Volumenvisualisierung

Studienarbeit

vorgelegt von

Walter Schwebs

Institut für Computervisualistik

Arbeitsgruppe Computergrafik

Betreuer und Prüfer: Prof. Dr. Stefan Müller

Juni 2004

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Anforderung

Die Visualisierung von Volumendaten spielt in den verschiedensten Disziplinen eine immer größere Rolle, denn für das Verständnis komplexer Zusammenhänge ist die 3D-Darstellung oft hilfreich. Dafür stehen verschiedene Verfahren zur dreidimensionalen Darstellung aus der Computergrafik zur Verfügung. Die klassischen, auf Strahlverfolgung basierenden Algorithmen, bieten hohe Qualität, benötigen jedoch auch eine hohe Rechenleisung. Auf der anderen Seite bietet texturbasiertes Volumen-Rendering höhere Performance und ermöglicht eine interaktive Manipulation der Darstellung, allerdings zu Lasten der Bildqualität. Im Rahmen dieser Arbeit sollen daher verschiedene Verfahren untersucht und gegenübergestellt werden, um schließlich ein Verfahren auszuwählen und zu implementieren.
Ziel dieser Arbeit ist die Implementierung eines Verfahrens zur Visualisierung von Volumendaten. Das System soll in der Lage sein, Beispieldatensätze einzulesen und geeignet zu visualisieren. Die Ergebnisse der Arbeit sollen an ausgewählten Datensätzen dokumentiert und diskutiert werden.

1.2 Gliederung der Arbeit

Ausgehend von genannter Aufgabenstellung gliedert sich diese Studienarbeit folgendermaßen.
Kapitel 2 gibt einen Überblick über die grundlegenden Begriffe und Verfahren der Volumenvisualisierung.
Kapitel 3 befasst sich mit dem Texturbasierten Rendering von 3D-Datensätzen.
Kapitel 4 geht auf die Implementierung eines Visualisierungssystems für Volumendaten ein.
Kapitel 5 bietet eine Programmbeschreibung des erarbeiteten Rahmenwerks, bevor in **Kapitel 6** die vorliegende Arbeit zusammengefasst und ein Ausblick auf zukünftige Entwicklungen gegeben wird.

Kapitel 2

Volumenvisualisierung

2.1 Grundlagen

Um Volumendaten visualisieren zu können, muss zuerst geklärt werden was Volumendaten überhaupt sind. Ein Volumen ist ein dreidimensionales Feld. Im Sinne der wissenschaftlichen Visualisierung sind die Werte dieses Feldes skalare Abtastwerte an den Schnittpunkten eines Gitters, das verschiedene Formen haben kann (siehe Abbildung 2.2). Dieses 3D-Feld wird repräsentiert als Stapel von 2D-Schichtbildern, die übereinander gelegt, das Volumen ergeben. Das Volumen wird also in viele Scheiben zerschnitten. Alle diese Bilder zusammen ergeben, wie in Abbildung 2.1 zu sehen, den Datensatz. Analog zum Pixel als kleinstes Bildelement, gibt es auch ein kleinstes Volumenelement. Jedes dieser Volumenelemente enspricht der Dichte des Ausgangsmaterials am Abtastpunkt. Allerdings gibt es zwei Sichtweisen.

- **1.Definition:** Eine *Voxelzelle* ist ein durch acht Stützpunkte gegebener Würfel, der Wert des Volumenelements wird interpoliert.
- **2.Definition:** Unter einem *Voxel* versteht man einen einzelnen Punkt innerhalb des Volumens, der Wert des Volumenelements ist konstant.

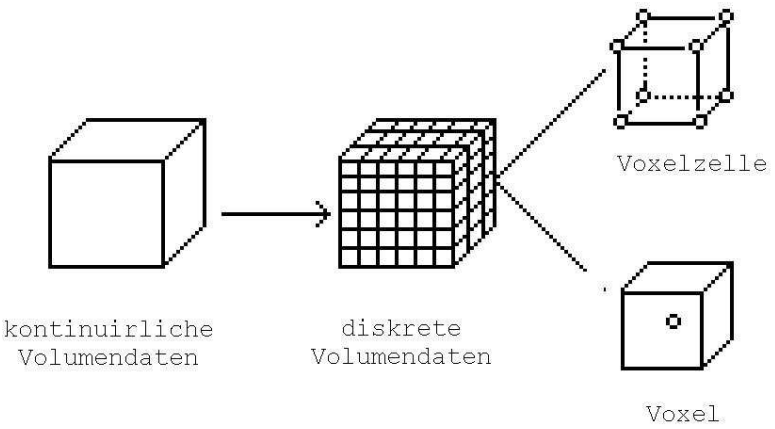

Abbildung 2.1: Der 3D-Datensatz

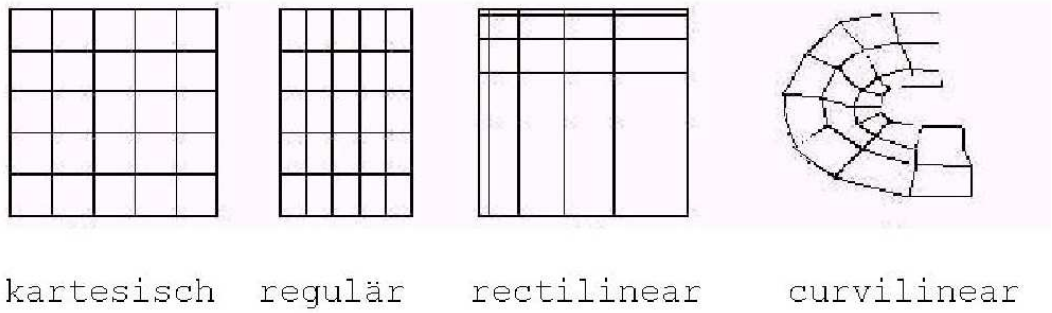

Abbildung 2.2: Verschiedene Gittertypen

- **Kartesische Gitter**
  Dies ist die einfachste Gitterstruktur. Die Zellen sind Würfel gleicher Größe und achsenparallel ausgerichtet.
- **Reguläre Gitter**
  Reguläre Gitter sind ähnlich zu kartesischen Gittern. Nur sind hier die Zellen keine Würfel, sondern Quader.
- **Rectilineare Gitter**
  Die Zellen sind wie bei regulären Gittern Quader. Allerdings sind die Abstände zwischen den Gitterpunkten entlang der Koordinatenachsen beliebig.
- **Curvilineare Gitter**
  Curvilineare Gitter entsprechen kartesischen Gittern, auf deren Gitterpunkte eine nicht-lineare Transformation angewendet wurde. Die Gitterstruktur kann somit Zellen enthalten, deren Flächen nicht planar sind.

2.2 Visualisierungs-Pipeline

Die Darstellung dreidimensionaler Datensätze erfordert einen mehrstufigen Prozess, der als Visualisierungs-Pipeline bezeichnet wird.

