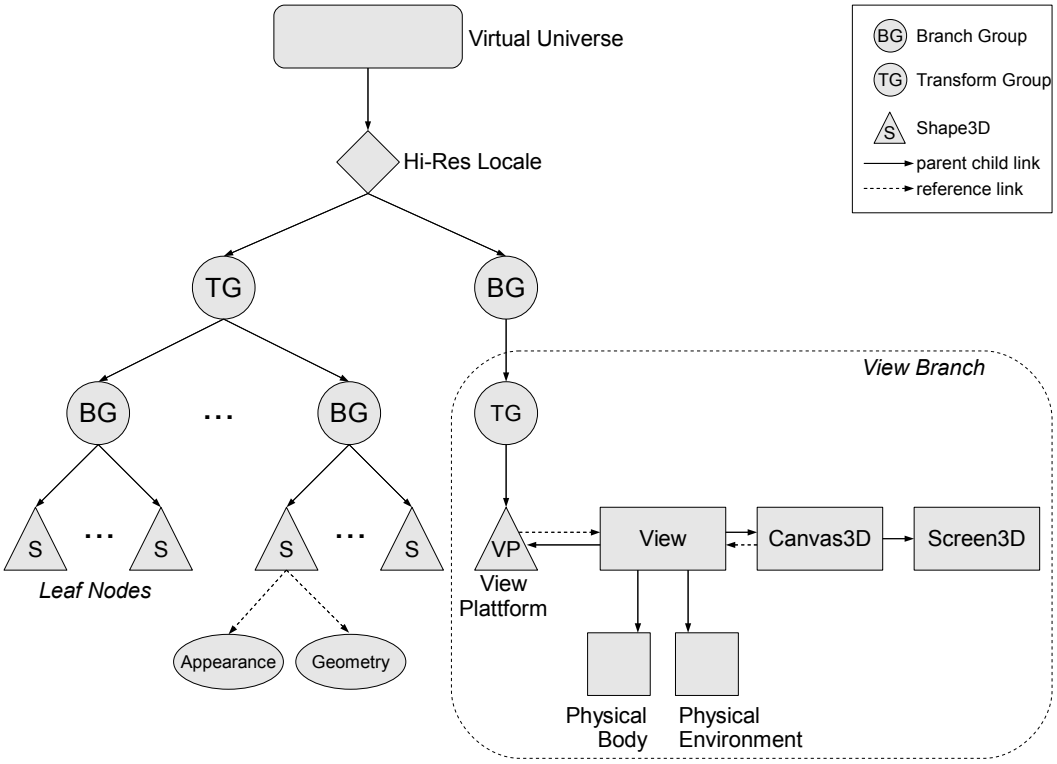


Abbildung 7.1.: Java3D Scene Graph

Ein immer häufiger anzutreffendes Paradigma ist das **Singleton Pattern**, welches verhindert, dass von einer Klasse mehr als eine Instanz erzeugt werden kann. Dies ist z.B. dann erwünscht, wenn es sich um eine speicherintensive Klasse mit statischen Informationen handelt, die als Nachschlagewerte dienen. Die Singleton Umsetzung in Java kann entweder über private Konstruktoren oder ausschließlich statische Methoden/Attribute erfolgen. In YaDiV wurde bspw. das DICOM Data Dictionary als Singleton implementiert, aber auch global einmalige Instanzen wie der ImageStack oder die MasterControl Klasse.

Damit eine Instanz eines Objektes auf Änderungen eines anderen reagieren kann, werden über das **Observable-Observer Pattern** Messages ausgetauscht. So trägt sich z.B. Viewport2d als Observer der globalen ImageStack Instanz ein, die u.a. die Nachrichten „DICOM Ladevorgang hat begonnen“, „Neues Schichtbild wurde geladen“ oder „DICOM Ladevorgang ist abgeschlossen“ verschickt. So kann der Benutzer im Viewport2d bereits geladene Schichtbilder ansehen, während der Ladevorgang noch andauert. Da YaDiV thread-basiert implementiert wurde und die GUI während länger andauernder Prozesse nicht blockiert, ist es wichtig das alle GUI-Elemente zur Erzeugung von Segmenten gesperrt werden wenn bspw. ein Gaußscher Weichzeichner oder ein Segmentierungsalgorithmus begonnen hat und noch nicht abgeschlossen wurde. Alle diese Nachrichten werden

über das Observable-Observer Pattern ausgetauscht.

Eine Variante des **Property-Change-Listener** kommt in der für YaDiV entwickelten Settings API (Kapitel 7.2.4) zum Einsatz. Während das Observable-Observer Pattern für die Benachrichtigung über globale Ereignisse wie Start oder Ende eines Filters verwendet werden, erlaubt die Settings API die Reaktion auf die Änderung einzelner Objektattribute. Jedes dieser Attribute bekommt dabei automatisch ein GUI Element über den Settings-Browser zugewiesen. Speziell angepasste GUI Elemente können ebenfalls über die Settings API angefordert werden, dabei wird ein **Factory Pattern** verwendet.

7.2.3. Messages

Nachrichten werden in YaDiV durch eine Instanz der Message-Klasse repräsentiert. Eine Nachricht besteht dabei immer aus einem eindeutigen Typ (int) und einem optionalen Objekt. Die Nachrichtentypen werden zentral verwaltet aber dezentral angemeldet; ein potentieller Nachrichtensender muss zunächst einen Nachrichtentyp registrieren.

Dazu gibt es in der Message-Klasse die statische Methode

```
int register_message(String message_name)
```

über die alle Nachrichtentypen registriert werden. Rückgabwert ist eine eindeutige ganzzahlige Kennung. Zur Laufzeit müssen dann nur noch die Kennungen verglichen werden. Die Zuordnung vom String zur Kennung wird in der Message-Klasse selbst einmalig gespeichert. Durch diesen Mechanismus ist das System einerseits flexibel um neue Nachrichtentypen erweiterbar, andererseits ist es immer noch möglich die menschenlesbare Kennung einer Nachricht abzurufen, bswp. bei der Fehlersuche. Zur Laufzeit bleiben die Typ Kennungen eindeutig. Bei einem erneuten Programmstart mit veränderter Registrierungsreihenfolge können sich die Kennungen jedoch gegenüber dem letzten Programmlauf ändern.

Tabelle 7.1 zeigt einen Auszug der Messages die beim Programmstart von YaDiV registriert werden.

Mit jeder Message kann auch ein Objekt versendet werden, dies ist aber optional und muss mit dem entsprechenden Message Observer vereinbart werden, da sich in dem Fall Sender und Empfänger über den Objekt-Typ einig sein müssen.

7.2.4. Settings

Neben der Message Klasse bildet die ursprünglich ausschließlich für YaDiV konzipierte Settings API einen zentralen Bestandteil in der Entwicklung. Aus Sicht des Benutzers ist ein Setting ein dynamisches Attribut, das zur Laufzeit an eine Instanz oder auch an eine

ID	Unique String	Beschreibung
0	ImageStack Clear	Schichtbilddaten wurden gelöscht
1	ImageStack LoadingStarted	Ladeprozess Anfang
2	ImageStack LoadingEnded	Ladeprozess Ende
3	ImageStack NewImageLoaded	Neues Bild wurde geladen
4	ImageStack FirstImageLoaded	Erstes Bild wurde geladen
5	ImageStack NewActiveImage	Neues aktives Bild
6	ImageStack NewSegemt	Neues Segment wurde erstellt
7	ImageStack DeleteSegemt	Segment wurde gelöscht
8	ImageStack ResizeStart	VoxelCube Skalierung Anfang
9	ImageStack ResizeEnd	VoxelCube Skalierung Ende
10	ImageStack GaussStart	Gauss Filter Anfang
11	ImageStack GaussStep	Gauss Filter Zwischenschritt (pro Schicht)
12	ImageStack GaussEnd	Gauss Filter Ende
13	ImageStack SegStart	Segmentierungsprozess Anfang
14	ImageStack SegEnd	Segmentierungsprozess Ende
15	ImageStack SelectXYZ	Neuer Seed wurde gewählt
16	Segment NameChanged	Segmentname geändert
17	Segment ColorChanged	Segmentfarbe geändert
18	Segment Changed	Segmentdaten geändert
19	Segment SizeChanged	Segmentgröße geändert
20	SegGen BB Changed	Bounding Box für Segment-Generator geändert
...	...	...

Tabelle 7.1.: Messages in YaDiV

Klasse selbst angehängt werden kann. Andere Objekte haben zusätzlich die Möglichkeit sich als Listener eines Objekts (oder einer Klasse) einzutragen, um auf Veränderung der Attribut-Werte reagieren zu können.

Ein mögliches Anwendungsbeispiel sind unterschiedliche GUI-Elemente, welche die selbe Funktion ausführen. So gibt es in YaDiV z.B. drei unterschiedliche Wege um die Texture3D-Darstellung einzuschalten: über das Hauptmenü, über einen graphischen Button im Viewport3d oder über einen Checkbox-Button im Viewport3d ToolPane Element. Alle diese Elemente zeigen aber auch gleichzeitig den Zustand an; der graphische Button ist bei aktivierter Texture3D-Darstellung hervorgehoben, Menü- und ToolPane Element zeigen in diesem Fall ein gesetztes „Häkchen“. Aktiviert der Benutzer die Texture3D-Darstellung über eines dieser drei Elemente, müssen die beiden anderen ebenfalls ihre Statusanzeige verändern, um in der Darstellung konsistent zu bleiben.

Die Settings API vereinfacht diese Situation, in dem sie zum einen die dafür benötigten Operationen zur Verfügung stellt und zum anderen die Verschränkung von Settings-

Attributen mit den für den jeweiligen Attribut-Typ üblichen GUI-Elementen bereits automatisch anbietet. Im konkreten Anwendungsfall definiert die Klasse *Tex3Renderer3d* ein Settings Attribut `OPT_RENDER_BG`, das von der *MenuBar*-, der *Viewport3d*- und der *ToolView3d*-Klasse jeweils mit den entsprechenden GUI-Elementen verknüpft wird. Jede Änderung des Attributes wirkt sich automatisch auf alle damit verschränkten GUI-Elemente aus. In den ersten zwei Zeilen von Listing 7.1 ist die Erzeugung von zwei mit Settings Attributen verschränkten GUI-Elementen exemplarisch zu sehen. Unabhängig von den Default-Elementen können auch eigene Bedien-Elemente definiert werden, diese müssen dann die entsprechenden Interface-Funktionen selbst implementieren.

Die Settings API ist bewusst auf wenige, einfache Attribut-Typen beschränkt: unterstützt werden boolean, integer, double, string, color, und enumeration Attribute. Um diese Attribute nutzen zu können, muss die entsprechende Klasse das *SettingsOwner*- oder das *SettingsListener*-Interface implementieren, je nachdem ob sie eigene Settings-Attribute besitzen oder auf die Statusänderungen von Settings-Attributen anderer Klassen reagieren möchte. Ein SettingsListener kann sich als Listener eines SettingsOwners oder der Klasse selbst anmelden und wird danach automatisch bei jeder Attribut Änderung informiert. Ein SettingsOwner ist dabei automatisch auch ein SettingsListener – der über die Änderungen der eigenen Attribute informiert wird – und muss zusätzlich einen eindeutigen Namen als Identifier zur Verfügung stellen.

Ein Settings-Attribut kann an eine Instanz oder an eine Klasse gebunden sein. Dabei dient das Klassen-Attribut für alle Instanzen als Vorgabe, solange diese kein eigenes Attribut mit dem gleichen Namen mit anderem Wert definiert haben. Bei einer Änderung des Klassenattributes werden folglich auch die Listener aller Instanzen aufgerufen, die dieses Klassenattribut verwenden. Intern werden die Listener zugriffseffizient in Hash-Tabellen verwaltet, so dass bei einer Attributänderung nur die für diese Instanz bzw. Klasse eingetragenen Listener durchlaufen und aufgerufen werden müssen.

Ein weiterer Bestandteil der API ist der sogenannte **Settings Browser**, der für jedes Objekt- oder Klassenattribut ein einfaches GUI Element einbindet. Auf diese Weise ist es sehr einfach möglich, einen Algorithmus mit unterschiedlichen Parametern zu testen, ohne für jede Variable erst ein neues GUI-Element schreiben zu müssen, da im Settings Browser bereits ein rudimentäres Eingabe-Element – hierarchisch und alphabetisch geordnet nach Klasse, Instanz- und Attributname – vorhanden ist. Abbildung 7.2 zeigt den geöffneten Settings Browser von YaDiV.

Genau wie die Message-Klasse verwendet auch die Settings-Klasse statische Methoden, um die Attribute zur Laufzeit zu registrieren bzw. um Attribut-Listener anzumelden. Ebenfalls implementiert ist ein Mechanismus, um Settings-Werte dauerhaft zu speichern und beim nächsten Programmstart automatisch wieder einzulesen.

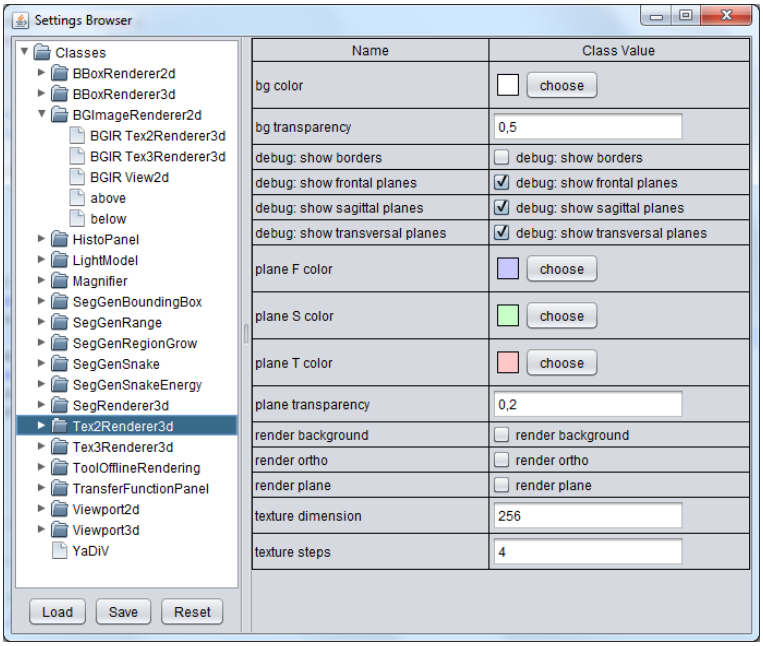


Abbildung 7.2.: Settings Browser in YaDiV

7.2.5. JGridMaker

Ein weiteres Konzept, das im Rahmen der YaDiV Entwicklung entworfen und anschließend als eigenes OpenSource Paket veröffentlicht wurde, ist die JGridMaker API. Um die Lesbarkeit des Quelltextes für das Layout von GUI-Elementen zu erhöhen, wurde einr API entworfen, die deren Anordnung in einer an HTML Tabellen angelehnten Syntax beschreibt.

Die *JGridMaker*-Klasse selbst ist dabei als Singleton konzipiert und stellt statische Methoden zum Layout zur Verfügung. Die API kapselt so die Funktionalität des Swing GridBag Layout von Java. Dieses ist sehr flexibel, jedoch ist die Gestaltung einer anspruchsvollen Oberfläche nur mit sehr eingeschränkt menschenlesbaren Quelltext möglich. Listing 7.1 zeigt ein einfaches Beispiel, in dem zwei über die Settings API erzeugte GUI Elemente mittels JGridMaker in einer 2x2 Tabelle angeordnet werden.

Hauptbestandteil der API ist der XML-Parser, der das als String übergebene Layout in das Swing GridBag Layout umsetzt. Ein Nebenprodukt des JGridMaker Konzeptes ist es, dass Funktionalität und Layout voneinander getrennt werden können, da ein einmal entworfenes Layout für eine bestimmte Form von Dialog als reine Textbeschreibung auch an anderen Programmstellen wiederverwendet kann. Zudem unterstützt die API auch die Anordnung und die Erzeugung der Elemente über eine ausgelagerte Textdatei, so dass GUI-Entwickler Design Änderungen vornehmen können ohne notwendigerweise den Quelltext verändern zu müssen.

```
final JEnumOptionComboBox jeocb_color_mode
        = new JEnumOptionComboBox(ir2d, BGImageRenderer2d.OPT_COLOR_MODE);
final JEnumOptionComboBox jeocb_filter
        = new JEnumOptionComboBox(v2d, Viewport2d.OPT_SCALE_FUNC);

final GMPanel left_panel = new GMPanel(); // add components
left_panel.add("jeocb_color_mode", jeocb_color_mode);
left_panel.add("jeocb_filter", jeocb_filter);

left_panel.set_layout( // layout components
        "<table width='100%' cellpadding='0' border='0'>"+
        "  <tr>"+
        "    <td anchor='west'>Color Mode</td>"+
        "    <td anchor='east'>::jeocb_color_mode::</td>"+
        "  </tr><tr>"+
        "    <td anchor='west'>Skalierung</td>"+
        "    <td anchor='east'>::jeocb_filter::</td>"+
        "  </tr>"+
        "</table>");
```

Listing 7.1: JGridMaker Beispiel

7.3. Wichtige Datenstrukturen und Klassen

Auch wenn diese Arbeit eher einen allgemeinen Überblick über das entwickelte Programm geben soll, ist es an dieser Stelle sinnvoll einige Datenstrukturen und Klassen zu erläutern, die bei der Implementierung eine zentrale Rolle einnehmen.

7.3.1. ImageStack und VoxelCube

Die Datengrundlage von YaDiV sind Schichtbilder im DICOM Format. Das DICOM Format speichert neben den reinen Bilddaten und rein technischen Informationen (Bittiefe pro Pixel, x- und y-Abstand zwischen zwei Pixeln, etc.) auch Metadaten wie Patientenname, -alter und andere untersuchungsrelevante Daten. Diese nicht-technischen Metadaten sind für die meisten Module von YaDiV von untergeordneter Bedeutung, dennoch soll auch auf diese Daten zugegriffen werden können.

Die zentrale Klasse in YaDiV, die alle Schichtbilder verwaltet, ist die *ImageStack*-Klasse („Bildstapel“). Der ImageStack ist sowohl für das Einlesen der Daten, als auch für deren Bereithaltung zur Laufzeit zuständig. Dabei werden die Metadaten in einer an das DICOM Format angelehnten Datenstruktur nebst Schnittstelle zur Verfügung gestellt. Die technisch relevanten Daten werden gesondert extrahiert und in geeigneten, zugriffs- und speicheroptimierten Datenstrukturen abgelegt.

Die eigentlichen Schichtbilddaten werden dabei in der *VoxelCube*-Klasse (bzw. Datenstruktur) abgelegt. Um später lästige (und zeitaufwendige) Fallunterscheidungen zu vermeiden, werden beim Aufbau des VoxelCube einige Konventionen eingehalten:

1. Unabhängig vom Format der Pixeldaten in der DICOM Datei werden die Intensitätswerte im VocelCube immer positiv ganzzahlig abgelegt und *window width*, *window center*, *intercept* und *slope* entsprechend angepasst. Dies vereinfacht nicht nur den Zugriff, sondern erlaubt auch effiziente Lookup-Tabellen für aufwendigere Berechnungen, in denen für jeden Intensitätswert der entsprechende Umrechnungswert abgelegt werden kann.
2. Unabhängig von der Scan-Ebene erfolgt die Ablage der Schichtbilder im VoxelCube immer so, dass transversale Schnitte in der xy-Ebene liegen, die mit der z-Achse absteigend angeordnet sind. Die Anordnung der Schichten entspricht immer HFP („Head-First-Prone", siehe Kapitel 3.1.5, (0018,5100) Patient Position).

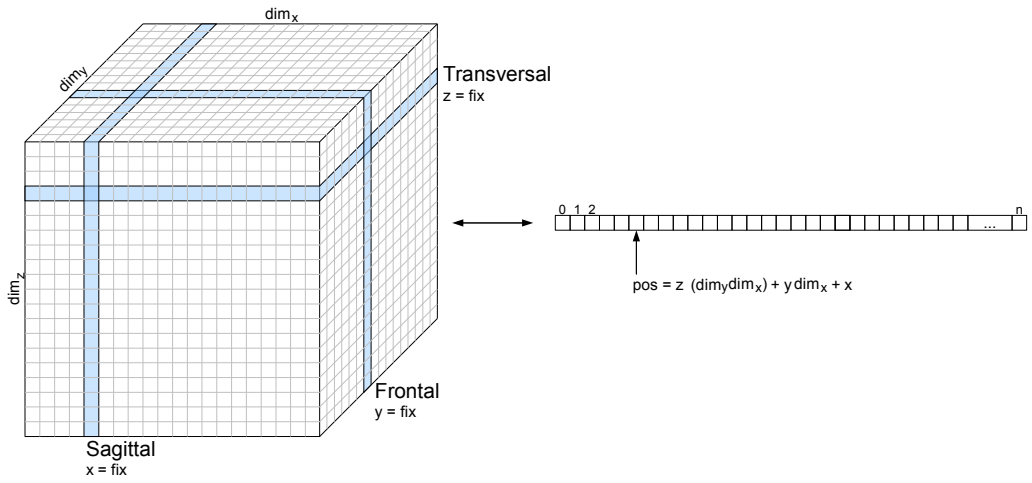


Abbildung 7.3.: Anordnung der Schichtbilder im VoxelCube und internes Array

Abbildung 7.3 skizziert die Ablage der Schichtbilder im VoxelCube. dim_x, dim_y und dim_z bezeichnen die Dimension des VoxelCube in x-, y- und z-Richtung. Durch die festgelegte Anordnung entspricht dim_x der Breite und dim_y der Höhe eines transversalen Schnittbildes, deren Anzahl gerade dim_z entspricht. Bei transversalen Schichtbildern ist die z-Koordinate fixiert, bei sagittalen die x- und bei frontalen die y-Koordinate.

Beim Aufbau des VoxelCube wird vom ImageStack zusätzlich ein Histogramm erstellt, das für jeden Intensitätswert zählt wie oft dieser im VoxelCube auftritt. Außerdem enthält der ImageStack eine Referenz auf alle für diesen Datensatz erzeugten Segmente.

