UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

Analyse statischer Volumendaten zur Entwicklung eines adaptiven Raycastingverfahrens

Studienarbeit

im Studiengang Computervisualistik

vorgelegt von

Marius Erdt

Betreuer: Dipl.-Inform. Matthias Biedermann
(Institut für Computervisualistik, AG Computergraphik)

Koblenz, im Juli 2006

Erklärung

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....
(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Zusammenfassung

Das performante Rendering großer Volumendaten stellt trotz stetig gestiegener Prozessorleistungen nach wie vor hohe Anforderungen an jedes zugrunde liegende Visualisierungssystem. Insbesondere trifft dies auf direkte Rendering-Methoden mithilfe des Raycasting-Verfahrens zu, welches zum einen eine sehr hohe Qualität und Genauigkeit der generierten Bilder bietet, zum anderen aber aufgrund der dafür nötigen hohen Abtastrate relativ langsam ist.

In dieser Studienarbeit wird ein Verfahren zur Beschleunigung des Raycasting-Visualierungsansatzes vorgestellt, das auf adaptivem Sampling beruht. Dabei werden statische Volumendaten zunächst in einem Vorverarbeitungsschritt einer Gradientenanalyse unterzogen, um so ein Interessensvolumen zu erstellen, das wichtige und weniger wichtige Bereiche kennzeichnet. Dieses Volumen wird anschließend von einem Raycaster genutzt, um adaptiv für jeden Abtaststrahl die Schrittweite zu bestimmen.

Schwerpunkte dieser Arbeit bilden die Integration des Algorithmus in das Visualisierungssystem *Galenus* und die Auswertung der Ergebnisse in Bezug auf Qualität, Geschwindigkeit und benötigten Speicherplatz. Des Weiteren wird die Frage behandelt, ob eine von der Transferfunktion unabhängige (d.h. nur auf den Skalarwerten basierende) Analyse von Volumendaten möglich und sinnvoll ist.

1 Einleitung

1.1 Motivation

In einer Vielzahl diverser Anwendungsbereiche entstehen heutzutage große Mengen an Volumendaten. Die Bandbreite reicht hier von Simulationsergebnissen aus der Physik und den Ingenieurwissenschaften über Messungen meteorologischer und geologischer Daten bis hin zu medizinischen Aufnahmen, wie sie etwa Computertomographie und Magnetresonanz-Tomographie liefern. Die Visualisierung dieser Daten spielt vor allem in der Medizin eine bedeutende Rolle, da sie die Diagnostik in vielen Fällen unterstützen und verbessern kann. Häufig steht jedoch einer echtzeitfähigen Darstellung die Komplexität der gelieferten Informationen entgegen. Im Gegensatz zur zweidimensionalen Bildgebung erreichen Volumendaten auch bei geringen Auflösungen schnell Dimensionen, die direkte Visualisierungsalgorithmen an ihre Grenzen stoßen lassen. Besonders fällt dies beim *Raycastingalgorithmus* ins Gewicht, da hier normalerweise jedes Volumenelement einzeln abgetastet wird. Für eine hochqualitative Darstellung, wie sie z.B. in der medizinischen Diagnostik erforderlich ist, gibt es allerdings keine gleichwertige Alternative zu diesem Verfahren. Insofern stellt die Erweiterung des Raycasting-Ansatzes durch eine adaptive Abtastung der zu rendernden Volumina eine deutliche Leistungssteigerung auch bei großen Datenmengen in Aussicht.

1.2 Ziel der Arbeit

Ziel dieser Arbeit ist die Entwicklung eines adaptiven Raycasting-Konzeptes und seine konkrete Realisierung auf Grundlage des Volumenvisualisierungssystems *Galenus*[1]. Implementierungsschwerpunkte bilden dabei die Erstellung eines Analysesystems, das statische Volumendaten in einem Vorverarbeitungsschritt bez. visueller Kriterien untersucht, und die Anpassung des Galenus-Raycasters, um die so gewonnenen Ergebnisse für adaptives Sampling zu nutzen.

Der Fokus der Arbeit liegt auf der Entwicklung von adaptivem *GPU*-Raycasting, da hier mit Aufkommen programmierbarer Grafikhardware bereits interaktive Geschwindigkeiten erreicht werden können. Das System ist daher auf die speziellen Eigenschaften der Grafikkarte wie beispielsweise begrenzten Videospeicher und Parallelisierung ausgerichtet. Eine CPU-Implementierung wurde daher nur zu Vergleichszwecken erstellt und ist nicht Bestandteil der Testserien.

1.3 Aufbau der Arbeit

Im Folgenden werden der Inhalt der einzelnen Kapitel kurz erläutert und die wichtigsten Themen in den Gesamtzusammenhang eingeordnet.

- Kapitel 2 gibt zunächst einen Überblick über das Prinzip und den theoretischen Hintergrund der *direkten Volumenvisualisierung*. Besondere Aufmerksamkeit gilt dabei dem Konzept des *Raycastings* und dessen Umsetzung auf der GPU.

[1] Galenus ist ein System zur Darstellung von Volumendaten, das im Rahmen des Projektpraktikums „Medizinische Visualisierung“ im Wintersemester 2005/2006 an der Universität Koblenz-Landau realisiert wurde [5].

- In Kapitel 3 wird eine Reihe von bekannten Beschleunigungsverfahren für Raycasting gezeigt und bewertet. Anschließend werden das in dieser Arbeit verwendete *adaptive Sampling* und seine Besonderheiten erläutert.
- Im 4. Kapitel steht die Varianzdetektion in Volumendatensätzen im Vordergrund. Hierzu werden vor allem *gradientenbasierte* Verfahren bzw. darauf aufbauende Ansätze vorgestellt.
- Kapitel 5 beschreibt die konkrete Implementierung und die Integration von adaptivem Sampling in den Galenus Raycaster.
- Im 6. Kapitel werden die Ergebnisse mittels verschiedener Datensätze zusammengestellt und bewertet.
- Kapitel 7 zieht ein abschließendes Fazit und gibt einen Ausblick auf Sinn und Zweck der weiteren Verwendung von adaptivem Sampling.

2 Direkte Volumenvisualisierung

Das Prinzip der direkten Volumenvisualisierung[2] basiert auf der zweidimensionalen Abbildung einer Menge von Skalarwerten, die auf einem räumlichen Gitter angeordnet sind. Den Skalarwerten werden dabei eine Farbe und Opazität (RGBA-Wert) zugeordnet, die durch das dem Volumen zugrunde liegende *optische Modell*[3] bestimmt werden. Die Darstellung erfolgt dann durch Integration der RGBA-Werte entlang der Blickrichtung[4].

Man unterscheidet bei der direkten Volumenvisualisierung zwischen Verfahren, die im Bild-, Objekt- oder Frequenzraum arbeiten. Im Bildraum wird die Darstellung Pixel für Pixel aufgebaut, indem jeweils Sichtstrahlen durch das Volumen geschickt und die Werte integriert werden. Ein klassisches Beispiel für ein Bildraumverfahren ist das Raycasting, welches im Rahmen dieser Arbeit als Bildgebungsverfahren benutzt wird. Im Gegensatz dazu wird beim Objektraumverfahren vom Objekt selbst ausgegangen, z.B. wenn das Volumen auf die Bildebene projiziert wird. So kann ein Teil zur Farbe mehrerer Pixel beitragen. Frequenzraumverfahren verarbeiten die Volumendaten in der Frequenzdomäne und transformieren das Ergebnis anschließend in den Bildraum.

In diesem Kapitel werden die Grundlagen der direkten Volumendarstellung zusammenfassend erläutert. Dazu gehören das Konzept der Visualisierungspipeline, das Volumenrenderingintegral sowie die Beschreibung des Renderns von Volumina mithilfe des Raycastingalgorithmus und des Emissions-Absorptionsmodells.

2.1 Die Volumenvisualierungspipeline

Ein zentrales Konzept im Bereich des Volumenrenderings stellt die *Volumenvisualisierungspipeline* dar. Ähnlich wie die Renderingpipeline der Computergrafik beschreibt sie den Datenfluss von abstrakten Eingabewerten hin zu einer visuellen Darstellung, wie sie etwa in Form von zweidimensionalen Abbildungen oder bewegten Sequenzen auf einem Bildschirm ausgegeben werden können.

Abb. 1 veranschaulicht den grundlegenden Ablauf für die direkte Volumenvisualisierung. An erster Stelle steht die Gewinnung von Rohdaten, beispielsweise durch eine CT- oder MRT-Aufnahme[5]. Diesen Schritt bezeichnet man als *Datenakquisition*.

[2] Darüber hinaus gibt es indirekte Verfahren, die in einem Vorverarbeitungsschritt aus den Volumendaten ein Oberflächenmodell (z.B. ein Dreiecksgitter) erstellen, welches anschließend auf herkömmliche Weise gerendert werden kann. Prominente Vertreter sind Marching Cubes [9] und Ansätze aus der Bildverarbeitung (Konturdetektion, Region Growing).

[3] Ein solches Modell stellt im Wesentlichen eine Annäherung der physikalischen Ausbreitung von Licht in einem Volumen dar. Es gibt zahlreiche Ansätze [10], die z.T. auch komplexe Effekte wie Lichtbrechung- und Streuung berücksichtigen. Die populärste Variante ist das *Emissions-Absorptionsmodell* (s. auch 2.3).

[4] Auch wenn Sichtstrahlen bei einigen direkten Volumenvisualisierungsverfahren nicht explizit Teil des Algorithmus sind (vor allem Verfahren, die im Frequenz- oder Objektraum arbeiten), basiert deren Darstellung dennoch auf der Auswertung des Volumenrenderingintegrals.

[5] Computertomographie und Magnetresonanz-Tomographie.

Abbildung 1: Die Visualisierungspipeline für direktes Volumenrendering.

Beim nun folgenden *Sampling / Filtering* wird aus den gewonnenen Daten ein dreidimensionales Gitter erstellt, das als Grundlage für die Visualisierung dient. Dabei wird der Datensatz meist reduziert und gefiltert, um die anschließende Darstellung zu optimieren.

Den *Voxeln*[6] wird daraufhin ein RGBA-Wert[7] zugeordnet, der sich aus der Anwendung eines *optischen Modells* ergibt. Diese als *Klassifikation* bezeichnete Abbildung geschieht konkret durch eine vorher entsprechend festgelegte (multidimensionale) *Transferfunktion*, die im einfachsten Fall als Lookup-Tabelle realisiert werden kann.

Beim *Compositing* wird letztendlich, z.B. durch Strahlintegration entlang der Blickrichtung, aus den gewonnenen Farbwerten das Bild aufgebaut.

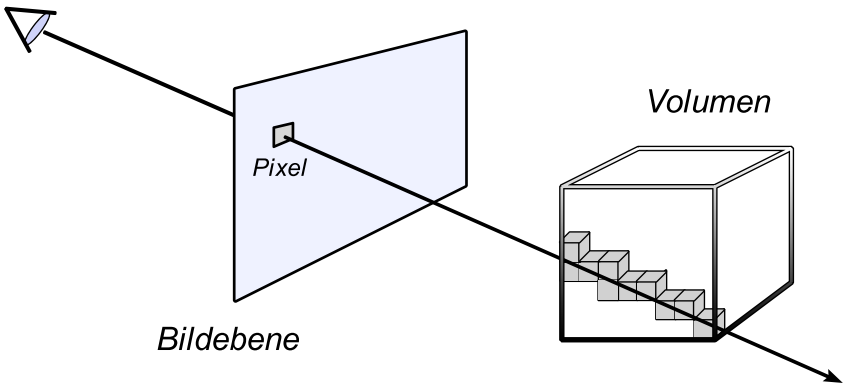

Abbildung 2: Das Raycasting-Prinzip: Für jedes Pixel wird ein Sichtstrahl durch das Volumen geschickt, wobei die dabei getroffenen Elemente den Farbwert bestimmen.

2.2 Raycasting

Raycasting ist ein direktes Visualisierungsverfahren, welches das Bild pixelweise aufbaut, indem Sichtstrahlen, die von einem festen Betrachtungspunkt aus durch die Zentren der Bildpixel verschickt werden, die Szene abtasten (s. Abb. 2).

Entlang jedes Strahls wird in gleichen Abständen[8] das Volumen aus den diskreten Gitterwerten rekonstruiert, beispielsweise durch trilineare Interpolation. Wie Abb. 3 zeigt, wird dabei aus den 8 umgebenden Knoten ein gewichteter Mittelwert

[6] Elemente des räumlichen Gitters.

[7] Bestimmt eine Farbe (red, green, blue) und eine Opazität (alpha).

[8] Adaptives Sampling basiert auf der Verwendung von unregelmäßigen Abständen, um Rechenzeit zu sparen (s. 3.3).

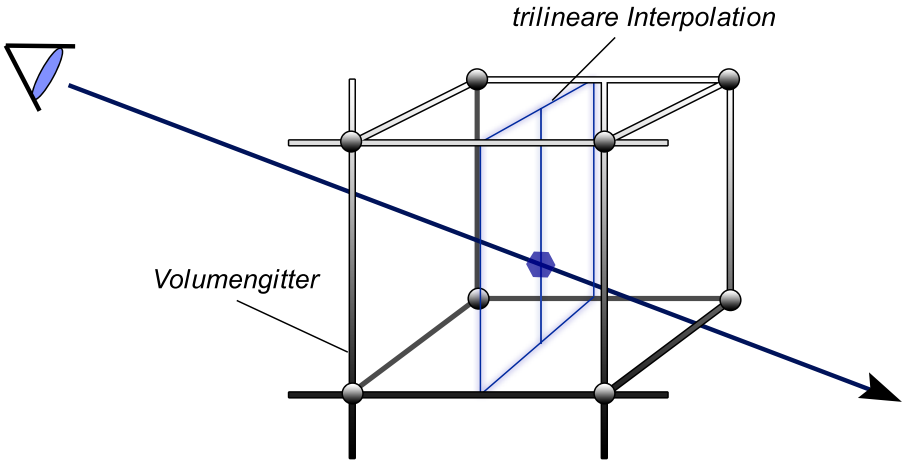

Abbildung 3: Entstehung eines Sichtstrahl-Samples durch trilineare Interpolation.

gebildet. Die Klassifikation, also die Zuordnung von Opazität und Farbe, kann entweder vor oder nach diesem Schritt erfolgen (pre-/postclassification). Die Farbe des jeweiligen Pixels wird nun durch Integration der so erhaltenen RGBA-Werte bestimmt. Abschnitt 2.3 zeigt eine näherungsweise Lösung dieses Problems anhand des Emissions-Absorptionsmodells.

2.3 Das Volumenrenderingintegral

Im *Emissions-Absorptionsmodell* wird das Volumen als Ansammlung von Licht emittierenden Partikeln mit einer bestimmten Dichte angesehen. Das heißt, es wird von jedem Element Licht erzeugt und auch absorbiert. Es können daher grundsätzlich alle Voxel der Szene zur Darstellung beitragen.

Beim Raycasting ergibt sich das Bild nun, indem für jeden Sichtstrahl berechnet wird, wie viel Licht der vom Strahl getroffenen Elemente beim Betrachter ankommt (Abb. 4). Diese Berechnung geschieht durch die Lösung des *Volumenrenderingintegrals*. Dabei werden der Sichtstrahl $\vec{x}(\delta)$ durch die Entfernung zum Augpunkt δ parametrisiert und die Skalarwerte des Volumengitters als $s(\vec{x}(\delta))$ bezeichnet. Aus der Anwendung des Emissions-Absorptionsmodells ergeben sich die entsprechende (Farb-)Emission als

$$c(\delta) := c(s(\vec{x}(\delta))) \tag{1}$$

und die Abschwächung mit

$$\kappa(\delta) := \kappa(s(\vec{x}(\delta))) \tag{2}$$

Das Volumenrenderingintegral für einen Strahl mit der Länge D_{max} lautet nun

$$C = \int_0^{D_{max}} c(\delta) \cdot e^{-\int_0^{\delta} \kappa(\delta')d\delta'} d\delta \tag{3}$$

Die Farbe eines Pixels berechnet sich also aus der Integration aller Emissionen, abgeschwächt durch alle Voxel, die zwischen dem Betrachter und der jeweiligen

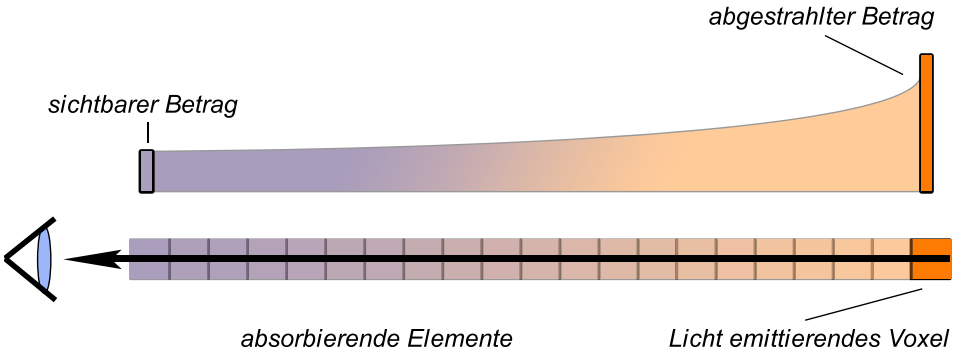

Abbildung 4: Licht wird im Volumen emittiert und erreicht abgeschwächt den Betrachter.