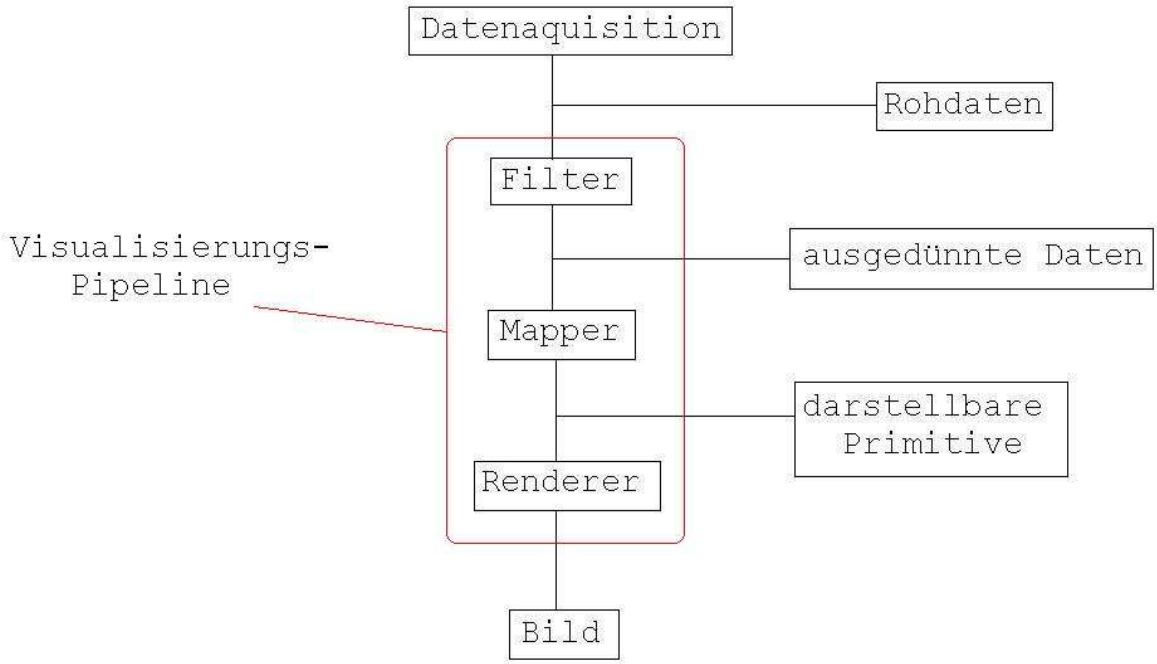

Abbildung 2.3: Die Visualisierungs-Pipeline

Datenaquisition

Die wissenschaftlichen Disziplinen, die dreidimensionale Datensätze liefern, die als Rohdaten dienen, sind vielfältig. Zu nennen sind zum Beispiel medizinische Visualisierungssysteme wie Computer Tomography oder Magnet Resonanz Imaging. Sowie bildgebende Systeme aus der Technik, wo solche Datensätze mittels Berechnung oder Simulation physikalischer Prozesse entstehen.

Filter

Diese Rohdaten können noch, für die Darstellung, irrelevante Punkte enthalten, die durch einen Filter entfernt werden.

Mapper

Der Mapper bringt diese Daten in eine darstellbare Repräsentation, wobei es sich um geometrische Primitive handelt, denen zusätzliche Attribute wie Farbe oder Transparenz zugeordnet werden können.

Renderer

Ein Renderer liefert schließlich das Ergebnisbild der Visualisierungs-Pipeline.

2.2.1 Verarbeitungsschritte

Die in den einzelnen Prozessstufen durchgeführten Verarbeitungsschritte, werden im folgenden kurz erläutert:

Rekonstruktion

Werden Daten aus einem physikalischen Prozess gewonnen, besteht eine Hauptaufgabe darin, diese in geeigneter Weise abzutasten, also zu diskretisieren. Bei der Visualisierung dieses Datensatzes wird eine Neuabtastung entsprechend der Lage des Volumens zur Bildebene notwendig(siehe Abbildung 2.4). Das Ziel der Rekonstruktion ist die Wiederherstellung des Orginalsignals aus den diskreten Gitterwerten. Dies erreicht man mittels einer Interpolationmethode, wie nearest Neighbor Interpolation, bi-oder tri-linearer Interpolation. Die Nearest Neighbor Methode ist die einfachste. Der

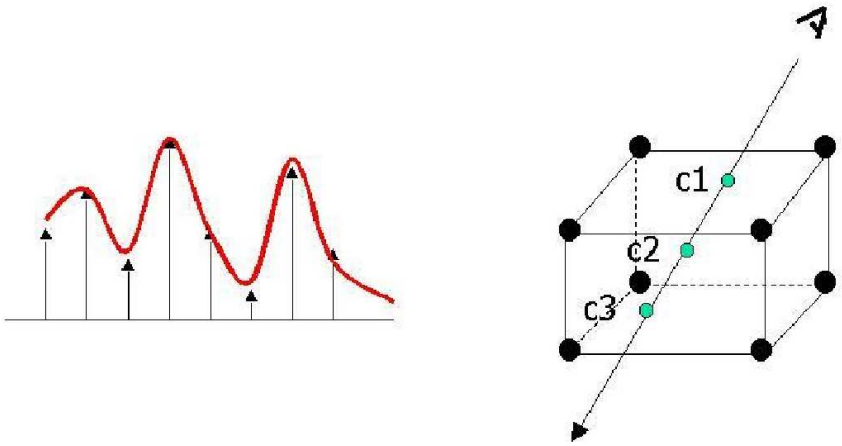

Abbildung 2.4: Darstellung der Rekonstruktion

für einen Punkt zu bestimmende Datenwert wird einfach durch den Datenwert des am nächsten liegenden Gitterpunktes approximiert. Bei der bi-linearen Interpolation wird der Dichtewert an einer Samplingposition aus den Dichtewerten der vier benachbarten Voxel bestimmt. Entsprechend werden bei der tri-linearen Interpolation acht benachbarte Voxel zur Berechnung herangezogen. Die Interpolation zwischen acht Dichtewerten ist am zeitaufwändigsten, allerdings liefert die tri-lineare Interpolation auch die besten Ergebnisse. Üblicherweise werden die Interpolationsmethoden als Filterfunktionen dargestellt(siehe Abbildung 2.5). Der tent-Filter entspricht zum Beispiel der tri-linearen Interpolation.

Klassifikation

Das Ziel der Klassifikation ist den Skalardataten visuelle Attribute, wie Farbe und Opazität, zuzuweisen. Dies wird durch Transfer-Funktionen realisiert. Hierbei wird unterschieden zwischen:

- **Präklassifikation** Anwendung der Transferfunktion vor der Rekonstruktion. Einfach zu realisieren, aber fehlerhaft gemäß dem Abtasttheorem.
- **Postklassifikation** Anwendung der Transferfunktion nach der Klassifikation. Korrekte Darstellung, aber schwierig zu realisieren.

Die Wahl der Klassifikationsart hat wesentlichen Einfluss auf das Ergebnissbild(siehe Abbildung 2.6).

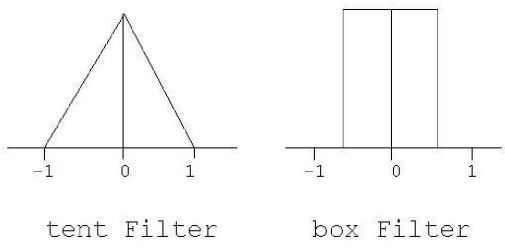

Abbildung 2.5: Zwei Rekonstruktionsfilter

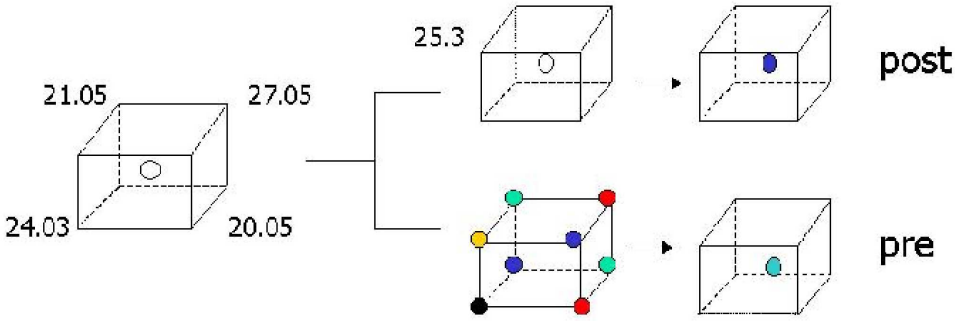

Abbildung 2.6: Vergleich von Prä-und Postklassifikation

Optisches Modell

Das optische Modell liefert eine wichtige theoretische Grundlage der Volumenvisualisierung. Bei den direkten Volumenvisualisierungsverfahren, die in dieser Arbeit im Vordergrund stehen, wird es realisiert durch das Emission/Absorption-Modell. Es geht von der Annahme aus, das jeder Punkt im Volumen Licht emmitieren und absorbieren kann und realisiert das physikalische Modell der Lichtausbreitung in transparenten Volumina. Um die Ausgabefarbe I zu berechnen, wird davon ausgegangen, dass der Sichtstrahl $\vec{x}(t)$ in Abhängigkeit vom Abstand zum Betrachter parametrisiert ist. An jedem Punkt des Strahls ist der Skalarwert gegeben durch $s(\vec{x}(t))$. Hinzu kommen der Farbkoeffizient, der dem emmitierten Licht entspricht und durch $c(s(\vec{x}(t)))$ dargestellt wird und der Absorptionskoeffizient durch $\tau(s(\vec{x}(t')))$ dargestellt. Somit ergibt sich für jeden Sichtstrahl das Volumen-Rendering-Integral durch folgende Gleichung.

$$I = \int_0^D c(s(\vec{x}(t)))e^{-\int_0^T \tau(s(\vec{x}(t')))dt'}dt$$

Da wir es allerdings mit einem diskreten Datensatz zu tun haben, ist diese Form des Volumen-Rendering-Integrals nicht anwendbar. Diese Gleichung muss also in eine diskrete Form gebracht werden.

