UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

GPU-Ray Textures zur interaktiven Steuerung von direktem Volumenrendering

Studienarbeit

im Studiengang Computervisualistik

vorgelegt von
Stefano Ligas

Betreuer: Dr. Matthias Raspe
(Institut für Computervisualistik, AG Computergraphik)

Koblenz, im August 2009

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	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

Die Visualisierung von Volumendaten findet unter anderem in der Medizin, bei der Abbildung von Geodaten oder bei Simulationen ihre Anwendung. Ein effizientes Verfahren zur Darstellung von Volumendaten bietet das Raycasting, das durch die hohe Leistung von Consumerhardware hervorragende Qualität und große Flexibilität in Echtzeit ermöglicht. Beim Raycasting-Verfahren werden Strahlen durch ein Volumen verfolgt und anhand (regelmäßiger) Samples entlang des Strahles Farb- und Opazitätswerte bestimmt. „Ray Textures“ [1] sind ein Konzept zur Steuerung verschiedener Strahlparameter durch das Einzeichnen beliebiger Bereiche auf einer Textur. Der bisherige Ansatz ist jedoch softwarebasiert und umfasst nur einen begrenzten Funktionsumfang. Ziel dieser Studienarbeit ist eine eigenständige Implementation eines GPU-Volumen-Raycasters und die Umsetzung des RayTexture Ansatzes komplett auf der GPU. Im Vordergrund steht dabei die Unterstützung (nahezu) beliebiger Pinselformen und -modi, das Mapping der 2D-Interaktion auf das 3D-Rendering und die Steuerung weiterer Strahlparameter in Echtzeit. Die Schwerpunkte der Studienarbeit sind im Einzelnen die Implementation eines GPU-Volumen-Raycasters, die Umsetzung des Ray Texture Ansatzes komplett auf der GPU, die Vorstellung der Ergebnisse anhand mehrerer Beispielszenarien und die Dokumentation der Ergebnisse.

1 Einleitung

1.1 Motivation

Aufgrund der stetig steigenden Leistung der Consumerhardware wird das Raycasting-Verfahren zur Darstellung von volumetrischen Daten immer wichtiger. Das Raycasting-Verfahren basiert auf Strahlen, die in einem Volumen verfolgt werden und die anhand kontinuierlicher Abtastpunkte jedem Bildpunkt eine Farbe zuordnen. Vor allem bei medizinischen Aufnahmen, wie sie Computertomographien (CT) und Magnetresonanz-Tomographie (MRT) liefern, kann die so erzeugte 3D-Sicht bei der Diagnostik helfen. Allerdings ist die Interaktion mit dem Volumen meist beschränkt oder nicht vorhanden. So ist zum Beispiel die genauere Betrachtung oder die Isolierung relevanter Körperpartien eher schwierig und nicht ohne weiteren Rechenaufwand möglich. Wünschenswert wäre es daher, das Volumen zur Laufzeit so zu verändern, dass es den Vorstellungen des Betrachters entspricht. „Ray Textures“ beschreiben ein Konzept, mit dem man durch das Einzeichnen beliebiger Bereiche auf einer Textur die im Raycasting verwendeten Strahlen manipulieren und so das Volumen anpassen kann. Durch die in der Textur eingezeichneten Stellen lassen sich somit beispielsweise bestimmte Bereiche hervorheben oder können als manuelle Rechenzeitbeschleunigung verwendet werden.

1.2 Ziel der Arbeit

Ziel der Arbeit ist die Implementierung eines Raycasters und der „Ray Textures“ komplett auf der GPU. Ebenfalls sollen die in [1] erwähnten Einschränkungen behoben und das Konzept um zusätzliche Flexibilität erweitert werden. Dabei fallen drei Aufgaben in den Vordergrund. Zum einen sollen weitere Modifikationen der Strahlparameter möglich sein und bereits bestehende implementiert und evaluiert werden. Zum anderen befasst sich diese Arbeit mit der Einschränkung in der Pinselauswahl. Hierbei soll sowohl die Pinselgröße als auch die Pinselform variabel sein. Weiterhin wird die Abbildung der Textur auf das dreidimensionale Objekt neu überdacht und intuitiver gestaltet. Anhand mehrerer Beispielszenarien werden diese drei Punkte vorgestellt und diskutiert.