Um eine möglichst menschenlesbare Darstellung zu erhalten, ähnelt das Interface des VoxelCube dem eines dreidimensionalen Arrays. Der Zugriff erfolgt lesend und schreibend über die Methoden

`int get_xyz(int x, int y, int z)` bzw.
`void set_xyz(int x, int y, int z, int value)`.

Intern wird jedoch aus Effizienzgründen ein eindimensionaler Array verwendet, in dem die Intensitätswerte nach Schichtbild und Zeile aufsteigend angeordnet vorliegen. Für optimierte Zugriffe ist es außerdem möglich, direkt auf den eindimensionalen Array zuzugreifen. Der Index I für den Intensitätswert an der Stelle (x, y, z) entspricht dann

$$I(x, y, z) = z * dim_x \cdot dim_y + y \cdot dim_x + x$$

Diese Art des Zugriffs ist sehr effizient, allerdings leidet die Lesbarkeit des Quelltextes. Daher sollte sie nur dann verwendet werden, wenn es sich um eine extrem laufzeitkritische Stelle handelt.

7.3.2. Segment und BitCube

Ein Segment in YaDiV wird durch die gleichnamige Klasse repräsentiert. Jedes Segment besitzt einen eindeutigen Namen, eine nicht notwendigerweise eindeutige Farbe (für die Visualisierung) und eine *BitCube*-Datenstruktur, die für jeden Voxel im VoxelCube festhält, ob dieser zum Segment gehört – oder nicht. Die Struktur muss zugriffs- und veränderungsoptimiert sein, da bei vielen typischen Aufgaben sehr oft die Segmentzugehörigkeit einzelner Voxel überprüft werden muss, z.B. in der Visualisierung (lesend) oder der Segmenterzeugung (schreibend). Da diese Information für jeden Voxel den Wert *true* oder *false* annimmt, scheint es nahezuliegen hierfür einen Array vom Typ boolean zu verwenden.

Ein solcher Array hat jedoch mehrere Nachteile. Segmente sind eine der grundlegenden Datenstrukturen. Zu einem geöffneten Datensatz können sehr viele Segmente angelegt worden sein, deren Erzeugung üblicherweise sehr zeitaufwendig ist. Die Datenstruktur muss daher sowohl zugriffs- als auch speichereffizient angelegt werden. Ein boolean kann zwar nur die Werte *true* oder *false* annehmen, wird aber intern in Java (genau wie in C++) durch eine 32 Bit[1] Ganzzahl repräsentiert. Ein *boolean* Array würde folglich das 32fache des eigentlich benötigten Speichers kosten.

Daher wird genau wie beim VoxelCube intern ein Array aus int Werten verwendet und jede einzelne Ganzzahl als Folge von 32 „gehört zum Segment" - oder „gehört nicht zum Segment"-Informationen interpretiert. Die Reihenfolge der Bits entspricht dabei der Reihenfolge der Intensitätswerte im VoxelCube, jedoch entspricht die Bitzahl pro Bildzeile immer einem Vielfachen von 32, eventuell vorhandene überzählige Bits bleiben unbelegt.

[1] Auf 64 Bit Systemen entsprechend 64 Bit, die BitCube-Klasse ist die einzige Klasse in YadiV die in einer 32- und einer 64 Bit Version existiert

Dadurch wird zwar der Speicheraufwand minimal erhöht, allerdings können in bestimmten Anwendungen erhebliche Performance-Vorteile erzielt werden.

Genau wie beim VoxelCube erfolgt der Zugriff auf die einzelnen Bits über die Methoden

`boolean get_xyz(int x, int y, int z)` und
`void set_xyz(int x, int y, int z, boolean value)`.

Auch beim BitCube ist es möglich, den intern verwendeten Array direkt zu manipulieren. Die zum Voxel (x,y,z) gehörenden Indize für den *int* Wert v_i und das entsprechende Bit v_b lassen sich dann durch

$$\begin{aligned} v_i &= z(dim_y * dim_x/32) + y(dim_x/32) + (x/32) \\ v_b &= x \bmod 32 \end{aligned}$$

bestimmen. In der Formel für die Berechnung von v_i kann ausgenutzt werden, dass bei der Teilung durch 32 der nicht ganzzahlige Rest entfällt, was dazu führt das sich diese Operation sehr effizient durch einen rechts-shift um 5 Bit implementieren lässt.

$\sim$a	Einerkomplement
$a\|b$	Binäre-Oder-Verknüpfung von a und b
$a\&b$	Binäre-Und-Verknüpfung von a und b
$a\hat{}b$	Binäre-Exklusiv-Oder Verknüpfung von a und b
$a >> n$	Rechts-Shift mit Vorzeichen (schnelle Division durch zwei)
$a >>> n$	Rechts-Shift ohne Vorzeichen
$a << n$	Links Shift (schnelle Multiplikation mit zwei)
int Integer.numberOfLeadingZeros(int i);	
int Integer.numberOfTrailingZeros(int i);	
int Integer.bitCount(int i);	
int Integer.reverse(int i);	
int Integer.rotateLeft(int i, int distance);	
int Integer.rotateRight(int i, int distance);	
int Integer.reverseBytes(int i);	
int Integer.highestOneBit(int i);	
int Integer.lowestOneBit(int i);	

Tabelle 7.2.: Nativ unterstützte Bit-Operationen in Java

Da Java native Bit-Operationen wie shift, binäres und/oder, Negation etc. direkt auf dem Prozessor ausführt, ist es möglich an dieser Stelle sehr nah an der Hardware zu implementieren. Dadurch lässt sich die Performance vieler Operationen, wie zum Beispiel einer einfachen clear() Methode, um Faktor 32 steigern. Tabelle 7.2 zeigt die in Java

verfügbaren Bit Operatoren und einige weniger bekannte statische Methoden der Integer Klasse, die ebenfalls hardwarenah mit normalen int Typen arbeiten. Die Verwendung dieser Methoden erschwert die Lesbarkeit des Quelltextes und sollte daher nur an besonders zeitkritischen Stellen erfolgen. Der erzielte Performance-Vorteil ist dann jedoch enorm. Einen sehr lesbaren Einstieg in die mit Bit-Operationen verbundenen Möglichkeiten bietet das „Hacker's Delight" Buch von Henry S. Warren [83].

Neben der Schnittstelle für den Zugriff auf einzelne Bit-Voxel bietet die BitCube Klasse einige weitere nützliche Methoden, wie z.B. die bitweise Addition oder Subtraktion eines anderen BitCubes oder eine automatische BoundingBox Berechnung. Alle diese Operationen nutzen die Speicherform des int Arrays und die damit verbundene Möglichkeit der nativen Programmierung aus.

7.4. Benutzerschnittstelle

Ziel dieses Kapitels ist es, einen Überblick über das Konzept der Benutzerschnittstelle von YadiV zu geben. Da die Schnittstelle Teil der Entwicklung ist und sich von Version zu Version verändern kann, sollen hier nur die bleibenden Strukturen beschrieben werden.

7.4.1. Das Hauptfenster

Für die Erzeugung der graphischen Benutzerschnittstelle wird die Java SWING API, falls vorhanden mit dem Nimbus Look and Feel, verwendet. Abbildung 7.4 zeigt das Hauptfenster des Programms, das aus dem View2d (mitte links), dem View3d (mitte rechts), dem Hauptmenü (oben), der ToolPane (unten links), dem Magnifier (unten rechts) und der Statuszeile (ganz unten) besteht.

Der **View2D** besteht aus drei graphischen Komponenten: die Titelzeile, die auch einige ausgewählte Bedienelemente wie den Zoom Faktor oder Knöpfe zum Umschalten zwischen transversaler, sagittaler und frontaler Ansicht enthält, einem Auswahlelement für das aktuelle Schichtbild und einem Fenster für die 2D-Visualisierung der ausgewählten Schicht. Befindet sich der Mauszeiger über einem Schichtbild, zeigt der **Magnifier** (unten rechts im Hauptfenster) eine Ausschnittvergrößerung und Informationen über den Voxel der sich unter dem Mauszeiger befindet.

Auch der **View3D** besitzt eine Titelzeile mit Bedienelementen, u.a. graphische Buttons für das Anzeigen der Texture Volume Rendering Darstellung oder den Wechsel in den Fullscreen Modus.

Die **ToolPane** ist in Karteikarten unterteilt, welche Informations- und Bedienelemente für unterschiedliche Komponenten von YaDiV enthalten:

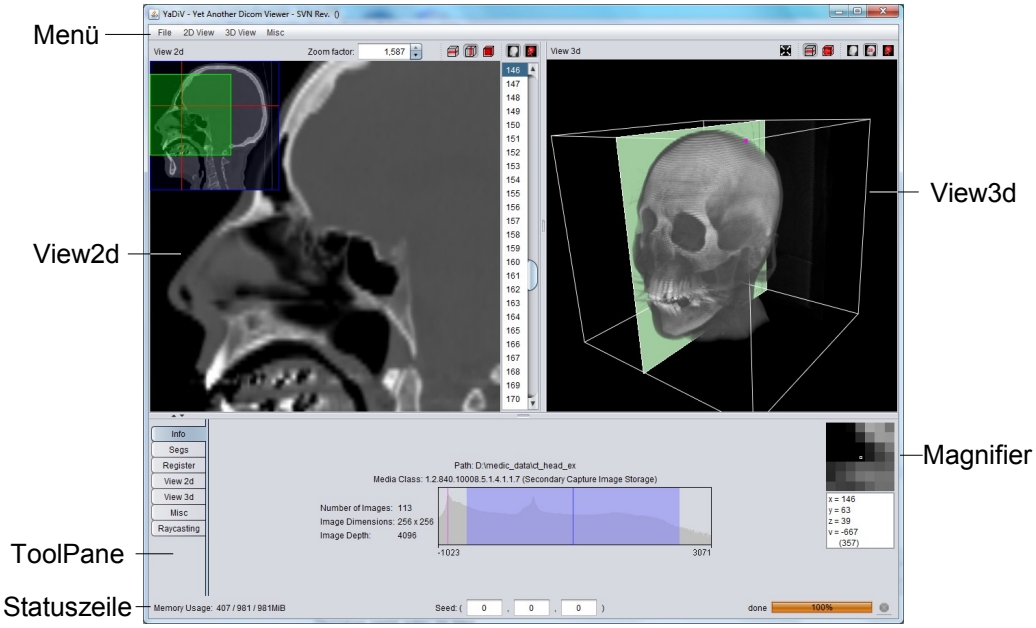


Abbildung 7.4.: YaDiV Hauptfenster

- Die Karteikarte **Info** zeigt einige ausgewählte Informationen über die aktuell geladene DICOM Serie an (Pfad, Media Class, Anzahl und Auflösung der Schichtbilder, Intensitätswert-Histogramm, Anzahl Intensitätsstufen, etc).
- Die **Segs** Karteikarte enthält alle GUI Elemente für die Segmenterzeugung und Segmentmanipulation.
- Hinter dem **Register** Karteikartenreiter verbergen sich die Bedienelemente für die haptische Registration von Segmenten.
- Alle Einstellungen zur 2D-Darstellung, z.B. Rekonstruktionsfilter oder Farbdarstellung verbergen sich hinter **View2d**.
- Die Karte **View3d** enthält Einstellungen zum View3d, z.B. Volume Texture Rendering, Segmentvisualisierungen, etc.
- Bedienelemente für Module, die sich nicht in eine der anderen Kategorien einteilen liessen (Gaußscher Filter, Resize, Segmentexport, ...), wurden unter **Misc** zusammengefasst.
- Die Karteikarte **Raycasting** enthält Einstellmöglichkeiten für das Ray Casting Modul, wie z.B. den Editor für die Transferfunktion.

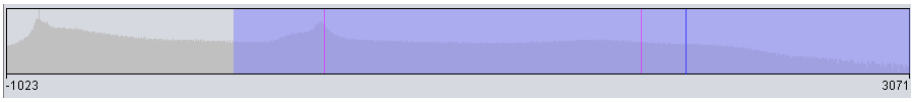
Die **Statuszeile** zeigt Informationen über den Speicherverbrauch und den zuletzt selektierten Voxel an. Ganz rechts ist ein Bedienelement, das den Fortschritt laufender Prozesse anzeigt und diese auch unterbrechen kann.

Weitere Komponenten wie der Settings Browser oder das DICOM Info Window sind nur über das Hauptmenü zu erreichen.

7.4.2. VoxelCube Histogram

Ein oft verwendetes Element in der Benutzerschnittstelle ist das VoxelCube Histogramm, das die Häufigkeit der Intensitätswerte im Volumendatensatz darstellt. Optional können auch das aktuelle VOI-Fenster dargestellt oder die Intensitätswerte der aktuell selektierten Voxel hervorgehoben werden.

(a) Original



(b) Logarithmus

Abbildung 7.5.: VoxelCube Histogramm

Für die Darstellung wird üblicherweise der Logarithmus der Häufigkeit eines Intensitätswertes verwendet, andernfalls würden die Spitzenwerte (üblicherweise Luft oder Voxel außerhalb des Aufnahmebereiches) die Darstellung dominieren, wie sich in Abbildung 7.5 erkennen lässt. Ohne eine logarithmische Werteskala ist das Diagramm (dunkelgrau im Bild) in großen Wertebereichen nicht mehr von der Nulllinie zu unterscheiden. Die dunkelblaue vertikale Linie gibt das Zentrum des VOI-Fensters an, das hellblau eingefärbt eingezeichnet wird. Die violett markierten Intensitätswerte entsprechen den Werten der aktuell selektierten Voxeln (im Beispiel zwei).

Anwendungen für das VoxelCube Histogramm sind u.a. der Dialog zur Auswahl des VOI-Fensters, die Min-Max basierte Segmentierung oder der Editor für die Transferfunktion beim Direct Volume Rendering.

7.4.3. DICOM Info Window

Neben dem Viewport2d bietet das DICOM Info Window (Abbildung Abbildung 7.6) die Möglichkeit, die eingelesenen Daten zu analysieren. Während der Viewport2d die Bilddaten anzeigt, zeigt das DICOM Info Window alle Data Elemente (siehe Kapitel 3.1.3, S. 23) eines Schichtbildes nach Ihrer Kennung geordnet an. Neben den patientenbezogenen

Metadaten sind dort auch die technischen Data Elemente wie Bildtiefe oder Pixelformat einsehbar, daher ist das DICOM Info Window ein wichtiges Werkzeug sowohl für Anwender als auch für Entwickler.

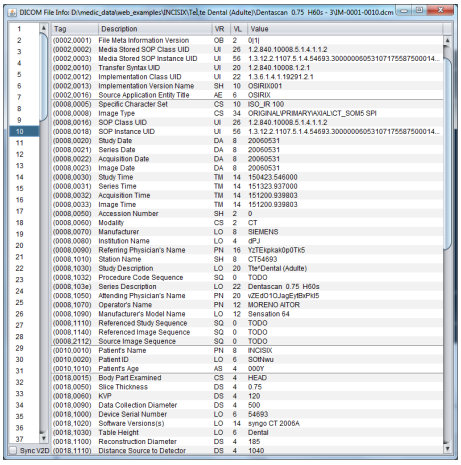


Abbildung 7.6.: DICOM Info Window

7.5. Visuelle Schnittstelle

Die visuelle Schnittstelle ist einer der fundamentalen Bestandteile von YaDiV. Sie soll es dem Anwender erlauben auf möglichst einfache und intuitive Art und Weise die Volumendaten zu analysieren und zu verstehen. Gleichzeitig ist sie die Voraussetzung für alle anderen Module. So lässt sich z.B. eine Region Grow Segmentierung nicht verwenden, ohne die Möglichkeit einen geeigneten Startpunkt zu setzen. Die Güte eines Registrierungsprozesses lässt sich nur dann intuitiv begreifen, wenn das Ergebnis angemessen visualisiert wird.

7.5.1. 2D Visualisierung

Obwohl der Schwerpunkt der Visualisierungskomponenten von YaDiV klar in der 3D-Darstellung liegt, ist eine hochqualitative 2D-Visualisierung unumgänglich. Zum einen liegen die Volumendaten in Schichtbildform vor, zum anderen erlauben die aktuellen Eingabegeräte wie die Computermaus eine präzise Ansteuerung einzelner Bildelemente nur in 2D. Vergleichbar ausgereifte Eingabemethoden für 3D-Daten befinden sich noch im Entwicklungsstadium sind (Kapitel 6.2, S. 108).

7.5.2. Viewport2d

Den ersten visuellen Zugang zu den vorliegenden Volumendaten stellt die Darstellung der Schichtbilder im View2d dar, der bereits während des Ladevorgangs die Anzeige der bisher eingelesenen Daten erlaubt. Neben der reinen Darstellung ist der Viewport2d zugleich auch Eingabeelement für verschiedene grundlegende Funktionen, wie der manuellen Segmentierung oder der Selektion einzelner Voxel.

Alle 2D-Visualisierungskomponenten unterstützen die Darstellung in transversaler, sagittaler und frontaler Ansicht. Eine flexibel gestaltete Rekonstruktionsfilter API erlaubt die Verwendung von verschiedenen Rekonstruktionsfiltern, bspw. beim Zoom auf einen bestimmten Bildbereich.

7.5.2.1. Layer und 2D-Renderer

Um auf zukünftige und spezialisierte 2D-Visualisierungen vorbereitet zu sein, wurde im Viewport2d ein Konzept implementiert, das es erlaubt mehrere Visualisierungsebenen übereinander zu legen. Die einzelnen Komponenten (sogenannte *Renderer2d* Instanzen) können dabei zur Laufzeit hinzugefügt, entfernt, und in ihrer Zeichenreihenfolge verändert werden.

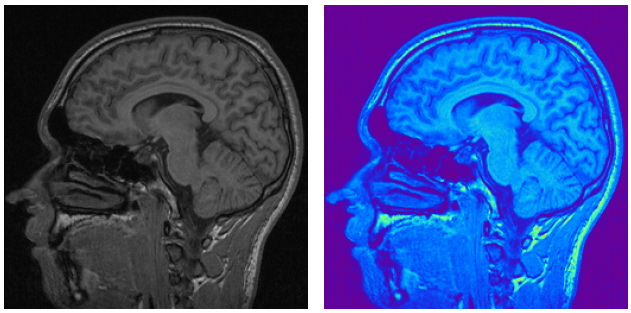
Von diesem Konzept machen auch die zur Grundfunktionalität gehörenden 2D-Visualisierungskomponenten Gebrauch, z.B. in der Darstellung der Schichtbilddaten oder die 2D-Segmentdarstellung. Beispiele für spezialisierte Renderfunktionalität sind u.a. die Visualisierung von Distanz und Geschwindigkeit der Active Contour Verfahren oder den aktuellen Versatz eines Atlas während des Registrierungsprozesses. Die Sichtbarkeit jeder Ebene lässt sich zur Laufzeit an- oder ausschalten.

Renderer2d Instanzen sind nicht notwendigerweise auf den Einsatz im Viewport2d beschränkt, bspw. benutzen auch die Volume Texturing Module die 2D-Hintergrund- und Segment-Renderer Funktionalität. Um unnötiges Neuzeichnen zu verhindern, wird intern ein mehrstufiges Nachrichtenkonzept verwendet. Wenn sich das Datenmodell einer *Renderer2d* Instanz ändert, löst diese eine `REQUEST_REDRAW` Nachricht aus, die von dem den Renderer enthaltenden Objekt empfangen und zu einem geeigneten Zeitpunkt bearbeitet werden muss.

7.5.2.2. 2D Schichtbilddarstellung

Die erste Grundfunktionalität des Viewport2d ist naturgegeben die Visualisierung einzelner Schichten aus dem Volumendatensatz. Das ImageSelector Bedienelement erlaubt die

Auswahl einzelner Bilder in allen drei orthogonalen Ansichten, wobei neben den unterschiedlichen Auflösungen auch die unterschiedlichen Abtastraten (pixel spacing vs. slice thickness) berücksichtigt werden müssen. Neben der Wahl der Projektionsebene bilden die Wahl des Farbmodells und das aktuelle VOI-Fenster im Intensitätswertebereich die Parameter für die 2D-Darstellung. Abbildung 7.7 zeigt ein Beispiel für die unterschiedlichen Farbmodelle anhand eines sagittalen MRT Schnittbildes.



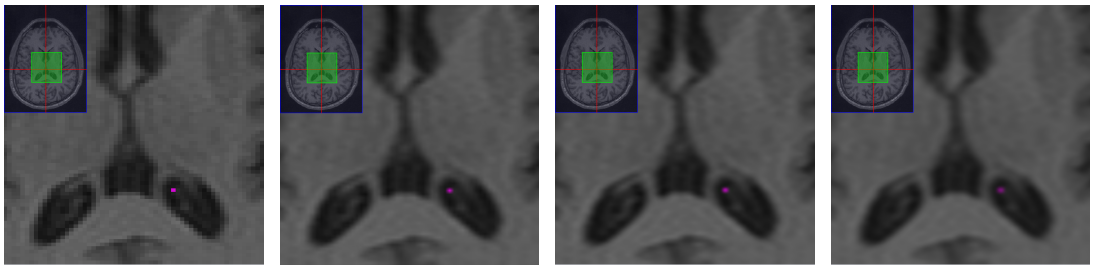
(a) Grauwertskala (b) Pseudofarben

Abbildung 7.7.: Grauwert- und Pseudofarbendarstellung im Viewport2d

Wenn die Schichtbildauflösung nicht der Anzeigeauflösung entspricht, kommt zusätzlich ein Rekonstruktionsfilter zum Einsatz, der vor allem für die Vergrößerung des Schichtbildes auf die reale Dimension auf dem Bildschirm notwendig ist. In der Java2D API sind bereits drei unterschiedliche Strategien zur Skalierung von Pixelbildern enthalten, die je nach Grafikkarte auch durch die Hardware unterstützt werden. Neben dieser Möglichkeit wurden für den Viewport2d zusätzlich ein eigener bilinearer, ein bikubischer und ein sinc-Filter implementiert.

