Volume Rendering Techniques for General Purpose Graphics Hardware

Christof Rezk Salama

Abstract: Techniken der Volumenvisualisierung werden zur räumlichen Darstellung dreidimensionaler Skalarfelder benötigt, wie sie beispielsweise in der Medizin in Form von tomographischen Schnittbildern entstehen. Die hier in einer Kurzfassung vorliegende Dissertation führt neue Methoden zur interaktiven Darstellung von Volumendaten auf universeller Hardware ein. Diese Methoden nutzen die in handelsüblichen Personal Computern (PC) vorhandenen Graphikkarten, die hauptsächlich für Computerspiele und Multimedia-Anwendungen entwickelt wurden. Das Ziel dieser Arbeit ist es, auf solch einer kostengünstigen Plattform eine Lösung zu finden, deren Bildqualität mit traditionellen Ray-Casting Verfahren vergleichbar ist, und die gleichzeitig eine echtzeitfähige Performanz erreicht. In diesem Zusammenhang wurden die Vor- und Nachteile traditioneller texturbasierter Ansätze im Hinblick auf Geschwindigkeit und Darstellungsqualität analysiert. Basierend auf dieser Analyse wurden neue effiziente Techniken der Volumenvisualisierung entwickelt, die speziell die Möglichkeiten moderner PC-Graphik-Hardware, wie mehrstufige Rasterisierung, *Pixel Shaders*, und *Dependent Textures*, ausnutzen.

1 Einführung

Maler und Bildhauer stellen ihre persönliche Wahrnehmung der Realität in Bildern und Skulpturen dar, die ihre individuellen Eindrücke und Gefühle widerspiegeln. Ähnlich den bildenden Künsten kommuniziert die wissenschaftliche Visualisierung Information, die einer persönlichen Interpretation bedarf. Im Gegensatz zu subjektiven Sinneseindrücken sind es hier wissenschaftliche Daten, die dem Darstellungsprozess zugrunde liegen. In beiden Fällen jedoch sollten ästhetische Ansprüche niemals vernachlässigt werden.

Als eine Folge der rasanten technologischen Entwicklung umfassen wissenschaftliche Daten heutzutage weit mehr Information als es überhaupt möglich ist, in einem statischen Bild zu vermitteln. Die wissenschaftliche Visualisierung hat sich daher zu einem kreativen und explorativen Prozess entwickelt, innerhalb dessen die Strukturen und Zusammenhänge, die in den Daten verbogen sind, aufgedeckt werden.

Die hier in einer Kurzfassung vorliegende Dissertation [RS02] beschäftigt sich mit der Visualisierung dreidimensionaler Skalarfelder, in der Regel als *Volumendaten* bezeichnet. Solche Volumendaten werden beispielsweise in der Medizin durch tomographische Aufnahmeverfahren (z.B. Computer- und Kernspintomographie) gewonnen. Sie entstehen aber auch bei der numerischen Simulation komplexer Vorgänge in Naturwissenschaft und Technik.

Da die Analyse solcher Volumendaten einen hohen Grad an Benutzerinteraktion erfordert, werden schnelle Darstellungsverfahren benötigt, die üblicherweise auf Spezialhardware oder teure Graphik-Workstations zurückgreifen. Dies wiederum verringert die Anwendbarkeit der Algorithmen aufgrund der hohen Kosten und der geringen Verfügbarkeit entsprechender Systeme. Der Fokus dieser Arbeit liegt daher auf Verfahren, interaktive und hochqualitative Volumenvisualisierung auf handelsüblicher PC Hardware zu ermöglichen. Dabei sollen spezielle Eigenschaften günstiger Konsumer-Graphikkarten, die hauptsächlich für Computerspiele und Multimedia-Anwendungen entwickelt wurden, effizient genutzt werden.