1.3 Aufbau der Arbeit

- Kapitel 2 erläutert zunächst einige Grundlagen zum direkten Volumenrendering. Dabei wird insbesondere auf das Raycasting-Verfahren eingegangen und dessen Implementierung auf der GPU.
- Im 3. Kapitel wird das „Ray Textures“-Konzept erläutert und mögliche Weiterentwicklungen vorgestellt.
- Das 4. Kapitel zeigt die Implementation, wobei näher auf die einzelnen Shader-Programme eingegangen wird.

- Anschließend werden im 5. Kapitel die Ergebnisse anhand mehrerer Beispielszenarien vorgestellt und Vor- und Nachteile der Implementation erörtert.
- In Kapitel 7 wird ein abschließendes Fazit gezogen und in einem Ausblick (Kapitel 6) mögliche Erweiterungen besprochen.

2 Volumenrendering

2.1 Volumetrische Daten

Analog zu den Pixeln im zweidimensionalen Bildaufbau gibt es bei einem Volumen ebenfalls Datenelemente, Voxel[1], die die Farb-, Helligkeitsinformationen evtl. auch Transparenzinformationen beinhalten. Vereinfacht kann man sich ein Volumen als Stapel von Bildern vorstellen, wobei man jeden rechteckigen Pixel in einen quaderförmigen Voxel überführt [2]. Die Höhe des Voxels entscheidet dabei, wie dick eine Schicht in diesem Stapel ist (Abbildung 1). Betrachtet man sich Abbildung 1 genauer, so erkennt man, dass volumetrische Daten nicht nur die Oberfläche beschreiben, sondern auch über Informationen der inneren Struktur der Objekte verfügen.

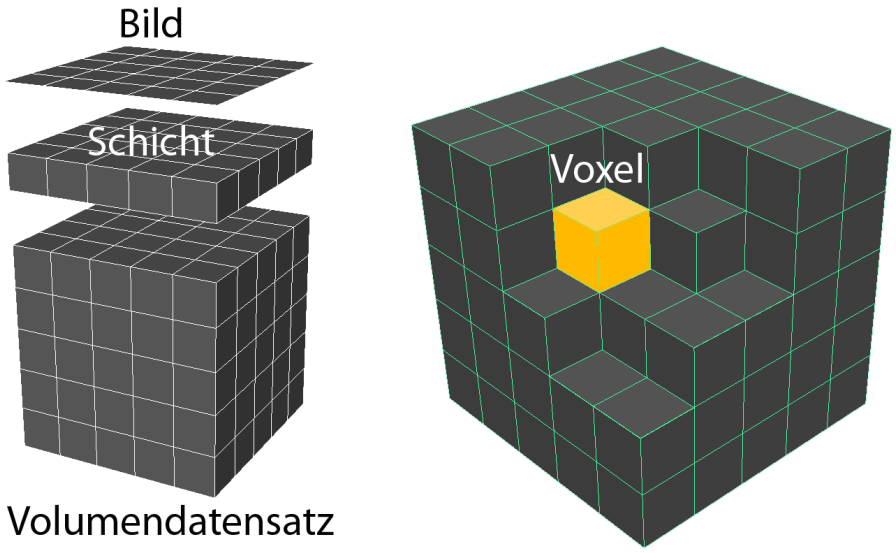

Abbildung 1: Schematischer Aufbau eines Volumendatensatzes.

Es gibt viele Gebiete, bei denen volumetrische Daten erzeugt werden, aber vor allem in der Medizin entstehen durch verschiedene Modalitäten solche Datensätze. Bei einer Computertomographie zum Beispiel verschießt man kontinuierlich Röntgenstrahlen in einem Halbkreis (0°-180°) durch den Körper. Je nachdem, wie dicht das getroffene Gewebe ist, wird der Strahl mehr oder weniger abgeschwächt. Mit Hilfe der restlichen Strahlung, die am anderen Ende ankommt, lässt sich durch verschiedene Techniken ein Schnittbild des Körpers rekonstruieren. Die Schnittbilder werden in regelmäßigen Abständen aufgenommen, bis der Bilderstapel die gescannte Körperpartie komplett wiedergeben kann [2].
Die so gewonnenen Aufnahmen stellen jedoch aufgrund der hohen Datenmengen sehr hohe Ansprüche an die Hard- und Software. Beispielsweise enthält ein

[1] Voxel abgeleitet von volume element

Abdomen-CT-Scan (Abbildung 2) mit einer Datentiefe von 16 Bit (12 Bit Nutzdaten), einer Auflösung von 512 x 512 pro Bild und mit insgesamt 800 Schnittbildern in einem Abstand von 0.5 mm eine gesamt Datengröße von 400 Mb [3].