Abbildung 7.8 zeigt die Auswirkung der verschiedenen Filter in der Ausschnittsvergrößerung eines transversalen Schnittbildes aus dem gleichen Datensatz wie in Abbildung 7.7. Zur leichteren Orientierung blendet der Viewport2d in diesem Modus den gewählten Ausschnitt in einer verkleinerten Schichtbildansicht ein (oben links in jedem Bild). Diese Vorschau verschwindet wenn der Ausschnitt nicht mehr verändert wird. Während der Unterschied zwischen der Nearest Neighbour Filterung und der bilinearen Methode noch deutlich zu erkennen sind, nehmen die sichtbaren Unterschiede zu den anderen Filtermethoden zunehmend ab – lediglich am selektierten Voxel im unteren Bildbereich rechts (eingefärbt in lila) lässt sich eine leichte Veränderung feststellen. In der Praxis wird daher meist ein bilinearer oder ein Nearest Neighbour Filter verwendet.

Unterstützt wird die Rekonstruktion durch ein selbst entwickeltes Verfahren, welches den speziellen Anwendungsfall der Schichtbilddatenvisualisierung ausnutzt. Bei dieser letzten Methode, im Folgenden 3-Ebenen Methode genannt, werden zusätzlich zur aktuellen Schicht auch die Bilder der darüber bzw. darunterliegenden Schicht verwendet und mit einer geringen Transparenz über das eigentliche Schichtbild gezeichnet. Dies erzeugt in



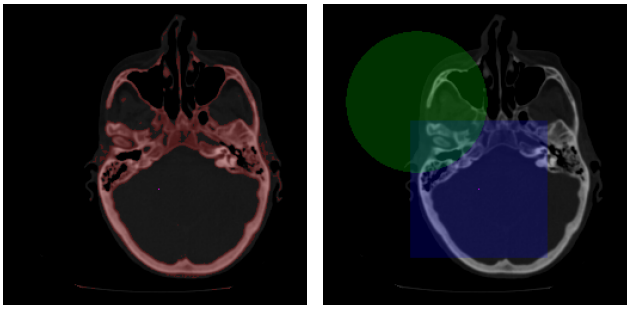
(a) Nearest Neighbour (b) Bilinear (c) Bikubisch (d) Sinc

Abbildung 7.8.: Viewport2d: Rekonstruktionsfilter für die Ausschnittvergrößerung

der Praxis einen zusätzlichen Glättungseffekt. Die 3-Ebenen Methode ist daher mit allen anderen Filterverfahren kombinierbar.

7.5.2.3. 2D-Segmentdarstellung

In der 2D-Segmentdarstellung werden die zu einem Segment gehörenden Pixel eines Schichtbildes optisch hervorgehoben, indem sie entsprechend der Segmentfarbe eingefärbt werden. Jedes Segment erhält ein zusätzliches Layer mit einer einstellbaren Transparenz, so dass die Volumendatenintensität weiterhin sichtbar bleibt. Durch die Transparenz ist es ferner möglich, mehrere sich teilweise überlappende Segmente in der 2D-Darstellung optisch auseinanderhalten zu können.



(a) Segmenthervorhebung (b) Segmentüberlappung

Abbildung 7.9.: 2D Segmentdarstellung

Abbildung 7.9 demonstriert dies an einem Beispiel. Das erste Bild (links) zeigt ein Min-Max Segment (rot eingefärbt), bei dem versucht wurde den Knochen zu selektieren. Gut zu erkennen ist, dass der Knochen rot eingefärbt wurde. Allerdings sind auch einige festere Knorpelteile sowie Teile der Matte, auf der der Patient gelegen hat, segmentiert worden (unterer Bildbereich). Das zweite Bild (rechts) demonstriert die Visualisierung von Segmentüberlappung. Für das bessere Verständnis wurden hier einfache geometrische Segmentformen gewählt: Segment A besteht aus einer Kugel (grün), Segment B aus einem

Würfel (blau). Der Überlappungsbereich erhält folglich eine Cyan Einfärbung. In allen Segmentbereichen sind die Intensitätswerte weiterhin gut sichtbar

7.5.3. Viewport3d

Der Viewport3d enthält u.a. die Java3D Zeichenfläche für viele andere Rendering Module, wie z.B. Texture2D- und Segmentrenderer.

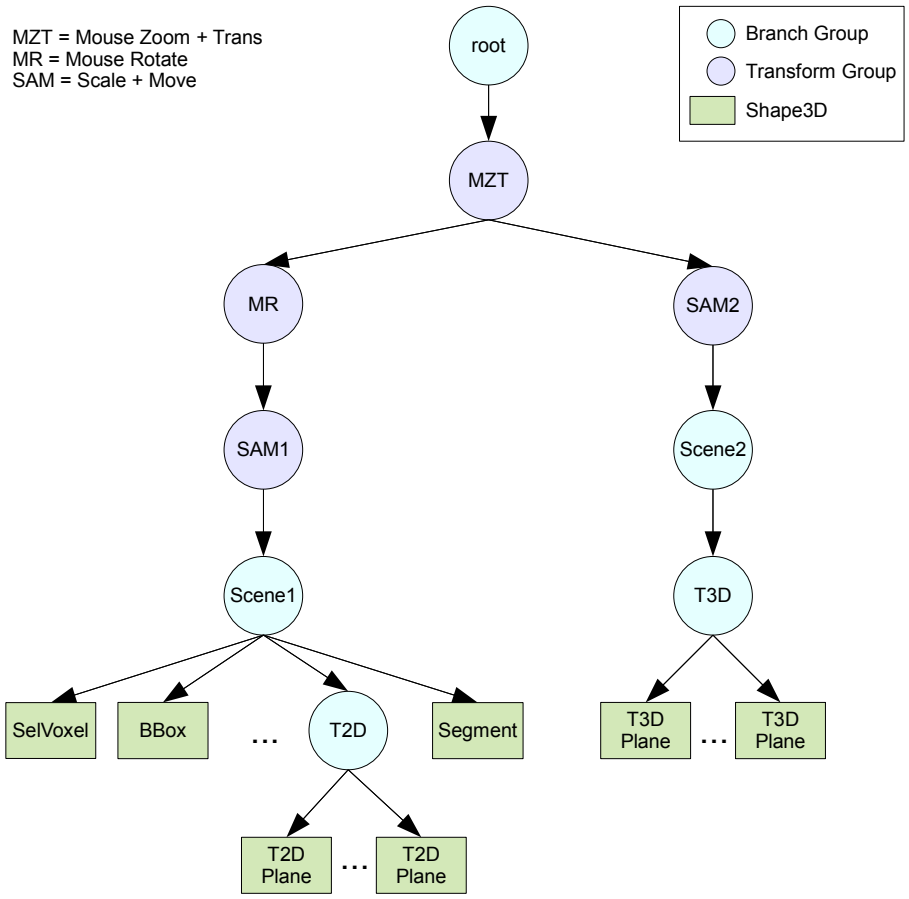


Abbildung 7.10.: Viewport3d Scenegraph

Abbildung 7.10 zeigt den Aufbau des Java3D Scene Graph in YaDiV. Direkt nach dem Wurzelknoten folgt eine Skalierung- und Translationsmatrix (MZT), welche die Verschiebung und die Skalierung (Zoom) der Benutzersicht enthält und sich damit auf alle Geometrien auswirkt. Unterhalb dieser Transformation teilt sich der Scenegraph in zwei Teile auf: der rechte Teil enthält die Geometrie Objekte für die Texture3D Darstellung, da diese im Gegensatz zu allen anderen Geometrien bei der Rotation des Augpunktes unbewegt bleiben, der linke Teilbaum enthält alle anderen Geometrien. Während im linken Teilbaum noch eine Matrix für die Rotation der Szene um Ihr Zentrum folgt, fehlt

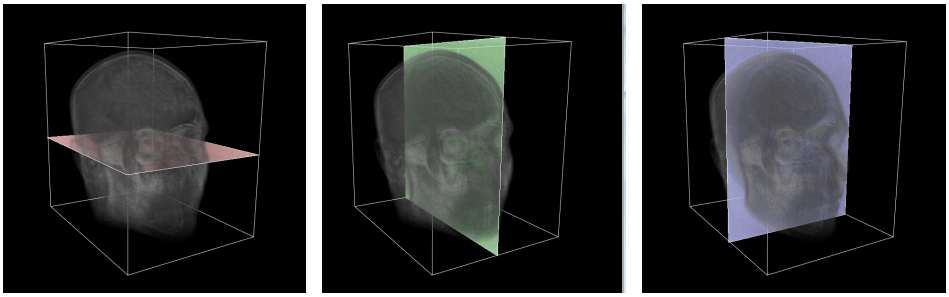
diese Matrix im rechten Teilbaum. Hier erfolgt die (inverse) Rotation direkt auf den 3D-Texturkoordinaten.

Eine wichtige Aufgabe übernehmen die Transformationsgruppen SAM1 und SAM2, die im jeweiligen Teilbaum eine Skalierung der Koordinaten auf den Sichtbereich und Verschiebung in das Zentrum der Zeichenfläche durchführen. Die Skalierung führt dabei gleichzeitig auch die notwendige Streckung bzw. Stauchung gemäß des je nach Volumendatensatz variierenden Voxel Spacing in x-, y- bzw. z-Richtung durch. Durch diese Anordnung im Scenegraph übernehmen die darunterliegenden Geometrien die reinen Positionskoordinaten des VoxelCube. Wenn z.B. ein Voxel an der Stelle (x, y, z) als Seed selektiert wird, kann der Seed-Renderer einfach ein Point3D Shape mit den gleichen Koordinaten erzeugen.

Die sichtbaren Objekte selbst sind Kindknoten von Scene1 oder Scene2. Letztere enthält ausschließlich die Projektionsgeometrien für die Texture3D Darstellung.

7.5.4. Darstellung einzelner Schichtbilder

Ein einfaches aber wichtiges Hilfsmittel für die Orientierung ist die Visualisierung einer einzelnen Schicht, bspw. derjenigen die gerade im View2D angezeigt wird. Dafür wird das entsprechende Bild als Textur auf ein ebenes Rechteck gelegt und mit einer einstellbaren Transparenz in den View3d integriert. Abbildung 7.11 zeigt ein Beispiel eines MRT Datensatzes mit Schichtbildern aus den drei orthogonalen Projektionsebenen.

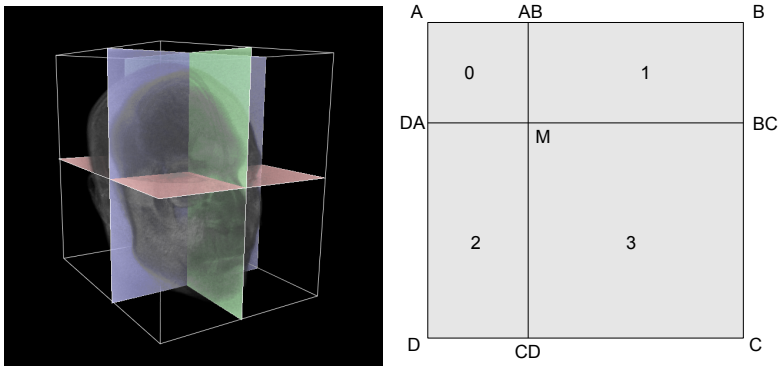


(a) Transversal (b) Sagittal (c) Frontal

Abbildung 7.11.: Texturdarstellung einzelner Schichtbilder

Eine häufig verwendete Variante der Schichtbildvisualisierung ist die sogenannte **Ortho Slice Darstellung**. Dabei werden je ein ausgewähltes transversales, sagittales und frontales Schichtbild gleichzeitig angezeigt. Die Ortho Slice Darstellung in YaDiV verwendet hierfür die zuletzt betrachteten Schichtbilder im View2d. Auch hier ist eine transparente Darstellung in vielen Anwendungen wünschenswert. Da transparente Objekte in absteigender Entfernung zum Betrachter gezeichnet werden müssen. ist für eine optimale Darstellung eine Zerlegung der Schnittebenengeometrie eines jeden Bildes entlang der

Schnittkante mit den anderen Schichtbildern erforderlich. Jedes Schichtbildobjekt wird aus vier einzelnen (Unter-)Objekten zusammengesetzt, skizziert in Abbildung 7.12.



(a) Visualisierung (b) Aufteilung der Geometrie

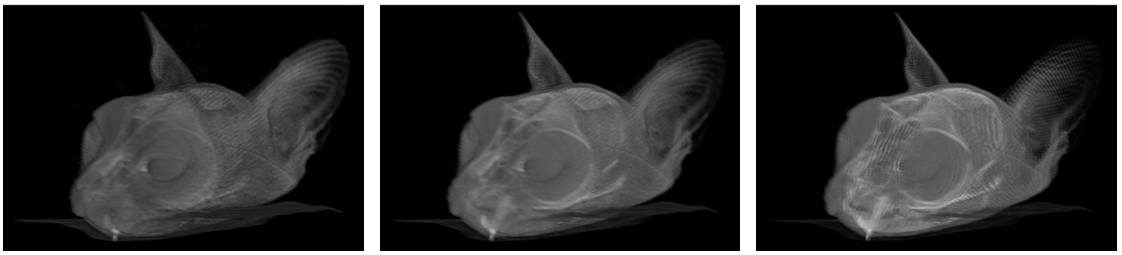
Abbildung 7.12.: Ortho Slice Darstellung

7.5.5. Texture2D Volume Rendering

Die Texture2D Volumendarstellung ermöglicht eine schnelle und auch auf älteren Rechnern lauffähige dreidimensionale Visualisierung der Volumendaten. In dieser Darstellung werden die transversalen, sagittalen und frontalen Schichtbilder als Texturen auf entsprechend orthogonal angeordnete Rechtecke in der 3D-Szene aufgetragen. Der Alpha (bzw. Transparenz-) Wert eines jeden Texturpixels wird dabei durch das aktuelle VOI-Fenster im Intensitätswerteraum bestimmt; liegt der Intensitätswert unterhalb der Fenstergrenze wird der Transparenzwert auf 0 (also durchsichtig), oberhalb auf 255 (opaque) gesetzt. Im Fensterbereich selbst wird die Transparenz linear interpoliert.

Da diese Darstellungsform sehr speicherintensiv ist, hat der Benutzer die Möglichkeit, über den Parameter des Ebenenabstandes die Anzahl der verwendeten Projektionsebenen zu reduzieren. Die bestmögliche Darstellung wird durch den Abstand 1 erreicht, d.h. jedes Schichtbild wird in der Darstellung verwendet. Ein Abstand von 2 halbiert die Anzahl der Schichtbilder, in dem nur jedes zweite Bild benutzt wird, etc. Neben der Schrittweite lassen sich auch das Farbmodell (Grauwerte, Falschfarben, Custom), die Grundfarbe der Projektionsebenen (falls keine Falschfarben-Darstellung gewählt wurde) und die globale Ebenen-Transparenz einstellen.

Abbildung 7.13 zeigt dies an einem Beispiel. Wie erwartet steigt der Speicherverbrauch linear mit der Anzahl der verwendeten Schichtebenen an. Bei der Framerate fällt ein Einbruch bei Abstand 1 auf, bedingt durch die Limitierung des dezidierten Grafikkartenspeichers (Testsystem: 512MB). Gut zu erkennen ist außerdem die fehlende (automatische) Alphawert-Korrektur, was dazu führt, dass im letzten Bild das Skelett durchscheint,



(a) Abstand 1 (443 MB / 10 FPS) (b) Abstand 2 (234 MB / 60 FPS) (c) Abstand 4 (109 MB / 118 FPS)

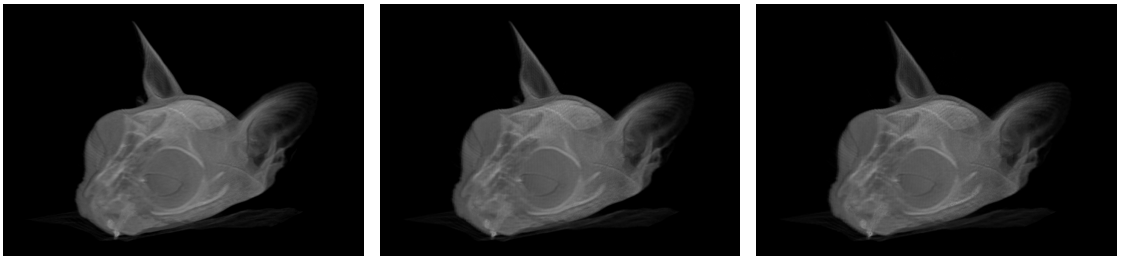
Abbildung 7.13.: Einfluss der Schrittweite in der Texture2D Volumen Darstellung[1]

während es im ersten Bild durch die größere Anzahl an Ebenen verdeckt wird. Die automatische Alphawert-Korrektur wurde nur in der Texture3D Volumen Rendering Methode integriert, die auch sonst viele Vorteile bietet.

7.5.6. Texture3D Volume Rendering

In der 3D-Texturdarstellung wird der gesamte Volumendatensatz als 3D-Textur interpretiert. Das hat zur Folge, dass sich die Projektionsebenen stets orthogonal zur Sichtrichtung des Betrachters anordnen lassen. Neben einer verbesserten Transparenzdarstellung und der Möglichkeit zur trilinearen Interpolation, hat dies insbesondere einen niedrigeren Speicherverbrauch in höheren Auflösungen zur Folge (siehe Kapitel 5.3.1.7).

Auch in der 3D-Texturdarstellung hat der Benutzer die Möglichkeit den Abstand zwischen den Projektionsebenen zu bestimmen. Dies spielt jedoch für den Speicherverbrauch eine untergeordnete Rolle, da der Volumendatensatz trotzdem komplett als 3D-Textur abgelegt werden muss. Dennoch ist eine Reduzierung der Projektionsebenen besonders auf älteren Rechnern hilfreich, um eine angenehme Framerate[1] zu erhalten.



(a) Abstand 1 (279 MB / 9 FPS) (b) Abstand 2 (276 MB / 18 FPS) (c) Abstand 4 (274 MB / 33 FPS)

Abbildung 7.14.: Einfluss der Schrittweite in der Texture3D Volumen Darstellung[1]

[1]Gemessen auf einer Notebookgrafikkarte (NVIDIA GeForce 9600M GT).

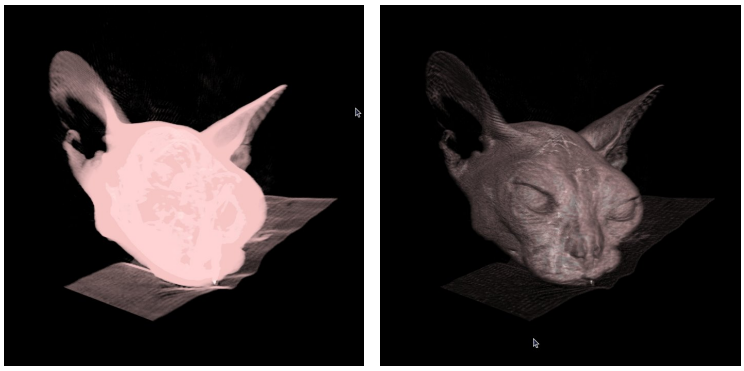
Auch dies lässt sich am besten anhand eines Beispiels erläutern. Abbildung 7.14 demonstriert die Auswirkung des Schrittweite-Parameters in der 3D-Texturdarstellung. Selbst bei der optisch höchsten Qualität bleibt der Speicherverbrauch nahezu konstant, dafür nimmt die Framerate erwartungsgemäß linear mit der Anzahl Projektionsebenen ab. Ebenfalls in dieser Abbildung zu sehen ist, dass der optische Transparenzeindruck gleich bleibt. Auch bei der Verwendung von weniger Schichten werden keine tieferlegenden Strukturen sichtbar.

In der Texture3D Darstellung sorgt eine automatische Alphawert Korrektur für einen gleichbleibenden Transparenzeindruck. In gewisser Weise lässt sich die Anzahl der Projektionsebenen als Abtastrate des Strahls in der Berechnung des Direct Volume Rendering Integrals auffassen. In [37] wird gezeigt, wie dafür die Alphawerte an die verwendete Abtastrate angepasst werden können. Dies kann einmalig vor der Erzeugung der Textur geschehen, indem die dafür verwendete VoxelColorTable modifiziert wird. Da dies nicht linear geschehen kann, wird die von Lichtenbelt et al. vorgeschlagene Formel zur Berechnung des korrigierten Alpha-Wertes

$$\alpha' = 1 - \sqrt[f]{1-\alpha} = (1-\alpha)^{\Delta s}$$

verwendet. Dabei entspricht f der Abtastrate und Δs dem Abtastabstand ($\frac{1}{f}$).

Ein Nachteil beider Verfahren ist die fehlende Schattierung , was zur Folge hat, dass die dritte Dimension erst durch die Bewegung intuitiv begreifbar wird. Um dies zu verbessern, wurde in YaDiV ein GPU-Fragment Shader integriert, der in Echtzeit die Berechnung der Normalen übernehmen kann. Auch an dieser Stelle wird dafür die Zentrale-Differenzen-Methode verwendet. Für die Berechnung der Schattierung werden die Normalen und die aktuell im Viewport3d gesetzten Lichtquellen (siehe Kapitel 7.5.8, S. 154) verwendet.



(a) Ohne Shading (b) Mit Shading

Abbildung 7.15.: Texture3D Volumen Darstellung mit und ohne Shading

Abbildung 7.15 zeigt die Auswirkung der Schattierung. Der Tiefeneindruck im rechten Bild ist bei nahezu gleichbleibenden Frameraten drastisch erhöht. Für die Implementation wurde die OpenGL Shading Language (GLSL) eingesetzt.

7.5.7. 3D Segmentdarstellung

Für die dreidimensionale Segmentdarstellung wurden unterschiedliche Formen der Visualisierung implementiert: die Darstellung als Punktwolke, die Segmentrand-Triangulation sowie die Segmentdarstellung im Texture2D bzw. -3D Volumen Rendering. Die Varianten sind nicht exklusiv, sondern lassen sich kombinieren.