Das Standardverfahren zur Generierung virtueller Bilder aus Volumendaten ist der *Ray-Casting* Algorithmus [KH93]. Die Bildsynthese basiert dabei auf dem physikalischen Modell der Ausbreitung des Lichts in transparentem Medium. Die Streuung des Lichts wird in diesem Fall vernachlässigt, und es wird nur die aktive Emission und Absorption betrachtet (*Emission-Absorption-Modell*). Eine sogenannte *Transferfunktion*, die vom Benutzer eingestellt wird, bildet die skalaren Datenwerte auf entsprechende Emissions- und Absorptionswerte (Farbe und Opazität) ab. Das vereinfachte Modell ohne Berücksichtigung der Streuung erlaubt es dann, die Lichtintensität entlang von Strahlen zu integrieren.

Der Ray-Casting Algorithmus berechnet für jeden Pixel des Ergebnisbildes einen Strahl durch den Volumendatensatz. Entlang dieses Strahls wird das Volumen mit äquidistanter Schrittweite neu abgetastet. Die einzelnen Abtastwerte werden dabei durch trilineare Interpolation bestimmt und anschließend durch die Transferfunktion auf entsprechende Emissions- und Absorptionswerte abgebildet. Der endgültige Farbwert des Pixels im Ergebnisbild wird schließlich durch numerische Integration des Emissions- und Absorptions-Modells errechnet.

Da die Neuabtastung des Datensatzes ein sehr rechen- und zeitaufwändiger Prozess ist, erreichen Ray-Casting-Verfahren in der Regel keine sehr hohen Bildraten. Um dem Benutzer eine Volumendarstellung anzubieten, die es ihm erlaubt interaktiv, d.h. in Echtzeit die Darstellungsparameter (beispielsweise den Blickwinkel oder die Transferfunktion) zu verändern, wurden hardwarebeschleunigte Verfahren entwickelt.

2 Universelle Graphik-Hardware

Eine Graphikkarte mit speziellen Prozessoren für die hardwarebeschleunigte Darstellung dreidimensionaler Szenen gehört heutzutage zu nahezu jedem Konsumer-PC. Diese Graphik-Prozessoren (GPUs) führen Berechnungen aus, die aus einer abstrakten polygonalen Szenenbeschreibung ein zweidimensionales Rasterbild generieren *(Rendering)*. Der Berechnungsvorgang ist als eine effiziente Verarbeitungskette, die sogenannte Graphik-Pipeline, realisiert. Wie in Abbildung 1 dargestellt kann diese Pipeline grob in drei verschiedene Stufen unterteilt werden.

Zu Beginn wird die gewünschte Szenengeometrie in planare Polygone zerlegt. Die *Eckpunkte* (Vertices) der Polygone werden als Datenstrom in die Pipeline eingegeben. Als erster Schritt in der Verarbeitungskette erfolgt die *Geometrieverarbeitung*. Die Eckpunk-

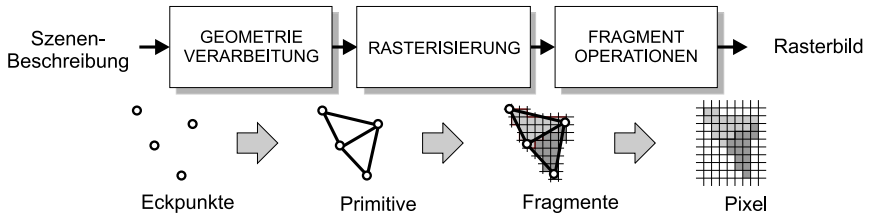

Abbildung 1: Die Graphik-Pipeline zum hardwarebeschleunigten Rendering.

te werden im dreidimensionalen Raum an ihre endgültige Position transformiert und zu *Primitiven* zusammengefasst. Als Primitive bezeichnet man geometrische Einheiten, die die Hardware weiterverarbeiten kann. In der Regel handelt es sich dabei um Dreiecke. In dem darauf folgenden *Rasterisierung*-Schritt werden anschließend diejenigen Pixel im Ergebnisbild ermittelt, die von dem entsprechenden Primitiv überdeckt werden. Das Primitiv wird somit in *Fragmente* zerlegt, die jeweils die Größe eines einzelnen Pixels haben. Die Farbe dieser Fragmente ergibt sich aus einer Kombination der Farbe des Dreiecks (Primärfarbe) und des interpolierten Farbwertes einer spezifizierten *Textur*, die ähnlich einer Tapete auf das Dreieck aufgebracht wird. Im letzten Schritt, den *Fragment Operationen*, werden die entstandenden Fragmente in das endgültige Bild geschrieben und dabei mit der bereits im Bild enthaltenen Information verknüpft. Das Ergebnis ist ein Pixelbild der eingegebenen Szenenbeschreibung.