Die diskrete Approximation wird zum Beispiel durch den *Over-Operator* realisiert, auf den im dritten Kapitel noch eingegangen wird..

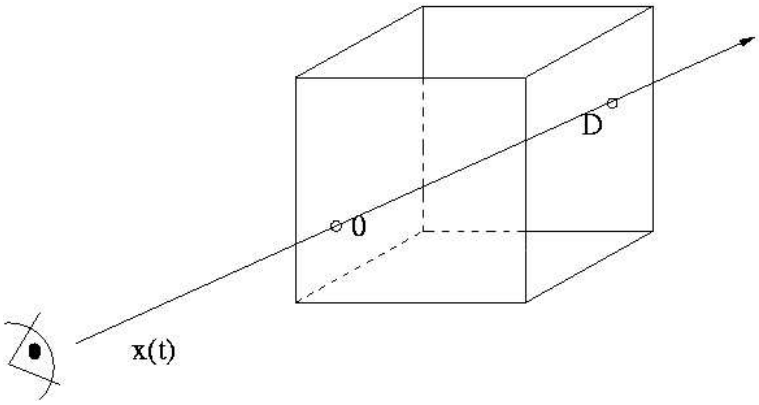

Abbildung 2.7: Das Volumen-Rendering-Integral

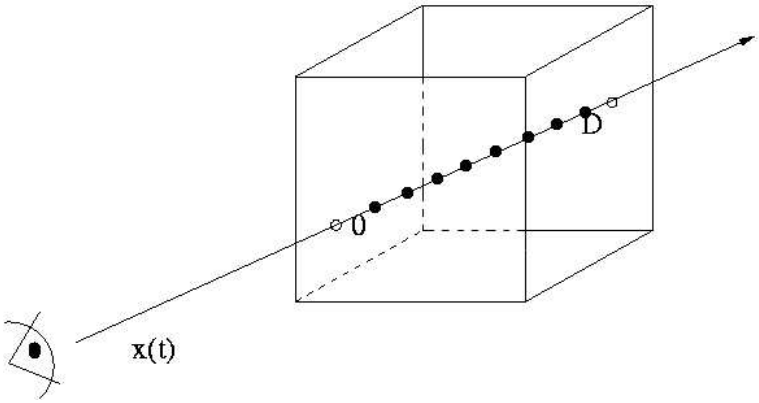

Abbildung 2.8: Diskrete Approximation des Volumen-Rendering-Integral

Beleuchtung

Die Exploration von 3D-Darstellungen wird durch eine geeignete Beleuchtung wesentlich unterstützt, da durch sie die Tiefenwahrnehmung der Volumina gefördert wird. In der Computergrafik findet vor allem das Phong-Modell Anwendung.

Composition

Unter der Composition versteht man das Zusammensetzen des 2D-Ergebnisbildes. Hier unterscheidet man zwischen **front-to-back**, wobei das Volumen bei der Mischung des Ergebnisbildes von vorne nach hinten traversiert wird und **back-to-front**, die Traversierung von hinten nach vorne durch den Datensatz.

Die Anordnung und Durchführung dieser Schritte hängen vom gewählten Verfahren ab, wie im nächsten Abschnitt erläutert wird.

2.3 Verfahren

Zunächst folgt eine grafische Übersicht der Volumenvisualisierungsverfahren.

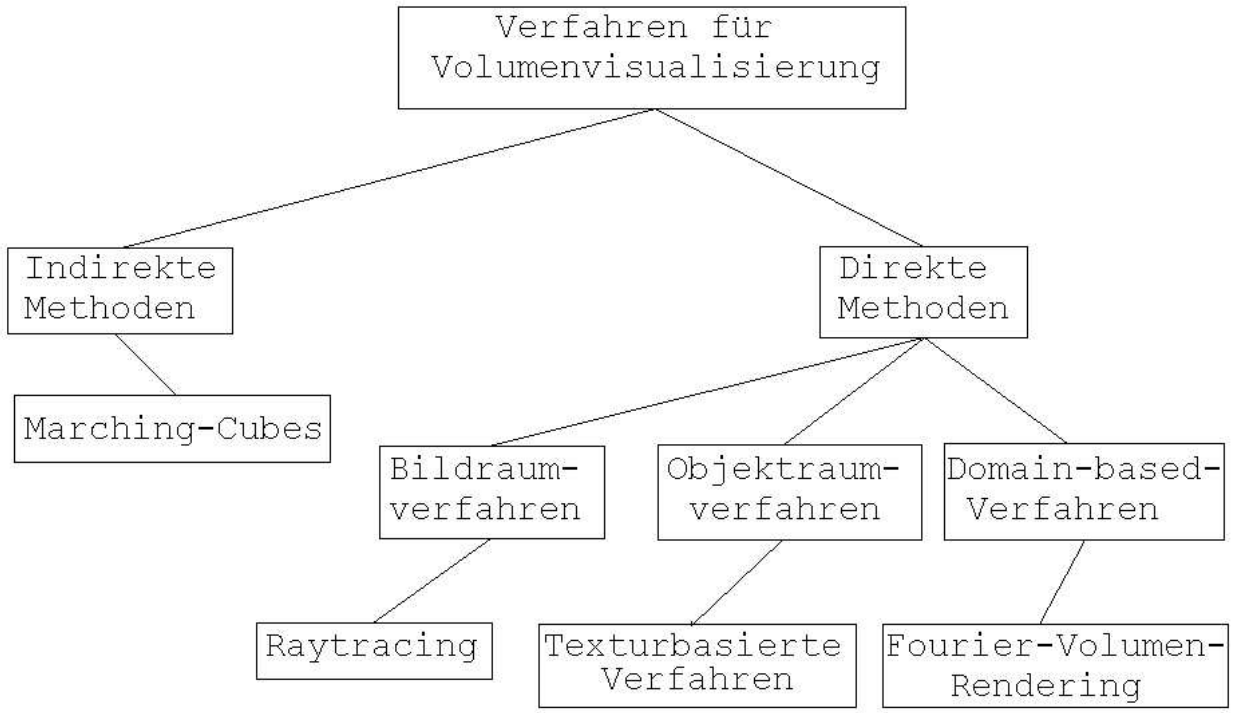

Abbildung 2.9: Übersicht über Volumenvisualisierungsverfahren

Indirektes Volumen-Rendering

Diese Verfahren sind dadurch gekennzeichnet, das eine Oberfläche innerhalb des Volumen bestimmt wird, als Isofläche bezeichnet, die dann mittels traditioneller Rendering-Verfahren dargestellt wird. Das Volumen wird also auf eine Flächenbeschreibung reduziert.
Hierzu gehört das Marching Cubes Verfahren, der bekannteste Vertreter dieser Art von Algorithmen. Hierbei wird das Volumen Voxelzelle für Voxelzelle durchwandert. Es folgt eine Klassifizierung der Eckpunke der Voxelzelle (als innerhalb oder außerhalb der Isofläche, wenn der Datenwert größer oder kleiner als der Isowert ist) aus der die Konfiguration für die Voxelzelle erfolgt. Grundlegend für dieses Verfahren ist die Überlegung, wie eine Fläche ein Voxel schneiden kann und wie dies zu approximieren ist. Abbildung 2.10 zeigt alle Fälle (unter Berücksichtigung der Symmetrie reduziert von 256 auf 15 Konfigurationen) die beim Marching-Cubes-Verfahren auftreten können.
Probleme, die mit diesem Verfahren einher gehen, sind vor allem Informationsverlust und eine aufwendige Vorverarbeitung.