Abbildung 2: [3]Abdomen-CT-Scan

2.2 Direktes Volumenrendering (DVR)

Das Prinzip der direkten Volumenvisualisierung[2] basiert auf der zweidimensionalen Darstellung eines volumetrischen Datensatzes. Allen DVR-Verfahren liegt ein optisches Modell zugrunde, bei dem grundsätzlich der Weg des Lichtes durch das Medium betrachtet wird[3][3]. Die Dichte des Mediums wird direkt durch Skalarwerte, die man den Voxeln zuweist, repräsentiert. Anhand des optischen Modells werden die Skalare entlang der Blickrichtung durch das Volumen integriert.
Der größte Vorteil des direkten Volumenrenderings besteht darin, dass die Visualisierung des Volumens ohne den Umweg einer zeitaufwändigen Oberflächenrepräsentation erfolgt. Es ermöglicht weiterhin eine qualitative Darstellung interner Strukturen, inklusive amorphen und transparenten Merkmalen [5].
Im Bildraum[4] wird für jedes Pixel der Darstellung ein Strahl durch das Volumen verfolgt. Durch die Integration der Voxel auf dem Strahl erhält jedes Pixel seine

[2]Es gibt auch indirekte Volumenrendering-Verfahren, bei denen aus dem Volumendatensatz eine Oberflächenrepräsentation berechnet wird. Der bekannteste Vertreter des indirekten Volumenrenderings ist das „Marching Cubes“-Verfahren [4].

[3]Das optische Modell wird genauer in Kapitel 2.5 erklärt.

[4]Da sich diese Arbeit auf das im Bildraum bezogene Raycasting (Kapitel 2.6) beschränkt, wird auf eine genauere Erläuterung anderer Verfahren im Frequenz- oder Objektraum verzichtet.

Farbe, wodurch das Bild entsteht. Der detailierte Ablauf eines direkten Volumenrenderings wird im nächsten Kapitel erläutert.

2.3 Volumenvisualierungspipeline

Abbildung 3: Die Visualisierungspipeline für direktes Volumenrendering

Die Volumenvisualisierungspipeline beschreibt den Ablauf von der Aufnahme der volumetrischen Daten bis hin zu deren dreidimensionalen Darstellung (Abbildung 3). Die Pipeline beginnt mit der Akquisition der Daten. Sie erfolgt durch externe Geräte, zum Beispiel in der Medizin durch CT MRT etc., und beschreibt im Wesentlichen die Erzeugung der Datensätze.
Für die Visualisierung wird der Volumendatensatz in ein Voxelgitter überführt. Je nach Blickrichtung und Abtastrate kann es beim anschließenden Sampling vorkommen, dass der Abtastpunkt auf dem verfolgten Strahl nicht genau auf einem der Gitterpunkte liegt. An solchen Stellen muss der Abtastpunkt mittels trilinearer Interpolation rekonstruiert werden. Bei einem GPU-basierten Verfahren kann dieser Schritt über eine 3D-Textur von der Grafikhardware übernommen werden.
Bei der Klassifikation erhalten die Voxel ihre Farbe und ihre Opazität. Dies geschieht in der Regel über eine Transferfunktion, worauf im nächsten Kapitel genauer eingegangen wird. Als Compositing wird die Integration der Strahlen entlang der Blickrichtung bezeichnet. Dadurch erhält man an jeder abgetasteten Stelle einen Farbwert, der anhand eines optischen Modells verrechnet und zuletzt zu der fertigen Darstellung zusammengefügt wird.