Strahlungsquelle liegen. Wie in 2.2 beschrieben, wird das Volumen beim Raycasting nicht kontinuierlich abgetastet, sondern in Intervallen. Daher wird das Volumenrenderingintegral hier numerisch angenähert. Wie in [3] gezeigt, lässt sich durch Riemannsummenbildung die Berechnung des Absorptionskoeffizienten sowie der emittierten Farben wie folgt vereinfachen

$$C \approx \sum_{i=0}^{D_{max}/d} \alpha_i C_i \prod_{j=0}^{i-1} (1 - \alpha_j) \tag{4}$$

wobei α die Opazität des Voxels und d die Länge der Abtastungsintervalle angeben.

Gleichung (4) lässt sich sehr effizient durch *alpha blending* umsetzen. Das Überblenden kann iterativ von vorne nach hinten (front-to-back) oder umgekehrt (back-to-front) geschehen, wobei Letzteres keine Zwischenspeicherung von α-Werten erfordert und deshalb bevorzugt verwendet wird[9].

Der Wert an der Stelle i ($i = n-1, .., 0$) ergibt sich beim back-to-front-blending aus der mit der Opazität α_i verrechneten Farbe C_i und dem bisherigen akkumulierten Wert C'_{i+1}.

$$C'_i = \alpha_i C_i + (1 - \alpha_i) C'_{i+1} \tag{5}$$

Für front-to-back-compositing ($i = 1, .., n$) erhält man folgende Gleichung

$$C'_i = C'_{i-1} + (1 - \alpha'_{i-1}) \alpha_i C_i \tag{6}$$

mit dem akkumulierten Opazitätswert α'_i

[9]Für *early ray termination* ist eine front-to-back-Reihenfolge nötig [s. (6)].

$$\alpha'_i = \alpha'_{i-1}(1 - \alpha'_{i-1})\alpha_i \tag{7}$$

2.4 GPU Raycasting

Raycastingalgorithmen bauen ein Bild durch eine Vielzahl unabhängiger Sichtstrahlen auf, die das Volumen abtasten. Dieses Prinzip legt eine Umsetzung auf die in hohem Maße parallelisierte, moderne Grafikhardware-Architektur nahe. So kann beispielsweise eine Nvidia 7900 GTX[10] Grafikkarte 24 Pixel pro Taktzyklus verarbeiten. Des Weiteren ist der hochperformante[11] Videospeicher heutzutage ausreichend groß ($\geq$ 512 MB), um auch komplexe Datensätze aufnehmen zu können. Die in den letzten Jahren stark gestiegene Flexibilität der Shader-Einheiten hat zudem die Implementierung von Raycasting auf der GPU stark vereinfacht. DirectX 9.0 Pixel Shadermodel 3.0 bietet hier Schleifenfunktionalität, inklusive Abbrüchen und Verzweigungen als wichtige Merkmale.

Für eine Raycasting-Implementierung auf der GPU muss zunächst der Datensatz als 3D-Textur in den Videospeicher der Grafikkarte geladen werden. Anschließend muss für jedes Pixel der Richtungsvektor des entsprechenden Sichtstrahls berechnet werden. Um dies zu erreichen, wird zunächst ein Würfel (bounding box) mit den Außmaßen des Volumens gezeichnet, der in sichtbare und verdeckte Seiten (front- und backfaces) aufgeteilt wird [6]. Die 3D-Texturkoordinaten der frontfaces werden nun als RGB-Farbwerte in einer 2D-Textur mit derselben Auflösung wie der Viewport gespeichert. Diese Farbanteile können so als Eintrittspunkte der Sichtstrahlen in das Volumen interpretiert werden. Analog werden auch die backfaces in eine 2D-Textur gerendert, dessen Komponenten die Austrittspunkte der Strahlen repräsentieren. Die Richtungsvektoren lassen sich durch Subtraktion der frontfaces von den backfaces und anschließende Normalisierung berechnen.

In einem Fragmentprogramm[12] wird nun der eigentliche Raycaster implementiert. Dazu wird das Voxelgitter in einer Schleife entlang der ermittelten Richtungsvektoren abgetastet. Da das Volumen als 3D-Textur vorliegt, kann die trilineare Interpolation der umgebenden Werte (s. Kap. 2.2) hardwareunterstützt und damit sehr effizient umgesetzt werden.

Die darauf folgende Anwendung der Transferfunktion (Postklassifikation) wird als Texturzugriff auf eine Lookup-Tabelle realisiert, welche die Abbildung auf Farb- und Opazitätswerte beinhaltet. In jedem Schleifendurchlauf werden diese akkumuliert, bis das Volumen durchlaufen oder eine bestimmte Opazitätsgrenze überschritten ist[13].

[10] Erschienen 2006.

[11] GDDR3-Speicher kann mit Realtaktraten von über 400 MHz betrieben werden (2006). Im Vergleich zur CPU-Anbindung kommen außerdem breitbandigere Speicherinterfaces zum Einsatz und erlauben so Durchsatzraten von bis zu 50 GB/s.

[12] In Assembler oder einer Shaderhochsprache, wie z.B. *Cg*, vorliegender Code, den Pixel-/Fragmentshader verarbeiten können.

[13] *Early ray termination* mit front-to-back-compositing (s. 3.1).

3 Beschleunigungsverfahren

Die Bildgenerierung findet beim Raycasting prinzipiell unter Berücksichtigung aller Elemente statt, was eine lineare Steigerung der Renderingzeit bei wachsender Komplexität des Volumens mit sich bringt. Dies bedeutet, dass bei einer Verdopplung der Auflösung eines dreidimensionalen Datensatzes die Darstellung um den Faktor 8 verzögert wird.

Zur Beschleunigung von Raycasting wurden daher zahlreiche Ansätze entwickelt, die meistens die Verringerung der räumlichen Komplexität des Datensatzes zum Ziel haben. Dies kann durch Überspringen leerer Bereiche, den Aufbau von hierarchischen Datenstrukturen, wie es beim Raytracing verwendet wird, und Ähnlichem geschehen. Nicht alle Vorgehensweisen sind dabei gleich gut für eine GPU-Implementierung geeignet, da hier die besonderen Eigenschaften der Grafikhardware, wie z.B. begrenzter Speicherplatz, berücksichtigt werden müssen.

Darüber hinaus wurden Ansätze entwickelt, die eine Geschwindigkeitserhöhung durch Reduzierung der Bildqualität erreichen, beispielsweise indem nicht alle Teile des Datensatzes berücksichtigt werden oder einfach durch Senkung der Abtastrate. So beschreibt M. Levoy [8], ausgehend von einer groben Bildauflösung, eine sukzessive Verbesserung der Darstellung durch Erhöhung der Anzahl an Abtaststrahlen in Bereichen hoher Komplexität. Der Schwerpunkt dieser Arbeit liegt allerdings auf der Beschleunigung von Raycasting bei möglichst gleichbleibender Renderingqualität. Daher werden die oben genannten Herangehensweisen hier nur erwähnt und im weiteren Verlauf nicht berücksichtigt.

In diesem Abschnitt werden einige der wichtigsten Beschleunigungsverfahren für Raycasting wie *early ray termination*, *empty space skipping* und *adaptives Sampling* vorgestellt. Das in dieser Arbeit entwickelte System wird dabei im letzten Teil beschrieben.

3.1 Early Ray Termination

Eine ohne großen Aufwand zu implementierende Beschleunigungstechnik für Raycasting ist *early ray termination*. Sie basiert auf dem Prinzip, dass die interessanten Teile eines Datensatzes meistens relativ lichtundurchlässig sind. Infolgedessen erreicht die akkumulierte Opazität bei front-to-back-Traversierung des Volumens schon in den ersten Schichten einen Wert von fast 1 (was mit vollständiger Lichtundurchlässigkeit gleichzusetzen ist). Das bedeutet, dass dahinter liegende Voxel nicht oder nur kaum zum Volumenrenderingintegral beitragen und deshalb vernachlässigbar sind. Insofern kann jeder Sichtstrahl abgebrochen werden, sobald eine bestimmte Opazitätsgrenze überschritten ist.

Da die Opazität α beim front-to-back-compositing [s. (7)] bei jedem Schritt aktualisiert und abgespeichert wird, lässt sich early ray termination durch eine einfache *if*-Abfrage im Fragmentprogramm umsetzen. Übersteigt die Opazität den vorher festgelegten Schwellenwert, wird die Schleife durch eine *break*-Anweisung verlassen und der Farbwert für das entsprechende Pixel direkt ausgegeben. Eine solche Vorgehensweise muss allerdings durch die Grafikhardware unterstützt werden und ist z.B. bei Nvidia ab der nv40-Serie bzw. bei ATI ab r520 implementiert.

Der große Vorteil dieser Technik liegt in der leichten Realisierung ohne langwierige Vorverarbeitung. Auch in bestehende Implementierungen kann dieses Verfahren

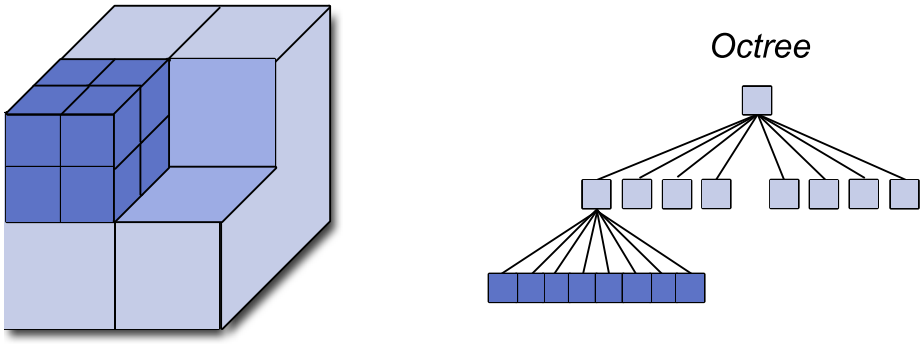

Abbildung 5: Zerlegung des Volumens und daraus resultierende Baumstruktur.

relativ einfach integriert werden. Der Geschwindigkeitszuwachs hängt jedoch in hohem Maße davon ab, wie viele lichtundurchlässige Voxel in der Szene vorkommen. Bei Datensätzen mit ausgedehnten Leerraumanteilen oder unter Verwendung einer stark transparenten Transferfunktion bringt dieser Ansatz daher kaum Vorteile. Des Weiteren werden Flusskontrollfunktionen in Shadern zwar inzwischen unterstützt, jedoch bringen diese noch große Leistungseinbußen mit sich, so dass sich die Verwendung von early ray termination auf der GPU sogar negativ auswirken kann.

3.2 Empty Space Skipping

Der Oberbegriff *empty space skipping* fasst eine Reihe von Verfahren zusammen, die es beim Volumenrendering ermöglichen, leere Bereiche[14] in der Szene zu erfassen und zu überspringen. Eine solche Vorgehensweise impliziert dabei die Verwendung einer zusätzlichen Datenstruktur, die in einem Vorverarbeitungsschritt unter Berücksichtigung des verwendeten Datensatzes angelegt wird.

3.2.1 Octree

In der Computergrafik werden häufig *Octree*-Datenstrukturen zur Beschleunigung von Raytracing hinzugezogen, ein Konzept, das auch im Volumenrendering eine wichtige Bedeutung erlangt hat. Hierbei wird der Datensatz rekursiv in jeweils 8 Teilvolumina gegliedert, bis eine bestimmte Grenze bzw. die unterste Ebene (d.h. Voxelebene) erreicht ist (s. Abb. 5). Diese räumliche Hierarchie wird in Form einer Baumstruktur, des Octree, abgebildet. In den Knoten werden dabei zusätzliche Informationen über die Unterbäume abgespeichert, die beispielsweise anzeigen, ob das darunterliegende Volumen leer ist oder nicht.

Eine solche Datenstruktur wird dann bei der Strahltraversierung genutzt, um zu entscheiden, ob ein Bereich keine wichtigen Daten mehr enthält und ausgelassen werden kann.

J. Krüger und R. Westermann [6] verwenden in ihrer GPU-Raycasting-Implementierung nur eine einzige Octree-Hierarchiestufe, was praktisch einem dreidi-

[14] Ein leerer Bereich im Kontext der Volumenvisualisierung enthält nur Voxel, deren Opazität gleich null ist.

mensionalen Raster mit einer konstanten Zellgröße entspricht. Für jeden Block der Größe 8^3 Voxel werden hier in einer RGB-3D-Textur der minimale und maximale vorkommende Skalarwert gespeichert (R- und G-Komponente). Darüber hinaus wird eine 2D-Textur angelegt, in der für alle solcher min-/max-Intervalle dokumentiert ist, ob sich nach Anwendung der Transferfunktion in dem entsprechenden Intervall ein sichtbarer Wert befindet.

Bei der Strahltraversierung können so leere Bereiche gezielt übersprungen werden, indem auf der Ebene des grob aufgelösten Rasters die min-/max-Werte abgefragt und als Index für die 2D-Textur verwendet werden. Diese gibt dann, wie beschrieben, endgültigen Aufschluss darüber, ob der betroffene Block ausgelassen werden kann.

Anhand einer Software-Implementation zeigt M. Levoy [7], dass der Octree-Ansatz grundsätzlich Geschwindigkeitssteigerungen um den Faktor 2 bis 5 zulässt. Der tatsächliche Zuwachs hängt allerdings stark von dem verwendeten Datensatz bzw. den Ausmaßen nicht sichtbarer Voxel ab.

Diese Ergebnisse lassen sich jedoch nicht ohne weiteres auf eine GPU-Realisierung übertragen. Auch aktuelle Grafikprozessoren stoßen bei der Traversierung einer Octree-Datenstruktur an ihre Grenzen, da hier eine Vielzahl von Abfragen im Shader nötig ist. Eine noch nicht effizient verwirklichte Flusskontrolle kann die gewonnene Renderzeit insofern schnell wieder zunichte machen. Außerdem muss der begrenzte Videospeicher der Grafikkarte berücksichtigt werden, da jede Hierarchieebene eine eigene 3D-Textur benötigt. Der Ansatz von Krüger und Westermann umgeht dieses Problem durch die Reduzierung auf nur eine Ebene, allerdings ohne dabei das volle Potential der Octree-Struktur auszureizen.

3.2.2 Bounding Volumes

Während das Volumen beim Octree-Ansatz rekursiv zerlegt wird, geht man bei Verwendung von *bounding volumes* vom Objekt, d.h. den tatsächlich sichtbaren Daten aus. Diese werden durch eine Geometrie umschlossen, um wichtige von leeren Bereichen zu trennen. Bei der Strahltraversierung werden dementsprechend die unwichtigen Teile übersprungen und nur innerhalb der bounding volumes weiter abgetastet.

H. Scharsach [14] implementiert diese Technik unter Ausnutzung der Vertexeinheit der Grafikkarte. Diese liegt normalerweise brach, da das eigentliche Raycasting in einem Fragmentprogramm stattfindet. Somit kann die bounding box, ohne Leistungseinbußen in Kauf zu nehmen, aus einer Vielzahl an Dreiecken aufgebaut werden und die Szene sehr genau umhüllen. Auf diese Weise kann die Anzahl an Freiräumen innerhalb dieser Grenzen auf ein Minimum reduziert werden. Der erforderliche zusätzliche Speicherplatzbedarf steigt zwar mit der Komplexität der umhüllenden Geometrie, jedoch ist dies im Vergleich zur Größe des Volumens meist vernachlässigbar.