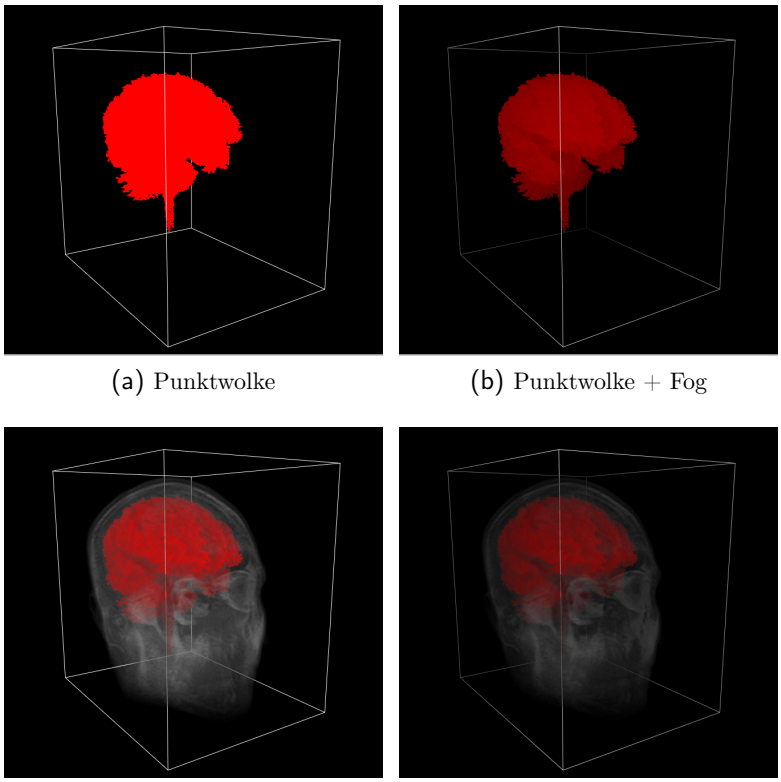
7.5.7.1. Darstellung als Punktwolke

Eine sehr schnelle Form der Visualisierung ist die Punktwolkendarstellung, bei der das Segment in (äquidistanten) Schritten abgetastet wird. Ist die Segmentzugehörigkeit für einen Voxel gegeben, wird in der 3D-Visualisierung ein Punkt mit den Voxelkoordinaten hinzugefügt. Das Ergebnis ist eine Punktwolke, die bei größtmöglicher Auflösung einen Punkt pro Segmentvoxel enthält[1].

Obwohl die Darstellung durch das fehlende Shading nur in der Bewegung als 3D-Struktur erkennbar ist, gibt es Anwendungen die von der Punktwolkendarstellung sinnvoll Gebrauch machen. In YaDiV wird sie z.B. für die interaktive Visualisierung während eines u.U. länger dauernden Segmentierungsprozesses eingesetzt, bei dem der Benutzer frühzeitig erkennen soll, ob das Ergebnis seinen Vorstellungen entspricht oder das Verfahren vorzeitig abgebrochen werden kann. Vor allem in Kombination mit dem 3D-Texture Renderer lässt sich so eine schnelle und interaktive Plausibilitätsprüfung durchführen.

Ein Beispiel für die Punktwolkendarstellung der grauen Gehirnsubstanz zeigt Abbildung 7.16. In seiner einfachsten Form erlaubt das Stillbild der Punktwolke nur sehr bedingt Rückschlüsse auf die dreidimensionale Form, da Bewegung und Tiefeneindruck fehlen. Eine einfache Möglichkeit einen Tiefeneindruck zu erzielen, ist das Hinzufügen eines Nebeleffektes in der Hintergrundfarbe, so dass die Punkte mit zunehmender Entfernung ausgeblendet werden. In Abbildung 7.16 b) sieht man, das der erzielte Effekt an eine schattierte Darstellung erinnert. Ein ähnlicher Effekt lässt sich erzielen wenn die Punktwolkendarstellung mit einer transparenten Texture3D Darstellung kombiniert wird. Die Kombination von Nebel und Texture3D Darstellung bietet hingegen keine optischen Vorteile, das resultierende Bild wirkt im wesentlichen etwas dunkler.

[1] Da viele aktuelle Grafikkarten auf die Darstellung von Polygonen und den Einsatz von Shadern optimiert werden, sind sehr große Punktwolken nicht immer die effizienteste Form der Visualisierung.



(a) Punktwolke (b) Punktwolke + Fog

(c) T3D (d) Punktwolke + Fog + T3d

Abbildung 7.16.: Segmentvisualisierung als Punktwolke

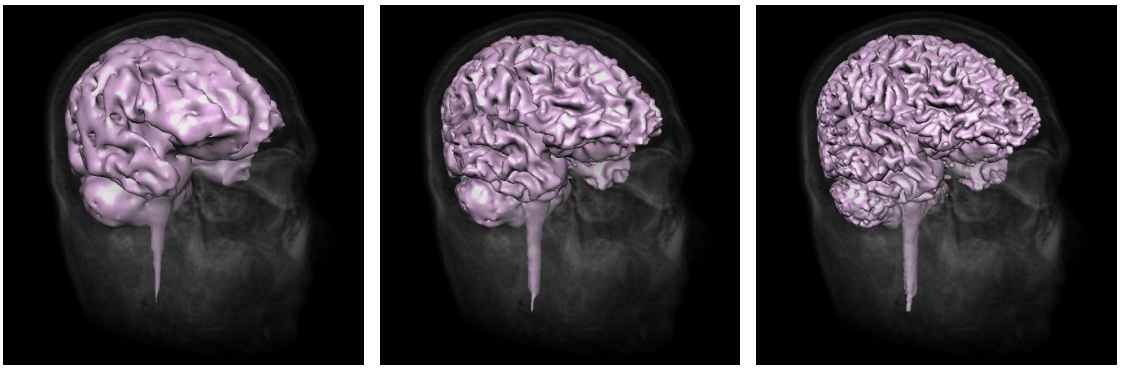
7.5.7.2. Marching Cube Darstellung

Eine optisch ansprechendere Form ein Segment darzustellen ist das Marching Cube Verfahren (siehe Kapitel 5.3.2.1). Dabei wird zuerst die Oberfläche des Segments polygonal als Dreiecksfläche approximiert und anschließend im Viewport3d angezeigt. Durch die Möglichkeit der schattierten Darstellung ist ein 3D-Eindruck auch bei unbewegten Bildern gegeben. Eine gegenüber dem Originalverfahren modifizierte Lookup-Tabelle garantiert in allen Fällen eine wasserdichte Randflächen-Triangulation.

Genau wie die Punktwolkendarstellung wird auch das MC Verfahren für die schnelle und interaktive Segmentvisualisierung, auch während der Segmenterzeugung, verwendet. Obwohl der Algorithmus sehr effizient arbeitet, kann es bei sehr komplexen Segmenten zu einer Verzögerung kommen, insbesondere wenn sehr viele Dreiecke erzeugt werden. Daher wurde eine automatische Verfeinerung (**refinement**) implementiert, bei der das Segment zuerst mit einer großen und sich dann in jedem Visualisierungsschritt verkleinernden Würfelgröße trianguliert wird.

Abbildung 7.17 zeigt dies an einem Beispiel. Bei Schrittweite 4 wird bereits nach 47ms

eine erste Approximation angezeigt. Verändert sich das Segment nicht weiter, wird automatisch die Schrittweite verringert. Wenn zwischendurch eine Segmentänderung eintritt, beginnt der Verfeinerungsvorgang von vorne, so dass der Benutzer immer ein schnelles Feedback der Veränderung erhält. Auch die höchste Auflösungsstufe (Schrittweite 1) wird trotz knapp 800.000 erzeugten Dreiecken inklusive Glättung in weniger als einer Sekunde erreicht. Tabelle 7.3 zeigt einige exemplarisch gewählte Zeitmessungen, die auf einem Laptop durchgeführt wurden. Je nach Größe des Volumendatensatzes schwankt dabei die Anzahl generierter Dreiecke zwischen 64.000 und 961.000 pro Sekunde. Selbst bei sehr großen Segmenten wie dem kompletten Skelett eines Menschen im Datensatz *Obelix_261* (siehe Anhang A) wird die schnelle Vorschau (Schrittweite 4) in weniger als einer Sekunde erreicht.



(a) SW4 (34356 Dreiecke/47ms) (b) SW2 (170296 Dreiecke/141ms) (c) SW1 (784532 Dreiecke/702ms)

Abbildung 7.17.: Automatische Verfeinerung beim MC Verfahren

Einen Sonderfall stellt der Rand der Volumendaten dar, da hier die Randfläche theoretisch nicht fortgesetzt werden kann. Da dies aber praktisch zu einer eher unschönen Randansicht führt, wird in einem Post-Processing Schritt der Rand abgeschlossen, bei dem alle Marching-Cube-Eckpunkte außerhalb des aufgenommenen Volumens als nicht zum Segment gehörend behandelt werden. Um diese künstlich konstruierten Randflächen optisch vom echten Segmentrand im Inneren unterscheiden zu können, werden diese durch eine etwas hellere Färbung gekennzeichnet.

Um den Speicherverbrauch zu reduzieren, wird eine indizierte Dreiecksfläche verwendet. Für die Überprüfung, ob ein Eckpunkt bereits als Index vorhanden ist, wird eine Hashing-Methode verwendet, bei der der Dreieckseckpunkt als Schlüssel und der Index (falls vorhanden) als Wert gespeichert wird. Da die erzeugten Randflächen jedoch bei komplexen Segmenten sehr groß werden können, ist die Verwendung einer einzigen Hash-Tabelle sowohl speicherintensiv als auch ineffizient. Dies lässt sich durch einen einfachen Trick beheben: Da das Marching Cube Verfahren den Volumendatensatz schichtweise durchläuft, können neu einzufügende Eckpunkte nur in der aktuellen oder der vorangegangenen Schicht bereits vorhanden sein, so dass statt einer sehr großen, zwei vergleichsweise kleine

Data Set	Segment	SW	#Δ	ms	#Δ/ms
CT_Head 256 x 256 x 113	bone	4	26568	135	197
		2	125616	203	618
		1	543192	765	710
	brain	4	13288	36	369
		2	78384	240	327
		1	474152	578	821
	jaw	4	984	15	64
		2	6300	10	610
		1	30924	49	635
VIX 512 x 512 x 250	bone	4	112192	322	348
		2	511588	1012	506
		1	2148580	3183	675
	flesh	4	89680	93	961
		2	372896	394	946
		1	1517008	2111	719
Obelix_261 512 x 512 x 520	body	4	240932	652	370
		2	1017592	2449	416
		1	4202888	8308	506
	skeleton	4	186112	926	201
		2	902504	2015	448
		1	4009216	6818	588

Testsystem: Notebook mit Intel Core 2 Duo (2.4GHz), 4GB RAM, NVidia 9400m

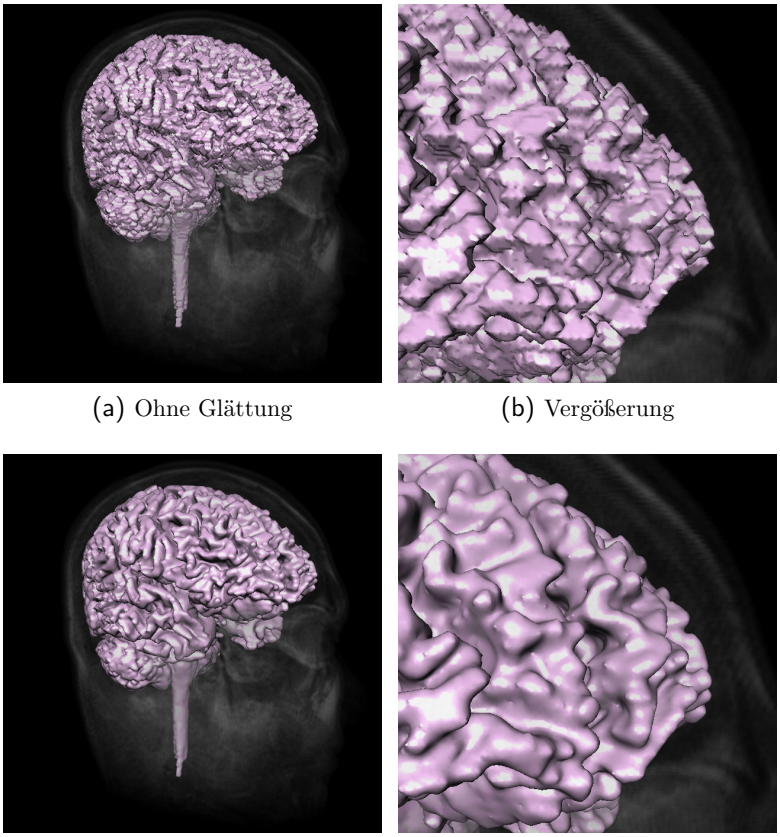
Tabelle 7.3.: YaDiV Marching Cube Performance

Hash-Tabellen verwendet werden können.

Auch die Berechnung des Case-Index für die Lookup-Tabelle lässt sich unter Ausnutzung der bekannten Durchlaufrichtung des MC Verfahrens optimieren. Da innerhalb einer Schicht zwei direkt aufeinanderfolgende Würfel immer eine gemeinsame Seite – also vier gemeinsame Eckpunkte – besitzen, lassen sich bei geschickter Indizierung vier Bit des zuvor berechneten Index durch eine Shift Operation auf den nächsten Fall übertragen, so dass die Hälfte der vergleichsweise aufwendigen Segmentzugehörigkeitsabfragen eingespart werden kann. Eine weitere Beschleunigung lässt sich erzielen, wenn vor der Maching Cube Phase die Eckpunkt-Koordinaten in der Lookup-Tabelle einmalig mit der aktuellen Schrittweite skaliert werden, so dass beim späteren Durchlauf nur die Translation mit der Würfelposition durchgeführt werden muss.

Beim Einsatz des MC Verfahrens für die Segmentvisualisierung können die Intensitätswerte nicht zum Glätten verwendet werden, da das Segment selbst nur die Werte 0 (gehört nicht zum Segment) und 1 (gehört zum Segment) abspeichert. Deshalb ist eine nachträg-

liche Glättung notwendig, für die ebenfalls nicht die Intensitätswerte aus den original Volumendaten verwendet werden sollen. Dahinter steckt die Philosophie, dass sich ein Segment lokal stark von den gegebenen Volumenstrukturen im Bilddatensatz entfernen darf, bspw. wenn ein Mediziner eine ihm bekannte aber durch eine zu niedrige Auflösung im Datensatz fehlende Struktur ergänzt oder durch die Aufnahmetechnik entstandene Artefakte mittels medizinischem Fachwissen erkennt und entfernt. Eine auf den Intensitätswerten beruhende Glättung würde in solchen Fällen die Darstellung verfälschen.



(a) Ohne Glättung (b) Vergößerung

(c) Mit Glättung (d) Vergößerung

Abbildung 7.18.: MC Segmentfläche mit und ohne Glättung

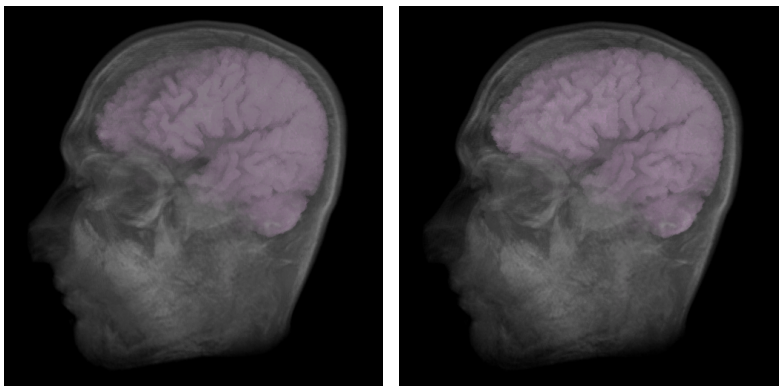
Als Glättungsmechanismus wurde daher eine eigene Methode verwendet, die eine rein geometrische Glättung der MC Dreiecksfläche durchführt. Vereinfacht dargestellt werden dabei für jeden Eckpunkt v die Mittelpunkte der angrenzenden Dreiecke betrachtet, deren geometrisches Mittel berechnet und v um einen konfigurierbar kleinen Anteil in dessen Richtung bewegt. Führt man diese Operation mehrfach hintereinander aus, stellt sich ein Glättungseffekt ein. Als praktischer Wert hat sich hier eine dreifache Glättung herausgestellt.

Etwas komplizierter wird diese Situation jedoch durch die unterschiedliche Abtastrate

des bildgebenden Verfahrens in x-, y- bzw. z- Richtung. Um zu vermeiden, dass die Verstärkung der Stufen bei einer niedrigeren Auflösung in einer Koordinatenebene den Glättungseffekt verfälscht, muss die Eckpunktverschiebung für alle drei Koordinaten einzeln und entsprechend des Voxel Spacing pro Koordinate öfter durchgeführt werden. Abbildung 7.18 demonstriert den Glättungseffekt für das Segment aus Abbildung 7.17. Die Stärke der Glättung entspricht in diesem Bild 3, sie lässt sich zur Laufzeit einstellen oder auch ganz ausschalten.

7.5.7.3. Segmente im Textur Volumen Rendering

Eine weitere Möglichkeit Segmente in die 3D-Ansicht zu integrieren ist die farbliche Hervorhebung der entsprechenden Voxel im 2D bzw. 3D Textur Rendering Modul. Um diese Visualisierung nutzen zu können, muss folglich die entsprechende Volumendarstellung aktiviert sein. Sie verursacht dann keinen zusätzlichen Speicherverbrauch und nur minimale Laufzeitkosten. Abbildung 7.19 zeigt ein Segment in einem MRT Datensatz in Texture2D und Texture3D Darstellung.



(a) Texture2D (b) Texture3D

Abbildung 7.19.: Segmentvisualisierung im Texture Renderer

Diese Form der Segmentdarstellung wird vor allem im Vorschau Modus des Volume Raycasting Verfahrens (siehe Kapitel 7.5.10) verwendet, um einen ersten Eindruck der späteren hochqualitativen Darstellung zu vermitteln.

7.5.8. Beleuchtung

Eines der wesentlichen Hilfsmittel zur intuitiven Erkennung einer dreidimensionalen Szene ist ein Beleuchtungsmodell, das zumindest eine einfache schattierte Darstellung unterstützt. Fast alle im Vorfeld getesteten Visualisierungs-Tools gehen von einer statischen

Lichtquelle[1] aus, so dass sich viele Rendering Techniken direkt darauf optimieren lassen. Da in YaDiV jedoch auch unterschiedliche Beleuchtungsmodelle erprobt und auf ihre Eignung für das Szenenverständnis oder andere Aspekte (z.B. den ästhetischen Eindruck) untersucht werden sollen, unterstützt der Viewport3d bis zu 8 Lichtquellen, welche über eine komfortable graphische Benutzerschnittstelle konfiguriert werden können.

Insgesamt gibt es vier verschiedene Arten von Lichtquellen:

- **Ambient Light** - erzeugt ein ambientes Hintergrundlicht ohne Position und Richtung.
- **Point Light** - ist eine punktförmige Lichtquelle an einer festen Raumposition, die Licht in alle Richtungen aussendet.
- **Spot Light** - besitzt ebenfalls eine festen Raumposition und zusätzlich eine Richtung und einen Austrittswinkel, so dass ein kegelförmiger Lichtwurf entsteht. Außerdem lässt sich die Abnahme der Lichtintensität zum Kegelrand über einen Parameter steuern.
- **Directional Light** - definiert eine Lichtquelle mit Richtung aber ohne Position. Das Licht kommt an jeder Position im Raum aus der gleichen Richtung.

Die Lichtkonfiguration beeinflusst dabei nicht nur die Darstellung im Viewport3d, sondern kann auch in das High-Quality-Raycasting Modul übernommen werden, so dass auch hier der Viewport3d eine schnelle Vorschaufunktion übernehmen kann.

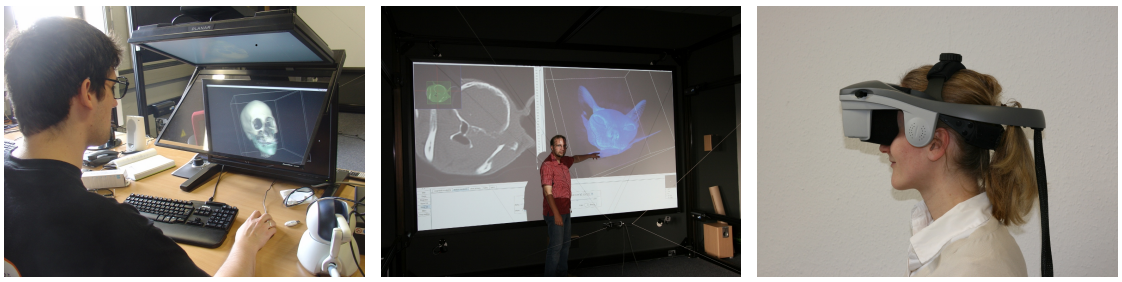
7.5.9. Stereo Visualisierung

Eine für viele Anwender völlig neue Darstellungsform für einen intuitiven räumlichen Eindruck der Volumendaten und Segmente ist die stereographische Visualisierung, die automatisch verwendet wird, wenn YaDiV eine entsprechende Hardwareunterstützung vorfindet. Im Gegensatz zur klassischen Darstellung, wo für die Anzeige auf einem normalen 2D-Monitor ein einziges Bild berechnet wird, muss in der stereographischen Ansicht jedes Bild aus zwei Betrachterpositionen (eine für das linke und eine für das rechte Auge) dargestellt werden. Das Ergebnis wirkt in etwa so, als wäre der normale Monitor ein Fenster hinter dem ein ein reales, dreidimensionales Objekt im Raum schwebt. Insbesondere komplex strukturierte Segmente mit einer Vielzahl an kleinen Details lassen sich so erheblich einfacher und besser interpretieren.

Notwendig dafür ist eine entsprechende Grafikkarte sowie ein stereographisches Display, bspw. ein 3D-Monitor oder ein Stereoprojektionssystem. Üblicherweise sind in diesem

[1] Üblicherweise eine punktförmige Lichtquelle im Auge oder ein gleichmäßig gerichtetes Licht von schräg oben.

Fall auch spezielle Brillen für den Betrachter notwendig. Eine Ausnahme bildet z.B. ein Head Mounted Display (HMD), das aufgrund seiner Konstruktion die Displays direkt vor den Augen des Betrachters positioniert. Abbildung 7.20 zeigt eine Auswahl an stereographischen Displays des Lehrstuhls für Graphische Datenverarbeitung, mit denen das Programm getestet wurde.