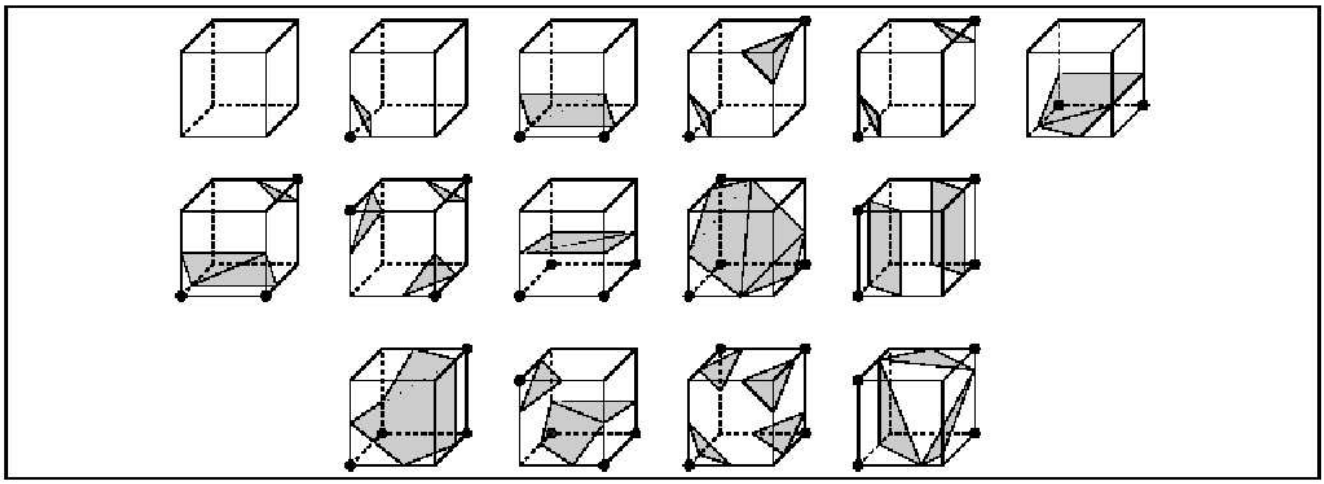

Abbildung 2.10: Marching-Cubes-Konfiguration

Direktes Volumen-Rendering

Direktes Volumen-Rendering, verzichtet auf eine intermediäre Repräsentation der Daten. Die Daten des Volumens werden direkt zur Darstellung verwendet und der 3D-Datensatz als ganzheitliches, transparentes Medium aufgefasst. Diese Verfahren lassen sich grob in folgende Klassen einteilen:

Bildraumverfahren

Die Bildraumverfahren zeichnen sich dadurch aus, das das Ergebnisbild in einzelne Pixel zerlegt wird. Für jedes dieser Ergebnispixel wird nun der Beitrag des Volumen zur Farbe des Pixels bestimmt. Als Beispielverfahren soll hier das Raytracing dienen.
Ausgehend von den einzelnen Pixeln der Bildebene werden Sehstrahlen durch das Volumen gesendet, in bestimmten Abständen werden Samples genommen. Die Rekonstruktion erfolgt also hier während der Integration. Der endgültige Pixelwert des Zielbildes wird aus den Werten der getroffenen Voxeln berechnet.

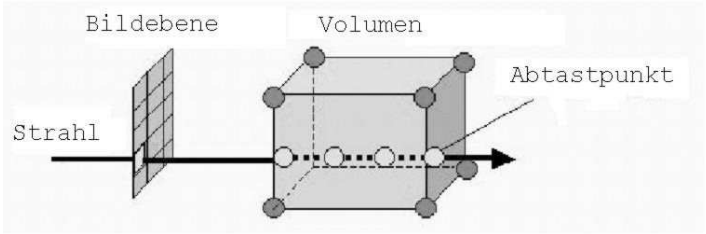

Abbildung 2.11: Das Prinzip des Raytracing

Objektraumverfahren

Einen alternativen Ansatz beschreiben die Objektraumverfahren. Ausgehend von den Voxeln des Volumens, wird für jeden Voxel dessen Beitrag zum Ergebnisbild bestimmt. Zu dieser Klasse von Verfahren gehört das Splatting, wobei die Voxel schichtweise auf das Ergebnisbild projeziert werden. Dabei wird der Beitrag eines Voxels zum Bild über mehrere Pixel verteilt. Für jeden Voxel wird in einer Tabelle ein Footprint gespeichert. Das Bild entsteht durch Compositing der transformierten Footprints.

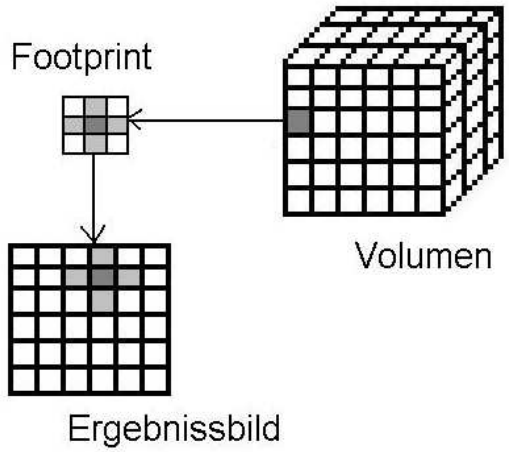

Abbildung 2.12: Das Prinzip des Splatting

Domain-basierte Verfahren

Diese Klasse von Algorithmen transferieren die Volumendaten zuerst in ein anderes System. Die Projektion der Voxel auf die Pixel des Zielbildes wird dann in diesem neuen System durchgeführt. Beim Frequenzraumverfahren erfolgt beispielsweise zunächst eine Transformation in den Frequenzraum.

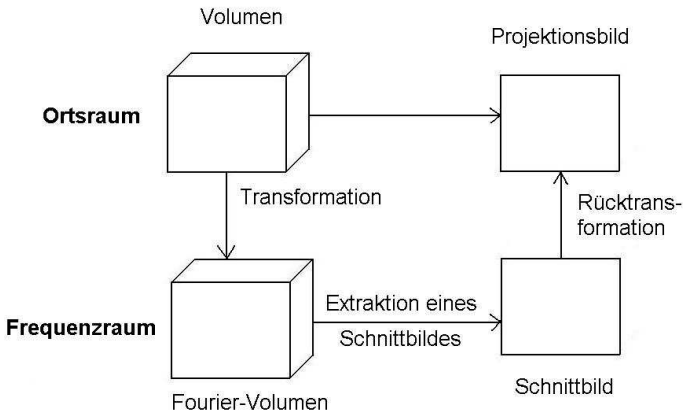

Abbildung 2.13: Das Frequenzraumverfahren

Kapitel 3

Texturbasiertes Volumen-Rendering

Das texturbasierte Volumen-Rendering aus der Familie der Objektraumverfahren, zeichnet sich vor allem durch die Ausnutzung gängiger Grafik-Hardware, wie Texture Mapping und Compositing Hardware, aus. Innerhalb des Volumens werden äquidistante Schnittebenen, auch als Slices bezeichnet, definiert. Diese Slices liefern Schnittpolygone, die nun back-to-front halbtransparent gezeichnet werden. Das Ergebnis entspricht näherungsweise der Auswertung des Volumenintegrals. Die Rekonstruktion erfolgt hier während des Texture Mapping.

3.1 3D-Textur basierte Verfahren

Das Volumen wird als ein 3D-Texturblock im Speicher gehalten. Die Slices werden parallel zur Bildebene (viewport-aligned) angeordnet. Wenn sich der Betrachterstandpunkt ändert, müssen die viewport-aligned Slices neu berechnet werden. Der Unterschied zwischen 3D-Texturen und den normalen 2D-Texturen besteht darin, das als Texturquelle keine 2D-Matrix, sondern eine 3D-Matrix dient. Die Slices werden also tri-linear interpoliert, dadurch ergibt sich die Möglichkeit die Bildqualität zu verbessern, indem einfach die Anzahl der Slices erhöht wird. Eines der größten Probleme bei diesem Verfahren ist die Größe des verfügbaren Texturspeichers, der oftmals die für das Volumen benötigte Kapazität nicht erreicht und auf älteren Rechnern werden 3D-Texturen gegebenenfalls überhaupt nicht unterstützt.