Bounding volumes bieten eine gute Möglichkeit empty space skipping performant zu realisieren. Bei Datensätzen, die zum großen Teil klar abzugrenzende Objekte wiedergeben (z.B. Repräsentationen von technischen Bauteilen in synthetischen Datensätzen), lässt sich so eine noch wesentlichere Steigerung der Darstellungsgeschwindigkeit als unter Verwendung des Octree-Ansatzes erzielen.

Anders verhält es sich, wenn das Volumen viele halbtransparente Voxel enthält, die Daten verrauscht sind oder ungünstig verstreut liegen. Die Gefahr, hier wichtige Elemente zu übergehen, ist hoch, da klare Grenzen nur schwer zu definieren sind. Die Erstellung des bounding volumes erfordert zudem eine Vorverarbeitung, die auf der Analyse der Originaldaten beruht und deshalb zeitaufwendig ist.

3.3 Adaptives Sampling

Adaptives Sampling beschreibt beim Raycasting die Anpassung der Abtastrate des Volumens an die tatsächlich vorhandenen Daten. Das heißt, in Bereichen mit konstanter Dichte werden nur grobe Schrittweiten verwendet, wohingegen Teile, in denen sich die Werte stark ändern, sehr genau abgetastet werden. Das Ziel ist dabei die Beschleunigung des Renderings bei gleichbleibender Qualität der Bilder.

Das im Rahmen dieser Arbeit entwickelte System implementiert eine solche Vorgehensweise. Folgende Besonderheiten lassen diese Technik — auch im Vergleich zu den oben beschriebenen Methoden — vielversprechend erscheinen. Zum einen ist empty space skipping gewissermaßen ein Bestandteil von adaptivem Sampling bzw. kann durch dieses erreicht werden, da leere Bereiche gleichzeitig von hoher Homogenität sind und somit nur mit einer geringen Frequenz abgetastet werden. Darüber hinaus ist ein Gewinn nicht nur in solchen leeren Regionen zu erwarten, sondern allgemein in Gebieten relativ konstanter Werte, d.h. auch innerhalb von Objekten.

Da adaptives Sampling eine Datenstruktur erfordert, welche die Variation der im Volumen vorkommenden Werte kennzeichnet, ist eine Vorverarbeitung der Originaldaten nötig. Dabei wird eine Interessenkarte erstellt, in der sich die Komplexität der darzustellenden Szene widerspiegelt. Bei der Strahltraversierung wird die Schrittweite dann unter Zuhilfenahme dieser Struktur bei jedem Samplingschritt bestimmt.

Im Hinblick auf eine effiziente GPU-Implementierung stehen hier zwei wichtige Richtlinien im Vordergrund

- Minimaler zusätzlicher Aufwand pro Samplingschritt im Fragmentprogramm, da schon bei moderaten Auflösungen millionenfach abgetastet werden muss. Insbesondere die Vermeidung von Flusskontrolle, wie *if*-Abfragen, *breaks* etc., welche die Leistung merklich vermindern.
- Beschränkung des Speicherplatzes der Datenstruktur auf ein im Vergleich zu den Originaldaten des Volumens deutlich reduziertes Maß.

Diese Anforderungen erschweren die Verwirklichung einer hierarchischen Speicherung der Interessensinformationen. Ein Octree-Ansatz zur Realisierung von adaptivem Sampling, wie ihn J. Danskin und P. Hanrahan [1] beschreiben, ist daher auf der GPU nicht derart erfolgversprechend, wie es die Softwareimplementierung zeigt. Das liegt zum einen daran, dass jede Hierarchieebene eine eigene 3D-Textur benötigt, da die heutige Grafikkartengeneration hier nur feste Auflösungen und Farbtiefen unterstützt. Zum anderen ist das Durchlaufen des Octree mit einer Vielzahl von *if*-Abfragen bei jedem Samplingschritt verbunden, was einen Einbruch der Leistung erwarten lässt, der in keinem Verhältnis zum erzielten Gewinn steht.

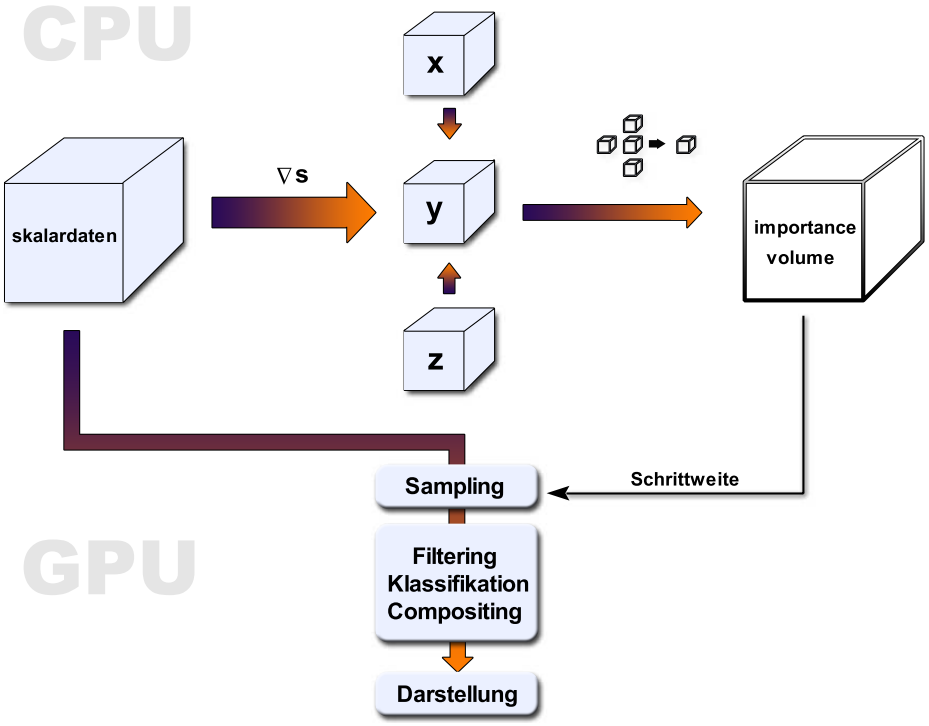

Abbildung 6: Das Importance Volume wird auf Basis der Originaldaten auf der CPU erstellt und liefert in der Samplingphase des Renderings für jeden Punkt die entsprechende Schrittweite.

3.3.1 Importance Volume

Roettger et al. [13] verwirklichen adaptives Sampling mittels eines sog. *Importance Volume* — einer 3D-Textur, die für jeden Punkt im Volumen die maximale isotrope Samplingdistanz angibt. Detailliertere Angaben, etwa wie dieses Volumen erstellt wird oder wie es genau aufgebaut ist, werden allerdings nicht gemacht.

Die Idee, eine dreidimensionale Textur zum direkten Zugriff auf Samplingfrequenzen zu verwenden, wird in dieser Arbeit aufgegriffen und bildet die Basis für die Verwaltung der in der Analyse gewonnenen Daten. Abb. 6 veranschaulicht den Ablauf von adaptivem Raycasting unter Verwendung eines Importance Volume.

Bei jedem Abtastschritt wird die 3D-Textur abgefragt und die Schrittweite für die aktuelle Position ausgelesen. Eine ausführliche Beschreibung von Erzeugung und Aufbau des Importance Volume wird in Abschnitt 5 im Rahmen der Implementierung gegeben.

3.3.2 Das Nyquist-Shannon-Theorem

Das Nyquist-Shannon-Theorem ist ein fundamentales Theorem der Signalverarbeitung und besagt, dass ein analoges, bandbegrenztes Signal bei der Digitalisierung mit einer Frequenz abgetastet werden muss, die größer als die doppelte vorkommende Frequenz des Eingangssignals ist, sofern man dieses später beliebig genau

rekonstruieren möchte. Betrachtet man ein Volumen als das Ergebnis der Abtastung eines kontinuierlichen Signals, so ist die höchste vorkommende Frequenz durch den abrupten Wechsel eines Wertes zwischen zwei benachbarten Voxeln gekennzeichnet [3]. Dies entspricht einer maximalen Frequenz von eins geteilt durch den Abstand der Voxel. Um Artefakte in der Darstellung zu vermeiden, muss daher generell mindestens zweimal pro Voxel abgetastet werden. Da beim adaptivem Sampling eine Homogenität in bestimmten Bereichen garantiert werden kann, wird das Nyquist-Shannon-Theorem auch bei einer niedrigeren Samplingrate nicht verletzt, da hier keine hohen Frequenzen vorkommen. Es ergibt sich allerdings ein Problem, sobald durch eine nichtlineare Transferfunktion hohe Frequenzen eingebracht werden. Regionen mit relativ konstanten Skalarwerten können so durch die Klassifikation stark heterogen werden. Wird die Abtastrate also nicht unter Berücksichtigung der Transferfunktion festgelegt, muss man mit Artefakten in den betreffenden Abschnitten rechnen. Eine solche Einbeziehung ist aber nicht unbedingt wünschenswert, da bei jeder Änderung der Transferfunktion das Importance Volume neu erstellt werden muss — ein Vorgang, der die Interaktion mit dem Volumen unakzeptabel verzögern kann. In dieser Arbeit wird deshalb untersucht, inwiefern eine Vorverarbeitung im Hinblick auf Qualität und Geschwindigkeit sinnvoll ist, die rein auf den Skalarwerten beruht.

3.3.3 Pre-Integration

Da adaptives Sampling die Verwendung von unterschiedlich langen Abtastintervallen beinhaltet, ist eine Anpassung der in Abschnitt 2.3 beschriebenen näherungsweisen Lösung des Volumenrenderingintegrals notwendig. Jedes Intervall muss vor dem Blending entsprechend gewichtet werden, um dickere von dünneren Schichten unterscheiden zu können.

Dies kann geschehen, indem der Opazitätswert für jeden Abschnitt in Bezug auf die minimale Schrittweite Δx_0 wie folgt neu berechnet wird [3]

$$\alpha_{corrected} = 1 - (1 - \alpha_{stored})^{\Delta x / \Delta x_0} \tag{8}$$

Eine andere Möglichkeit ist die von Engel et al. [2] entwickelte *Pre-Integration*, die auf der Vorabintegration aller theoretisch möglichen Intervalle beruht. Für jedes Skalarpaar s_{front} und s_{back} eines Abschnitts sowie für die jeweilige Länge d wird das Volumenrenderingintegral näherungsweise gelöst. Dies geschieht unter Annahme, dass das Skalarfeld zwischen s_{front} und s_{back} stückweise linear ist. Die berechneten Werte werden anschließend in einer Tabelle gespeichert und beim Raycasting abgerufen. Da die Transferfunktion schon bei der Erstellung dieser Tabelle im Rahmen der Vorintegration berücksichtigt wird, muss sie hier nicht mehr angewandt werden.

Die Opazität ergibt sich in Abhängigkeit von s_{front}, s_{back} und d durch

$$\alpha(s_{front}, s_{back}, d) \approx 1 - exp\left(-\frac{d}{s_{back} - s_{front}} \left(A(s_{back}) - A(s_{front})\right)\right) \tag{9}$$

mit der akkumulierten Abschwächung von 0 bis zum Skalarwert s:

$$A(s) := \int_0^s \kappa(s')ds'$$

Analog lässt sich die Farbe berechnen, wobei hier anzumerken ist, dass die Abschwächung innerhalb eines Strahlsegments vernachlässigt wird. Dies ist jedoch eine durchaus gebräuchliche Annäherung im Kontext der Postklassifikation und wirkt sich erst bei großen Produkten $\kappa(s)d$ aus [2].

$$C(s_{front}, s_{back}, d) \approx \frac{d}{s_{back} - s_{front}} \left(K(s_{back}) - K(s_{front})\right) \tag{10}$$

mit

$$K(s) := \int_0^s \kappa(s')c(s')ds'$$

$A(s)$ und $K(s)$ lassen sich sehr einfach berechnen, da die Skalarwerte meistens diskretisiert vorliegen und die Integrale damit zu einer Summe werden. Für den Fall einer 1 : 1-Abbildung durch die Transferfunktion ergibt sich für C gerade ein mit der Länge des Abschnitts skalierter Mittelwert.

Pre-Integration hat den Vorteil, dass durch die Vorabberechnung der Intervalle auch hohe Frequenzen, die durch die Transferfunktion verursacht werden können, mit berücksichtigt werden. So muss die Samplingrate in den betroffenen Bereichen nicht erhöht werden, obwohl es das Nyquist-Shannon-Theorem eigentlich fordert. Dies bedeutet aber auch eine Abhängigkeit der Lookup-Tabelle von der Transferfunktion, die so nicht wünschenswert ist.

4 Volumenanalyse

Wie in Abschnitt 3.3 beschrieben, benötigt die Realisierung von adaptivem Sampling einen Vorverarbeitungsschritt, in dem das Originalvolumen auf homogene Bereiche hin analysiert wird. Die Ergebnisse werden anschließend genutzt, um das Importance Volume zu erstellen, welches die Schrittweiten für das Raycasting speichert.

Dieses Kapitel gibt einen Überblick einiger für die Volumenvisualisierung wichtiger Verfahren zur Bestimmung von Variationen in Skalarfeldern. Der Schwerpunkt liegt hierbei auf Methoden, welche die räumlichen Ableitungen des Volumens approximieren, da solche Operationen relativ einfach durch Filter realisiert werden können.

4.1 Gradienten

In vielen Anwendungen der Volumenvisualisierung spielt die Berechnung der *Gradienten* eines Volumens eine überaus wichtige Rolle, da diese Voraussetzung für die meisten Segmentierungs- und Merkmalidentifikationsverfahren bzw. für die Oberflächennormalengewinnung sind. Der Gradient ist mathematisch wie folgt definiert [12]

Ist $f : \Re^3 \rightarrow \Re$ ein *Skalarfeld*, so heisst

$$\nabla f := (f_x, f_y, f_z)^T = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)^T \tag{11}$$

Gradient von f. Es ist $\nabla f : \Re^3 \rightarrow \Re^3$ ein *Vektorfeld*[15]. Sein Betrag ist definiert als

$$|\nabla f| := \sqrt{f_x^2 + f_y^2 + f_z^2} \tag{12}$$

Jedem Skalarfeld ist demnach ein Vektorfeld zugeordnet, welches das Maß und die Richtung der größten Änderung des Feldes beschreibt.

Ein Volumen kann im Kontext der Visualisierung als diskretes, dreidimensionales Skalarfeld angesehen werden. In diesem Fall ist der Gradient ein Vektor, der für jeden Punkt in Richtung des maximalen Anstiegs der Werte zeigt. Er wird laut Definition aus den partiellen Ableitungen bezüglich x,y und z gebildet. Man spricht deshalb auch von der *ersten Ableitung* des Skalarfeldes. Da an diskreten Stellen jedoch nicht differenziert werden kann, geschieht die Berechnung des Gradienten nur näherungsweise. Im Folgenden werden einige Verfahren beschrieben, die dies z.T. unter Verwendung von Filtermasken realisieren.

4.1.1 Zentraldifferenzen

Eine einfache Methode zur Berechnung des Gradienten an einem bestimmten Punkt ist die Bildung von *Zentraldifferenzen*. Dabei werden die partiellen Ableitungen in x-,y- und z-Richtung durch Bildung der Differenzen von einander gegenüberliegenden Voxeln auf den entsprechenden Achsen abgeschätzt.

[15] Vektorfelder sind Funktionen $\vec{v} : \Re^3 \rightarrow \Re^3$, also z.B. Geschwindigkeitsfelder, Gravitationsfelder.

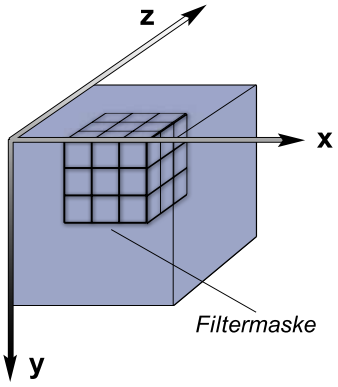

Abbildung 7: Orientierung der Achsen innerhalb des Volumens und zugehörige Filtermaske.

$$\nabla f(x,y,z) = \begin{pmatrix} f(x+1,y,z) - f(x-1,y,z) \\ f(x,y+1,z) - f(x,y-1,z) \\ f(x,y,z+1) - f(x,y,z-1) \end{pmatrix} x,y,z \in N \tag{13}$$

Pro Achse wird also nur ein Voxelpaar gleichgewichtet berücksichtigt, was auf der einen Seite der Laufzeit entgegenkommt, andererseits die Berechnung sehr rauschanfällig macht. Gerade im Hinblick auf MRT-Daten, die immer mit einer gewissen Varianz behaftet sind, scheint dieses Filter deshalb wenig geeignet für eine Homogenitätsanalyse. Eine Verbesserung stellt die Kombination mit einem glättenden Operator in Aussicht, wie der nächste Abschnitt zeigt.