(a) Planar Monitor (b) Stereoprojektion (c) HMD

Abbildung 7.20.: YaDiV Stereovisualisierung)

Mechanismen zur Stereo Visualisierung (siehe auch Kapitel 5.3.4, S. 105) werden von Java3D direkt unterstützt und wenn vorhanden automatisch vom Viewport3d angefordert. Dazu muss zunächst überprüft werden, ob das aktuell verwendete Canvas3d Objekt die Stereodarstellung unterstützt. Der *View Branch* des Scenegraph (siehe Abbildung 7.1) enthält das *Physical Body* Modell, das u.a. Anzahl und Position der Augen des Betrachters enthält, die zur Laufzeit angepasst werden können. In der normalen Fensterdarstellung ohne Stereo Effekt wird die CYCLOPEAN_EYE_VIEW Einstellung verwendet, d.h. ein einziges Betrachterauge in der Mitte des Kopfes. Im Stereo Modus werden stattdessen zwei Augpunkte im *Physical Body* positioniert, wobei darauf geachtet werden muss, dass das linke Auge auch tatsächlich links vom rechten gesetzt wird. Da dies auch von der Verkabelung zwischen Grafikkarte(n) und Display(s) abhängt, lassen sich die Augpunktpositionen zur Laufzeit vertauschen und auch der Augenabstand ist vom Anwender konfigurierbar.

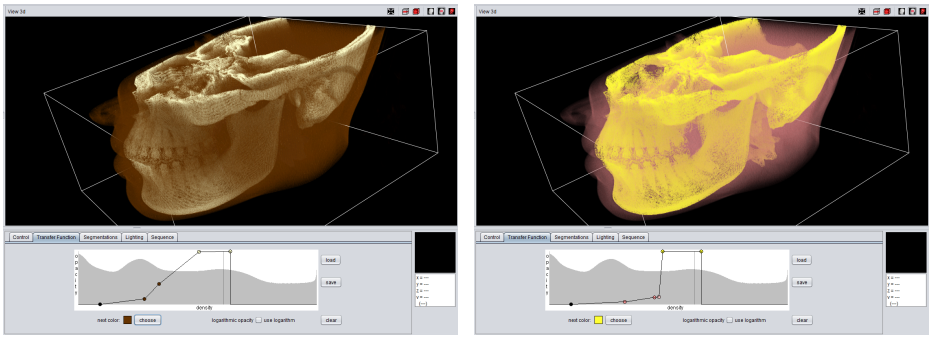
Wenn die Grafikkarte keine native Stereo Visualisierung (Quadbuffer-Stereo) unterstützt, aber über einen zweiten Monitorausgang verfügt, lässt sich die stereographische Darstellung dadurch erzielen, dass ein weiteres Canvas3D Objekt auf dem zweiten Monitor plaziert wird. Dies wird jedoch nur im Fullscreen Modus unterstützt, da anderenfalls auch die GUI Elemente auf beiden Bildschirmen angezeigt werden müssten. Beide Canvas3D Instanzen referenzieren dabei die selbe *VirtualUniverse* Instanz und benutzen die LEFT_EYE_VIEW- bzw. RIGHT_EYE_VIEW *ViewPolicy* von Java3D.

7.5.10. Volume Raycaster

Die schnelle und auf den ersten Eindruck beeindruckend realistische Texture3D Volumendarstellung zeigt insbesondere in der Vergrößerung unangenehme Artefakte. Dies liegt sowohl an der festen Abtastrate, als auch an der auf Ganzzahlen beschränkten Berechnung der Farb- und Transparenzwerte.

Ein wesentlich besseres Bild lässt sich mit einem Raycasting Verfahren erzielen, wie es u.a. in der Arbeit von Maximilian Müller [52] implementiert wurde. Das dort entwickelte Ray Casting Modul integriert sich nahtlos in die YadiV Plattform und nutzt dabei die bereits vorhandenen Datenstrukturen und GUI Elemente aus. Neben verschiedenen Optimierungsstrategien ist das Ergebnis über eine Vielzahl von Parametern, wie der Abtastrate, der Transferfunktion oder der Gewichtung einzelner Segmente, kontrollierbar. Um deren Einfluss auf das endgültige Bild, dessen Berechnung je nach Volumengröße von einigen Sekunden bis zu Minuten dauern kann, schon vorher abschätzen zu können, wird die Texture3D Volumendarstellung für eine schnelle Vorschau eingesetzt.

Den größten Einfluss auf die Darstellung hat naturgemäß die Wahl der Transferfunktion, die sich über einen graphischen Editor einstellen lässt (Abbildung 7.21). Dabei wird eine stückweise lineare Funktion verwendet, deren Stützpunkte die gewählte Farbe für den zugehörigen Intensitätswert (x-Achse) anzeigen. Die y-Achse gibt die Transparenz (0 = durchsichtig) an. Der Editor erlaubt sowohl das Ändern oder Löschen bestehender als auch das Hinzufügen neuer Stützpunkte. Die Auswirkung auf die Texture3D Volumendarstellung ist interaktiv und selbst bei sehr großen Datensätzen flüssig.



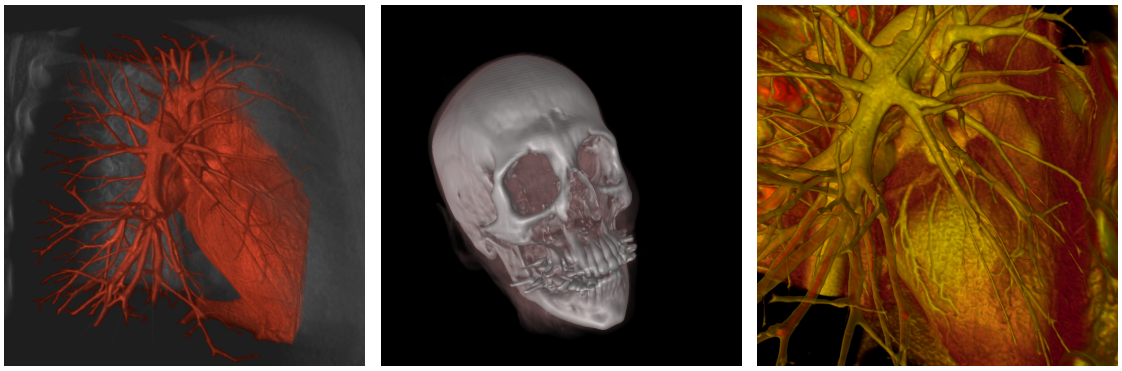
(a) (b)

Abbildung 7.21.: Graphischer Editor für die Transferfunktion

Neben der Bildauflösung lässt sich auch die Anzahl Threads für die Bildberechnung einstellen, da diese je nach System und Auslastung variieren kann. Da sich das Ray Casting Verfahren gut parallelisieren lässt, ist eine lineare Beschleunigung zur Anzahl der Threads feststellbar, falls diese nicht die Zahl der Prozessorkerne übersteigt.

Zur Vermeidung von Artefakten lässt sich die Abtastrate stufenlos einstellen. Eine weitere Möglichkeit ist die Wahl eines anderen Rekonstruktionsfilters. Dabei wird der Filter nicht nur zur Interpolation der Farb- und Alphawerte genutzt, sondern auch auf die Gradienten bzw. Normalen angewendet. Experimente haben jedoch gezeigt, dass der höhere Rechenaufwand aufwendiger Filterkerne nur bedingt Einfluss auf die Bildqualität besitzt ([52], Kapitel 5.1, Seite 46).

Ebenfalls konfigurieren lässt sich der Einfluss der Segmente in der Darstellung, so dass sich Segmente gezielt hervorheben oder verdecken lassen. Ein Anwendungsbeispiel wäre das segmentierte Skelett eines menschlichen Körpers, das mit niedriger Transparenz in einem umgebenden fast durchsichtigen Körper sichtbar sein soll. Um dies zu erreichen lässt sich sowohl die globale Transparenz einstellen, als auch ein zusätzlicher Alpha Wert für jedes einzelne Segment.



(a) (b) (c)

Abbildung 7.22.: Ergebnisse des (normalen) Ray Casting Moduls *Quelle: [52]*

Die durch das Ray Casting Modul erzeugten Bilder sind im direkten Vergleich zu den Textur Renderer Verfahren von deutlich besserer Qualität, Abbildung 7.22 demonstriert dies eindrucksvoll anhand von drei Beispielen. Die Bildberechnung ist selbst in der Grundkonfiguration hinreichend schnell, um hochwertige Bilder aus den teilweise sehr komplexen Datensätzen zu generieren. Nach Abschluss der Masterarbeit wurde das Ray Casting Modul weiter entwickelt und insbesondere der Memory Footprint reduziert. Außerdem lassen sich inzwischen im Viewport3d Bewegungen aufzeichnen und die so entstehende Bildsequenz mit einstellbarer Framerate mit dem Ray Casting Modul rendern, so dass auch Animationen in hoher Qualität und Auflösung erstellt werden können.

Bei der Konzeption dieses Moduls wurde versucht, einen Mittelweg zwischen leichter Erweiterbarkeit und akzeptabler Performance zu finden. So lassen sich bspw. die unterschiedlichen Verfahren zur Normalenberechnung über ein Strategie-Pattern austauschen. Optimierungen wie das Empty Space Skipping oder die Early Ray Termination sind unabhängig von der konkreten Farbberechnung bei der Strahlenverfolgung zuschaltbar.

(a) (b) (c)

Abbildung 7.23.: Ray Casting mit Lit Sphere Maps

Um die Erweiterungsfähigkeit des Ray Casting Moduls zu testen, wurde nachträglich eine Unterstützung für Lit Sphere Maps (basierend auf [76] bzw. [7], siehe auch Kapitel 5.3.3) integriert. Bei dieser Methode lassen sich unterschiedliche kreisförmige Texturen als Schattierungseffekt verwenden, womit sich sehr unterschiedliche Bildeindrücke erzielen lassen, wie Abbildung 7.23 anhand eines Beispiels eindrucksvoll demonstriert. In den ersten beiden Bildern wurden jeweils die gleichen Texturen (mit unterschiedlicher Transparenz) eingesetzt. Im dritten Bild wurde eine glasartige Textur für Hounsfield-Werte im Bereich des Knochens angegeben. Weitere Effekte wie ein illustrativer- oder handgezeichneter Zeichenstil lassen sich ebenfalls durch einfaches Austauschen der Texturen umsetzen.

Dazu wurde für die Benutzerschnittstelle ein zusätzlicher graphischer Transferfunktionen-Editor in das Modul integriert, der es erlaubt die Texturen (wieder in Verbindung mit einem Alpha-Wert) an die einzelnen Intensitätswerte zu binden. Auch eine zweite Erweiterung für die Kontur-Verstärkung ließ sich ohne Probleme umsetzen, ohne die bereits bestehenden Klassen des Ray Casting Moduls zu verändern. Die implementierten Optimierungsstrategiegen konnten auch hier genutzt werden.

7.6. Segmentierung

Nachdem in den vorangegangenen Kapiteln die Visualisierung von Segmenten beschrieben wurde, soll es im Folgenden um deren Erzeugung gehen. YaDiV unterstützt unterschiedliche Segmentierungsverfahren, die sich über eine gemeinsame Schnittstelle in der Toolbar in das Programm integrieren. Die Einsatzgebiete, Ergebnisse und Laufzeiten der einzelnen Algorithmen sind dabei sehr unterschiedlich.

7.6.1. Segmenterstellung und Boolsche Operationen

Da in den meisten praktischen Anwendungsfällen ein einzelnes Segmentierungsverfahren nicht ausreicht, wurde ein Mechanismus implementiert, über den sich unterschiedliche Algorithmen kombinieren lassen. Jedes Segmentierungsmodul an sich erzeugt zunächst kein eigenes Segment, sondern eine sogenannte Selektion – auch als temporäres Segment bezeichnet. Diese Selektion kann zu einem echten Segment hinzugefügt (boolsches ODER), davon entfernt (boolsches NOT) oder als neues Segment übernommen werden. Der Prozess ist in Abbildung 7.24 schematisch dargestellt. Da es auch möglich ist ein Segment in die Selektion zu kopieren (und umgekehrt), lassen sich über diesen Mechanismus auch die Segmente selbst miteinander verknüpfen.

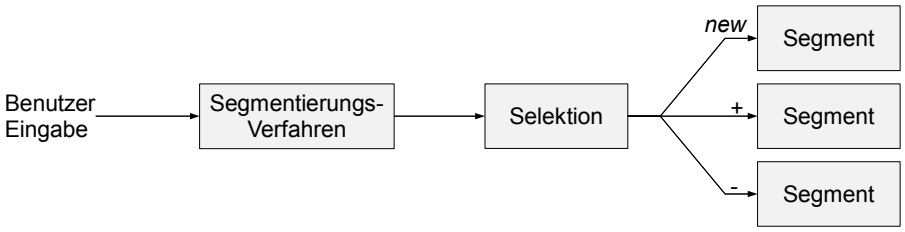


Abbildung 7.24.: Prozess der Segmenterstellung

Alle Operationen profitieren von der im Segment verwendeten BitCube Datenstruktur (siehe Kapitel 7.3.2), die eine sehr hardwarenahe und effiziente Umsetzung der boolschen Operationen ermöglicht. Eine von vielen Anwendern gewünschte Funktion ist es, die Segmentierung auf einen vorgegebenen Bereich beschränken zu können. Alle Segmentierungsverfahren lassen sich daher auf ein einfach einzustellendes Begrenzungsvolumen beschränken.

7.6.2. Min-Max Modul

Bei diesem Segmentierungsverfahren wird eine obere bzw. untere Schranke zur Definition eines zum Segment gehörenden Intensitätsintervalls verwendet, die sich über ein Slider Element einstellen lassen. Da dieser Algorithmus sehr schnell arbeitet ist die Auswirkung auf die Selektion interaktiv, so dass jede Schrankenveränderung sofort sichtbar wird.

7.6.3. Region Grow Modul

Für die Region Grow Segmentierung müssen zuvor ein oder mehrere Startpunkt(e) gewählt werden, die im Viewport2d selektiert werden können. Zusätzlicher Eingabeparameter ist die Varianz, die die erlaubte Abweichung der Intensität der Nachbarvoxel gegenüber der des Startvoxels (als seed bezeichnet) festlegt. Werden mehre Startvoxel verwendet, gilt

hier die Durchschnittsintensität. Dabei kann der Benutzer wählen, ob sich die Varianz auf die reale Intensität oder den auf das VOI-Fenster skalierten Wert bezieht. Zusätzlich lässt sich eine Menge von Segmenten angeben, die die Ausbreitung des Region Grow blockieren (sogenannte *blocking segments*). Während der Ausbreitung wird die Selektionsdarstellung ständig aktualisiert, so dass der Verlauf des Prozesses als Animation sichtbar ist. Der Benutzer kann den Prozess jederzeit abbrechen, z.B. wenn er sieht, dass sich das Verfahren in einen unerwünschten Volumenbereich ausbreitet.

In der Praxis führt eine einzige Region Grow Segmentierung nur sehr selten direkt zum Ziel, meist lässt das gewünschte Gebiet jedoch aus mehreren einzelne Segmentierungen mit geringerer Varianz kombinieren.

7.6.4. Snake Modul

Das in Kapitel 4.4.2 beschriebene aktive Konturverfahren wird in YaDiV mit dem in der englischen Literatur häufig verwendeten Begriff *Snake* Modul bezeichnet. Über die Benutzerschnittstelle lassen sich der Einfluss der Konturkrümmung, die Ausbreitungsgeschwindigkeit und die *stopping*-Funktion auswählen. Als Startkontur können ein oder mehrere Kreiskonturen mit einstellbarem Radius oder die Kontur eines bereits vorhandenen Segments verwendet werden. Das Verfahren kann sowohl im 2D- als auch im 3D-Modus mit einer vom Benutzer gewählten Anzahl Iterationen ausgeführt werden.

Weitere, technische Einstellungen können über den SettingsBrowser eingestellt werden, wie z.B. die Breite des Narrow Bands oder die Anzahl Iterationsschritte bis zur Reinitialisierung der Distanzfunktion. Diese Einstellungen sind eher für die Entwickler gedacht und blieben daher dem normalen Anwender verborgen.

7.6.5. Energy Snake Modul

Diese Philosophie wurde auch in der Benutzerschnittstelle des Energy-Snake Moduls verfolgt, das die in Kapitel 4.4.3 vorgestellte Energy Minimization Methode verwendet. Auch hier werden dem Benutzer die wichtigsten Parameter wie die Gewichtung der inneren bzw. äußeren Energie, des Volumens oder der Oberfläche zugänglich gemacht. Weiterhin kann zwischen unterschiedlichen HeaviSide Funktionen gewählt werden. Diese wurden als abstraktes Interface implementiert und können jederzeit erweitert werden. Zusätzlich kann der Benutzer wählen, ob der Algorithmus die Intensitäts- oder die durch das VOI-Fenster skalierten Werte verwenden soll.

Auch hier kann wie im Snake Verfahren eine Menge an Kreisscheiben oder ein Segment als Initialisierung gewählt werden. Die manuell gewählten Startpunkte lassen sich dafür komfortabel im Viewport2d als Seeds setzen. Dafür wird das gleiche Eingabekonzept wie

bei der Wahl der Seeds für den Region Grow Algorithmus oder zur Zentrierung des VOI-Fensters auf einen gewählten Voxelwert verwendet.

7.6.6. Atlasbasierte Segmentierung

Die Benutzerschnittstelle für die atlasbasierte Segmentierung ist naturgemäß etwas umfangreicher und ist in mehrere Karteikarten (engl.: tabs) unterteilt. Neben einer Auswahlmöglichkeit für den zu verwendenden Atlas und einer optionalen Starttransformation gibt es auch Möglichkeiten zur Visualisierung der Atlas-Transformation und des Verbund-Histogramms (siehe auch S. 70).

Die für die affine und elastische Teil der Registrierung notwendigen Parameter lassen sich über zwei getrennte Tabs einstellen. Für die affine Registrierung handelt es sich um die minimalen bzw. maximalen Rotationswinkel, Skalierungsfaktoren bzw. Translationswerte (jeweils in x-, $y-$ und $z-$Richtung bzw. beim Rotationswinkel die jeweilige Achse). Zusätzlich lässt sich die Anzahl der verwendeten Voxel für die Berechnung der *mutual information* angeben. Um das Verfahren zu Beschleunigen kann der Benutzer außerdem die Anzahl der parallel verwendeten Threads vorgeben.

Der Dialog für die elastische Registrierung fällt vergleichsweise einfach aus und beschränkt sich im Wesentlichen auf die Dimensionsangabe des initialen Kontrollpunktgitters und die Anzahl an Verfeinerungsstufen. Zusätzlich kann ein Faktor für die Kontrollpunktverschiebung (in Gradientenrichtung) gewählt werden, der eine stärkere oder schwächere Verformung pro Iterationsschritt erlaubt.

7.6.7. Manuelle Segmentierung

Auch die fortschrittlichsten Segmentierungsverfahren führen nicht immer direkt zum Ziel und der medizinische Anwender möchte bzw. muss lokale Korrekturen vornehmen. Im Fall von besonders komplexen Traumata führt kein automatisiertes Verfahren, sondern ausschließlich händisches Segmentieren zum Ziel. Hierbei spielen die medizinische Fachkenntnis sowie die Intuition des Arztes eine große Rolle. Daher wurde auch eine Möglichkeit zur manuellen Segmentierung implementiert.

Der Anwender kann dabei wie bei einem traditionellen Zeichenprogramm unterschiedliche Pinsel wählen und das Segment schichtweise einzeichnen bzw. Teile davon löschen. Auch hier wird wie in allen anderen Segmentierungsmethoden immer zuerst in die Selektion gezeichnet, die dann mit bestehenden Segmenten kombiniert werden kann. Auf diese Weise ist es auch möglich Korrekturen an einem bereits bestehenden Segment vorzunehmen. Ein wichtiges und oft verwendetes Hilfsmittel ist zudem eine einstellbare Varianz, die die unter dem Pinsel liegenden Voxel nur dann selektiert, wenn deren Intensitätswerte in

der durch die Varianz definierten Abweichung zur Intensität des Mittelpunkts des ersten Pinselkontaktes liegen. Das Ergebnis ist eine wesentlich komfortablere Handhabung, da geringfügige Abweichungen in der Mausbewegung abgefangen werden.

7.7. Haptische Schnittstelle

Die vollständige Integration der haptischen Schnittstelle ist noch nicht abgeschlossen, dennoch gibt es bereits jetzt die Möglichkeit, mit den Volumendaten haptisch zu interagieren. Dabei kann der Benutzer sowohl zwischen unterschiedlichen haptischen Renderingmethoden (Abbildung 7.25, siehe auch Kapitel 6.4.2 bzw. 6.4.3) wählen als auch Materialparameter, wie die Steifigkeit der Objekte, konfigurieren.

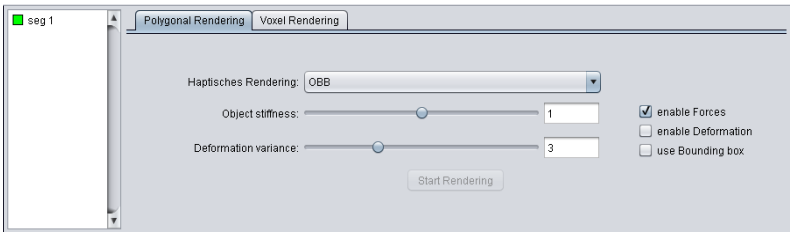
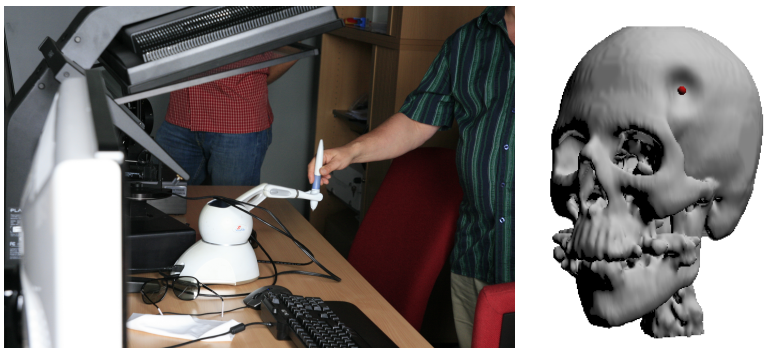


Abbildung 7.25.: GUI der haptischen Schnittstelle

Über das haptische Eingabegerät können Segmente erfühlt, bewegt und wahlweise auch elastisch deformiert werden, (siehe Abbildung 7.26). Zusätzlich lässt sich festlegen, ob das im Datensatz aufgenommen Volumen (bzw. dessen Bounding Box) eine natürliche Begrenzung für die Objektmanipulation darstellt. Falls die haptisch unterstützte elastische Deformation gewählt wurde, kann zur Laufzeit auch der Deformationsradius frei gewählt werden.