3 Texturbasierte Ansätze

Will man eine Hardware, wie sie oben beschrieben ist, für die Darstellung volumetrischer Information nutzen, so stellt man zunächst fest, dass keine echten volumetrischen Rendering-Primitive unterstützt werden. Als Abhilfe wird daher eine Hilfsgeometrie verwendet. Das volumetrische Objekt wird, wie in Abbildung 2 skizziert, in eine hohe Anzahl polygonaler Schichten zerlegt, die anschließend mit der entsprechenden Bildinformation texturiert werden. Es gibt unterschiedliche texturbasierte Ansätze, die sich hauptsächlich darin unterscheiden, auf welche Weise diese Zerlegung in polygonale Schichten erfolgt.

3.1 Verfahren mit zweidimensionalen Texturen

Zweidimensionale Texturen (2D-Texturen) sind gewöhnliche Pixelbilder, die während der Rasterisierung auf die Flächen der Polygone abgebildet werden. Dazu wird für jeden Eckpunkt eines Polygons zusätzlich zu den Ortskoordinaten auch noch ein Texturkoordinaten-Paar angegeben. Während der Texturierung interpoliert der Grafik-Prozessor für jedes einzelne Fragment innerhalb eines Dreiecks entsprechende Texturkoordinaten aus den Werten, die an den Eckpunkten gegeben sind (Interpolation in *baryzentrischen* Koordina-

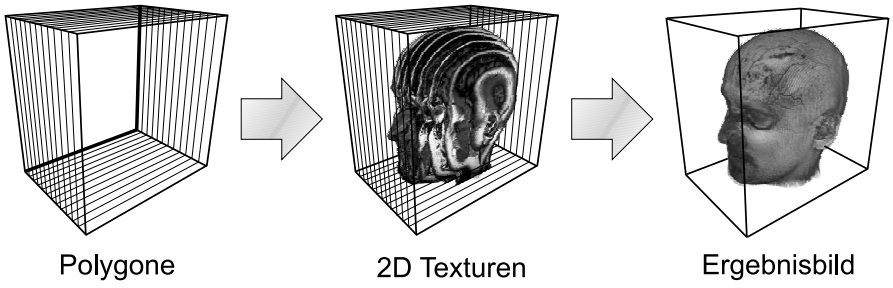

Abbildung 2: Die Zerlegung des Volumendatensatzes in achsenparallele, polygonale Schichten.

ten). Anhand dieser Texturkoordinaten wird wiederum ein Farbwert aus der Textur durch bilineare Interpolation bestimmt. Dieser interpolierte Farbwert wird schließlich mit der primären Farbe des Fragments kombiniert.

Um die hardwarebeschleunigte, bilineare Interpolation der 2D-Texturen auszunutzen, wird der Volumendatensatz in achsenparallele Schichten zerlegt, wie in Abbildung 2 dargestellt. Je nachdem entlang welcher Koordinatenachse man das Volumen unterteilt, gibt es drei mögliche Schichtenstapel zur Auswahl. Für die Darstellung werden dann jeweils diejenigen Schichten gewählt, bei denen der Winkel zwischen ihrer Normalenrichtung und der aktuellen Blickrichtung minimal ist. Auf diese Weise wird gewährleistet, dass der Betrachter niemals zwischen einzelnen Schichten hindurch blicken kann. Diese Vorgehensweise hat allerdings den Nachteil, dass der Volumendatensatz dreifach im Speicher vorliegen muss, da für jede Schichtrichtung ein separater Satz Texturen benötigt wird.