4.1.2 3D-Sobel

Der aus der Bildverarbeitung bekannte *Sobel-Operator* verbindet die Annäherung der partiellen ersten Ableitungen mit einer leichten Glättung der Werte. Er eignet sich deshalb, im Vergleich zur Bildung von Zentraldifferenzen, besonders für die Detektion homogener Bereiche bzw. zur Verstärkung von Objektgrenzen [11].

Die Ableitungen in x-,y- und z-Richtung lassen sich durch drei $3 \times 3 \times 3$ Filtermasken berechnen. Je nach Gewichtung der umgebenden Voxel lässt sich dabei die Stärke der Glättung anpassen, was wiederum in verschiedenen Masken resultiert. (14) zeigt ein konkretes Filter für die Ableitung nach x und den Blick entlang der positiven z-Achse (Abb. 7).

$$\begin{bmatrix} -0.03 & 0 & 0.03 \\ -0.06 & 0 & 0.06 \\ -0.03 & 0 & 0.03 \end{bmatrix} \begin{bmatrix} -0.06 & 0 & 0.06 \\ -0.12 & 0 & 0.12 \\ -0.06 & 0 & 0.06 \end{bmatrix} \begin{bmatrix} -0.03 & 0 & 0.03 \\ -0.06 & 0 & 0.06 \\ -0.03 & 0 & 0.03 \end{bmatrix} \tag{14}$$

Die Ableitungen in y- und z-Richtung lassen sich entsprechend durch Rotation der Maske berechnen (s. Anhang A).

Der Sobel-Operator bietet eine gute Qualität im Hinblick auf Varianzdetektion bei gleichzeitig vernünftiger Performanz. Zwar werden insgesamt drei Filtermasken zur Berechnung der Ableitungen benötigt, allerdings können diese in einem Durchlauf des Volumens angewandt werden. Ein 3D-Sobel-Filter wird daher in dieser Arbeit im Rahmen der Vorverarbeitung zur Gradientenextraktion genutzt (s. 5.2).

4.2 Höhere Ableitungen

Neben den partiellen ersten Ableitungen eines Skalarfeldes werden in der Volumenvisualisierung häufig auch höhere Ableitungen, etwa zur Berechnung von Krümmungsinformationen, genutzt. Vor allem im nicht fotorealistischen Rendering spielt dies eine wichtige Rolle, da so die plakative Darstellung von Silhouetten oder Oberflächenstrukturen erleichtert wird. Schließlich können höhere Ableitungen auch zur detaillierten Bestimmung von lokalen Änderungen in den Volumendaten verwendet werden, was sie für eine Homogenitätsanalyse interessant erscheinen lässt.

4.2.1 Die Hesse-Matrix

Die Hesse-Matrix H eines Skalarvolumens f fasst die partiellen zweiten Ableitungen wie folgt zusammen, wobei aufgrund von Symmetrieeigenschaften die Berechnung von sechs Komponenten ausreichend ist.

$$H = \nabla g = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial y \partial z} \\ \frac{\partial^2 f}{\partial z \partial x} & \frac{\partial^2 f}{\partial z \partial y} & \frac{\partial^2 f}{\partial z^2} \end{pmatrix} \tag{15}$$

Kindlmann et al. [4] berechnen mithilfe von Hesse-Matrix und Gradient hochqualitative Krümmungsinformationen. Die nötigen partiellen Ableitungen aus g und H werden dabei durch Anwendung von Filterkernen berechnet, die aus Splines verschiedenen Grades und deren Ableitungen resultieren. Der Aufwand für diese Berechnungen ist, vor allem im Vergleich zu den relativ geringen Anforderungen der Gradientenmethoden aus den Abschnitten 4.1.1 und 4.1.2, beträchtlich. Die Kosten dieses Verfahrens stehen daher, im Hinblick auf die Charakterisierung homogener und varianzreicher Regionen, in keinem angemessenen Verhältnis.

4.2.2 Der Laplace-Operator

Der Laplace-Operator Δ ergibt sich aus der Summe der partiellen zweiten Ableitungen eines Skalarfeldes und entspricht damit der Spur der Hesse-Matrix [12]

$$\Delta f := \nabla \nabla f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} \tag{16}$$

Die zweiten Ableitungen in x-,y- und z-Richtung können so durch nur eine Filtermaske approximiert werden. (17) zeigt eine Beispielmaske für den zweidimensionalen Fall

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \Rightarrow \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{17}$$

Hauptnachteil des diskreten Laplace-Operators ist seine, im Vergleich zu den Differenzenoperatoren erster Ordnung, höhere Rauschanfälligkeit. Bei Betrachtung der Filtermaske wird schnell klar, dass einzelne Störpunkte gegenüber Segmentgrenzen unverhältnismäßig stark detektiert werden. Insofern ist der Laplace-Operator trotz guter Laufzeiteigenschaften im Kontext dieser Arbeit weniger erfolgversprechend als die in Kap. 4.1 beschriebenen Ansätze.

5 Implementierung

In diesem Kapitel wird das in dieser Arbeit entwickelte Konzept für adaptives Sampling erläutert und seine Integration in das Visualisierungssystem Galenus beschrieben. Wichtige Design-Entscheidungen der einzelnen Phasen werden dabei anhand der vorher festgelegten Ziele und Anforderungen begründet.

Als Erstes wird das Vorverarbeitungsmodul dargestellt, welches eine gradientenbasierte Analyse des Volumens verwirklicht und mittels dieser Informationen das Importance Volume aufbaut. Anschließend wird die Erweiterung des GPU-Raycasters um adaptives Sampling gezeigt.

5.1 Anforderungen

Grundlegende Anforderung an jedes Beschleunigungssystem ist zunächst eine merkliche Steigerung der Leistung gegenüber dem *brute-force*-Ansatz, da die meisten Verfahren eine Vorverarbeitung voraussetzen (s. Kap. 3) und der damit verbundene Mehraufwand gerechtfertigt werden muss. Darüber hinaus sollte der zu entwickelnde Ansatz keine fundamentalen Einschränkungen bezüglich der zu verwendenden Datensätze, der Hardware-Anforderungen oder Ähnlichem notwendig machen. Da adaptives Sampling auf der Beurteilung von Homogenität beruht, sind des Weiteren alle Entscheidungen zwischen Qualität und Geschwindigkeit flexibel, d.h. mithilfe von benutzergesteuerten Schwellenwerten zu realisieren.

5.2 Gradientenanalyse

In einem ersten Schritt müssen die Originaldaten des Volumens auf homogene Bereiche hin untersucht werden. Dies geschieht durch eine Gradientenanalyse unter Verwendung des in Kap. 4.1.2 beschriebenen dreidimensionalen Sobel-Operators, der sich durch seine geringe Rauschempfindlichkeit auszeichnet. Da die Vorverarbeitung ausschließlich auf der CPU läuft, kann hier zudem relativ großzügig vom Hauptspeicher Gebrauch gemacht werden, der im Vergleich zum Videoram durchschnittlich um mindestens den Faktor 4 größer ist.

Zunächst wird das Volumen mit der in Anhang A aufgeführten Maske gefaltet (s. Abb. 8). Die sich daraus ergebenden partiellen Ableitungen in x-, y- und z-Richtung werden nun in drei *float*-Arrays zwischengespeichert. Für eine Varianzanalyse ist nur der Betrag des Gradienten interessant, deshalb wird die Richtung im weiteren Verlauf nicht berücksichtigt.

5.3 Aufbau des Importance Volume

Aus den in der Gradientenanalyse gewonnenen Daten wird nun das Importance Volume aufgebaut, welches in Form einer 3D-Textur die isotropen Schrittweiten für jeden Samplepunkt innerhalb des Datensatzes beinhaltet. Oberste Priorität bei der Erstellung einer solchen Struktur haben die Faktoren Speicherplatz und Shaderkomplexität, die es zu minimieren gilt. Daher wird auf eine Abhängigkeit der Schrittweite von der Richtung des Sichtstrahls verzichtet. Eine solche Berücksichtigung würde zwar gegenüber einer isotropen Angabe die homogenen Bereiche noch genauer approximieren, allerdings auf Kosten weiterer Abfragen im Fragmentprogramm und eines erhöhten Speicherplatzbedarfs. Größtes Problem ist hierbei, dass die Schrittweite bei jedem Shaderdurchlauf in Abhängigkeit der Samplingposition

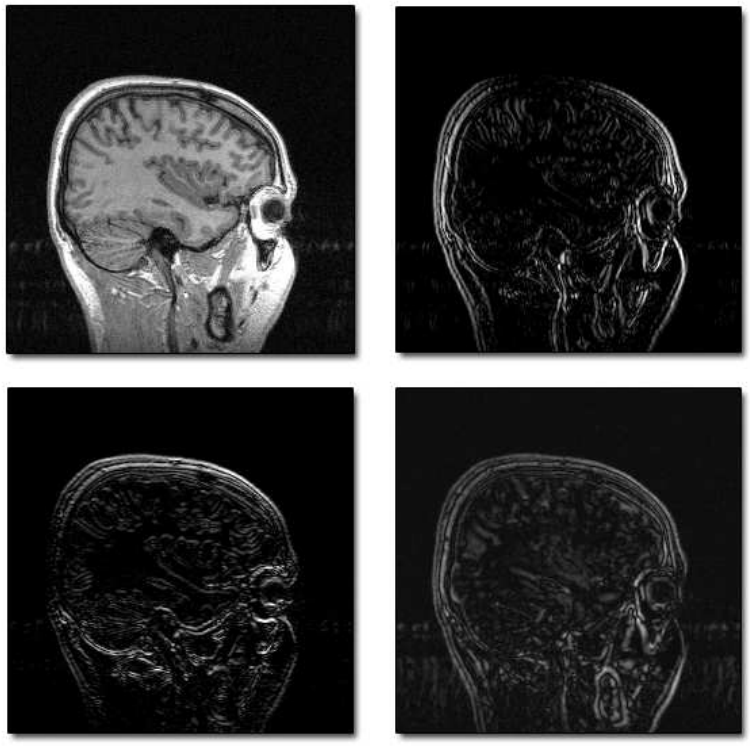

Abbildung 8: MRT-Datensatz und Gradienten. Originalbild (oben links) und die Ableitungen in x-,y- und z-Richtung.

und des Richtungsvektors neu berechnet werden müsste, um die in Abb. 9 gezeigten Fälle zu unterscheiden. Eine Steigerung der Performanz ist deshalb auf diese Weise nicht zu erwarten.

Das Importance Volume kann durch die Beschränkung auf isotrope Distanzen durch eine 8-Bit-Textur realisiert werden, da selbst bei umfangreichen Datensätzen kaum mit homogenen Regionen über einer Größe von 255^3 Voxeln zu rechnen ist. Selbst wenn dies der Fall wäre, macht es Sinn, die maximale Distanz auf einen niedrigeren Wert zu beschränken, wie der Erstellungsprozess im Folgenden noch zeigt.

Die Auflösung der Textur wird des Weiteren benutzergesteuert festgelegt, um eine Auswahl zwischen Erstellungsgeschwindigkeit bzw. Speicherplatz und Genauigkeit der Datenstruktur zur Verfügung zu stellen. In der Folge wird die Distanz nun in Blöcken $\#b$ angegeben, die, je nach Auflösung, das Volumen mehr oder weniger grob abdecken (s. Abb. 10). Die Schrittweite ergibt sich dabei durch Multiplikation von $\#b$ mit der Breite eines Blocks d. Tatsächlich variiert die Länge eines Strahlsegments innerhalb eines Blockes zwischen d und $\sqrt{3} \cdot d$, allerdings ist dies ein vernachlässigbarer Fehler und führt im schlimmsten Fall zu einigen zusätzlichen Samples in sehr großen Regionen. Idealerweise lässt sich die Länge mithilfe von d und dem Richtungsvektor des Sichtstrahls berechnen, was aber unverhältnismäßig aufwendig ist.

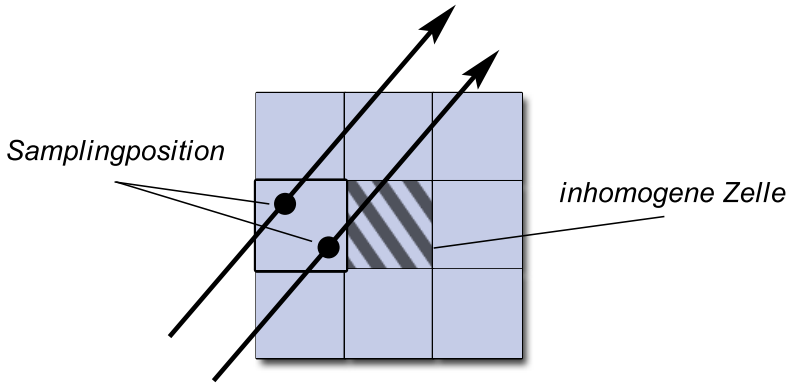

Abbildung 9: Die Samples zweier Sichtstrahlen fallen in dieselbe Importance-Volume-Zelle. Nur der rechte Strahl schneidet den benachbarten inhomogenen Block.

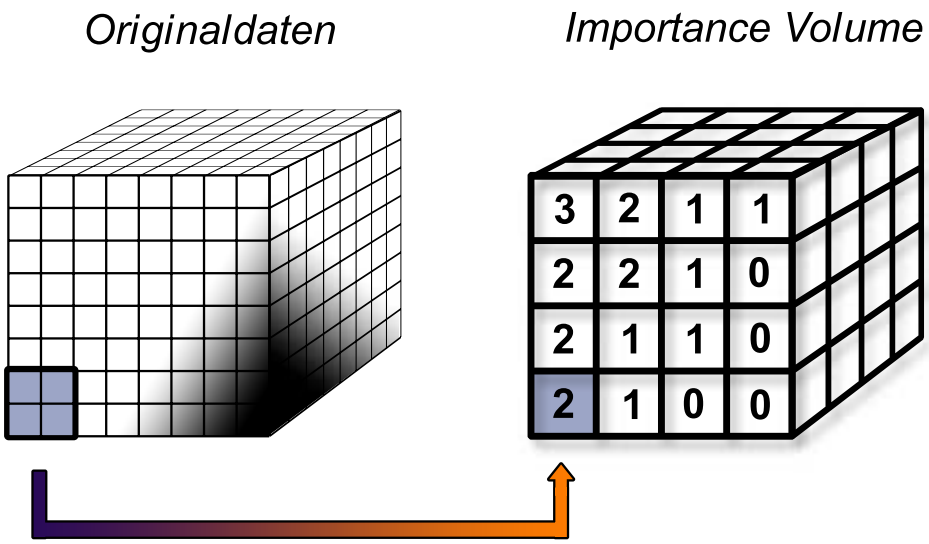

Abbildung 10: Das Importance Volume als Repräsentation des Originalvolumens. Jeder Eintrag gibt die Entfernung zu heterogenen Regionen in Blöcken an.

Vor der Erstellung des Importance Volume gibt der Benutzer zunächst die gewünschte Auflösung sowie einen Schwellenwert an, der die maximal zulässige Änderungsrate innerhalb eines homogenen Bereiches festlegt (s. Abb. 11). Daraufhin wird über die drei Arrays mit den Ableitungen des Volumens iteriert und der höchste vorkommende Gradient pro Importance-Volume-Block berechnet. Liegen alle Gradienten eines solchen Abschnitts unter dem Schwellenwert, so wird im y-Array an der entsprechenden Stelle der nun nicht mehr benötigte Wert mit einer 1 überschrieben. Für den gegenteiligen Fall wird hier eine 0 eingetragen.