2.4 Transferfunktion

Eine Transferfunktion erfolgt über eine bzw. vier Lookup-Tabellen (eine Tabelle pro Koeffizienten RGBA), bei der jedem Skalarwert eine bestimmte Farbe und Opazität zugeordnet wird. Sollte es für einen Skalarwert keinen Eintrag in der Tabelle geben, wird der nächstliegende Farb- und Opazitätswert dafür verwendet (clampen) [3].
Die Vorgehensweise einer Transferfunktion soll nun anhand medizinischer Beispiele veranschaulicht werden. Wie oben erwähnt, enthalten die Aufnahmen von Computertomographien Dichtewerte. Für eine einfache Darstellung verwendet man eine Lookup-Tabelle, bei der die Dichtewerte als RGBA-Wert interpretiert werden, d.h. dass sich die resultierende Darstellung auf Grauwerte beschränkt. Da auch im Alphakanal die Dichteinformation enthalten ist, variiert aufgrund des optischen Modells der Einfluss der einzelnen Dichtewerte auf die Farbgebung der fertigen

Visualisierung. Daraus folgt, dass dichteres Gewebe für mehr Opazität sorgt, was auch im Umkehrschluss gilt (Kapitel 2.5).
Es besteht auch die Möglichkeit, die Transferfunktion im Nachhinein zu verändern, wenn man beispielsweise in den Körper hineinsehen möchte, ohne die davorliegenden Daten aus dem Volumen zu entfernen, was zu einen möglichen Informationsverlust führen könnte oder wenn man Areale mit verschiedenen Dichtewerten unterschiedlich einfärben möchte, um sie besser zu unterscheiden. Dazu können verschiedene Methoden benutzt werden, um die Transferfunktion automatisch oder manuell zu bestimmen (VolVisVorlesung und Folien MedMax). Bei letzterem kann man zum Beispiel auf die Manipulation der Transferfunktion auf Basis der statistischen Häufigkeit der einzelnen Skalarwerte (Histogramm) zurückgreifen, welche auch in dieser Arbeit verwendet wird. Der Vorteil besteht darin, dass sich vielfach vorkommende Dichtewerte, wie zum Beispiel bei der Haut, schnell ermitteln lassen und man so deren Opazität reduzieren kann, um die dahinterliegenden Strukturen (Knochen) zu visualisieren. Allerdings ermöglicht diese Transferfunktion nur einfache Abbildungen, da die Abgrenzung gleicher Strukturen bzw. Dichtewerte, die eigentlich verschiedene Gewebearten repräsentieren, nicht möglich ist. Durch die Einbeziehung weiterer Unterscheidungskriterien, wie den Gradienten oder den Abstand von einer Gewebeart zur anderen, kann man eine bessere Darstellung erzielen [3] [5].

2.5 Volumenrenderingintegral

Das optische Modell kann je nach Verfahren oder Komplexität der Darstellung anders aussehen. Grundsätzlich wird der Weg des Lichtes durch das Medium betrachtet und es wird die Strahlungsenergie bestimmt, die beim Betrachter ankommt [3]. Dabei gibt es mehrere Faktoren, die zu beachten sind, die im Folgenden näher erläutert werden.
Von Absorption spricht man, wenn Licht auf dem Weg zum Betrachter verloren geht bzw. durch andere Objekte absorbiert wird. Sofern das Licht auf die anderen Objekte auftrifft, wird dieses nicht nur absorbiert, sondern auch in verschiedene Richtungen abgelenkt. Dieser Vorgang wird als Streuung bezeichnet. Bei der Emission wird jedes Objekt als selbstleuchtende Lichtquelle betrachtet. In anderen Worten: Die Emission beschreibt, wie viel Licht von einem Objekt „erzeugt“ wird. Schatten wird bei einer Visualisierung immer dann erzeugt, wenn der Weg zur Lichtquelle durch andere dichtere Materialien verdeckt wird.
Das hier vorgestellte optische Modell liegt in einer vereinfachten Form vor, da eine Licht-Streuung und die Berechnung von Schatten nicht berücksichtigt werden. Im Emissions-Absorptionsmodell (Abbildung 4) geht man davon aus, dass jedes Partikel im Volumen Licht emittiert, was wiederum auf dem Weg zum Betrach-

[5] Eine genauere Erläuterung würde allerdings den Rahmen dieser Arbeit überschreiten, weshalb an dieser Stelle auf [2] verwiesen wird.