Ein weiterer Nachteil der Verfahren mit 2D-Texturen ist eine verminderte Bildqualität, die dadurch entsteht, dass die trilineare Interpolation beim Ray-Casting lediglich durch eine bilineare Interpolation approximiert wird. Wie in Abbildung 3 demonstriert, sind visuelle Bildartefakte sichtbar, die auf die statische Zerlegung des Volumendatensatzes in Schichten zurückzuführen sind. Eine Erhöhung der Abtastrate (durch Erhöhung der Schichtzahl), die zur Vermeidung dieser Artefakte notwendig wäre, ist bei diesem Verfahren aus Performanz-Gründen nicht möglich, denn die fehlende Bildinformation zur Texturierung der zusätzlichen Schichten kann zur Laufzeit nicht schnell genug berechnet werden.

3.2 Verfahren mit dreidimensionalen Texturen

Auf High-End Graphik-Workstations – und mittlerweile auch auf Konsumer-Grafikkarten – gibt es neben 2D-Texturen auch 3D-Texturen, die trilineare Interpolation in Hardware unterstützen. In diesem Fall handelt es sich um dreidimensionale Texturobjekte, jedoch nicht um dreidimensionale Rendering-Primitive. Die Zerlegung des Datensatzes in polygonale Schichten muss daher nach wie vor erfolgen. Die Texturkoordinaten an den Eck-

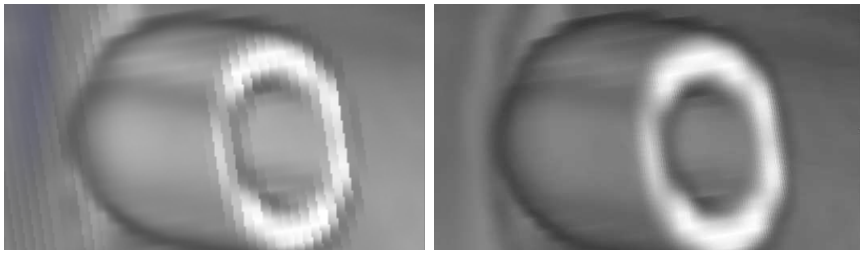

Abbildung 3: Im Vergleich zum Ray-Casting *(rechts)* treten beim 2D-texturbasierten Verfahren *(links)* deutliche Bildartefakte auf.

punkten der Polygone bestehen nun allerdings aus drei Komponenten. Die Textur für das Polygon wird dabei sozusagen aus einem dreidimensionalen Texturblock *herausgeschnitten*.

Da nun beliebige Schnittebenen aus dem dreidimensionalen Texturblock als Polygon-Textur verwendet werden können, bietet es sich an, anstatt der achsenparallelen Schichten das Volumen in Schichten parallel zur aktuellen Bildebene zu zerlegen. Wie in Abbildung 4 skizziert, bedeutet dies allerdings, dass die Zerlegung in Polygone jedes Mal neu berechnet werden muss, wenn sich die Position der virtuellen Kamera relativ zum Volumen ändert. Da der Prozess der Schnittberechnung vom Zentralprozessor (CPU) ausgeführt wird und den Graphik-Prozessor (GPU) nicht weiter belastet, hat die Neuberechnung der Polygone keine signifikante Auswirkung auf die Darstellungsgeschwindigkeit.

Die Vorteile der Verwendung von 3D-Texturen sind offensichtlich. Visuelle Bildartefakte, wie sie bei der Verwendung von 2D-Texturen auftreten, können durch Erhöhung der Schichtenzahl effektiv vermieden werden. Die Anzahl der Schichten kann bei diesem Verfahren beliebig erhöht werden, da die Texturinformation für alle Schichten zur Laufzeit interpoliert wird. Als weiterer Vorteil fällt die erwähnte Verdreifachung des Volumenda-

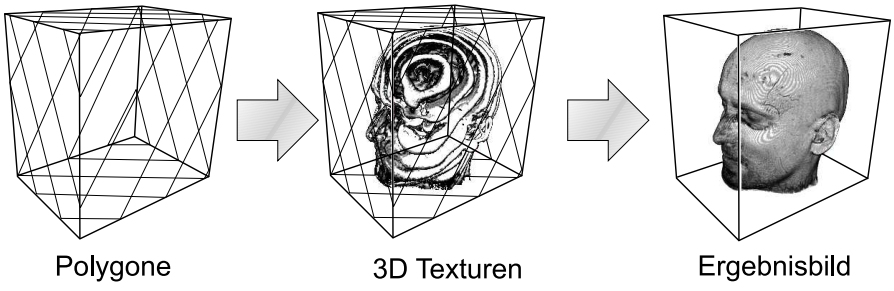

Abbildung 4: Die Zerlegung des Volumens in polygonale Schichten parallel zur Bildebene.

tensatzes im Speicher weg.