Nun wird das Importance Volume als dreidimensionales Array mit der vom Benutzer vorgegebenen Auflösung erstellt und parallel zum y-Array durchlaufen. Jeder

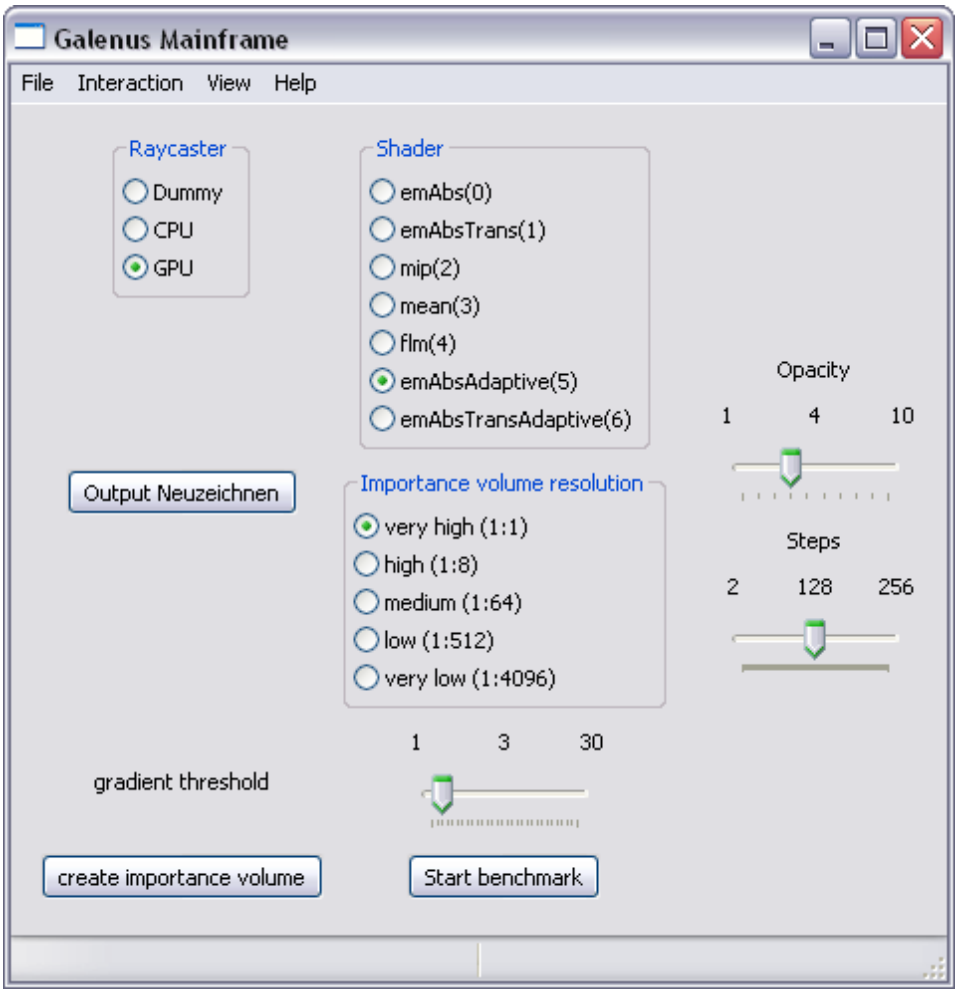

Abbildung 11: Galenus GUI mit adaptivem Sampling. Die Auflösung des Importance Volume kann von einer $1:1$-Abbildung (*very high*) bis zu $1/16^3$ des Originalvolumens (*very low*) reguliert werden. Unter *gradient threshold* wird der maximal zulässige Gradient innerhalb eines Blockes festgelegt.

Eintrag ergibt sich hier aus der Betrachtung der direkten Nachbarschaften. Zuerst wird der korrespondierende Eintrag im y-Array gesucht und auf die Markierung hin geprüft. Enthält dieser eine 0, so wird der Vorgang abgebrochen, eine Schrittweite von 0 eingetragen und mit der nächsten Stelle fortgefahren. Ist die Markierung dagegen eine 1, werden sukzessive alle Nachbarblöcke betrachtet, bis entweder eine 0 vorkommt oder ein Grenzwert an Iterationen erreicht ist. Der Abstand zur Ursprungsstelle wird dabei gespeichert und am Ende im Importance Volume vermerkt.

Die Anzahl an Tests für ein Volumen der Größe n^3, einer Blockgröße von d^3 Voxeln und der Tiefe der Nachbarschaftsuntersuchung T (= Distanz in Blöcken) ist ohne Randbehandlung[16] mindestens

$$\sum_{i=1}^{(n/d)^3} (2T_i + 1)^3 \cdot d^3 \tag{18}$$

Bei hohen Auflösungen kann dies die Berechnungszeit inakzeptabel steigern. Deshalb liegt hier die Implementierung eines Schwellenwertes nahe, welcher die Tiefe der Untersuchung für jeden Eintrag beschränkt. Eine Grenze von 40 hat sich im Laufe der Tests als ausreichend erwiesen. Bei den meisten Datensätzen wird dieser Wert ohnehin nie erreicht.

Das Importance Volume liegt nun als dreidimensionales Array vor und wird in eine 3D-Textur geschrieben, auf die der GPU-Raycaster Zugriff hat. Galenus benutzt OpenGL[17] als Application Programming Interface (API), daher geschieht die Erstellung der Textur durch die hier zur Verfügung gestellten Funktionen (Code 1).

Code 1: Erzeugung Importance Volume Textur

```
1   // generate texture names
2   glGenTextures(1, &importanceVolumeTextureNames);
3   glBindTexture(GL_TEXTURE_3D,
4                 importanceVolumeTextureNames);
5   glTexParameteri(GL_TEXTURE_3D,
6                   GL_TEXTURE_MIN_FILTER,
7                   GL_NEAREST); // select nearest voxel
8   glTexParameteri(GL_TEXTURE_3D,
9                   GL_TEXTURE_MAG_FILTER,
10                  GL_NEAREST); // select nearest voxel
11  glTexParameteri(GL_TEXTURE_3D,
12                  GL_TEXTURE_WRAP_R, GL_CLAMP);
13  glTexParameteri(GL_TEXTURE_3D,
14                  GL_TEXTURE_WRAP_S, GL_CLAMP);
15  glTexParameteri(GL_TEXTURE_3D,
16                  GL_TEXTURE_WRAP_T, GL_CLAMP);
```

[16] Voxel, die außerhalb des Volumens liegen, müssen nicht mit berücksichtigt werden. Je nach Tiefe der Nachbarschaftsuntersuchung reduziert sich daher die Anzahl an Berechnungen.

[17] Die Open Graphics Library (OpenGL) ist eine plattformunabhängige 2D- und 3D-Grafik-API.

```
17        glTexImage3D(GL_TEXTURE_3D, 0, GL_LUMINANCE,
18                     width, height, depth, 0,
19                     GL_LUMINANCE, GL_UNSIGNED_BYTE,
20                     importanceVolumeVector);
```

Mit *glTexImage3D* wird aus den in *importanceVolumeVector* angegebenen *unsigned char* Werten eine 3D-Textur erstellt, die jedes Element als einzelnen 8-Bit-Helligkeitswert speichert. Zur Auswahl des Importance-Volume-Blocks, welcher der Samplingposition am nächsten liegt, wird für die Interpolation GL_NEAREST festgelegt.

5.4 Galenus Raycaster

Galenus ist ein Visualisierungssystem, das in der Lage ist, Datensätze aus .pgm- und DICOM[18]-Schichtbildern bzw. aus dem .raw-Format zu verarbeiten und auf Pixel-Shadermodel-3.0-kompatibler[19] Grafikhardware zu rendern. Es stehen verschiedene Darstellungsmöglichkeiten zur Verfügung, darunter eine Visualisierung anhand des Emissions-Absorptionsmodells.

5.4.1 Shader

Um adaptives Sampling auf der GPU zu verwirklichen, müssen die Pixelshader, die das Raycasting implementieren, entsprechend angepasst werden. Code 2 zeigt das Fragmentprogramm für 8-Bit-Datensätze und Emissions-Absorptionsmodell.

Code 2: Emissions-Absorptionsshader (8-Bit)

```
1  void main(float2 texCoords2D : TEXCOORD0,
2                uniform sampler2D frontTexture,
3                uniform sampler2D backTexture,
4                uniform sampler3D volumeTexture,
5                uniform sampler3D importanceVolumeTexture,
6                uniform float4 all,
7                out float4 outColor : COLOR)
8  {
9        float resolution = all.x;
10       float opacityfac = all.y;
11       float3 frontColor= tex2D(frontTexture, texCoords2D).rgb;
12       float3 backColor = tex2D(backTexture,  texCoords2D).rgb;
13       float3 direction = normalize(backColor - frontColor);
14       float m_output = 0.0;
15       float textureValue;
```

[18]DICOM (Digital Imaging and Communication in Medicine) ist ein Standard zur Speicherung und zum Austausch radiologischer Daten.

[19]Galenus läuft prinzipiell auch auf Grafikkarten, die das Shadermodel lediglich in der Version 2.0 unterstützen. Allerdings muss die Schrittweite dann sehr grob gewählt werden, da hier nur eine eingeschränkte Menge an Instruktionen zur Verfügung steht. Für adaptives Sampling ist sogar zwingend Shadermodel 3.0 erforderlich. Die Variabilität der Abtastrate bewirkt hier, dass die Anzahl an Schleifendurchläufen nicht von vornherein feststeht. Dies ist in der Spezifikation von SM 2.0 nicht vorgesehen.

```
16        float opacity;
17        float importanceVolumeDistance;
18        float dirlength = length(backColor - frontColor);
19        float stepsize;
20        float d;
21
22        for (dirlength; dirlength > 0.0; ){
23           textureValue = tex3D(volumeTexture,  frontColor).x;
24           //weight opacity
25           opacity = (textureValue/opacityfac);
26           //importance volume lookup
27           importanceVolumeDistance =
28                tex3D(importanceVolumeTexture, frontColor).r;
29           // rescale
30           importanceVolumeDistance *= 255;
31           // set absolute distance
32           d = resolution * importanceVolumeDistance;
33           // set next stepsize
34           stepsize = 0.01 + 0.01 * d;
35           // integrate constant part
36           textureValue *= (d+1);
37           //raycast
38           m_output = (opacity * textureValue +
39                      (1.0 - opacity)*m_output);
40           // adjust ray length
41           dirlength -= stepsize;
42           //next position
43           frontColor = frontColor + (stepsize * direction);
44        }
45        float3 outVec = (float3(m_output, m_output, m_output));
46        outColor = float4(outVec, 1.0);
47  }
```

Der Unterschied zum Raycastingshader ohne adaptives Sampling besteht zunächst in der Übergabe einer weiteren Textur, die das Importance Volume enthält (*importanceVolumeTexture*, Zeile 5). Zusätzlich wird deren Auflösung im Vergleich zum Originalvolumen in der ersten Komponente des Vektors *all* angegeben (Zeile 9). Die Traversierung des Sichtstrahls erfolgt nun in der *for*-Schleife des Programms. Nach der Bestimmung von Farb- und Opazitätswert wird die Schrittweite als Blockanzahl aus dem Importance Volume gelesen und anschließend auf einen Bereich zwischen 0 und 255 skaliert. Die Multiplikation mit der Auflösung und der minimalen Samplingdistanz ergibt schließlich die Entfernung zum nächsten Sample.

Nun wird der Farbwert mit der Länge des Abtastintervalls unter Vernachlässigung der Abschwächung verrechnet und in back-to-front-Reihenfolge mit der bisher akkumulierten Farbe durch Blending vermengt. Danach muss der Strahl um die entsprechende Schrittweite verkürzt und die nächste Position in den Vektor *frontColor* geschrieben werden.

Für nichtlineare Transferfunktionen reicht die hier gezeigte einfache Gewichtung nicht aus, sondern sollte, wie in Kapitel 3.3.3 beschrieben, durch Pre-Integration der einzelnen Intervalle gelöst werden (eine entsprechende Erweiterung des Shaders durch Pre-Integration findet sich in Anhang B). Die Laufzeiten der beiden Verfahren

variieren allerdings nur marginal und die prozentualen Geschwindigkeitszuwächse sind, wie es bei einer Änderung pro Sample zu erwarten ist, vergleichbar. Daher beruhen die in Abschnitt 6 dokumentierten Messungen auf dem bereits beschriebenen Code.

Neben dem oben genannten Fragmentprogramm wurden noch weitere Shader aus dem Galenus-System angepasst, darunter der Code für die Visualisierung von 16-Bit-Daten mit und ohne Transferfunktion. Die für das adaptive Sampling wichtigen Teile sind jedoch identisch mit der in Code 2 gezeigten Vorgehensweise, deshalb werden sie an dieser Stelle nicht aufgeführt.

6 Ergebnisse

In diesem Kapitel werden die Ergebnisse des adaptiven GPU-Raycasters hinsichtlich Qualität und Leistung vorgestellt. Dabei kommen unterschiedliche Datensätze aus bildgebenden Modalitäten wie MRT und CT zum Einsatz:

- *KopfRed*:
  Teilaufnahme (MRT) eines Kopfes mit $256^2 \cdot 38$ Voxeln und 8-Bit Format.
- *Kopf*:
  MRT-Aufnahme der Größe 256^3 und 8-Bit Format.
- *Engine*:
  CT-Aufnahme eines Motorblocks mit $256^2 * 128$ Elementen und 8-Bit Format [15].
- *Aneurysma*:
  C-Arm-Röntgenaufnahme von erweiterten ateriellen Blutgefäßen der rechten Kopfhälfte. Auflösung: 256^3 / 8-Bit [15].

6.1 Performanz

Um eine angemessene Beurteilung des in dieser Arbeit entwickelten Verfahrens gewährleisten zu können, werden alle Zeitmessungen anhand einer über 100 Frames umfassenden Sequenz durchgeführt. Es werden dabei zwei verschiedene Testrechner verwendet, die sich insbesondere hinsichtlich der Grafikkarte unterscheiden. Die Flusskontrolle ist beispielsweise beim 7800-GTX-Renderer etwas effizienter implementiert, als es bei der 6800-Serie der Fall ist. Daher ist hier auch eine etwas deutlichere Leistungssteigerung zu erwarten.

Folgende Testrechner kamen zum Einsatz

- *Bob*
  Prozessor: AMD Athlon 64 X2 Dual Core Processor 4400+
  Arbeitsspeicher: 2048 MB RAM
  Grafik: NVIDIA GeForce 7800 GTX SLI
  Video Speicher: 256.0 MB
  Betriebssystem: Microsoft Windows XP Professional
- *Fel*
  Prozessor: Pentium 4 3.0 GHz 800 MHz fsb
  Arbeitsspeicher: 1024 MB RAM
  Grafik: NVIDIA GeForce 6800 GT
  Video Speicher: 256.0 MB
  Betriebssystem: Microsoft Windows XP Home

Die folgenden Abbildungen stellen die Performanz in Bildern pro Sekunde der Berechnungszeit für das Importance Volume in Sekunden gegenüber. Die verwendeten Auflösungen variieren dabei zwischen einem $1:1$ (very high)- bis $1:16^3$ (very low)-Verhältnis von Originaldaten und Importance Volume.

Jedes Diagrammpaar zeigt die Geschwindigkeit für die entsprechende Importance-Volume-Auflösung und den eingestellten Gradientenschwellenwert sowie die für die Erstellung des Importance Volume auf der CPU dazugehörige Zeit, die ohne die Dauer für die Bildung der Gradienten mittels 3D-Sobel-Filterung zu verstehen ist. Je nach Datensatz und Testrechner erhöht sich so die Phase für die Vorverarbeitung um einen fixen Wert. Da diese Angaben unabhängig von der gewählten Auflösung des Importance Volume sind und noch Spielraum für Optimierungen bieten, werden sie an dieser Stelle getrennt von den übrigen Ergebnissen aufgeführt.

- *Bob*
  Aneurysma: +48s.
  Engine: +24s.
  KopfRed: +6s.
- *Fel*
  Kopf: +57s.

Bei den Geschwindigkeitsdiagrammen ist zu beachten, dass sich die Messungen für jeden Datensatz an einem Richtwert orientieren, der durch einen Shader ohne Implementierung von adaptivem Sampling entstanden ist. Dieser Wert ist in den Abbildungen als horizontale Linie eingetragen.

6.1.1 *Kopf*-Datensatz auf Rechner *Fel*

Abb. 12 und 13 zeigen die Ergebnisse für den *Kopf*-Datensatz auf einer GeForce 6800 GT.