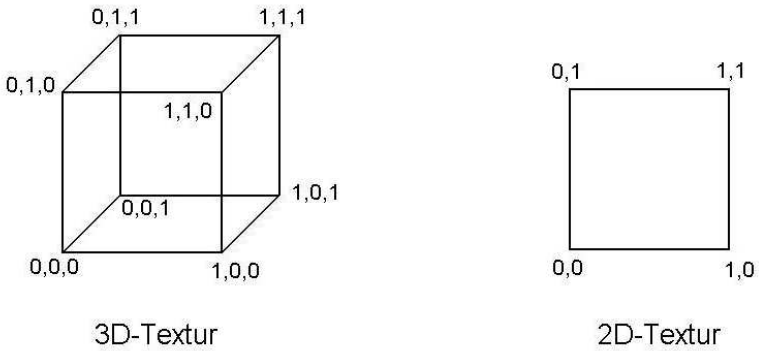

Abbildung 3.1: Vergleich der 3D-Textur mit der 2D-Textur

3.2 2D-Textur basierte Verfahren

Werden hingegen 2D-Texturen verwendet ist eine Modifikation des Algorithmus notwendig. Das Volumen wird als Stapel von 2D-Texturen im Speicher gehalten. Die Slices werden nun parallel zu einer der Hauptachsen des Volumens angeordnet. Daraus resultiert eine Verdreifachung des benötigten Texturspeichers, da für jede Achse ein eigener Stapel von 2D-Texturen benötigt wird(Abbildung 3.2). Für das Rendern des Volumens wird immer derjenige Stapel aus-

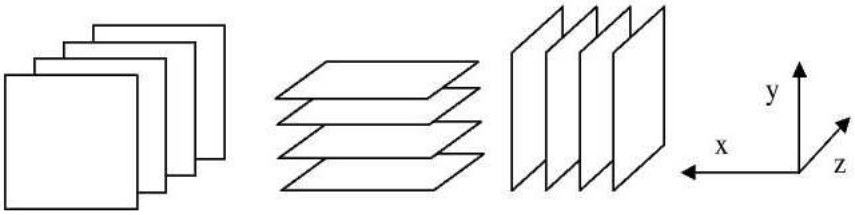

Abbildung 3.2: 2D-Textur-Stacks

gewählt, der möglichst senkrecht zur Blickrichtung ist. Ausserdem muss beachtet werden, dass je nachdem wo sich der Betrachterstandpunkt befindet, der Stack von vorne nach hinten oder umgekehrt gerendert werden muss. Die Schichten werden nun bi-linear interpoliert, dadurch verschlechtert sich die Bildqualität. Allerdings kann durch den Einsatz von Multitexturen die Bildqualität durch zusätzliche Zwischenschichten erhöht werden.

Alpha-Blending

Beiden texturbasierten Ansätzen verbindet das Rendern durch Alpha-Blending. Wobei die mit semitransparenten Texturen versehenen Slices nacheinander in den Bildspeicher gezeichnet werden. Allerdings bietet das Alpha-Blending verschiedene Möglichkeiten die Pixel der einzelnen Slices zu mischen.
Meist wird hierzu der schon erwähnte Over-Operator verwendet. Hierbei wird die Farbe c eines Pixels, durch folgende Gleichung bestimmt.

$$c = a_s c_s + (1 - a_s) c_d$$

Hierbei entspricht der a-Wert der Transparenz eines Pixels. Das Kürzel d kennzeichnet die Werte, die bereits im Bildspeicher enthalten sind und das Kürzel s kennzeichnet die Werte, die in den Bildspeicher geschrieben werden.

Abbildung 3.3: Visualisierung eines CT-Scans mit 2D-Texturen

3.3 Non-polygonal Isosurfaces

Neben der indirekten Methode zur Visualisierung von Isoflächen, existiert auch ein von Westermann und Ertl entwickelter texturbasierter Ansatz.
Der Datensatz wird dabei nicht in einem Vorverarbeitungsschritt mittels einem Isowert klassifiziert, sondern während der Rasterisierung. Praktisch geschieht dies, indem das OpenGl Alpha Blending mit dem OpenGl Alpha Test vertauscht wird. Dieser Test bewirkt, das nur Fragmente gerendert werden, die dem Isowert entsprechen.

3.3.1 Beleuchtung der Isosurfaces

Das Beleuchtungsmodell nach Phong bietet eine Simulation der Beleuchtung von Oberflächen. Wobei drei Anteile betrachtet werden:

$$I = I_{ambient} + I_{diffus} + I_{spekular}$$

ambiente Reflexion

bezeichnet den Anteil des Umgebungslichts und ist konstant

$$I_{ambient} = k_a = const.$$

diffuse Reflexion

Der diffuse Anteil ergibt sich gemäß des *Lambertschen Gesetzes* durch

$$I_{diffus} = I_d k_d (\vec{N} \circ \vec{L})$$

I_d beschreibt die Farbe des diffusen Lichts und k_d bezeichnet einen materialabhängigen Parameter. $\vec{L}$ stellt den Vektor in Richtung der Lichtquelle dar, $\vec{N}$ ist der Normalenvektor der beleuchteten Fläche.

spiegelnde Reflexion

Der spekulare Term beschreibt Reflexionen auf der Fläche und berechnet sich durch

$$I_{spekular} = I_s k_s (\vec{V} \circ \vec{R})^n$$

Dabei ist $\vec{V}$ der Vektor zum Beobachter und $\vec{R}$ der Reflexionsvektor von $\vec{L}$ an $\vec{N}$ in der Ebene, welche von $\vec{L}$ und $\vec{N}$ aufgespannt wird. Mit dem Exponenten *n* kann die Größe des Reflexionskegels kontrolliert werden.

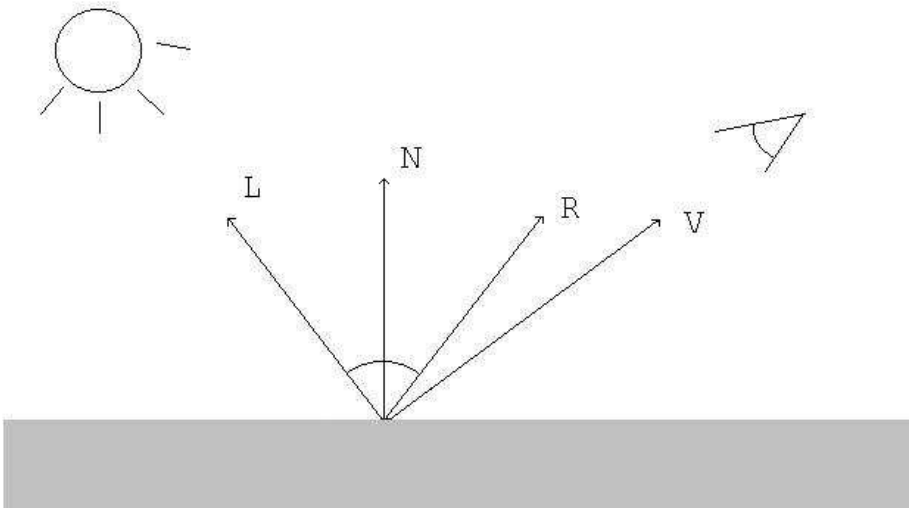

Abbildung 3.4: Das Phong-Modell

Um nun dieses Beleuchtungsmodell auf Volumendaten zu übertragen, sind einige Anpassungen notwendig.
Das Phong-Modell benötigt die Oberflächennormale für die Berechnung der Beleuchtung. Bei Voxeldaten verwendet man den Gradientenvektor des Voxels.
Zusätzlich kann man den Berechnungsaufwand des Phong-Modells wesentlich reduzieren, wenn man nur eine Lichtquelle mit konstanter Lichtrichtung betrachtet. Verzichtet man weiterhin auf den spekularen Anteil und ersetzt $I_d = 1$, erhält man folgende Gleichung:

$$I = (\vec{N} \circ \vec{L})$$

Dieses Skalarprodukt wird während der Textur-Anwendung bestimmt.