Das 3D-texturbasierte Verfahren erreicht relativ hohe Performanz auf High-End Grafik-Workstations mit entsprechend schnellem Speicherbus. Portiert man das Verfahren in gleicher Weise auf den PC, so stellt man erhebliche Leistungseinbußen fest, insbesondere bei großen Datensätzen. Bei Volumina, die aufgrund ihrer Größe nicht mehr vollständig in den (begrenzten) lokalen Speicher der Grafikkarte passen, muss der 3D-Texturblock in kleinere Subvolumina (*Bricks*) zerlegt werden. Dieser Vorgang wird *Bricking* genannt. Während des Renderings wird ein Brick nach dem anderen in den lokalen Grafikspeicher geladen. Anschließend werden die entsprechenden Polygone gezeichnet und der nächste Brick eingelagert. Bei dieser Vorgehensweise ist der größte Teil des Grafikspeichers mit einem einzigen Texturblock gefüllt. Es kommt dadurch zu erheblichen Verzögerungen, weil einerseits die GPU warten muss, bis der gesamte Brick eingelagert ist. Andererseits muss der Speicherbus warten bis die GPU alle Schichten gezeichnet hat, bevor der nächste Brick eingelagert werden kann. Hinzu kommt die Tatsache, dass bei derart großen 3D-Texturen der lokale Textur-Cache praktisch nutzlos ist, da er für 2D-Texturen begrenzter Größe ausgelegt ist.

3.3 Verfahren mit zweidimensionalen Multitexturen

Für die effiziente Darstellung virtueller Szenen in Computerspielen ist es oft notwendig, mehrere Texturen auf eine einzige Fläche anzuwenden, beispielsweise um vorberechnete Beleuchtungsinformation (Light Maps) mit den üblichen Wand-Texturen (Decal Textures) zu kombinieren. Die GPUs auf modernen PC-Grafikkarten bieten hierzu Multitexturen an und ermöglichen es somit während eines Durchlaufs durch die Grafik-Pipeline mehrere Texturen auf ein Fragment anzuwenden.

2D-Multitexturen können genutzt werden, um das in Abschnitt 3.1 vorgestellte 2D-texturbasierte Verfahren zu modifizieren, so dass zwischen den ursprünglichen Schichten zur Laufzeit beliebig viele zusätzliche Schichten eingefügt werden können. Dadurch können die typischen Bildartefakte, wie sie oben beschrieben sind, vermieden werden. Die Texturinformation für diese zusätzlichen Schichten wird während der Rasterisierung aus den zwei benachbarten Original-Schichten interpoliert. Das Ergebnis ist eine vollständige trilineare Interpolation, wobei zwei bilineare Interpolationen durch die beiden Multitexturen und der dritte lineare Interpolationsschritt durch die Farbkombination berechnet werden.

Der Vorteil dieser Technik gegenüber dem 3D-texturbasierten Verfahren ist eine bessere Ausnutzung der Speicherbandbreite bei großen Datensätzen. Um eine Schicht zeichnen zu können, benötigt die GPU lediglich die zwei benachbarten Texturen. Nachdem diese beiden Texturen in den lokalen Grafikspeicher eingelagert wurden, kann die GPU mit der Rasterisierung beginnen. Da die beiden Texturen nur einen geringen Teil des Grafikspeichers benötigen, können zeitgleich die nächsten Texturen über den Bus transferiert werden. Auf diese Weise erreicht man eine parallele Auslastung von Speicherbus und Gra-

fikprozessor. Im Ergebnis erhält man im Vergleich zum 3D-texturbasierten Verfahren eine erheblich höhere Performanz, obwohl das 2D-Multitexturbasierte Verfahren den Datensatz dreifach im Speicher halten muss.