Zunächst fällt auf, dass adaptives Sampling unter allen Importance-Volume-Auflösungen langsamer als der herkömmliche Shader ist, sofern der Gradientenschwellenwert niedriger als 5 ist. Dies ist auf die ineffiziente Flusskontrolle zurückzuführen, welche die Leistung auf der 6800 GT um knapp 10% verringert. Da der MRT-Datensatz ein nicht zu vernachlässigendes Rauschen beinhaltet, kann dieser Nachteil bei einem relativ geringen Schwellenwert noch nicht durch adaptives Sampling ausgeglichen werden. Erhöht man jedoch die Toleranz, sind deutliche Leistungssteigerungen erkennbar. Diese betragen bis zu 23% unter *very high*, 48% bei *high*, 45% unter *medium*, 30% bei *low* sowie 18% unter *very low*. Es muss dabei vermerkt werden, dass der Gradientenschwellenwert bei kleinen Blockgrößen des Importance Volume nicht beliebig gesteigert werden kann. Die Berechnungszeit für eine $1:1$-Abbildung von Voxeln und Importance-Volume-Zellen zeigt bei linearer Erhöhung des Grenzwertes eine exponentielle Zunahme. Deshalb wird unter *very high* schon ab einer Schwelle von 6 abgebrochen, was einer Rechendauer von 280 Sekunden entspricht. Dem gegenüber bleibt der Zuwachs bei einer *medium*-Auflösung selbst bei einem Grenzwert von 15 unter 4 Sekunden. Unter *very low* sind die Importance-Volume-Blöcke schließlich so groß, dass kaum ein Zeitunterschied bei Veränderung des *gradient threshold* festzustellen ist.

6.1.2 *KopfRed*-Datensatz auf Rechner *Bob*

Abb. 14 sowie 15 zeigen die Ergebnisse des reduzierten Kopf-Datensatzes auf einem GeForce-7800-GTX-SLI-Verbund.

Ähnlich wie bei der vorherigen Messung bleibt die Geschwindigkeit bei adaptivem Sampling zunächst hinter dem Richtwert zurück, auch wenn der Vorsprung hier etwas geringer ausfällt. Bei sukzessiver Erhöhung des Schwellenwertes sind jedoch drastische Geschwindigkeitszuwächse in einer Größenordnung von bis zu $2,87$ unter *medium* und noch $1,61$ unter *very low* zu verzeichnen. Dies ist vor allem damit zu erklären, dass der Datensatz prozentual mehr Freiräume enthält, als es bei dem oben gezeigten Beispiel der Fall ist.

Die Berechnungszeit für das Importance Volume zeigt auch bei dieser Messung einen exponentiellen Zuwachs bei hohen Auflösungen, bzw. ein nahezu konstantes Verhalten bei großen Blöcken, d.h. geringen Auflösungen.

6.1.3 *Engine*-Datensatz auf Rechner *Bob*

Die Ergebnisse für den *Engine*-Datensatz werden in Abb. 16 vorgestellt. Im Unterschied zu den in den Abschnitten 6.1.1 und 6.1.2 aufgeführten Volumina handelt es sich hierbei jedoch um eine CT-Aufnahme. Dies hat spürbare Auswirkungen auf die Verwendung von adaptivem Sampling, da solche CT-Datensätze eine deutlich geringere Rauschbelastung aufweisen. Aus diesem Grund liegt die Rechenzeit unter *very high* schon bei einem Schwellenwert von 1 deutlich über 400 Sekunden, unter *high* sind es noch 336 Sekunden. Erst ab einer Blockgröße von 4^3 (*medium*) sinkt sie auch bei höheren Grenzwerten unter 10 Sekunden. Dies kann damit erklärt werden, dass große Bereiche des Volumens praktisch keine Varianz aufweisen und sich somit sehr große Schrittweiten ergeben. Ist die Auflösung des Importance Volume hier zu hoch, steigt der Aufwand für jeden Block nichtlinear an.

Ein ähnliches Bild zeigen die Geschwindigkeitsmessungen, die von Anfang an eindeutig über dem Richtwert liegen. Der Abstand zwischen varianzreichen und homogenen Regionen ist hier offenbar so hoch, dass eine klare Trennung schon bei sehr niedrigen Gradientenschwellen möglich ist. Ab einem Wert von 4 führt eine weitere Steigerung daher kaum noch zu mehr Leistung.

Insgesamt kann die Darstellungsrate bei diesem Datensatz um einen Faktor von bis zu $2,98$ unter *medium* und noch $1,84$ unter *very low* verbessert werden.

6.1.4 *Aneurysma*-Datensatz auf Rechner *Bob*

In Abb. 17 sind die Ergebnisse für den *Aneurysma*-Datensatz dokumentiert. Auch hier handelt es sich um eine CT-Aufnahme, wobei sich das feingliedrige Aderngeflecht im Vergleich zum Motorblock sehr viel weniger vom Hintergrund abhebt.

Die Messungen zeigen bei einer linearen Erhöhung des Schwellenwertes einen nahezu proportional verlaufenden Anstieg der Geschwindigkeit und der Berechnungszeit. Dies bestätigt, dass die Varianz in diesem Datensatz ebenfalls etwa linear verteilt ist. Die Leistungsteigerung erreicht bei einem Grenzwert von 15 einen Faktor von $3,77$, was dem höchsten Zuwachs von allen getesteten Datensätzen entspricht. Allerdings ist zu erwarten, dass dies mit einer kontinuierlichen Verschlechterung der Bildqualität bzw. dem Wegfall von Details einhergeht, da hier, im Gegensatz zu den anderen Tests, keine Grenze zu erkennen ist, die einen Wechsel von Hintergrundvarianz zu den wichtigen Daten andeutet.

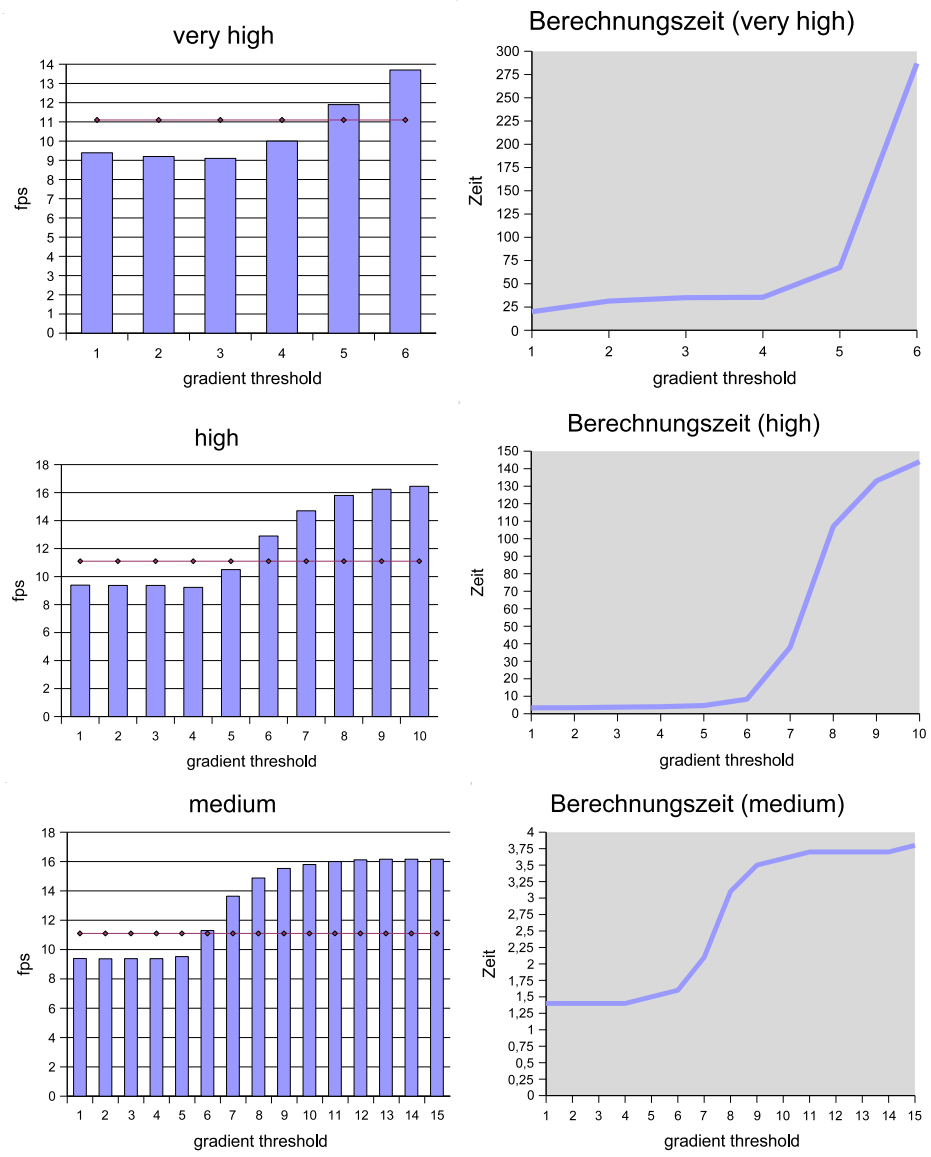

Abbildung 12: *Kopf*-Datensatz auf Rechner *Fel*. Die horizontale Linie zeigt den Richtwert eines Shaders ohne adaptives Sampling (Abbildungen links).

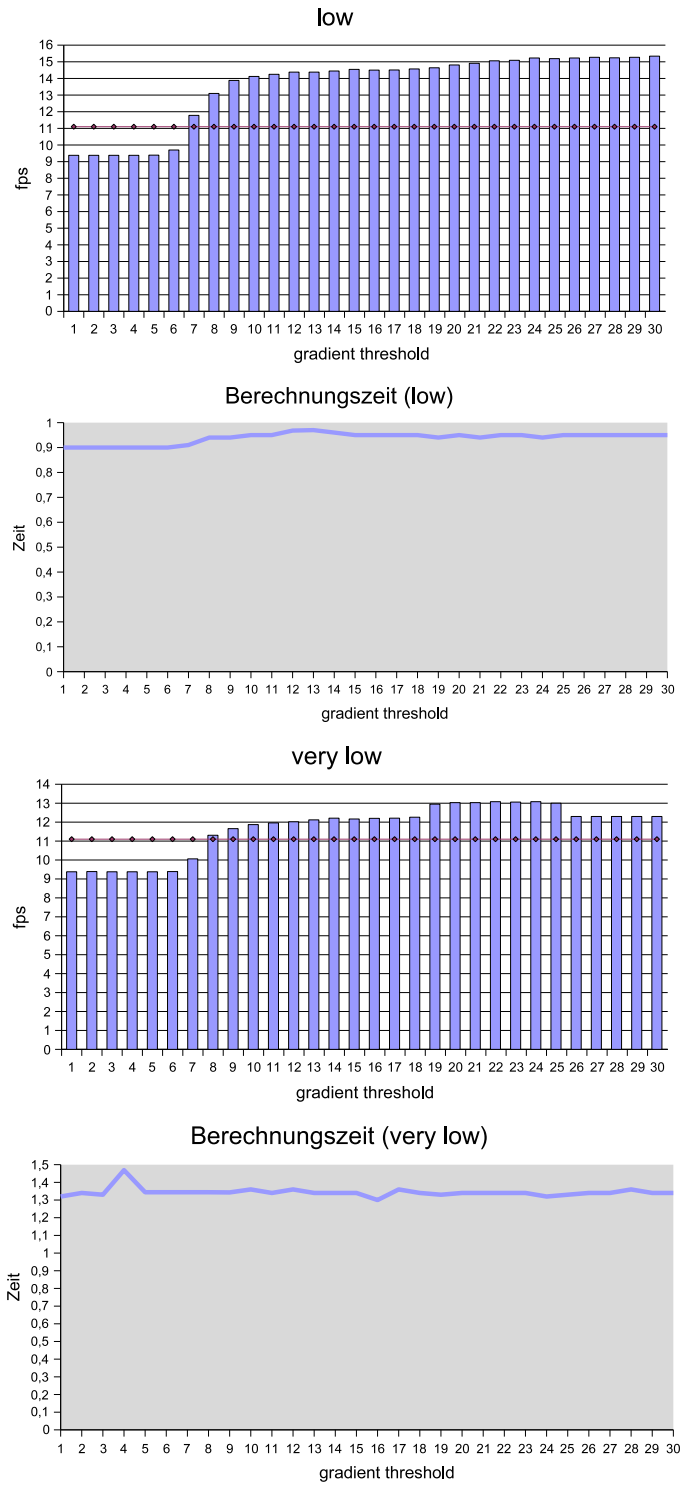

Abbildung 13: *Kopf*-Datensatz auf Rechner *Fel*. Low und very low.

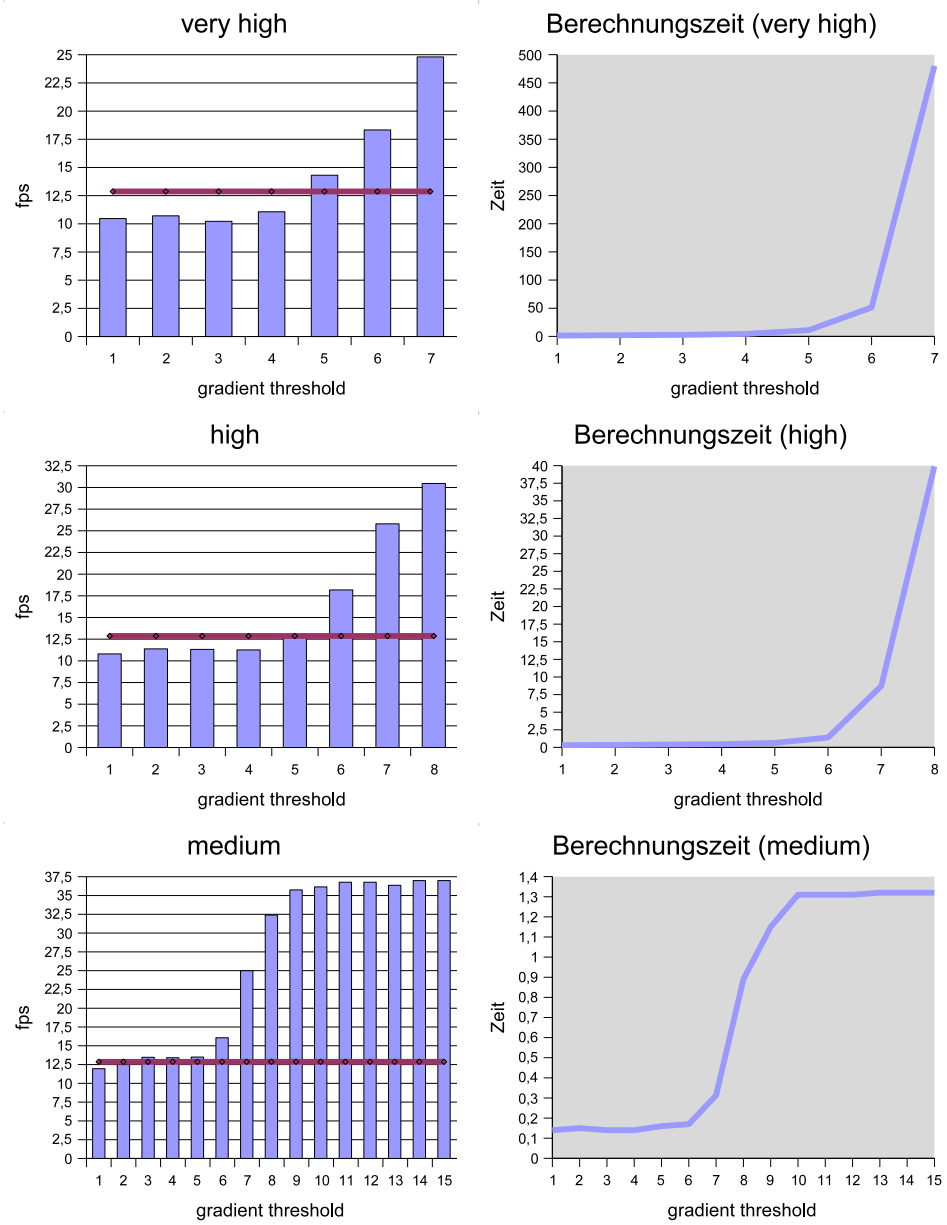

Abbildung 14: *KopfRed*-Datensatz auf Rechner *Bob*. Very high bis medium.