Kapitel 4

Implementierung

Im Rahmen dieser Studienarbeit wurde ein 2D-Textur basierter Volumen-Renderer in c++, basierend auf dem OpenGL-Api implementiert. Als Betriebssystem wurde SuSe Linux in der Version 9.0 genutzt. Der Benutzer hat dabei die Möglichkeit zwischen zwei Render-Modi zu unterscheiden.

4.1 Eingabedaten

Um einen 3D-Datensatz zu visualisieren, muss er zuerst in das Programm eingelesen werden. Dazu existiert die Funktion `readDataset()`, die den Datensatz in ein Array lädt. Die hier verwendeten Datensätze haben meist eine Größe von 128*128*64 Voxel (x,y,z). Die Skalardaten des 3D Feldes sind auf einem kartesischen Gitter gegeben. Der zu ladende Datensatz wird über eine *.raw Datei ausgewählt. Anschließend werden die Skalardaten aus der *.raw Datei eingelesen. Diese sind binär gespeichert und liegen im Datentyp `unsigned char` vor.

4.2 Darstellung semitransparenter Volumina

Folgende Methoden waren zur Realisierung des Algorithmus nötig:

4.2.1 Methoden

initVolume()

Wie im letzten Abschnitt erläutert wurde, wird der Datensatz als ein Stapel von 2D-Texturen aufgefaßt. In der Funktion `initVolume()` werden die Texturen für die drei Schichtenstapel , mittels Luminanz-Alpha-Texturen, generiert und im jeweiligen Datenfeld `GLuint Richtung_der_Slices[Anzahl_der_Voxel]` abgelegt. Jeder Stapel ist dabei entlang einer der drei Hauptachsen ausgerichtet. Folgender OpenGL-Befehl beschreibt die notwendige Initialisierung am Beispiel einer Textur in Z-Richtung.

```
glTexImage2D(GL_TEXTURE_2D,0,GL_LUMINANCE_ALPHA,XMAX,YMAX,0,GL_LUMINANCE_ALPHA,
GL_UNSIGNED_BYTE,zbuffer);
```

wichStack()

Wie im vorigen Kapitel beschrieben, ist es notwendig, denjenigen Textur-Stapel auszuwählen, der möglichst senkrecht zur Blickrichtung ist.
Die Funktion `whichStack()` liefert für einen gegebenen Betrachterstandpunkt den korrekten Schichtenstapel zurück. Dies geschieht durch Auswahl des Schichtenstapels, dessen Normalenvektor mit der Blickrichtung den kleinsten Winkel aufweist. Diese Funktion wird unverändert auch bei der Darstellung non-polygonaler Isosurfaces benutzt.

renderVolume()

Die Funktion `renderVolume()` zeichnet schließlich den Datensatz. Dies erfolgt durch Übereinanderblenden texturierter Polygone. Die Polygone sind dabei rechteckförmig und werden mit den in der `initVolume()`-Methode generierten Texturen belegt.
Durch folgende OpenGL-Befehle werden nun das Alpha-blending für das back-to-front-Compositing realisiert.

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
```

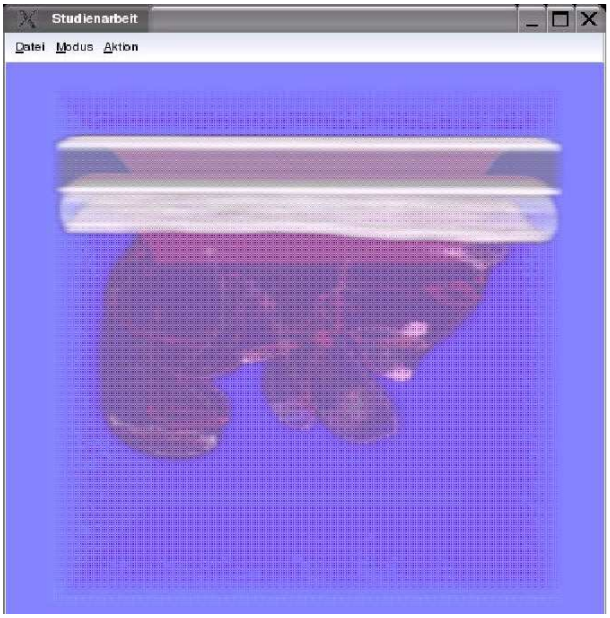

Abbildung 4.1: Ergebnis des implementierten Algorithmus

4.3 Darstellung beleuchteter Isosurfaces

Der zu visualisierende Datensatz wird ebenfalls, wie im vorherigen Kapitel beschrieben, durch die `readDataset()`-Methode eingelesen.

4.3.1 Methoden

getGradient()

Die Gradientenberechnung erfolgt für jeden Voxel in der Funktion `getGradient()` mittels Zentraldifferenzen. Am Rand des Volumendatensatzes werden Vorwärts-und Rückwärtsdifferenzen verwendet. Anschließend folgt die Normalisierung und die Speicherung in einer Textur.
Folgender Pseudocode beschreibt die Berechnung des Gradientenvektors `G(x,y,z)`, durch Zentraldifferenzen.
`G(x,y,z) =`

```
[(f(x+1,y,z)-f(x-1,y,z))/2,
 (f(x,y+1,z)-f(x,y-1,z))/2,
 (f(x,y,z+1)-f(x,y,z-1))/2]
```

initIsoVolume()

Da bei der Darstellung non-polygonaler-Isosurfaces zusätzlich noch der Gradientenvektor in der Textur gespeichert werden muss, werden nun statt Luminanz-Alpha-Texturen, RGBA-Texturen verwendet. Die Komponenten des Voxel-Gradienten werden in die RGB-Kanäle geschrieben und der Skalarwert des Volumen in den A-Kanal. Folgender Befehl generiert die geforderte Textur.