(a) (b)

Abbildung 7.26.: Haptik für die Reponierung und lokale Deformation

Die Schnittstelle wurde so konzipiert, dass neue Modelle und Eingabegeräte leicht hinzugefügt werden können. Derzeit implementiert sind die Unterstützung für das Novint

Falcon sowie das Sensable Phantom Omni und ein weiteres Sensable 6DOF Phantom. Ein Anbindung des Inca 6D der Firma Haption ist zur Zeit in Arbeit.

8. Rückblick und Zusammenfassung

Mit der Entwicklung von YaDiV wurde eine freie und offene Plattform geschaffen. Dabei spiegeln die meisten Module in YaDiV (bspw. die Visualisierung oder die Segmentierung) den Stand der aktuellen internationalen Forschung wider. Ziel war es, die Entwicklung neuer Techniken im Bereich Visualisierung und Segmentierung zu unterstützen. Durch die Unabhängigkeit von Hardware und Betriebssystem ist das Programm sehr flexibel einsetzbar, was insbesondere interdisziplinären Projekten mit mehreren Forschungspartnern zugute kommt. Die Unterstützung aktueller VR-Umgebungen mit stereographischer Visualisierung und haptischen Eingabegeräten, erlaubt die Erprobung neuer Wege in der Mensch-Maschine-Schnittstelle.

Die Unabhängigkeit von Hardware und Betriebssystem wurde vor allem durch die Wahl der Programmiersprache Java erreicht. Dabei wurde zu Beginn der Entwicklung in Fachkreisen diskutiert, ob diese Entscheidung nicht zu große Nachteile bzgl. der Performance mit sich bringen würde. Im weiteren Verlauf hat sich jedoch gezeigt, dass die Geschwindigkeit von YaDiV mit vielen freien und kommerziellen Programmen, sowohl in der Segmentierung als auch in der Visualisierung, vergleichbar ist. Oft wirkt YaDiV hier sogar etwas schneller[1]. Dies wurde vor allem durch eine effiziente Programmierung und die konsequente Multi-Thread-Unterstützung erreicht. Durch die animierte Darstellung aller Prozesse während deren Ausführung (z.B. Segmentierungsmethoden oder Veränderung der Transferfunktion), ist eine interaktive und dadurch intuitive Datenexploration und -manipulation möglich.

Während der Entwicklung wurde die Plattform mehrmals grundlegend neu konzipiert und den Bedürfnissen der neuen Module angepasst. Dabei wurde immer versucht, die Modulbestandteile, die auch für andere Anwendungen von Interesse sind, in die eigentliche Haupt-API zu übernehmen, zum Teil in etwas abstrakterer, dafür allgemeingültiger Form. Es hat sich gezeigt, dass viele, zum Teil sehr unterschiedliche Module, immer wieder die gleichen Werkzeuge einsetzen. Beispiele hierfür sind Interpolationstechniken (für Sub-Voxelauflösung aber auch Gradienten- oder Farbmittlung), die Bestimmung von Gradienten (durch zentrale Differenzen oder von Neumann Methode) oder der Einsatz von

[1] Da die Funktionen der Programme sich immer etwas unterschieden, ist ein objektiver Vergleich nur schwer durchführbar.

Distanzfeldern (Aktive Konturverfahren, Haptik). Hier erwies es sich als sinnvoll und notwendig, eine Bibliothek mit durchdachten, sowohl Speicher- als auch Laufzeit-effizienten Klassen und Methoden als integralen Bestandteil der Plattform zu definieren.

Dass diese Strategie erfolgreich war, ließ sich u.a. an der Frequenz der Re-Design-Zyklen und des Erfolgs der einzelnen Teilprojekte feststellen. Während der ersten Projektarbeiten musste auch die Plattform selbst ständig angepasst und erweitert werden, im späteren Verlauf wurde dies immer seltener notwendig. Ein gutes Beispiel hierfür war die Entwicklung des Ray Casting-Moduls, bei der die eigentliche YaDiV-API nur minimal angepasst wurde. Da das Modul auf einer breiten Werkzeugpalette aufbauen konnte, war bereits nach kurzer Zeit eine lauffähige Version mit sehr eindrucksvollen Resultaten entstanden. Nach Abschluss des Ray Casting Moduls wurde zu Testzwecken versucht, das bestehende Verfahren durch illustrative Rendering-Techniken (Kapitel 5.3.3) zu ergänzen, was nach nur wenigen Stunden problemlos möglich war. Dies ist nur ein Beispiel, dass die einfache Erweiterbarkeit der Plattform demonstriert. Das Ziel, die Entwicklung neuer Techniken im Bereich Visualisierung und Segmentierung zu unterstützen und durch eine breite Palette an Werkzeugen zu beschleunigen, wurde somit erreicht.

Das Programm bietet inzwischen eine breite Palette an Modulen zur Segmentierung (MinMax, Region Grow, Snake, Energy Snake, atlasbasiert, manuell) und Visualisierung (Direktes Volumen-Rendering mit Texture2D, Texture3D, Ray Casting, Indirektes Rendering über modifizierte MC und Dual MC Technik) der Volumendaten und bietet sich damit gerade auch durch die Plattformunabhängigkeit als Basis für interdisziplinäre Projekte an. Alle API Bestandteile wurden möglichst sauber und erweiterbar gehalten und unter Verwendung von JAVADOC dokumentiert. Tabelle 8.1 zeigt einige Statistiken des aktuellen Stands von YaDiV (ohne Haptik und atlasbasierte Segmentierung, die sich noch jeweils in dem für die Entwicklung vorgesehenen Branch befinden).

Number of packages	15
Number of classes	330
Total Lines of Code	34162

Tabelle 8.1.: YaDiV Code Statistik

Insbesondere die derzeitigen medizinischen Anwender von YaDiV haben das Programm während der Entwicklung getestet und wertvolles Feedback gegeben. Ihrer Mithilfe gilt daher mein ganz besonderer Dank.

9. Ausblick

Auch wenn mit der aktuellen Version bereits ein stabiler Entwicklungsstand erreicht wurde, gibt es noch vieles, was im Anschluss an diese Arbeit geleistet werden kann.

Plugin Konzept

Eine größere strukturelle Verbesserung könnte die Einführung eines modernen Plugin-Konzeptes sein, in dem ähnlich wie in Applikationen wie Firefox oder der Entwicklungsumgebung eclipse neue Komponenten zur Laufzeit hinzugefügt und auch entfernt werden können. Ein derartiges Konzept hätte zwei große Vorteile:

1. Ein stabiles Plugin-Konzept beugt der Gefahr vor, dass durch eine verteilte Entwicklung, wie sie in Open Source Community Projekten üblich ist, Methoden und Klassen von einzelnen Gruppen, die an einem speziellen Modul arbeiten, in das Hauptprogramm einfließt und so die bestehende Struktur über die Zeit aufweicht.

2. Die Entwicklung der Plattform selbst würde sich auf eine überschaubare Schnittstelle mit einer umfangreichen, allgemein nützlichen Grundfunktionalität beschränken, während speziellere Module (bspw. eine Anwendung, die in erster Linie für Tierärzte interessant ist) unabhängig vom Hauptprogramm entwickelt, gepflegt und evtl. sogar vermarktet werden können.

Beides käme der längerfristigen Produktpflege sehr zugute und würde die Wartbarkeit der Plattform weiter verbessern. Ein solches Konzept ist jedoch nicht trivial und setzt sehr erfahrene Programmierer voraus. So müssen Abhängigkeiten beachtet werden, da Teile der Funktionalität von Modul A das Vorhandensein von Modul B voraussetzen, ohne das Programmierer und Nutzer durch eine komplizierte Schnittstelle überfordert werden. Denkbar wäre hier evtl. eine Kooperation mit Medizinern für die Formulierung der Bedürfnisse und Software-Ingenieuren für die Umsetzung durch aktuelle Design-Konzepte. Auch die Verwendung von bereits entwickelten Plugin-Lösungen wie die eclipse Plugin-API oder das Netbeans Konzept sollte eingehend geprüft werden.

Haptik und Stereovisualisierung als Benutzerschnittstelle

Wie bereits in Kapitel 6.4.5 angedeutet, besteht noch viel Forschungsbedarf über den Einsatz von Haptik als neues Element in der intuitiven Mensch-Maschine-Kommunikation.

Gespräche mit medizinischen Partnern aus der Kiefer-, Mund- und Gesichtschirurgie der MHH haben gezeigt, dass ein großer Bedarf daran besteht, durch neue, intuitive Eingabekonzepte bestehende Aufgaben aus der klinischen Praxis zu vereinfachen und gänzlich neue zu ermöglichen. In der Zukunft werden Konzepte aus der sogenannten Virtuellen Realität viel stärker in diese Bereiche Einzug erhalten. Gerade weil dort mit 3D Volumendaten gearbeitet wird, ist die Beschränkung auf 2D Eingabe- (Maus) und Ausgabe-(Monitor) Geräte zur Exploration und Manipulation der gewonnen Daten unbefriedigend und letztlich auch ungeeignet.

Eine multimodale haptische Plattform könnte z.B. eine intuitive Vorregistrierung eines unsegmentierten Volumendatensatzes mit einem vollständig segmentierten Modell ermöglichen. Der Benutzer könnte die Daten haptisch unterstützt drehen, skalieren, „übereinanderschieben“ und bei Bedarf lokal verformen (Abbildung 9.1).

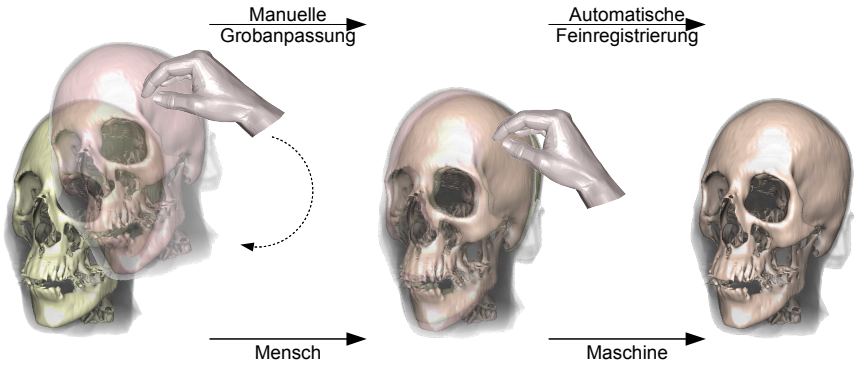


Abbildung 9.1.: Mögliche Anwendung von Haptik in halbautomatischer Registrierung/Segmentierung

Diese manuelle Vorregistrierung kann dann als Eingabe für einen automatischen atlasbasierten Matching-Algorithmus oder als Startsegment für ein Aktiv-Kontur-Verfahren dienen. Damit würden die Vorteile beider Welten kombiniert; mit einer menschennahen Schnittstelle fällt es dem Anwender leicht, eine schnelle Grobanpassung vorzunehmen – die voxelgenaue Feinanpassung bleibt den Algorithmen überlassen, deren sonstige Schwäche, gegen ein lokales Extremum zu konvergieren, so zu einem weiteren Vorteil wird. Die dafür notwendigen Bausteine (atlasbasierte Segmentierung, konturbasierte Segmentierung, stereographische Visualisierung und eine haptische Schnittstelle) sind bereits in YaDiV implementiert.

GPU oder Multiprozessor?

Haptik, anspruchsvolle Segmentierungsverfahren und hochqualitative Visualisierung stellen hohe Anforderungen an die Effizienz der eingesetzten Algorithmen. Versuche haben gezeigt, dass durch eine alternative Umsetzung in C/C++ zwar eine Verbesserung der Performance (Faktor 3–5) möglich ist, aber auch hier schnell die Grenzen der derzeitigen Systeme erreicht werden.

Ein vielversprechender Ansatz ist daher die konsequente Weiterentwicklung des bereits verwendeten Multi-Thread Konzepts, dass plattformunabhängig eine gute Ressourcennutzung garantiert. Dennoch bleiben zeitkritische Aufgaben, die von einer hardwarenäheren Umsetzung auf Multikern-CPU oder den Kernen einer modernen GPU profitieren würden. Viele in der letzten Zeit erschienenen Arbeiten setzen deshalb GPU Optimierung ein, die jedoch, trotz Entwicklungen wie Cuda, sowohl vergleichsweise umständlich zu programmieren als auch relativ eng an eine bestimmte Hardware gebunden sind. Während auf der einen Seite die Grafikprozessoren immer allgemeiner programmierbar werden, gibt es gleichzeitig die Entwicklung, dass aktuelle Hauptprozessoren aus immer mehr Rechenkernen bestehen und so ebenfalls parallelisierbare Aufgaben übernehmen können.

Eine mögliche Lösung deutet sich mit der Entwicklung von OpenCL (Open Computing Language) an. OpenCL wird als offener Standard von der Khronos Group entwickelt, die mehrere etablierte Standards wie OpenGL oder das Austauschformat Collada pflegt. Das Ziel von OpenCL ist es, eine Programmierplattform für CPU, GPU und DSP zu schaffen, die es dem Entwickler erlaubt, unabhängig von der tatsächlich vorhandenen oder später verwendeten Hardware massiv-parallele Algorithmen zu entwerfen. In OpenCL entwickelte Programme (so genannte Kernel) können zur Laufzeit auf unterschiedliche vorhandene OpenCL-fähige Geräte verteilt werden, so dass beim Anwender vorhandene CPU und GPU Hardware simultan zur Beschleunigung der Verfahren genutzt werden kann.

OpenCL könnte sich somit als „Königsweg“ erweisen, der es möglich macht, die Unabhängigkeit von Hardware und Betriebssystem weiter zu verfolgen und gleichzeitig extrem performante Lösungen anbieten zu können. Derzeit existieren verschiedene Open Source Projekte, die eine Java Anbindung von OpenCL zum Ziel haben, als Beispiele seien hier JOCL, libCLcalc oder der OpenCL-Zweig des Native Libraries For Java Projekts genannt.

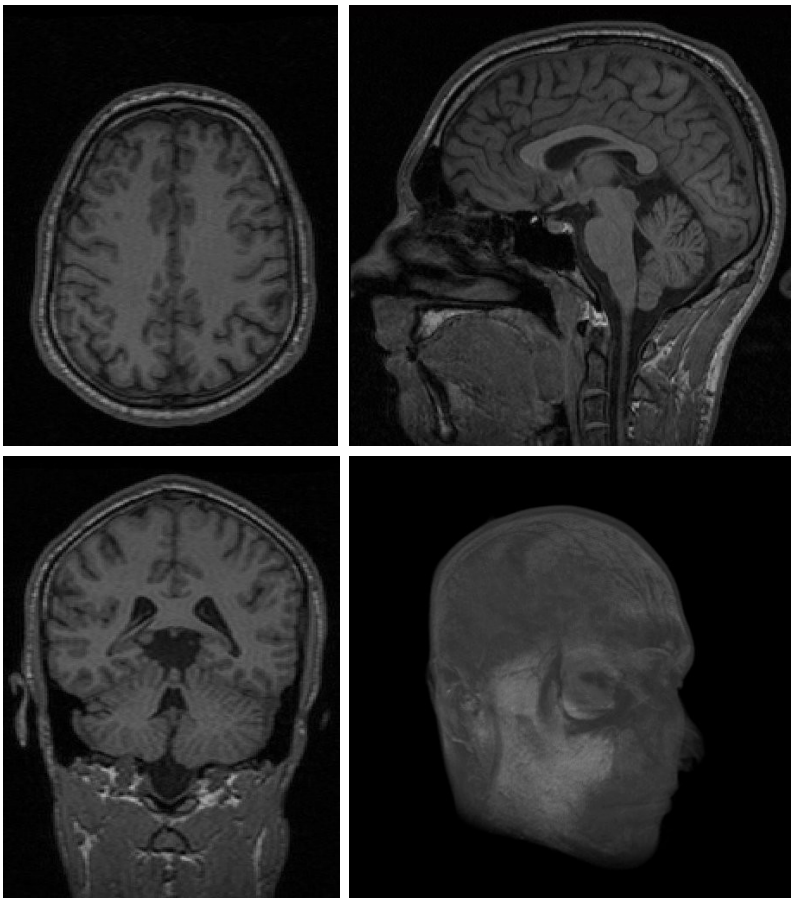
A. Datensätze

Die in dieser Arbeit gezeigten Bilder, die auch zur Messung von Frameraten oder Rendering-Zeiten verwendet wurden, beruhen auf freien oder anonymisierten Volumendatensätzen, von denen im Folgenden einige exemplarisch vorgestellt werden.

Obelix_261 (CT)
Ganzkörper CT Scan

512 x 512 x 520
Bits Stored: 2^{12}
Unsigned Little Endian

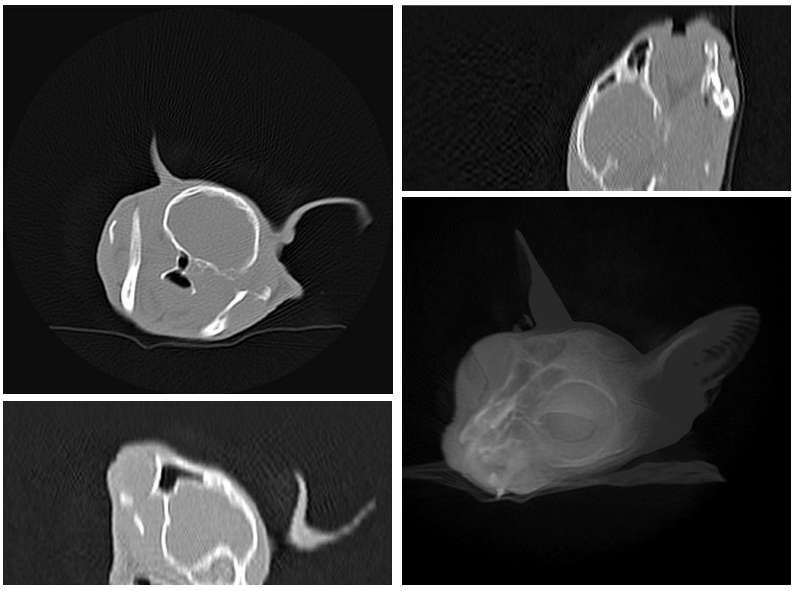
Quelle: Internet



MRT_Head (MRT)
Kopf

160 x 256 x 256
Bits Stored: 2^{12}
Unsigned Little Endian

Quelle: MHH



Cat (CT)
Kopf einer Katze

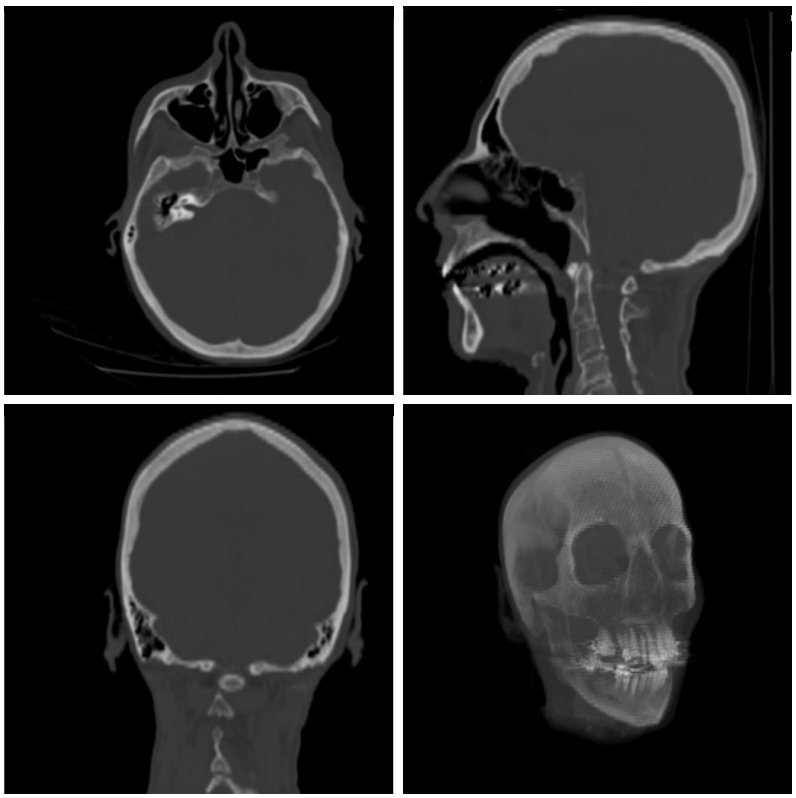
512 x 512 x 142
Bits Stored: 2^{16}
Signed Little Endian

Quelle: Tierklinik

VIX (CT)
Füße

512 x 512 x 250
Bits Stored: 2^{12}
Signed Little Endian

Quelle: Internet



CT_Head (CT)
Kopf

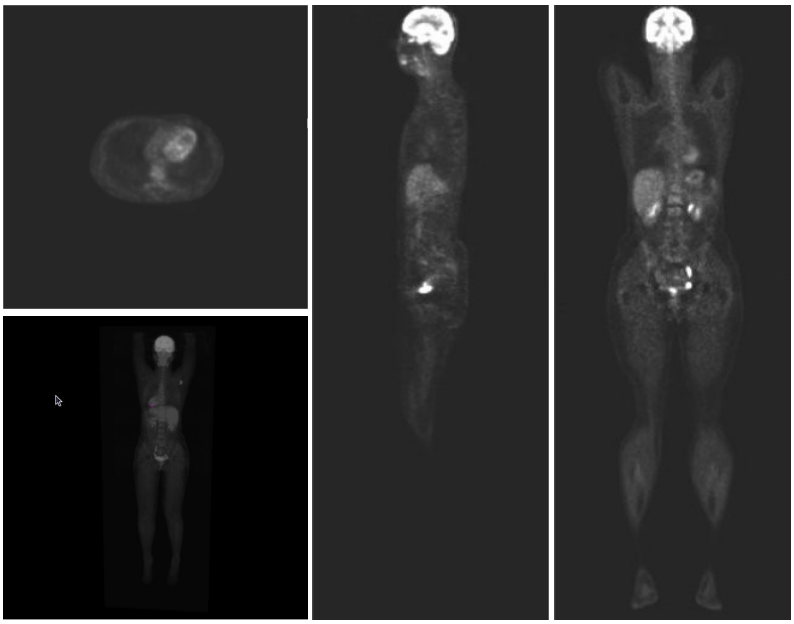
256 x 256 x 113
Bits Stored: 2^{12}
Unsigned Little Endian