4 Transferfunktionen

Mit dem Einstellen der Transferfunktion (*Klassifikation*), bestimmt der Benutzer, welche Teilbereiche im Volumendatensatz ganz ausgeblendet werden sollen, welche Teile opak oder semi-transparent dargestellt werden und welche Farben die einzelnen Bereichen haben sollen. Die Transferfunktion wird üblicherweise anhand einer Farbtabelle spezifiziert und ist eines der wichtigsten Mittel zur Analyse des Datensatzes. Daher ist es notwendig dem Benutzer die Möglichkeit zu bieten, die Transferfunktion in Echtzeit zu modifizieren und das Ergebnis seiner Modifikation unmittelbar auf dem Bildschirm zu betrachten. Dabei ist es von entscheidender Bedeutung an welcher Stelle innerhalb der Grafik-Pipeline die Transferfunktion angewendet wird.

Unter **Präklassifikation** versteht man die Anwendung der Transferfunktion *vor* der Interpolation der Skalarwerte. Für *jeden Abtastpunkt* des diskreten Volumendatensatzes wird somit ein Emissions- und ein Absorptionswert aus der gegebenen Farbtabelle bestimmt. Die Implementierung einer solchen Transferfunktion ist denkbar einfach. Im trivialen Fall kann beispielsweise der ursprüngliche Skalarwert durch die entsprechenden Emissions- und Ansorbtionswerte ersetzt werden, die dann direkt in die Textur geschrieben werden. Alle Interpolations-Operationen werden dann auf den Emissions- und Absorptionswerten durchgeführt.

Präklassifikation hat jedoch den Nachteil, dass die Bildqualität durch Aliasing-Artefakte erheblich gemindert wird. Diese in Abbildung 5 dargestellten Bildfehler lassen sich mit Hilfe des Abtasttheorems erklären. Wir interpretieren den Volumendatensatz als eine diskrete Repräsentation eines dreidimensionalen Signals und gehen davon aus, dass sich das kontinuierliche Skalarfeld aus dem diskreten Abtastwerten verlustfrei rekonstruieren lässt.

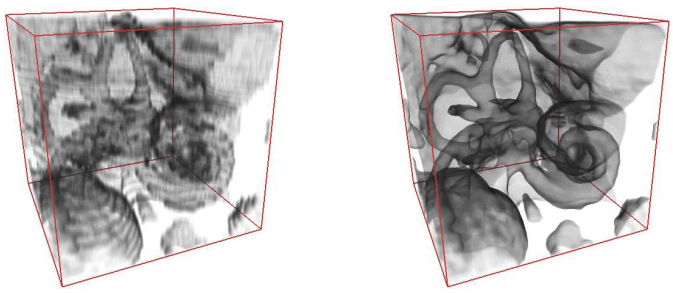

Abbildung 5: Vergleich der Bildergebnisse bei Präklassifikation *(links)* und Postklassifikation *(rechts)*.

Wir gehen somit implizit davon aus, dass das ursprüngliche, kontinuierliche Signal keine Frequenzen enthält, die eine Abtastung mit einer kleineren Schrittweite erfordern würden. Desweiteren nehmen wir an, dass die Approximation des zur exakten Rekonstruktion benötigten Sinc-Filters durch einen Dreiecksfilter für die gewünschte Bildqualität ausreichend ist.

Wenn man Präklassifikation durchführt, ersetzt man für jeden diskreten Abtastpunkt den Skalarwert s durch eine Transferfunktion $T(s)$. Die Transferfunktion $T(s)$ verändert dabei in der Regel das Frequenzspektrum des Signals, beispielsweise wenn sie scharfe Übergänge zwischen transparenten und opaken Bereichen enthält. Das zugrundeliegende Gitter wird dabei jedoch nicht verändert, obwohl die Berücksichtigung der höheren Frequenzen eine kleinere Schrittweite bei der Diskretisierung fordern würden. Das Ergebnis sind die für Aliasing typischen Stufeneffekte.

Die Alternative zur Präklassifikation ist die Anwendung der Transferfunktion *nach* der Interpolation. Es wird also für *jedes Fragment* ein Emissions- und ein Absorptionswert aus der gegebenen Farbtabelle bestimmt. Dies wird als **Postklassifikation** bezeichnet.