```
glTexImage2D(GL_TEXTURE_2D,0,GL_RGBA,XMAX,YMAX,0,GL_RGBA, GL_UNSIGNED_BYTE,zbuffer);
```

In dieser Code-Zeile wird eine Textur für den Texturstapel in Z-Richtung generiert. Die Parameter `XMAX,YMAX` entsprechen der Anzahl der Schnittebenen in X-und Y-Richtung. Weiterhin muss dieser Aufruf für jede Schnittebene in Z-Richtung erfolgen.
Die Textur-Stacks werden, wie bei der `initVolume()`-Methode in Datenfelder abgelegt.

renderIsoVolume()

Der Alpha-Test wird hier verwendet um alle Pixel zu entfernen, die nicht zur Isofläche gehören. In der vorliegenden Implementierung wird er gesetzt mit dem OpenGL-Aufruf:
`glAlphaFunc(GL_GREATER, IsoValue)`
Das heisst, es werden nur Fragmente gezeichnet, deren Alpha-Wert größer als IsoValue ist.
Da die Beleuchtungsberechnung durch Auswertung des Skalarprodukts

$$I = (\vec{N}^{\circ}\vec{L})$$

, während der Texturanwendung geschieht, wird die OpenGL-Extension `GL_ARB_texture_env_dot3` notwendig. ARB steht dabei für Architecture Review Board, das leitende, für OpenGl zuständige Organ. Diese Extension stellt die Operation zur Berechnung des Skalarprodukts während der Rasterisierung zur Verfügung.

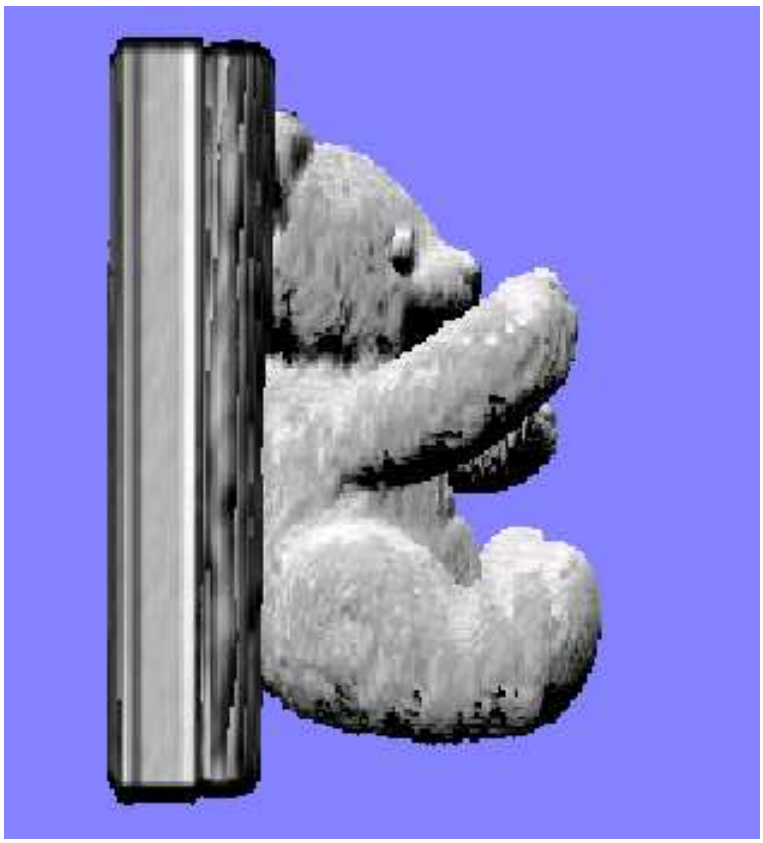

Abbildung 4.2: Darstellung mittels Iso-Flächen

Kapitel 5

Programmbeschreibung und Interaktion

Dieser Abschnitt erläutert alle Funktionalitäten und die Handhabung des Programms, dazu gehört hauptsächlich eine Beschreibung der Benutzerschnittstelle und der einzelnen Menüs.

5.1 Benutzerschnittstelle

Die Benutzerschnittstelle nutzt die QT-Library der Firma Trolltech.
Das Hauptfenster der Applikation ist in den Dateien *MyMainWindow.qt.cpp* und *MyMainWindow.qt.h* implementiert. Diese Klasse ist von der QT-Klasse *QMainWindow* abgeleitet und benutzt Standard-QT-Widgets, um sämtliche Menüpunkte zur Handhabung der Funktionalität und zur Interaktion mit dem Programm zur Verfügung zu stellen.

5.1.1 Menüs

- Das Menü *Datei*
  LoadFile: Ermöglicht das Laden der Volumendaten.
  Exit: Beendet das Programm.

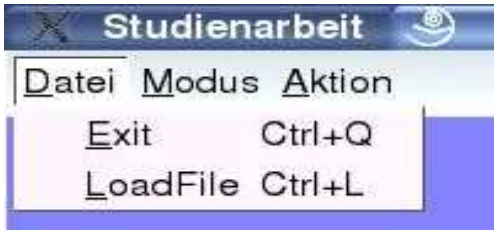

Abbildung 5.1: Das Datei-Menü

- Das Menü *Modus*
  Ermöglicht die Auswahl zwischen den beiden implementierten Darstellungs-Modi. Der semitransparenten Visualisierung der 3D-Datensätze über den Button *Textur*. Sowie der Darstellung beleuchteter Isofälchen mittels des *Iso*-Button.

- Das Menü *Aktion*
  Rendern: Visualisierung des gewählten Datensatzes.
  Isowert ändern: Erlaubt im *Iso-Mode*, das Ändern des Isowertes.

Abbildung 5.2: Das Modus-Menü

Abbildung 5.3: Das Aktion-Menü

5.2 Interaktion

Bei der Darstellung der Volumina stehen folgende Interaktionsmöglichkeiten zur Verfügung.

Rotation

Mittels gedrückter linker Maustaste läßt sich das Volumen um beliebige Achse rotieren.

Beleuchtung im Iso-Mode

Im *Iso-Mode* läßt sich zusätzlich die Position der Lichtquelle mittels gedrückter rechter Maustaste verstellen.