Quelle: Internet

Melanix_102 (PET)
Ganzkörper PET Scan

168 x 168 x 715
Bits Stored: 2^{16}
Signed Little Endian

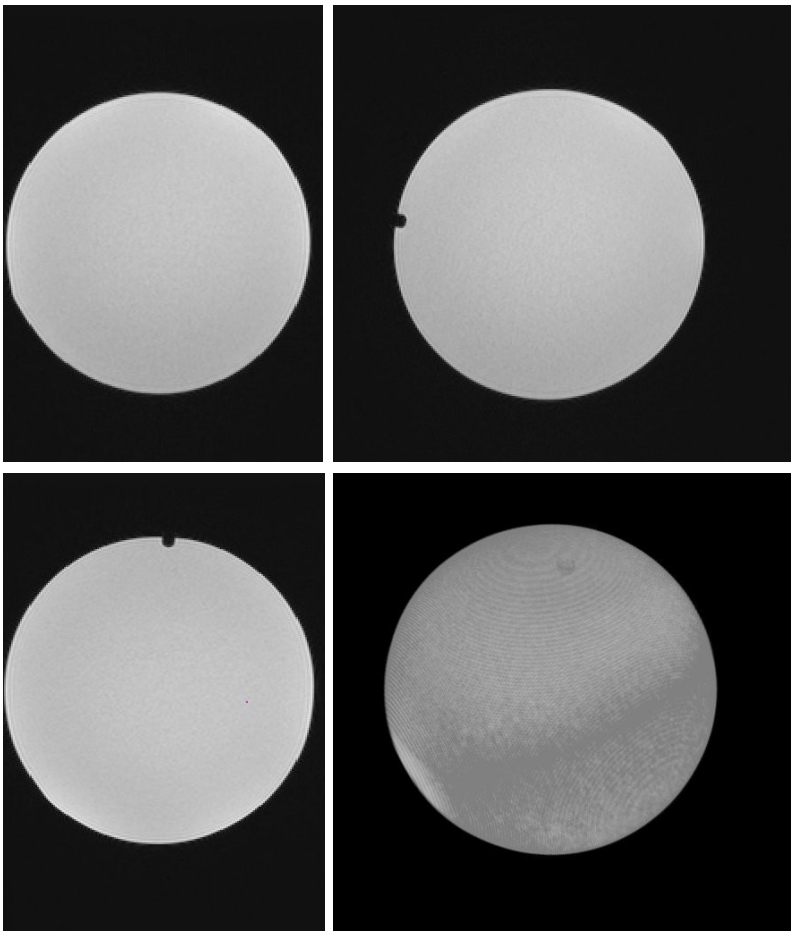
Quelle: Internet



Phantom (MRT)
Messkörper

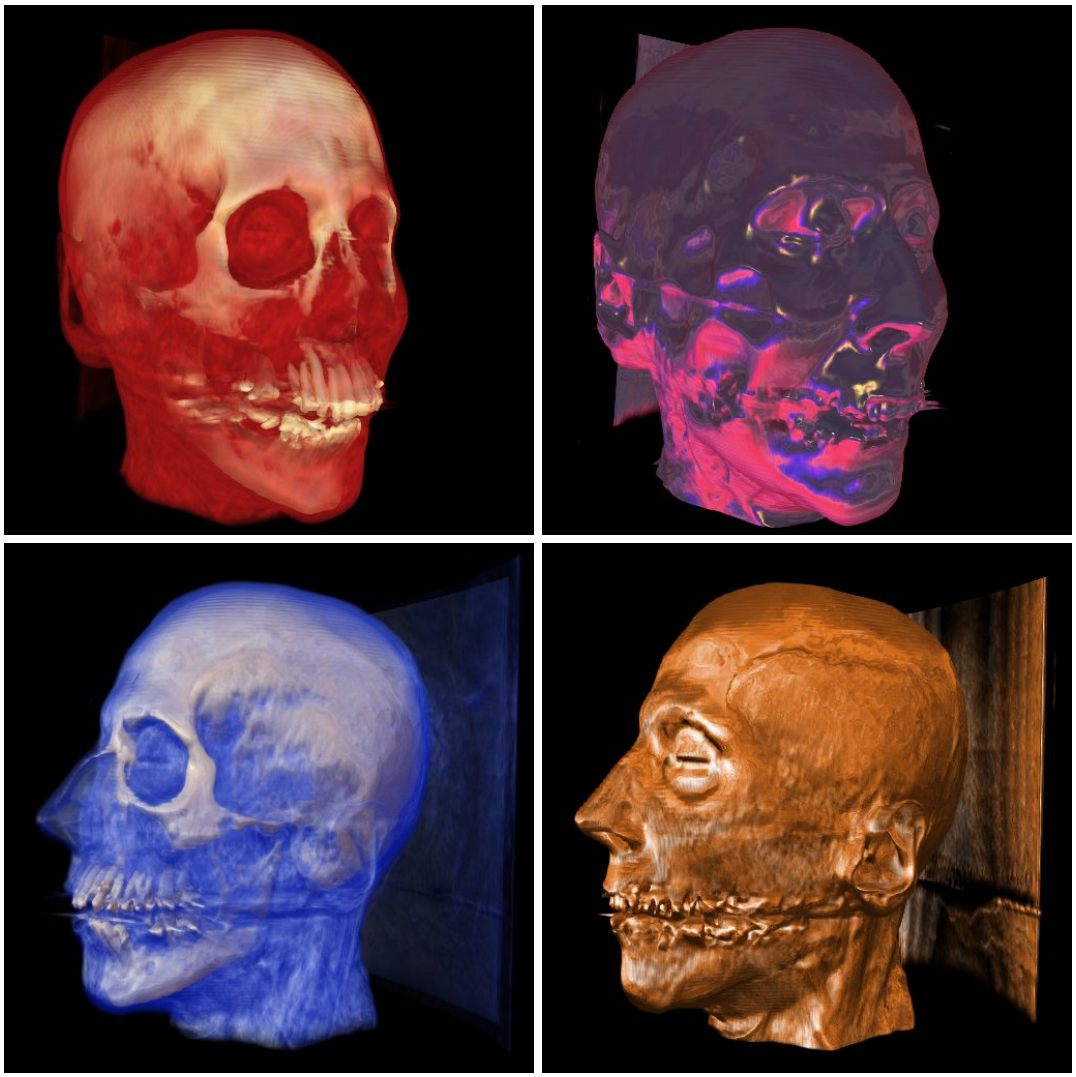
176 x 256 x 256
Bits Stored: 2^{12}
Unsigned Little Endian

Quelle: MHH

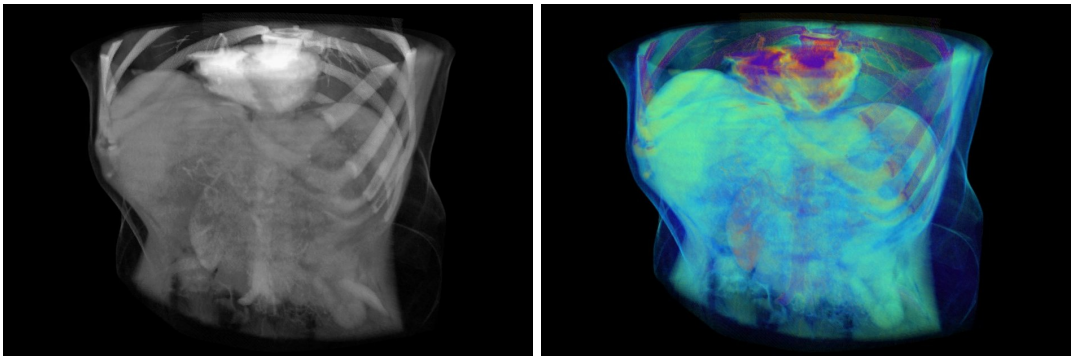


B. Bilder

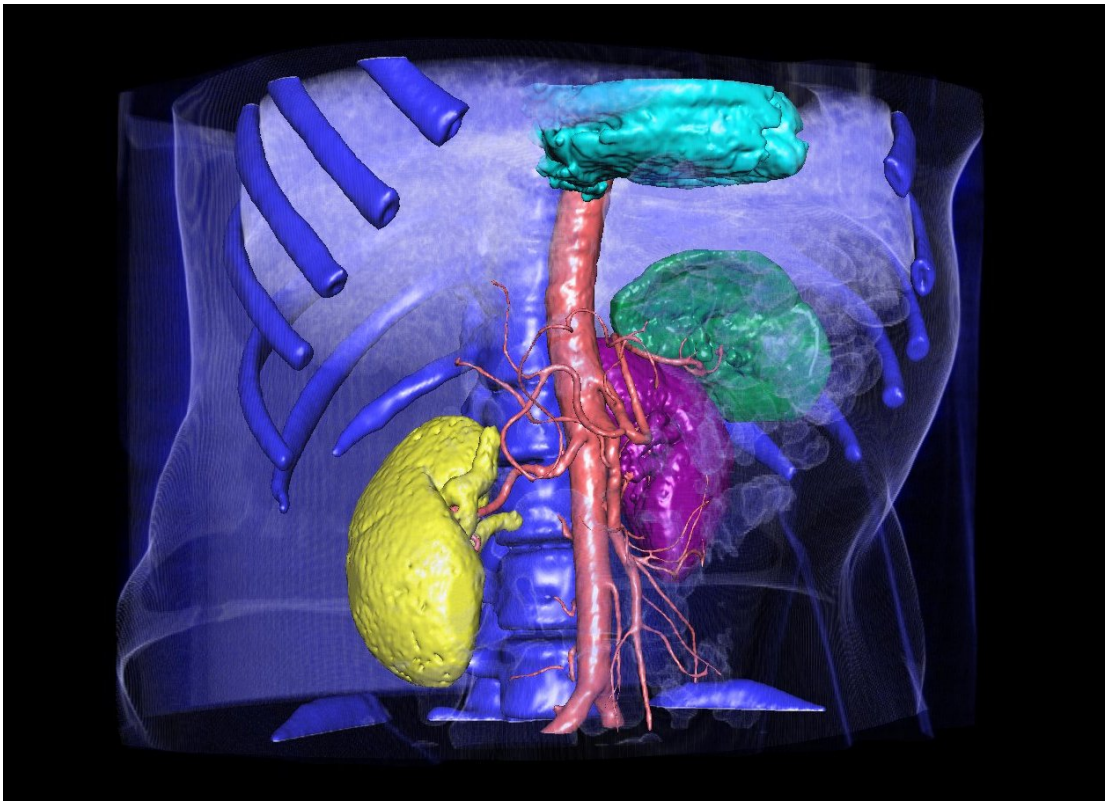
Dieser Abschnitt enthält Bilder, die mit den Modulen zur Volumen- und Segmentvisualisierung von YaDiV berechnet wurden und aus Platzgründen nicht im Hauptteil der Arbeit aufgenommen werden konnten.



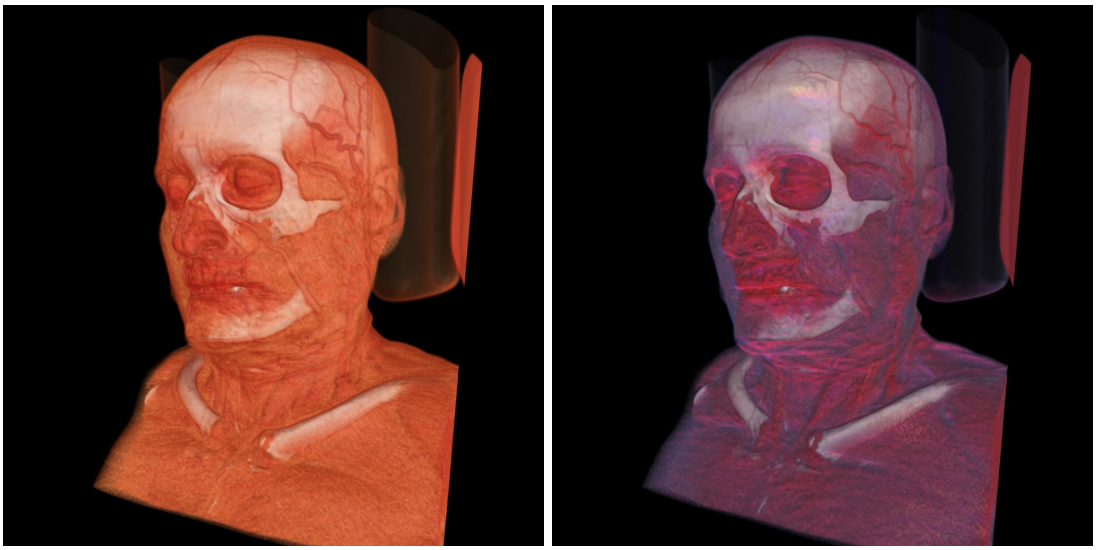
Unterschiedliche Zeichenstile durch Lit Sphere Mapping (CT-Head)



Texture3d Darstellung des Abdomen (ohne Shading) in Graustufen- und Pseudocolor-Darstellung

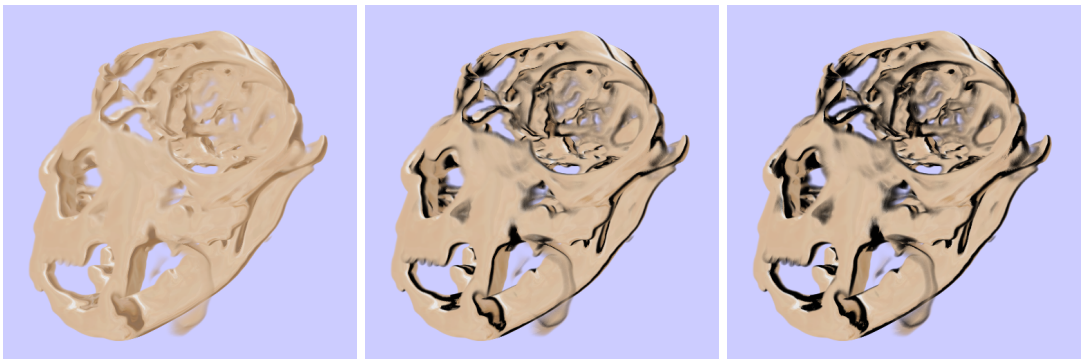


Kombination unterschiedlicher Methoden zur Visualisierung des Thorax: Die CT-Rohdaten wurden mit Texture3D-Volumenrendering unter Verwendung einer Transferfunktion mit hoher Transparenz visualisiert. Die Segmente – hier: beiden Nieren (Ren sinister und Ren dexter), Herz (Cor), Milz (Lien) und die Arorta – liegen als MC-Dreiecksflächen vor. Die bemerkenswert detaillierte Darstellung erreicht auf einem modernen Grafik-PC mit 12.5 FPS (in der stereographischen Darstellung) interaktive Frameraten.

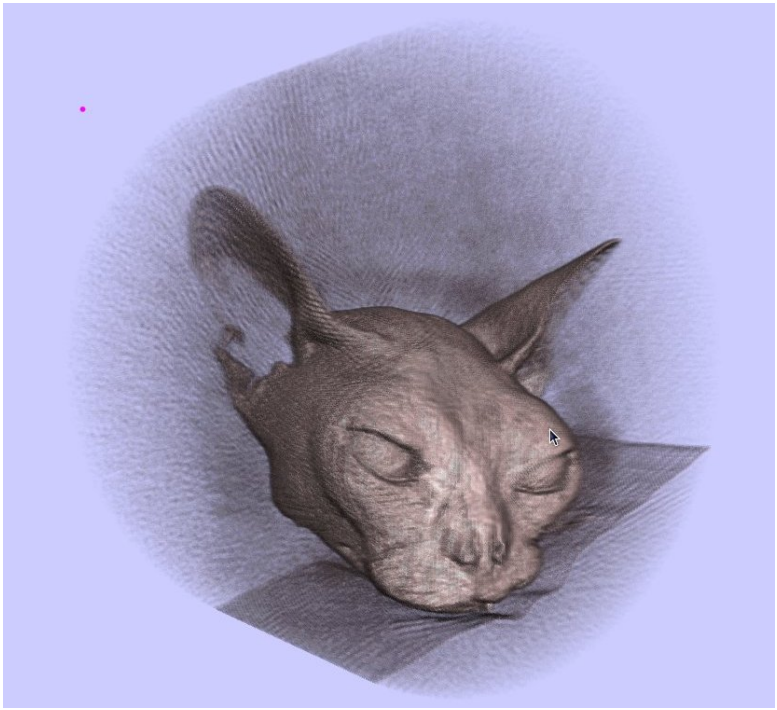


Volumen Ray Casting mit Lit Sphere Rendering: Identische Transferfunktion mit unterschiedlichen Lit Sphere Maps (Manix)

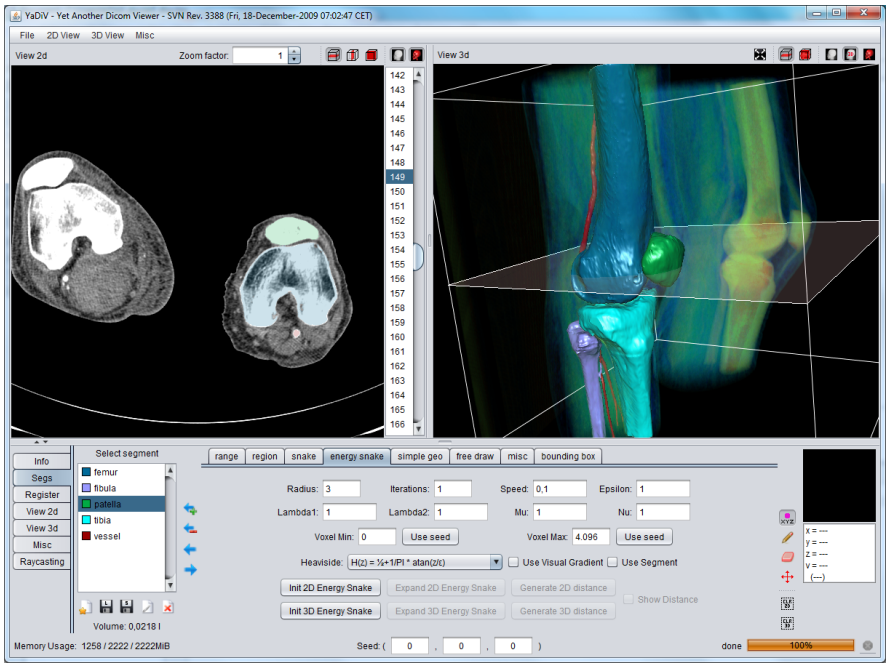
Krümmungsbasierte Konturhervorhebung im Ray Casting Verfahren



Visualisierung eines Katzen-Schädels im Cartoon-Stil mit unterschiedlichen Einstellungen zur Konturhervorhebung



Texture3D Volumendarstellung mit Shading. In dieser VOI-Einstellung gut zu erkennen sind die durch Quantenrauschen entstandenen Scan-Artefakte



Oberfläche von YaDiV: Segmente im Kniebereich

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[1] Allerkamp, Dennis: *Generation of Stimuli Supporting Tactile Perception of Textiles in a VR System*. Hannover, Germany, Leibniz Universität Hannover, Welfenlab, Dissertation, 2009 108

[2] Basdogan, Cagatay ; Sedef, Mert ; Harders, Matthias ; Wesarg, Stefan: VR-Based Simulators for Training in Minimally Invasive Surgery. In: *Computer Graphics and Applications* 27 (2007), Nr. 2, S. 54–66. http://dx.doi.org/10.1109/MCG.2007.51 111

[3] Bashore, Thomas M. ; Bates, Eric R. ; Berger, Peter B. ; A.Clark, David ; Cusma, Jack T. ; Dehmer, Gregory J. ; Kern, Morton J. ; K.Laskey, Warren ; Laughlin, Martin P. O. ; Oesterle, Stephen ; Popma, Jeffrey J. ; O'Rourke, Robert A. ; Abrams, Jonathan ; Bates, Eric R. ; Brodie, Bruce R. ; Douglas, Pamela S. ; Gregoratos, Gabriel ; Hlatky, Mark A. ; Hochman, Judith S. ; Kaul, Sanjiv ; Tracy, Cynthia M. ; Waters, David D. ; Winters, William L. J.: American College of Cardiology/Society for Cardiac Angiography and Interventions Clinical Expert Consensus Document on Cardiac Catheterization Laboratory Standards: A report of the American College of Cardiology Task Force on Clinical Expert Consensus Documents endorsed by the American Heart Association and the Diagnostic and Interventional Catheterization Committee of the Council on Clinical Cardiology of the AHA. In: *Journal of the American College of Cardiology* 37 (2001), Nr. 8, S. 2170–2214 26

[4] Blinn, James F.: What Is a Pixel? In: *IEEE Computer Graphics and Applications* 25 (2005), Nr. 5, S. 82–87. http://dx.doi.org/10.1109/MCG.2005.119 33

[5] Boulos, Solomon: Notes on efficient ray tracing. In: *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*. New York, NY, USA : ACM, 2005, S. 10 90

[6] Bronstein, I.N. ; Semendjajev, K.A.: *Taschenbuch der Mathematik*. Frankfurt am Main, Germany : Harri Deutsch Verlag, 1997 39

[7] Bruckner, Stefan ; Gröller, Meister E.: Style Transfer Functions for Illustrative Volume Rendering. In: *Computer Graphics Forum* 26 (2007), September, Nr. 3, S. 715–724 105, 159