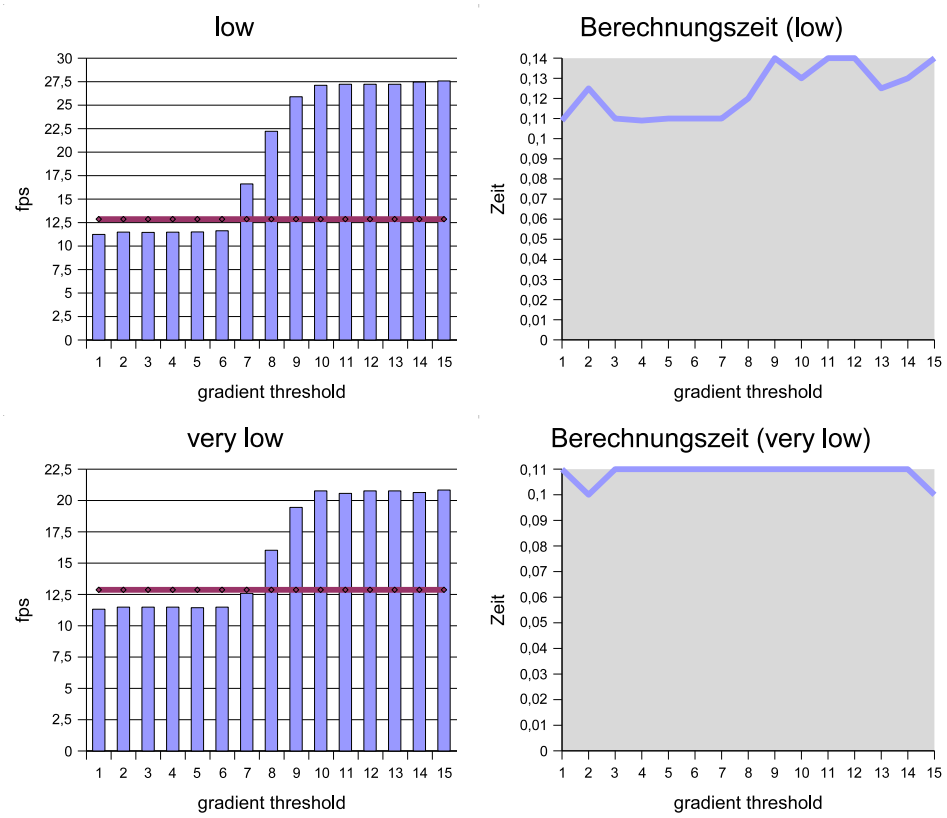

Abbildung 15: *KopfRed*-Datensatz auf Rechner *Bob*. Low und very low.

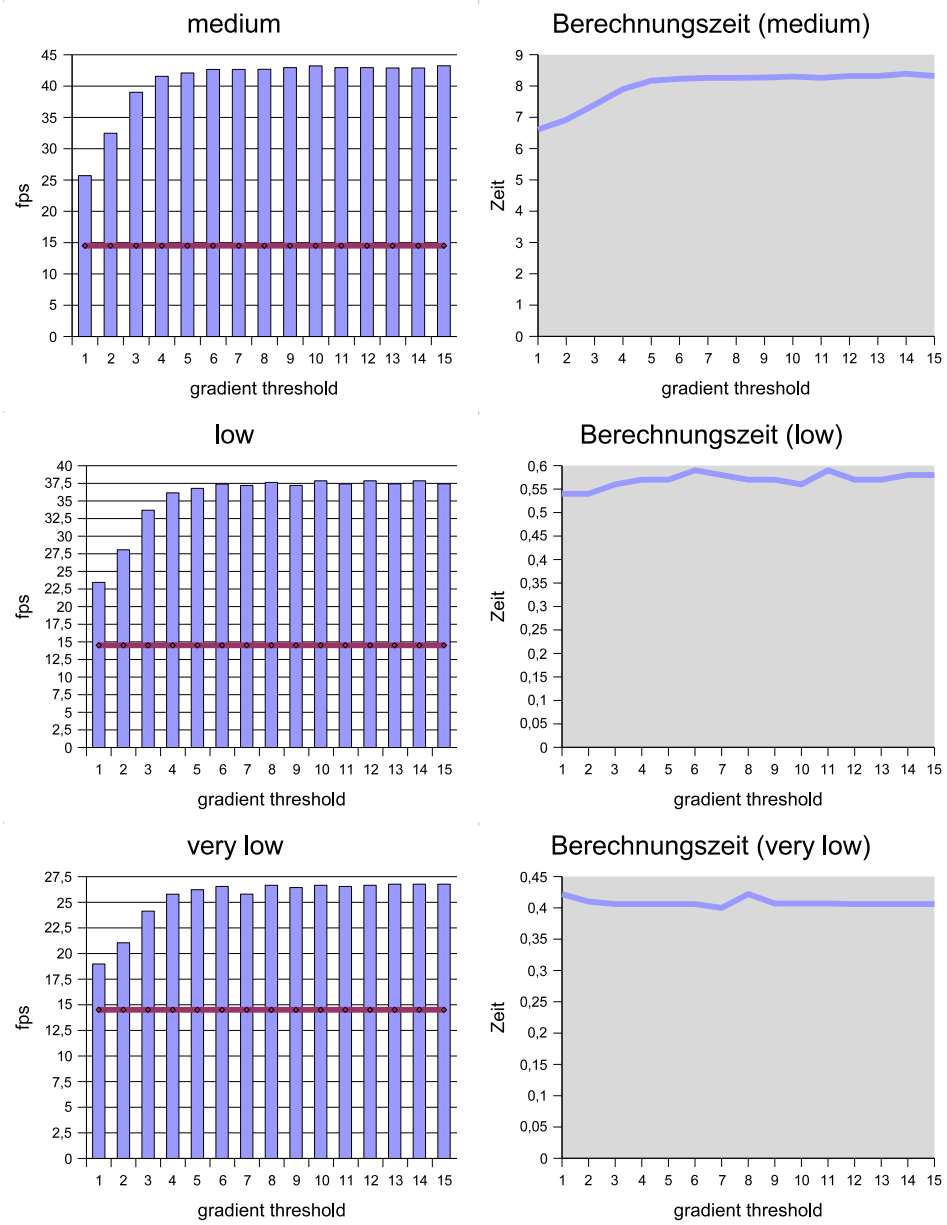

Abbildung 16: *Engine*-Datensatz auf Rechner *Bob*. Medium bis very low.

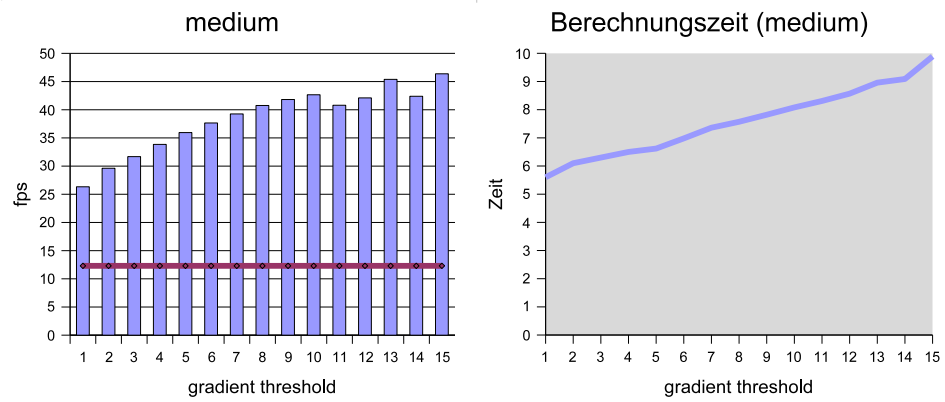

Abbildung 17: *Aneurysma*-Datensatz auf Rechner *Bob*. Auflösung: Medium.

6.2 Qualität

Die Bewertung der Qualität erfolgt in diesem Abschnitt mittels der Bildung von Differenzbildern zwischen den Darstellungen mit und ohne adaptives Sampling. Es ist dabei jeweils angegeben, welche Blockgröße im Importance Volume verwendet wird bzw. welcher Gradientenschwellenwert zum Einsatz kommt.

Um eine visuelle Beurteilung auch bei sehr geringen Unterschieden zu ermöglichen, wird der Kontrast bei den Differenzabbildungen z.T. stark erhöht. Zusätzlich wird ein Verlauf von Weiß nach Schwarz eingeblendet, der in einem unveränderten Bild linear ist und sich unter Erhöhung des Kontrastes entsprechend verschiebt. Auf diese Weise kann das Maß der Verfremdung anschaulich nachvollzogen werden.

6.2.1 *Engine*-Datensatz

Abb. 18 zeigt den *Engine*-Datensatz mit und ohne adaptives Sampling. Dabei stellt sich heraus, dass bei einer *medium*-Blockgröße selbst bei einem Schwellenwert von 10 keinerlei Artefakte zu erkennen sind. Die Messungen zeigen hier ab einem Wert von 5 praktisch keine Veränderung in der Darstellungsgeschwindigkeit und Berechnungszeit. Dies lässt auf eine Konstanz des Importance Volume und damit der Qualität ab dieser Schwelle schließen.

Der visuelle Eindruck bestätigt sich bei Betrachtung der Differenzbilder für die Grenzen 3 und 5. Nur unter äußerster Steigerung des Kontrastes sind hier Unterschiede zum Original zu erkennen. Analog verhält es sich für die verbleibenden Auflösungen, die auch keine sichtbaren Veränderungen zur Folge haben. Der *Engine*-Datensatz liefert damit die visuell überzeugendsten Ergebnisse der Testreihen.

6.2.2 *KopfRed*-Datensatz

Der *KopfRed*-Datensatz weist im Vergleich zu den CT-Aufnahmen ein höheres Hintergrundrauschen auf, das sich auch in den visuellen Ergebnissen niederschlägt. So macht sich mit zunehmendem Schwellenwert schrittweise eine Blockbildung bemerkbar, die vor allem an nichtabrupten Übergängen auffällt (s. Abb. 19). Diese Artefakte sind bei höheren Auflösungen des Importance Volume generell weniger präsent, da sich die Schrittweite hier genauer an den Originaldaten orientieren kann.

Bei sehr großen Blöcken (*low* und *very low*) kommt es meist erst bei hohen Schwellenwerten zu Störungen, wobei diese dann sehr deutlich sichtbar sind. Dies ist darauf zurückzuführen, dass die Varianz innerhalb eines Blockes fast immer über dem Grenzwert liegt und somit die minimale Schrittweite verwendet wird.

6.2.3 *Kopf*-Datensatz

Die Darstellung des *Kopf*-Datensatzes verhält sich anlalog zum *KopfRed*-Volumen. Bei hohen Schwellenwerten zeigen sich Blöcke an Übergängen, die nur unzureichend durch das Importance Volume abgebildet werden (s. Abb. 20).

Um eine zufriedenstellende Qualität zu erreichen, ist eine Orientierung an den Messergebnissen aus Kap. 6.1 sinnvoll. Nimmt man die Stelle, ab der sich die Darstellungsgeschwindigkeit nicht mehr wesentlich erhöht, als Maß für den Grenzwert, so erzielt man meistens ansprechende Ergebnisse. Dies kann damit erklärt werden, dass zunächst homogene und varianzreiche Regionen relativ klar voneinander getrennt sind. Eine weitere Steigerung des Schwellenwertes macht sich dann nur noch unwesentlich in der Leistung bemerkbar, bzw. führt irgendwann zu einer Zusammenfassung von Bereichen mit kontinuierlichen Übergängen.

6.2.4 *Aneurysma*-Datensatz

Die Performanzmessungen des *Aneurysma*-Datensatzes haben bereits angedeutet, dass mit steigendem Grenzwert das Verschwinden von Details zu befürchten ist. Abb. 21 bestätigt diese Erwartung. Während sich der visuelle Unterschied bei einer Schwelle von 1 noch in Grenzen hält, werden bei einer Erhöhung des Wertes auf 5 ganze Gefäßäste ignoriert.

6.3 Zusammenfassung

In den Testreihen wurde deutlich, dass die Wahl des richtigen Schwellenwertes von entscheidender Bedeutung sowohl für die Geschwindigkeit als auch für die Qualität ist. Eine Grenze von 3 bei CT-Datensätzen bzw. 10 bei den MRT-Aufnahmen hat sich dabei als angemessen erwiesen. Will man blockbildende Artefakte gänzlich ausschließen, so sollte zudem eine hohe Auflösung des Importance Volume gewählt werden.

Den besten Kompromiss stellt aber in allen Beispielen die *medium*-Auflösung dar, die fast immer die höchsten Frameraten erzielt und auch eine überzeugende Qualität liefert, sofern der Grenzwert nicht zu hoch ist. Im Vergleich zu den *high*- und *very-high*-Blockgrößen steigt unter diesen Bedingungen außerdem die Berechnungsdauer für das Importance Volume nicht in einem exponentiellen Verlauf an. Des Weiteren beträgt der zusätzlich belegte Speicherplatz nur 1/64 gegenüber einem Volumen mit 8-Bit-Format und entsprechend weniger bei einem 12- oder 16-Bit-Datensatz.

Grundsätzlich bleibt die Zeit für die Erstellung des Importance Volume hinter der Dauer der Gradientenbestimmung zurück. Eine Ausnahme bildet hier allenfalls, wie beschrieben, eine falsche Wahl des Schwellenwertes bei den Auflösungen *high* und *very high*, was aber normalerweise vermieden werden kann. Insofern sollte die Berechnungsdauer kein entscheidendes Kriterium gegen die Anwendung von adaptivem Sampling sein.

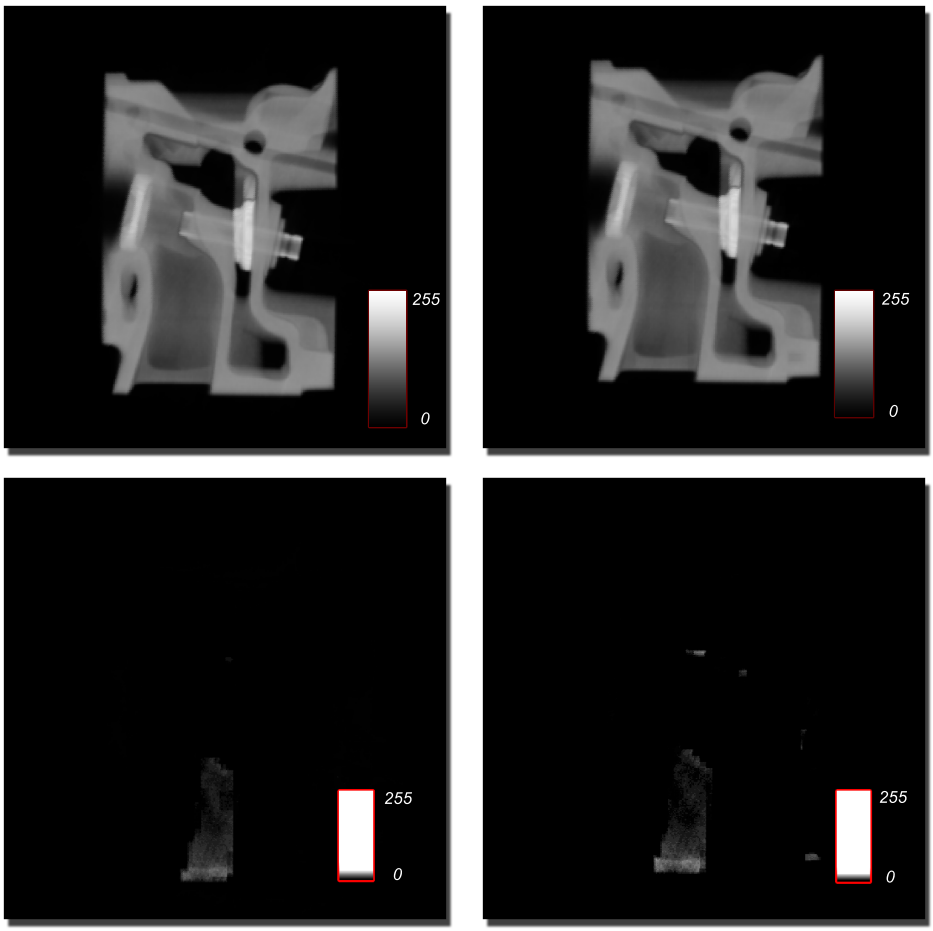

Abbildung 18: *Engine*-Datensatz. Von oben links nach unten rechts: Original ohne adaptives Sampling. Adaptives Sampling mit Schwellenwert von 10 und *medium*-Auflösung. Kontrastverstärkte Differenzbilder mit Schwellenwert von 3 bzw. 5 und *medium*-Auflösung.