Eine effiziente Implementierung von Postklassifikation auf universeller Graphikhardware ist schwierig, da es nicht möglich ist aus der Pipeline Zwischenergebnisse (z.B. Fragment-Werte) zu extrahieren, diese in Software zu modifizieren und anschließend wieder in die Pipeline einzufügen. Es müssen also vorhandene Mechanismen genutzt werden um die Transferfunktion innerhalb der Pipeline anzuwenden.

Die Lösung dazu heißt *Dependent Textures*, wiederum ein Konzept das durch Computerspiele motiviert ist. Es handelt sich dabei um eine Erweiterung der Multitexturen. Nachdem das Primitiv rasterisiert wurde, wird für jedes Fragment ein Texturkoordinatenpaar (s, t) interpoliert. Anhand dieser Koordinaten wird aus der ersten Textur ein RGBA-Farbwert[1] interpoliert. Die zweite Textur ist die Dependent Texture, denn sie bezieht ihre Texturkoordinaten nicht aus den interpolierten Werten der Eckpunkte, sondern aus dem Farbwert der ersten Textur. Die R und A Komponenten des Farbwertes der ersten Textur werden als Texturkoordinaten für die zweite Textur interpretiert. Aus der zweiten Textur wird anhand dieser Koordinaten wiederum ein Farbwert bilinear interpoliert.

Um das Konzept der Dependent Textures zur Implementierung einer Transferfunktion auf Fragment-Ebene zu verwenden, wird der ursprüngliche Skalarwert s in die erste Textur geschrieben. Die Dependent Texture enthält die Farbtabelle der Transferfunktion und verwendet somit dem interpolierten Skalarwert s als Koordinate. Das Ergebnis ist die Anwendung der Transferfunktion $T(s)$ separat auf jedes Fragment.

Der Vorteil der Postklassifikation ist eine höhere Genauigkeit bei der Abtastung. Durch Erhöhung der Schichtenzahl können auch hohe Frequenzen in der Transferfunktion berücksichtigt werden. Abbildung 5 zeigt den Vergleich der Bildqualität zur Präklassifikation. Die typischen Aliasing Artefakte werden somit effektiv vermieden.

Ein besonderer Vorteil der Implementierung mittels Dependent Textures ist die Tatsache, dass auch zwei- oder mehrdimensionale Transferfunktionen möglich sind. Beispielsweise kann neben dem ursprünglichen Skalarwert s auch der Betrag des Gradienten $|\nabla s|$ zur

[1] RGBA steht für die einzelnen Farbkomponenten Rot, Grün, Blau und Alpha (= Opazität).

Abbildung 6: Volumenvisualisierung mit Transferfunktion $T(s)$ über den Skalarwert *(links)*, mit gradientengewichteter Transferfunktion $T(s, |\nabla s|)$ *(Mitte)*, und mit zusätzlicher Beleuchtung *(rechts)*.

Klassifikation herangezogen werden. Abbildung 6 demonstriert den Nutzen dieser Vorgehensweise. Durch eine Transferfunktion $T(s, |\nabla s|)$ können somit Bereiche aufgrund ihrer Homogenität eingefärbt oder transparent gemacht werden.

5 Lokale Beleuchtung

In den bisher beschriebenen texturbasierten Verfahren entstand die im Bild dargestellte Lichtintensität allein aus der Eigenemission des Volumendatensatzes. Für die bessere Wahrnehmung räumlicher Strukturen ist es wünschenswert, auch die Beleuchtung von externen Lichtquellen zu berücksichtigen.

Das bekannte Blinn-Phong-Beleuchtungsmodell beschreibt die ambiente, diffuse und spekulare Beleuchtung einer Oberfläche als eine Funktion des Oberflächennormale $\vec{n}$, der Position der Lichtquelle $\vec{l}$ und der aktuellen Blickrichtung $\vec{v}$. Um dieses für Oberflächen geeignete Beleuchtungsmodell auf Volumendaten anzuwenden, wird die Oberflächennormale $\vec{n}$ durch den normalisierten Gradientenvektor ∇s ersetzt. Dies ist durch die Tatsache motiviert, dass der Gradientenvektor an jedem Punkt im Skalarfeld in die gleiche Richtung zeigt wie die Normalen auf die Isofläche. Für jeden Abtastpunkt wird der Gradientenvektor vorberechnet (beispielsweise durch zentrale Differenzen) und zusätzlich in der Textur gespeichert.