[8] Böttcher, Guido ; Allerkamp, Dennis ; Glöckner, Daniel ; Wolter, Franz-Erich: Haptic Two-Finger Contact with Textiles. In: *The Visual Computer* 24 (2008), October, Nr. 10, 911–922 108

[9] Buzug, Thorsten M.: *Einführung in die Computertomographie, Mathematisch-physikalische Grundlagen der Bildrekonstruktion.* Berlin/Heidelberg, Germany : Springer-Verlag, 2004. – ISBN 978–3–540–20808–2 29

[10] Cabral, Brian ; Cam, Nancy ; Foran, Jim: Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In: *VVS '94: Proceedings of the 1994 symposium on Volume visualization.* New York, NY, USA : ACM, 1994. – ISBN 0–89791–741–3, S. 91–98 93

[11] Carneiro, Bernardo P. ; Silva, Cláudio T. ; Kaufman, Arie E.: Tetra-cubes: an algorithm to generate 3D isosurfaces based upon tetrahedra. In: *Brazilian Symposion on Computer Graphics and Image Processing, SIBGRAPI 96* 21 (1996), S. 205–210 101

[12] Chan, Tony F. ; Vese, Luminita A.: Active contours without edges. In: *IEEE Transactions on Image Processing* 10 (2001), Nr. 2, S. 266–277. `http://dx.doi.org/10.1109/83.902291` 62, 64

[13] Chen, Jim X. ; Chen, Chunyang: *Foundations of 3D Graphics Programming: Using JOGL and Java3D.* 2. Springer Publishing Company, Incorporated, 2008. – ISBN 9781848002838 126

[14] Chernyaev, Evgeni V.: Marching Cubes 33: Construction of Topologically Correct Isosurfaces / CERN. 1995. – Forschungsbericht 99

[15] Collignon, A. ; Maes, F. ; Delaere, D. ; Vandermeulen, D. ; Suetens, P. ; Marchal, G.: Automated multimodality image registration based on information theory. In: *XIVth international conference on information processing in medical imaging - IPMI'95* Bd. 3, Kluwer Academic Publishers, 1992, S. 263–274 72

[16] Dössel, Olaf: *Bildgebende Verfahren in der Medizin.* Springer Verlag, 2000. – ISBN 3–540–66014–3 13, 16

[17] EIZO NANAO Corporation: Accurate Grayscale on medical display monitors. Version: 2004. `http://www.eizo.com`. 2004. – White paper. – Available online (20 pages) 83

[18] Foley, James D. ; Dam, Andries van ; Feiner, Steven K. ; Hughes, John F.: *Computer graphics: principles and practice.* 2. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1990. – ISBN 0–201–12110–7 90, 106

[19] Färber, M. ; Hoeborn, E. ; Dalek, D. ; Hummel, F. ; Gerloff, C. ; Bohn, C.-A. ; Handels, H.: Training and Evaluation of Lumbar Punctures in a VR-Environment using 6DOF Haptic Device. In: *Medicine Meets Virtual Reality – Studies in Health Technology and Informatics* 16 (2008), Nr. 132, S. 112–114 111

[20] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA : Addison-Wesley, 1995. – ISBN 0201633612. – 27th Printing, November 2003 127

[21] Gelder, Allen van ; Kim, Kwansik: Direct volume rendering with shading via three-dimensional textures. In: *VVS '96: Proceedings of the 1996 symposium on Volume visualization*. Piscataway, NJ, USA : IEEE Press, 1996. – ISBN 0–89791–865–7, S. 23–ff. 95

[22] Giansiracusa, Robert S. ; Kalet, Ira J.: Prism/DICOM Coordinate Definitions and Interconversion / Radiation Oncology Department, University of Washington. University of Washington, Seattle, Washington, USA, 2003 (2003-08-01). – Technical Report 27

[23] Giess, C. ; Evers, H. ; Meinzer, H.-P.: Haptic Volume Rendering in Different Scenarios of Surgical Planning. In: *Proceedings of the Third PHANTOM Users Group Workshop*, 1998, 19–22 110

[24] Gottschalk, S. ; Lin, M. C. ; Manocha, D.: OBBTree: a hierarchical structure for rapid interference detection. In: *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM, 1996. – ISBN 0–89791–746–4, S. 171–180 114, 115

[25] Guercke, Richard: *Boundary Extraction from Voxel Data Using Contour Lines*. Hannover, Germany, Leibniz Universität Hannover, Welfenlab, Masterarbeit, November 2007 99

[26] Handels, Heinz: *Medizinische Bildverarbeitung: Bildanalyse, Mustererkennung und Visualisierung für die computergestützte ärztliche Diagnostik und Therapie*. Version: 2, 2009. `http://dx.doi.org/10.1007/978-3-8348-9571-4`. – DOI 10.1007/978–3–8348–9571–4. ISBN 978–3–8348–9571–4 47, 68, 74

[27] Harders, M. ; Székely, G.: Improving Medical Segmentation with Haptic Interaction. In: *IEEE Virtual Reality (VR 2002)*, 2002, S. 243–250 110

[28] Kajiya, James T. ; Herzen, Brian P. V.: Ray tracing volume densities. In: *SIGGRAPH Comput. Graphics* 18 (1984), Nr. 3, S. 165–174. `http://dx.doi.org/10.1145/964965.808594` 90

[29] Kaufman, Arie E.: *Volume Visualization (Tutorial)*. Los Alamitos, CA, USA : IEEE Computer Society Press, 1991. – ISBN 0818690208 9

[30] Kiefer, J. ; Wolfowitz, J.: Stochastic estimation of a regression function. In: *Annals of Mathematical Statistics* 23 (1952), Nr. 3, S. 462–466. `http://dx.doi.org/10.1214/aoms/1177729392` 76

[31] Kim, Sang-Youn ; Park, Jinah ; Kwon, Dong-Soo: The Real-Time Haptic Simulation of a Biomedical Volumetric Object with Shape-Retaining Chain Linked Model. In: *IEICE Transactions on Information and Systems* E88-D (2005), Nr. 5, S. 1012–1020. `http://dx.doi.org/10.1093/ietisy/e88-d.5.1012` 119

[32] Krüger, A. ; Stampe, K. ; Hertel, I. ; Strauss, G. ; Preim, B.: Haptische Interaktion zur Planung von Nasennebenhöhlen-Operationen. In: Horsch, A. (Hrsg.) ; Deserno, T.M. (Hrsg.) ; Handels, H. (Hrsg.) ; Meinzer, H.-P. (Hrsg.) ; Tolxdorff, T. (Hrsg.): *Bildverarbeitung für die Medizin 2007*, Springer, 2007, S. 303–307 111

[33] Lacroute, Philippe ; Levoy, Marc: Fast volume rendering using a shear-warp factorization of the viewing transformation. In: *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM, 1994. – ISBN 0–89791–667–0, S. 451–458 92

[34] Lefohn, Aaron E. ; Cates, Joshua E. ; Whitaker, Ross T.: Interactive, GPU-Based Level Sets for 3D Segmentation. In: *MICCAI (1)*, 2003, S. 564–572 68

[35] Levoy, Marc: Display of Surfaces From Volume Data. In: *IEEE Computer Graphics and Applications* 8 (1988), Nr. 3, S. 29–37 90

[36] Lewiner, Thomas ; Lopes, Hélio ; Vieira, Antônio W. ; Tavares, Geovan: Efficient Implementation of Marching Cubes' Cases with Topological Guarantees. In: *journal of graphics, gpu, and game tools* 8 (2003), Nr. 2, S. 1–15 99

[37] Lichtenbelt, Barthold ; Crane, Randy ; Naqvi, Shaz: *Introduction to volume rendering*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1998. – ISBN 0–13–861683–3 148

[38] Lin, Ming C. ; Otaduy, Miguel: *Haptic Rendering: Foundations, Algorithms and Applications*. Natick, MA, USA : A. K. Peters, Ltd., 2008. – ISBN 1568813325 107

[39] Lin, Ming C. ; Otaduy, Miguel: *Haptic Rendering: Foundations, Algorithms and Applications*. Natick, MA, USA : A. K. Peters, Ltd., 2008. – ISBN 1568813325, 9781568813325 112

[40] Loose, Reinhard ; Wucherer, Michael: Anwendung ionisierender Strahlung in der medizinischen Forschung. In: *Fortschr Röntgenstr* Bd. 174, 2002, S. 1191–1193 16

[41] Lorensen, William E. ; Cline, Harvey E.: Marching cubes: A high resolution 3D surface construction algorithm. In: *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* Bd. 21. New York, NY, USA : ACM Press, July 1987. – ISSN 0097–8930, S. 163–169 96, 97

[42] Maintz, J.B.A. ; Viergever, M.A.: A Survey of Medical Image Registration. In: *Medical Image Analysis* 2 (1998), Nr. 1, S. 1–36 69

[43] Malladi, Ravikanth ; Sethian, James A. ; Vemuri, Baba C.: Shape Modeling with Front Propagation: A Level Set Approach. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17 (1995), Nr. 2, S. 158–175. `http://dx.doi.org/10.1109/34.368173` 56, 57

[44] Marschner, Stephen R. ; Lobb, Richard J.: An evaluation of reconstruction filters for volume rendering. In: *VIS '94: Proceedings of the conference on Visualization '94*. Los Alamitos, CA, USA : IEEE Computer Society Press, 1994. – ISBN 0–7803–2521–4, S. 100–107 35, 37

[45] Matthijs, Paul: Grayscale resolution: How much is enough? Version: 2003. `http://www.barcomedical.com`. 2003. – White paper. – Barco, Available online (33 pages) 83

[46] Max, Nelson: Optical Models for Direct Volume Rendering. In: *IEEE Transactions on Visualization and Computer Graphics* 1 (1995), Nr. 2, S. 99–108. `http://dx.doi.org/10.1109/2945.468400` 87

[47] McCormick, B.H. ; DeFanti, Thomas A. ; (Eds.), Maxine D. B.: Visualization in Scientific Computing. In: *ACM Computer Graphics* 21 (1987), Nr. 6 9

[48] McNeely, William A. ; Puterbaugh, Kevin D. ; Troy, James J.: Six degree-of-freedom haptic rendering using voxel sampling. In: *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press/Addison-Wesley Publishing Co., 1999. – ISBN 0–201–48560–5, S. 401–408 113

[49] Meijden, O. van d. ; Schijven, M.: The value of haptic feedback in conventional and robot-assisted minimal invasive surgery and virtual reality training: a current review. In: *Surgical Endoscopy* 23 (2009), June, Nr. 6, S. 1180–1190. `http://dx.doi.org/10.1007/s00464-008-0298-x` 111

[50] Meyer, Robert: *Entwurf und Implementierung eines atlasbasierten Segmentierungsverfahrens für medizinische Volumendaten*. Hannover, Germany, Leibniz Universität Hannover, Welfenlab, Masterarbeit, Dezember 2009 70, 76, 77

[51] Mitchell, Don P. ; Netravali, Arun N.: Reconstruction filters in computer-graphics. In: *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM, 1988. – ISBN 0–89791–275–6, S. 221–228 36

[52] Müller, Maximilian: *High-Quality Raycasting Verfahren zur Visualisierung von segmentierten medizinischen Voxeldaten*. Hannover, Germany, Leibniz Universität Hannover, Welfenlab, Masterarbeit, April 2009 38, 91, 157, 158

[53] Montani, Claudio ; Scateni, Riccardo ; Scopigno, Roberto: A modified look-up table for implicit disambiguation of Marching Cubes. In: *The Visual Computer* 10 (1994), December, Nr. 6, S. 353–355 99

[54] Morneburg, Heinz (Hrsg.): *Bildgebende Systeme für die medizinische Diagnostik*. Publicis MCD Verlag, Erlangen, 1995 (3). – ISBN 978–3–89578–002–8 13

[55] Mulder, W.A. ; Osher, S. ; Sethian, James A.: Computing interface motion in compressible gas dynamics. In: *Journal of Computational Physics* 100 (1992), Nr. 2, S. 209–228. `http://dx.doi.org/10.1016/0021-9991(92)90229-R` 55

[56] Mumford, David ; Shah, Jayant: Optimal approximations by piecewise smooth functions and associated variational problems. In: *Communications on Pure and Applied Mathematics* 42 (1989), Nr. 5, S. 577–685. `http://dx.doi.org/10.1002/cpa.3160420503` 62

[57] NEMA: *DICOM Standard*. `http://medical.nema.org/`. Version: 2008 22, 24, 25, 26, 27, 28, 31, 83

[58] Neumann, László ; Csébfalvi, Balázs ; König, Andreas ; Gröller, Eduard: Gradient Estimation in Volume Data using 4D Linear Regression. In: Gross, M. (Hrsg.) ; Hopgood, F. R. A. (Hrsg.): *Computer Graphics Forum (Eurographics 2000)* Bd. 19(3), 2000, S. 351–358 41

[59] Nielson, Gregory M.: Dual Marching Cubes. In: *VIS '04: Proceedings of the conference on Visualization '04*. Washington, DC, USA : IEEE Computer Society, 2004. – ISBN 0–7803–8788–0, S. 489–496 102, 104

[60] Nissen, Steven E. ; Jr., John W. H. ; Simon, Rüdiger: Introduction and background: The International Angiographic Compression Study. In: *J Am Coll Cardiol* 35 (2000), Nr. 5, S. 1367–1369 26

[61] Osher, Stanley ; Sethian, James A.: Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. In: *Journal of Computational Physics* 79 (1988), Nr. 1, S. 12–49. `http://dx.doi.org/10.1016/0021-9991(88)90002-2` 53, 55

[62] Otmaier, Tobias ; Deml, B. ; Kübler, Bernhard ; Passig, Georg ; Reintsema, Detlef ; Seibold, Ulrich: Robot Assisted Force Feedback Surgery. Version: 2007. http://dx.doi.org/10.1007/978-3-540-71364-7. In: *Advances in Telerobotics* Bd. 31. Springer Berlin / Heidelberg, 2007. – DOI 10.1007/978–3–540–71364–7. – ISBN 978–3–540–71363–0, S. 361–379 111

[63] Palmer, Ian: *Essential Java 3D fast: developing 3D graphics applications in Java*. New York, NY, USA : Springer-Verlag New York, Inc., 2001. – ISBN 1–85233–394–4 126

[64] Radon, Johann: Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten. In: *Berichte Sächsische Akademie der Wissenschaften - Math.physik Klasse 69*. Leipzig, Germany, 1917, S. 262–277 15

[65] Rogers, David F.: *Procedural elements for computer graphics (2nd ed.)*. New York, NY, USA : McGraw-Hill, Inc., 1998. – ISBN 0–07–053548–5 78

[66] Rohlfing, Torsten ; Brandt, Robert ; Menzel, Randolf ; Russakoff, Daniel B. ; Maurer, Calvin R. Jr.: Quo Vadis, Atlas-Based Segmentation? In: Suri, Jasjit (Hrsg.) ; Wilson, David L. (Hrsg.) ; Laxminarayan, Swamy (Hrsg.): *The Handbook of Medical Image Analysis – Volume III: Registration Models*. New York, NY, USA : Kluwer Academic / Plenum Publishers, August 2005, Kapitel 11, S. 435–486 69

[67] Rueckert, D. ; Sonoda, L.I. ; Denton, E. ; Rankin, S. ; Hayes, C. ; Leach, M.O. ; Hill, D. ; Hawkes, D.J.: Comparison and evaluation of rigid and non-rigid registration of breast MR images. In: *Medical Image Computing and Computer-Assisted Interventation — MICCAI'98*, Springer Berlin / Heidelberg, 1998. – ISBN 978–3–540–65136–9 74

[68] Sabella, Paolo: A rendering algorithm for visualizing 3D scalar fields. In: *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM, 1988. – ISBN 0–89791–275–6, S. 51–58 88

[69] Salisbury, Kenneth ; Conti, Francois ; Barbagli, Federico: Haptic Rendering: Introductory Concepts. In: *IEEE Computer Graphics and Application* 24 (2004), Nr. 2, S. 24–32. http://dx.doi.org/10.1109/MCG.2004.1274058 108, 110

[70] Sarnow, Dominik: *Analyse von Direct Volume Rendering Verfahren zur Visualisierung von Voxeldaten*. Hannover, Germany, Leibniz Universität Hannover, Welfenlab, Diplomarbeit, Dezember 2007 87

[71] Sarrut, D. ; Miguet, S.: Similarity Measures for Image Registration. In: *European Workshop on Content-Based Multimedia Indexing*. Toulouse, France : IHMPT–IRIT, 1999, S. 263–270 69

[72] Scheuermann, Björn: *Analyse von Direct Volume Rendering Verfahren zur Visualisierung von Voxeldaten*. Hannover, Germany, Leibniz Universität Hannover, Welfenlab, Diplomarbeit, September 2008 53, 55, 57, 60, 61, 64, 65

[73] Schroeder, Will ; Martin, Ken ; Lorensen, Bill: *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. 4. Kitware, Inc., 1997. – ISBN 1–930934–19–X 122

[74] Serra, J.: *Image Analysis and Mathematical Morphology*. London, UK : Academic Press, 1982. – ISBN 0–12–637240–3 31, 33

[75] Shirley, Peter ; Tuchman, Allan: A polygonal approximation to direct scalar volume rendering. In: *SIGGRAPH Computer Graphics* 24 (1990), Nr. 5, S. 63–70. http://dx.doi.org/10.1145/99308.99322 100

[76] Sloan, Peter-Pike J. ; Martin, William ; Gooch, Amy ; Gooch, Bruce: The lit sphere: a model for capturing NPR shading from art. In: *GRIN'01: No description on Graphics interface 2001*. Toronto, Ont., Canada, Canada : Canadian Information Processing Society, 2001. – ISBN 0–9688808–0–0, S. 143–150 104, 159

[77] Smith, Alvy R.: A Pixel Is Not A Little Square, A Pixel Is Not A Little Square, A Pixel Is Not A Little Square! (And a Voxel is Not a Little Cube) / Microsoft. 1995 (TM 6). – Technical Memo 33

[78] Srinivasan, Shankar ; Mital, Dinesh P. ; Haque, Syed: A quantitative analysis of the effectiveness of laparascopy and endoscopy virtual reality simulators. In: *Computers & Electrical Engineering* 32 (2006), Nr. 4, S. 283–298. http://dx.doi.org/10.1016/j.compeleceng.2005.11.001 111

[79] Tarel, Jean-Philippe ; Boughorbel, Sabri: On the choice of similarity measures for image retrieval by example. In: *MULTIMEDIA '02: Proceedings of the tenth ACM international conference on Multimedia*. New York, NY, USA : ACM, 2002. – ISBN 1–58113–620–X, S. 446–455 69

[80] Terzopoulos, Demetri ; Kass, Michael ; Witkin, Andrew: Snakes: Active Contour Models. In: *International Journal of Computer Vision* Bd. 1, 1987, S. 321–331 52, 53

[81] Teschner, M. ; Kimmerle, S. ; Zachmann, G. ; Heidelberger, B. ; Raghupathi, Laks ; Fuhrmann, A. ; Cani, Marie-Paule ; Faure, François ; Magnenat-Thalmann, Nadia ; Strasser, W.: Collision Detection for Deformable Objects. In: *Eurographics State-of-the-Art Report (EG-STAR)* Eurographics Association, Eurographics Association, 2004, S. 119–139 120

[82] Vollmer, Marc C.: *Entwurf und Implementation einer haptischen Schnittstelle für YaDiV*. Hannover, Germany, Leibniz Universität Hannover, Welfenlab, Masterarbeit, Oktober 2009 115, 117, 118

[83] Warren, Henry S.: *Hacker's Delight*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2002. – ISBN 0201914654 137

[84] Wasilewski, M.: *Active Contours using Level Sets for Medical Image Segmentation*. `http://www.postulate.org/segmentation/segmentation.pdf` 58, 65

[85] Westover, Lee: Footprint evaluation for volume rendering. In: *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM, 1990. – ISBN 0–89791–344–2, S. 367–376 91

[86] Westover, Lee: *Splatting: A Parallel, Feed-Forward Volume Rendering Algorithm*. USA, University of North Carolina at Chapel Hill, Diss., Juli 1991 91

[87] Wilson, Orion ; VanGelder, Allen ; Wilhelms, Jane: Direct Volume Rendering via 3D Textures. Santa Cruz, CA, USA : University of California at Santa Cruz, 1994. – Forschungsbericht 94, 95

[88] Wollny, Gert ; Kruggel, Frithjof: Computational Cost of Non-Rigid Registration Algorithms Based on Fluid Dynamics. In: *IEEE Transactions on Medical Imaging* 21 (2002), Nr. 8, S. 946–952 72

[89] Yu, Yifan: *Algorithmen zur Randflächenapproximation von Volumenkörpern in Voxeldarstellung*. Hannover, Germany, Leibniz Universität Hannover, Welfenlab, Diplomarbeit, Februar 2007 99, 100, 101

[90] Zhao, Hong-Kai ; Chan, Tony ; Merriman, Barry ; Osher, Stanley J.: A variational level set approach to multiphase motion. In: *Journal of Computational Physics* 127 (1996), Nr. 1, S. 179–195. `http://dx.doi.org/10.1006/jcph.1996.0167` 64

[91] Zilles, C.B. ; Salisbury, J.K.: A constraint-based god-object method for haptic display. In: *Intelligent Robots and Systems, IEEE/RSJ International Conference on* 3 (1995), S. 3146. `http://dx.doi.org/10.1109/IROS.1995.525876` 116

Curriculum Vitae

Name: Karl-Ingo Friese
Geboren am: 2. Oktober 1971 in Hildesheim

Bildungsgang

1991	Abitur an der Robert Bosch Gesamtschule Hildesheim
2002	Diplom in Mathematik (mit Studienrichtung Informatik) an der Universität Hannover

Beruflicher Werdegang

1994-2001	Studentische Hilfskraft
2002-2005	Wissenschaftlicher Mitarbeiter LUH (Projekt Interaktiver Landschaftsplan)
2005-2006	Wissenschaftlicher Mitarbeiter LUH (3D Biomedical Imaging, Vorlesungskonzeption und Lehre)
2007	Wissenschaftlicher Mitarbeiter LUH Organisation der Cyberworlds 2007 International Conference
2008-2009	Wissenschaftlicher Mitarbeiter TUC / LUH