5.3 Ergebnisbilder

Es folgt nun eine kleine Auswahl an Bildern, die mit dem implementierten Programm visualisiert wurden.

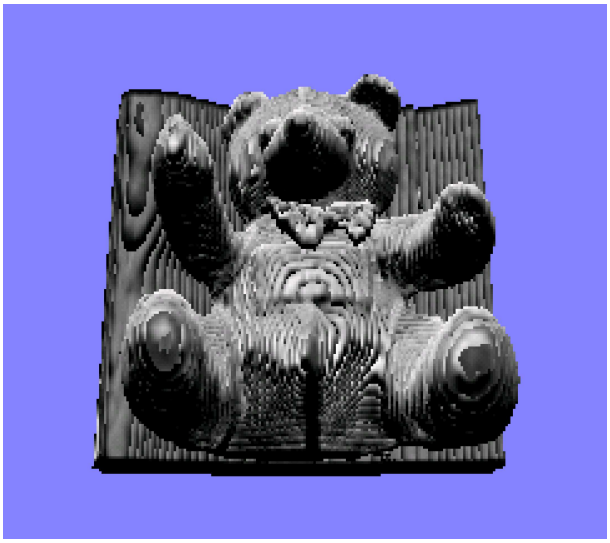

Abbildung 5.4: Darstellung der Rotation des Volumens und Variation des Isowertes

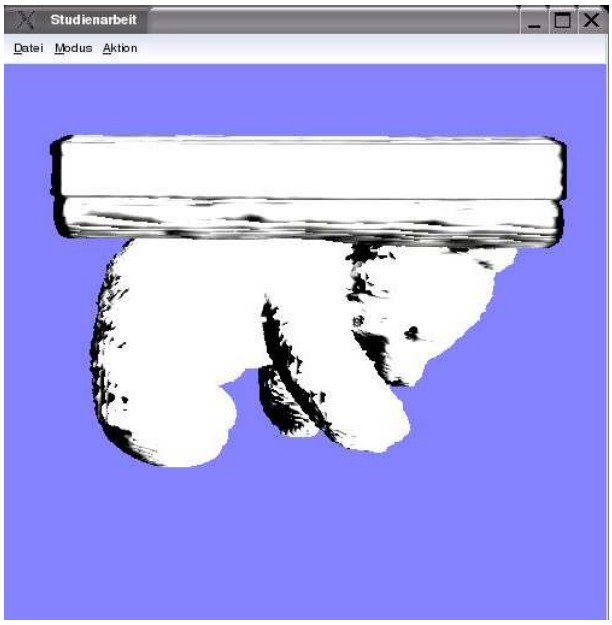

Abbildung 5.5: Veränderung der Beleuchtungssituation

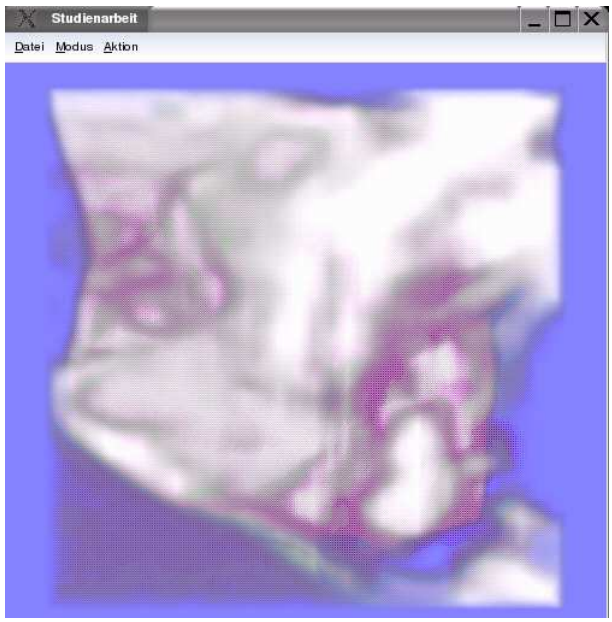

Abbildung 5.6: Darstellung eines CT-Scans vom Innenohr

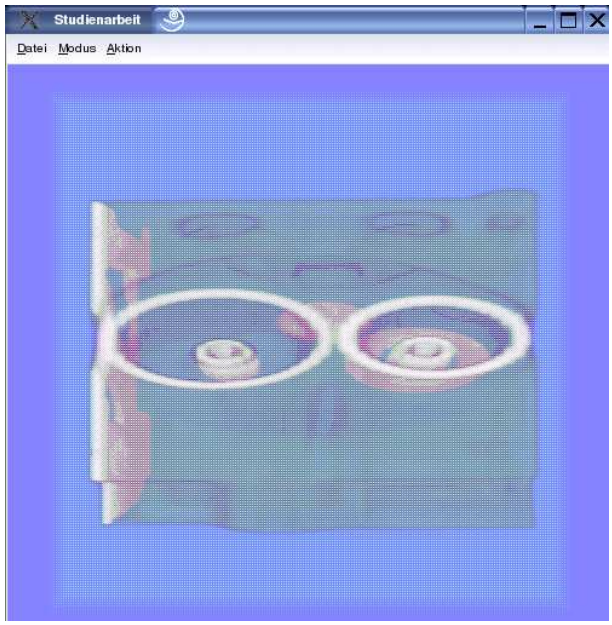

Abbildung 5.7: Darstellung eines CT-Scans von einem Motor, wobei der Datensatz die Größe $256x256x110$ Voxel hat.

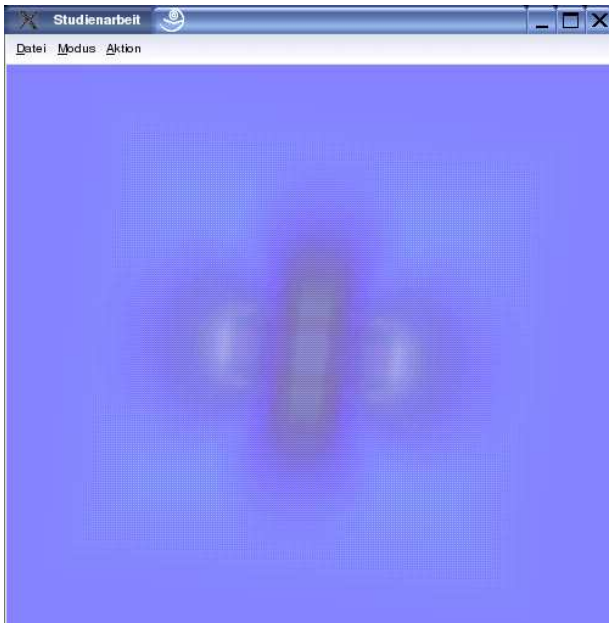

Abbildung 5.8: Semitransparente Darstellung der harmonischen Funktion, wobei der Datensatz die Größe $16x16x16$ Voxel hat.

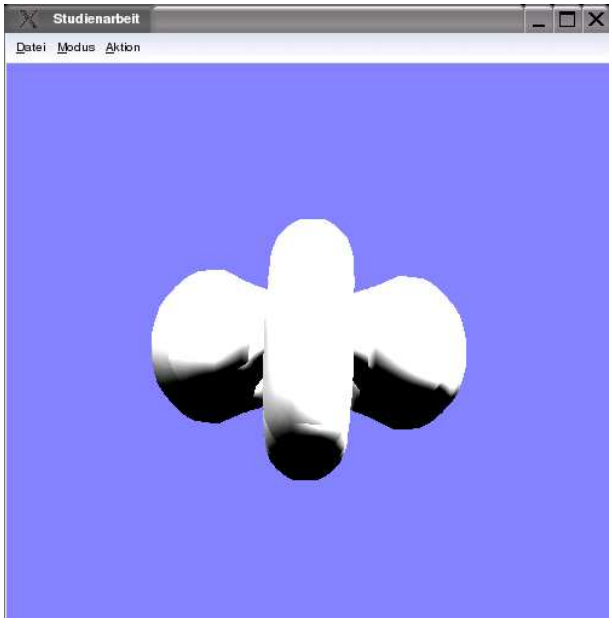

Abbildung 5.9: Darstellung der harmonischen Funktion im Iso-Mode, wobei der Datensatz die Größe $16x16x16$ Voxel hat.

Kapitel 6

Zusammenfassung und Ausblick

Diese Studienarbeit bietet einen Einblick in das Gebiet der Visualisierung von Volumendaten. Dabei wurde zunächst mit einem allgemeinen Überblick in den Themenbereich begonnen und anschließend konkrete Verfahren zur Volumenvisualisierung vorgestellt.
Schließlich wurde auf die Implementierung eines Programm eingegangen, das die Visualisierung von Volumendaten ermöglicht.

Die Funktionalität des Programms ist in vielerlei Hinsicht erweiterbar. So könnte man die Exploration der Volumendaten auf vielfältige Weise ergänzen. Beispielsweise durch Implementierung von Clipping-Ebenen, die störende Teile des Volumens wegschneiden, um bestimmte Subvolumen sichtbar zu machen, wie zum Beispiel in Abbildung 5.1. zu sehen ist.
Ein weiterer wichtiger Gesichtspunkt, der bei dem vorgestellten Programm noch nicht ausreichend berücksichtigt wurde, ist die Performance. Volumenvisualisierung ist höchst rechenintensiv, dies zeigt sich zum Beispiel in der langsamen und zähen Darstellung der Ergebnisbilder in diesem Programm. Moderne Hardware bietet enormes Verbesserungspotential.

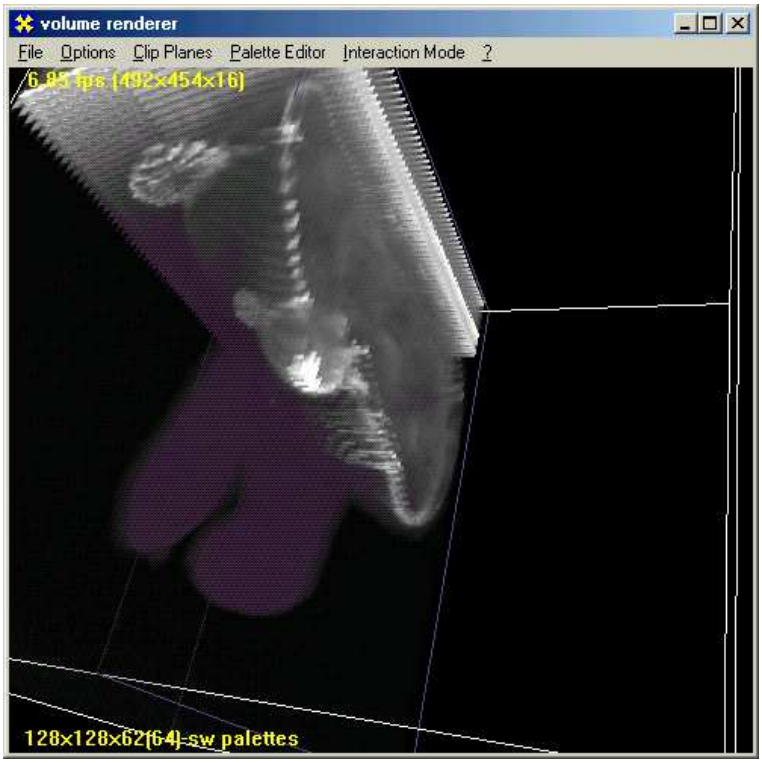

Abbildung 6.1: Volumenvisualisierung mit Clipping-Ebenen

Literaturverzeichnis

[EE02] Klaus Engel and Thomas Ertl. Interactive high-quality volume rendering with flexible consumer graphics hardware. 2002. EUROGRAPHICS‘02,pages 25-48.

[Eng02] Klaus Dieter Engel. Strategien und algorithmen zur interaktiven volumenvisualisierung indigitalen dokumenten. Stuttgart, 2002. Universität Stuttgart. Doktorarbeit.

[Rat98] Ralf Ratering. Texturbasiertes volumen-rendering medizinischer bilddaten. Bonn, 1998. RheinischeFriedrich-Wilhelms-Universität Bonn. Diplomarbeit im Fach Informatik.

[RS01] Christof Rezk-Salama. Techniken der volumenvisualisierung auf universell einsetzbarer graphik-hardware. Erlangen, 2001. Technische Fakultät der Universität-Erlangen-Nürnberg. Doktorarbeit.