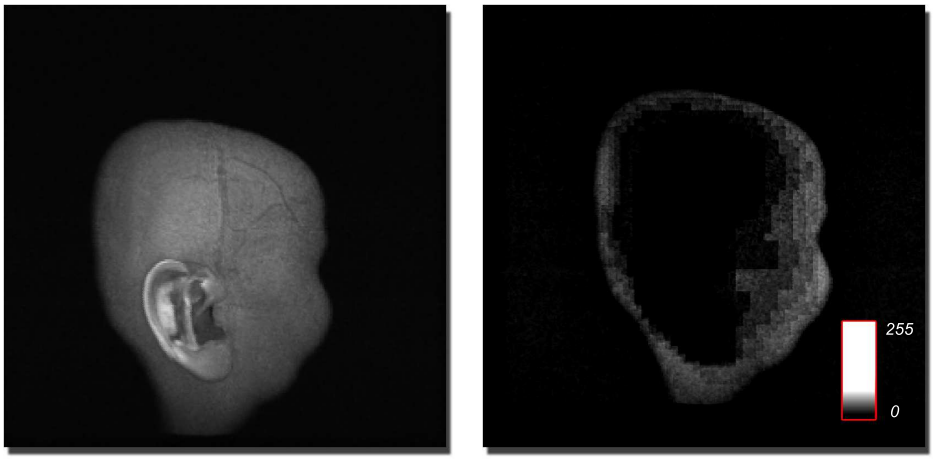

Abbildung 19: *KopfRed*-Datensatz. Original (links) und kontrastverstärktes Differenzbild mit Schwellenwert 10 und Auflösung *high*.

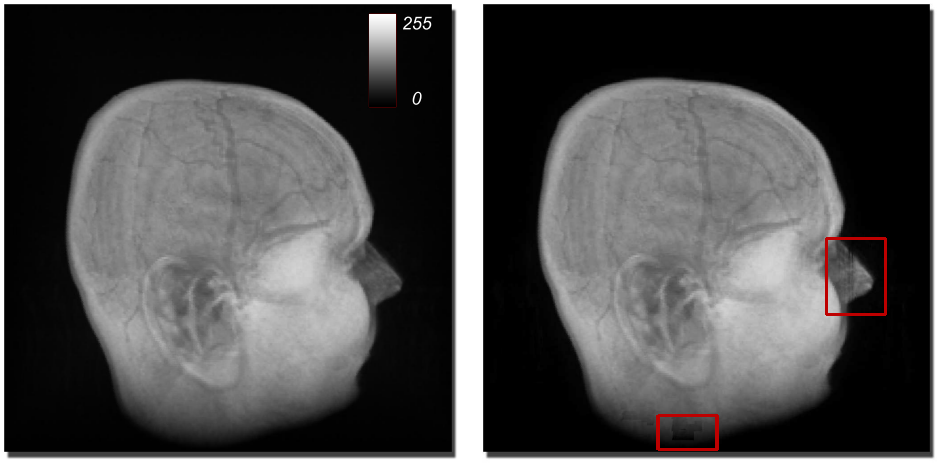

Abbildung 20: *Kopf*-Datensatz. Originalbild ohne (links) und mit adaptivem Sampling (Schwellenwert: 15, Auflösung: *medium*). Die Artefakte im rechten Bild sind mit bloßem Auge zu erkennen.

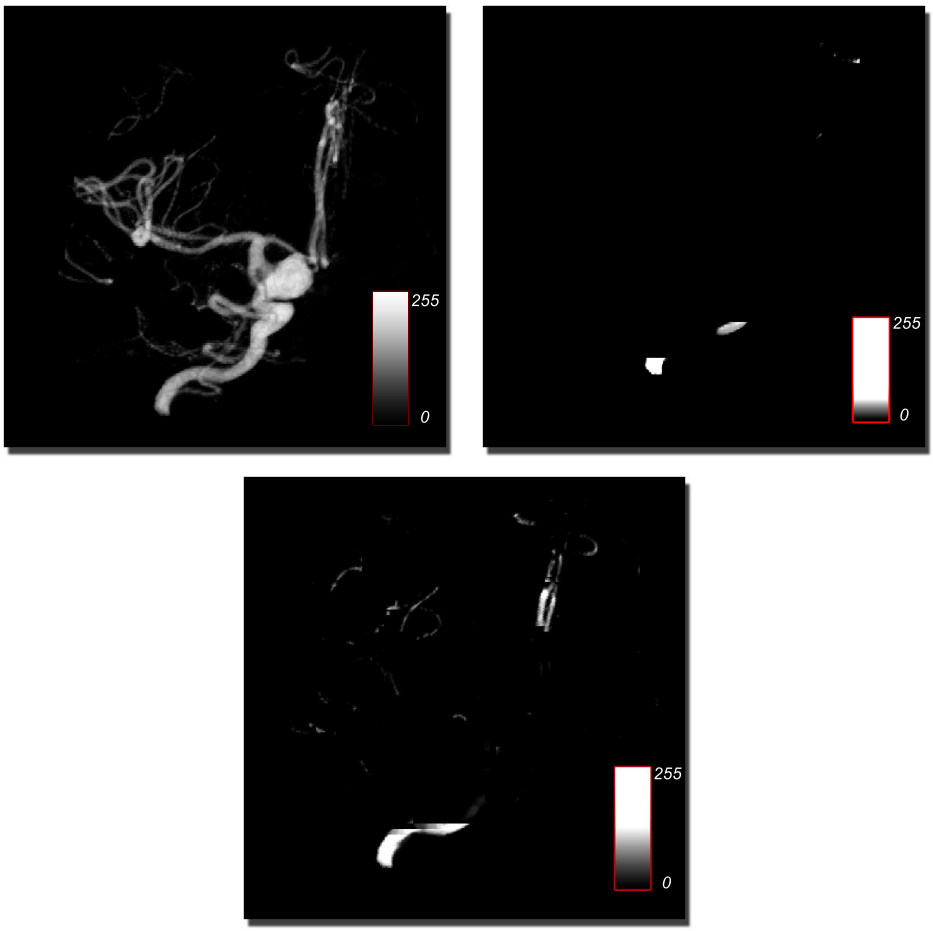

Abbildung 21: *Aneurysma*-Datensatz. Originalbild (oben links) und kontrastverstärkte Differenzbilder mit Schwellenwert 1 (oben rechts) und 5 (beide *medium*-Auflösung).

7 Bewertung und Ausblick

In dieser Arbeit wurde gezeigt, dass die Verwendung von adaptivem Sampling auf der GPU grundsätzlich Geschwindigkeitssteigerungen um den Faktor 2 bis 4 zulässt. Der tatsächliche Zuwachs hängt dabei in erster Linie von Art und Aufbau des Datensatzes ab. So kommt die geringe Varianz von leeren Regionen in CT-Aufnahmen und synthetischen Volumina dem Ansatz deutlich zugute, da der Gradientenschwellenwert hier sehr niedrig gewählt werden kann und damit ein schnelles Durchlaufen dieser Teile gewährleistet ist.

Datensätze, die eine über das ganze Volumen verteilte hohe Änderung der Werte aufweisen, sind dagegen weniger geeignet für adaptives Sampling. Zum einen ist es hier schwieriger einen geeigneten Grenzwert zu finden, der einen guten Kompromiss zwischen Qualität und Geschwindigkeit darstellt. Zum anderen werden die meisten Einträge im Importance Volume ohnehin leer bleiben, da das Prinzip des adaptiven Sampling auf dem Vorhandensein von homogenen Bereichen aufbaut. Insofern muss abgewogen werden, ob ein Datensatz die genannten Voraussetzungen erfüllt und ob sich eine Vorverarbeitung lohnt.

Die Frage, ob eine Analyse von Volumendaten allein auf den Skalarwerten basieren kann, hängt, wie schon beschrieben, direkt von der Transferfunktion ab. Es ist durchaus möglich, dass durch eine nichtlineare Transferfunktion ein hohes Maß an Varianz in einen zuvor homogenen Bereich eingebracht wird. Bei Anwendung von adaptivem Sampling kann deshalb die Darstellung verfälscht werden. Wie sich in den Tests gezeigt hat, kommen die größten Schrittweiten zwar innerhalb von leeren, d.h. unwichtigen Regionen des Volumens vor, während die interessanten Teile nach wie vor genau abgetastet werden. Es ist allerdings nicht auszuschließen, dass durch adaptives Sampling Details verloren gehen.

Eine weitere leichte Steigerung der Performanz des in dieser Arbeit vorgestellten Verfahrens ist durch zukünftige Grafikprozessoren zu erwarten, die z.B. die Flusskontrolle noch effizienter implementieren und so den dadurch entstehenden knapp 10-prozentigen Geschwindigkeitsverlust verringern. Es ist jedoch fraglich, ob jemals der gleiche prozentuale Zuwachs wie auf der CPU erreicht werden kann, da das Konzept der Renderingpipeline eine hohe Parallelisierung der Fragmentshadereinheiten vorsieht. Die schlechtere Cacheausnutzung durch das gleichzeitige Senden von Strahlen mit unterschiedlichen Schrittweiten bringt hier eine gewisse Limitierung mit sich.

Anhang

A 3D-Sobel-Filterkern

Ableitung in x-Richtung:

$$\begin{bmatrix} -0.03 & 0 & 0.03 \\ -0.06 & 0 & 0.06 \\ -0.03 & 0 & 0.03 \end{bmatrix}_{z-1} \begin{bmatrix} -0.06 & 0 & 0.06 \\ -0.12 & 0 & 0.12 \\ -0.06 & 0 & 0.06 \end{bmatrix}_{z} \begin{bmatrix} -0.03 & 0 & 0.03 \\ -0.06 & 0 & 0.06 \\ -0.03 & 0 & 0.03 \end{bmatrix}_{z+1}$$

Ableitung in y-Richtung:

$$\begin{bmatrix} -0.03 & -0.06 & -0.03 \\ 0 & 0 & 0 \\ 0.03 & 0.06 & 0.03 \end{bmatrix}_{z-1} \begin{bmatrix} -0.06 & -0.12 & -0.06 \\ 0 & 0 & 0 \\ 0.06 & 0.12 & 0.06 \end{bmatrix}_{z} \begin{bmatrix} -0.03 & -0.06 & -0.03 \\ 0 & 0 & 0 \\ 0.03 & 0.06 & 0.03 \end{bmatrix}_{z+1}$$

Ableitung in z-Richtung:

$$\begin{bmatrix} -0.03 & -0.06 & -0.03 \\ -0.06 & -0.12 & -0.06 \\ -0.03 & -0.06 & -0.03 \end{bmatrix}_{z-1} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}_{z} \begin{bmatrix} 0.03 & 0.06 & 0.03 \\ 0.06 & 0.12 & 0.06 \\ 0.03 & 0.06 & 0.03 \end{bmatrix}_{z+1}$$

Alle Masken bezüglich Blick entlang der positiven z-Achse.

B Shader mit Pre-Integration

Code 3: Emissions-Absorptionsshader mit Pre-Integration (8-Bit)

```
1  void main(float2 texCoords2D : TEXCOORD0,
2                uniform sampler2D frontTexture,
3                uniform sampler2D backTexture,
4                uniform sampler3D volumeTexture,
5                uniform sampler3D importanceVolumeTexture,
6                uniform sampler3D lookUpTable,
7                uniform float4 all,
8                out float4 outColor : COLOR)
9  {
10       float resolution = all.x;
11       float opacityfac = all.y;
12       float3 frontColor= tex2D(frontTexture, texCoords2D).rgb;
13       float3 backColor = tex2D(backTexture,  texCoords2D).rgb;
14       float3 direction = normalize(backColor - frontColor);
15       float3 index;
16       float m_output = 0.0;
17       float textureValue;
18       float opacity;
19       float importanceVolumeDistance;
20       float dirlength = length(backColor - frontColor);
21       float stepsize;
22       float d;
23       float oldTexValue = 0.0;
```

```
24
25        for (dirlength; dirlength > 0.0; ){
26            textureValue = tex3D(volumeTexture,  frontColor).x;
27            //weight opacity
28            opacity = (textureValue/opacityfac);
29            //importance volume lookup
30            importanceVolumeDistance =
31                 tex3D(importanceVolumeTexture, frontColor).r;
32            // set absolute distance
33            d = resolution * importanceVolumeDistance;
34            // set index for lookup table
35            index = (float3(textureValue, oldTexValue, d));
36            // rescale
37            d *= 255;
38            // set next stepsize
39            stepsize = 0.01 + 0.01 * d;
40            // get line segment from lookup table
41            textureValue = tex3D(lookUpTable, index).r;
42            //raycast
43            m_output = (opacity * textureValue +
44                       (1.0 - opacity)*m_output)
45            // set latest texture value for the next iteration
46            oldTexValue = textureValue;
47            // adjust ray length
48            dirlength -= stepsize;
49            //next position
50            frontColor = frontColor + (stepsize * direction);
51        }
52        float3 outVec = (float3(m_output, m_output, m_output));
53        outColor = float4(outVec, 1.0);
54    }
```

Literatur

[1] Danskin, J. ; Hanrahan, P. : Fast Algorithms for Volume Ray Tracing. In: *Proc. Symposium on Volume Visualization ´92* (1992), S. 91 – 98

[2] Engel, K. ; Kraus, M. ; Ertl, T. : High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading. In: *Eurographics / SIGGRAPH Workshop on Graphics Hardware 01* (2001), S. 9 – 16

[3] Engel, K. ; Hadwiger, M. ; Kniss, J. M. ; Lefohn, A. E. ; Salama, C. R. ; Weiskopf, D. : Real-Time Volume Graphics. In: *Course 28 Siggraph 2004* (2004)

[4] Kindlmann, G. L. ; Whitaker, R. T. ; Tasdizen, T. ; Möller, T. : Curvature-Based Transfer Functions for Direct Volume Rendering: Methods and Applications. In: *IEEE Visualization* (2003), S. 513 – 520

[5] Koblenz-Landau, U. : *WS05 / 06 Praktikum Medizinische Visualisierung.* Mai 2006. – http://www.uni-koblenz.de/FB4/Institutes/ ICV/AGMueller/Teaching/WS0506/PraktikumMedVis, letzte Änderung: 23.10.05

[6] Krüger, J. ; Westermann, R. : Acceleration Techniques for GPU-based Volume Rendering. In: *Proceedings of IEEE Visualization ´03* (2003), S. 287 – 292

[7] Levoy, M. : Efficient Ray Tracing of Volume Data. In: *ACM Transactions on Graphics* 9 (1990a), Juli, Nr. 3, S. 245 – 261

[8] Levoy, M. : Volume Rendering by Adaptive Refinement. In: *The Visual Computer* 6 (1990b), Nr. 1, S. 2 – 7

[9] Lorensen, W. E. ; Cline, H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In: *ACM SIGGRAPH Computer Graphics* 21 (1987), Juli, Nr. 4, S. 163 – 169

[10] Max, N. : Optical Models for Direct Volume Rendering. In: *IEEE Transactions on Visualization and Computer Graphics* 1 (1995), Nr. 2, S. 99 – 108. – ISSN 1077 – 2626

[11] Meissner, M. ; Guthe, S. ; Strasser, W. : Interactive Lighting Models and Pre-Integration for Volume Rendering on PC Graphics Accelerators. In: *Proc. Graphics Interface '02* (2002), S. 209 – 218

[12] Merziger, G. ; Wirth, T. : *Repetitorium der höheren Mathematik.* Am Bergfelde 28, 31832 Springe : Binomi Verlag, 1999. – ISBN: 3-923 923-33-3

[13] Roettger, S. ; Guthe, S. ; Weiskopf, D. ; Ertl, T. ; Strasser, W. : Smart Hardware-Accelerated Volume Rendering. In: *Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization* (2003)

[14] Scharsach, H. : Advanced GPU Raycasting. In: *Proceedings of CESCG 2005* (2005), S. 69 – 76

[15] Tübingen, U. : *Visualisierung wissenschaftlicher Daten.* Mai 2006. – http://www.gris.uni-tuebingen.de/areas/scivis/volren/index.html, letzte Änderung: 21.03.05, Kontakt: Dirk Bartz