Parametrisiert man die berechnete Gradientenrichtung in Polarkoordinaten φ und ϑ, so kann bei konstanter Beleuchtungsumgebung der gesamten Beleuchtungsterm als Funktion der Normalenrichtung vorberechnet und als eine Transferfunktion $T(\varphi, \vartheta)$ über die beiden Winkel φ und ϑ interpretiert werden. Eine derartige zweidimensionale Transferfunktion kann analog zu der in Abschnitt 4 beschriebenen Implementierung mit Hilfe von Dependent Textures realisiert werden.

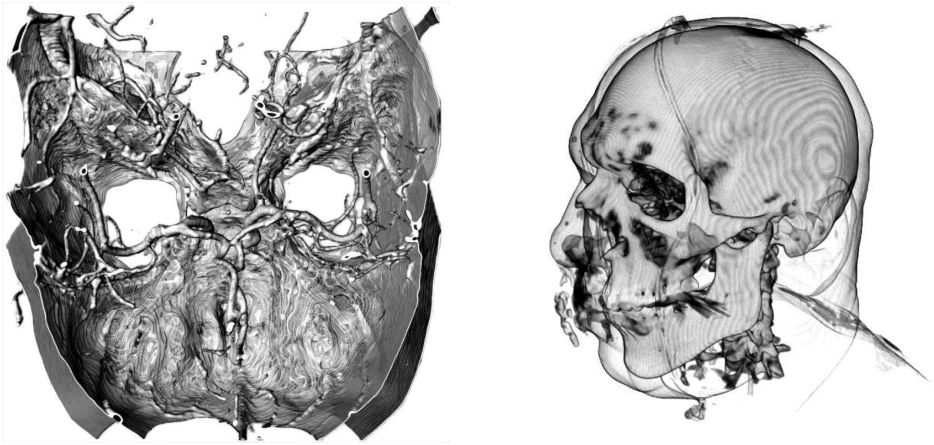

Abbildung 7: *Links:* Volumenvisualisierung der Blutgefäße im Gehirn aus CT Angiographie-Daten (Neuroradiologie, Kopfklinik Erlangen). *Rechts:* Computertomographie des Kopfes des *Visible Human* (National Library of Medicine, Maryland, USA).

6 Zusammenfassung

Die im Rahmen dieser Disseration erarbeiteten hardwarebeschleunigten Techniken ermöglichen hochqualitative Volumenvisualisierung mit echtzeitfähiger Performanz auf handelsüblicher PC-Hardware. Die Verfahren wurden an unterschiedlichen realen Visualisierungsproblemen in der Medizin getestet. Beispielhaft zeigt Abbildung 7 die Visualisierung von Blutgefäßen im menschlichen Gehirn und die Darstellung eines CT-Datensatzes aus dem Visible Human Project. Detaillierte Berichte über die medizinischen Anwendungen, weitere Forschungsergebnisse, sowie eine vollständige Literaturliste können in der Komplettfassung der Dissertation [RS02] nachgelesen werden.

Literatur

[KH93] Kajiya, J. und Herzen, B. V.: Ray tracing volume densities. In: *Proc. SIGGRAPH*. 1993.

[RS02] Rezk-Salama, C.: *Volume Rendering Techniques for General Purpose Graphics Hardware*. PhD thesis. Univ. Erlangen-Nürnberg. 2002.

Christof Rezk-Salama hat an der Universität Erlangen-Nürnberg Informatik studiert. Im März 2002 promovierte er am *Lehrstuhl für Graphische Datenverarbeitung* in Erlangen als Stipendiat im Graduiertenkolleg *Dreidimensionale Bildanalyse und Synthese*. Nach der Promotion arbeitete er ein Jahr als Entwicklungsingenieur bei Siemens Medical Solutions, Erlangen. Seit Oktober 2003 ist er als wissenschaftlicher Mitarbeiter am *Lehrstuhl für Computergraphik und Multimediasysteme* der Universität Siegen tätig.