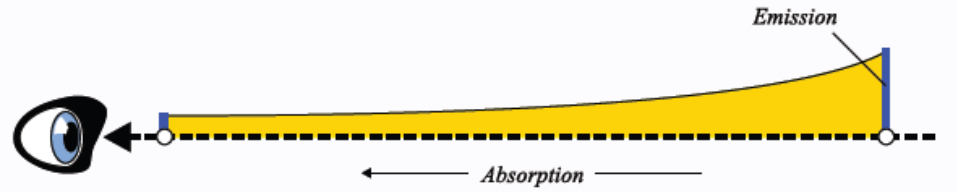

Abbildung 4: [3] Partikel emittiert Licht und erreicht abgeschwächt den Betrachter.

ter durch die Dichte der anderen Partikel absorbiert wird [3]. Die folgende Formel beschreibt diesen Vorgang:

$$C = \int_0^{Dmax} c(\delta) \cdot e^{-\int_0^{\delta} \kappa(\delta')d\delta'}$$

δ ist die Entfernung der aktuellen Position zum Augpunkt, $c(\delta)$ beschreibt die Eigenemission der aktuellen Position und $\kappa(\delta)$ den Wert der Absorption für die zu betrachtende Stelle auf dem Strahl.
Der Term innerhalb des Integrals beschreibt, wieviel Strahlung von dem Voxel beim Betrachter ankommt, wobei dieser Schritt für jede weitere Stelle auf dem Strahl durchgeführt wird. Dadurch, dass zwei Integrale in diesem Modell verwendet werden, nimmt die Berechnung der Emission und der Absorption zu viel Zeit in Anspruch. Eine Erweiterung der Formel für die Darstellung von Schatten und Streuung würde bedeuten, dass für jede Position entlang des Sehstrahls wiederum eine Integration entlang eines Lichtstrahls durchgeführt werden müsste, wobei dieser Vorgang sich für jede weitere Lichtquelle wiederholen würde. Für eine Streuung müsste sogar über die gesamte Umgebung integriert werden [3]. Die zusätzlichen Integrationen machen beide Verfahren zu aufwändig und rechenintensiv, sodass sie in Echtzeit kaum berechnet werden können.
Dadurch, dass beim Raycasting in Intervallen abgetastet wird, was als Diskretisierung bezeichnet wird, können die Integrale beseitigt werden, wodurch die Formel vereinfacht wird. Nach der Umformung der Formel nach [2], ergibt sich:

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

Durch die Diskretisierung kann man das Integral als Summe schreiben und den exponentiellen Term durch eine Multiplikation ersetzen. In dieser Formel beschreibt das α die Opazität, C die Farbe und d ist erneut die Entfernung der aktuellen Position zum Augpunkt. Darüber hinaus lässt sich diese Formel effizient durch Alphablending[6] implementieren:

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

[6]Das Alphablending ist eine Technik, bei der der Alphakanal bestimmt, wie transparent/opak ein Pixel dargestellt werden soll. Diese Technik wird meistens bei Überlagerungen von Bildern genutzt.

In dieser Arbeit wird diese Formel für das Raycasting-Verfahren verwendet, da sie auf das front-to-back-Sampling aufbaut. Hierbei werden die Strahlen von vorne nach hinten abgetastet, wodurch eine „early-ray termination“ (Kapitel 3.1.2) möglich ist. Diese kann in Verbindung mit dem Ray Textures-Konzept genutzt werden.

2.6 GPU-Raycasting

Beim Raycasting werden Strahlen durch das Volumen verfolgt und in gleichmäßigen Schritten abgetastet. Um die Richtung der Strahlen zu bestimmen, sind mehrere Vorverarbeitungsschritte nötig. Hierbei wird beim GPU-Raycasting zunächst eine Boundingbox in Form eines Würfels erstellt, bei der jede Ecke mit einer Farbe codiert wird, die für einen entsprechenden Vektor im Raum steht. Der Eckpunkt mit der Farbe rot (in RGB 1,0,0) beispielsweise beschreibt einen Vektor in X-Richtung. Die Grafikhardware interpoliert anschließend automatisch die Farben zwischen den Eckpunkten, sodass an jeder Stelle des Würfels eine Koordinate definiert wird. Der nächste Schritt besteht darin, sowohl die sichtbaren Seiten des Würfels (frontfaces) als auch die dahinterliegenden Seiten des Würfels (backfaces) zu rendern (Abbildung 5). Beide Ansichten werden daraufhin jeweils in eine 2D-Textur gespeichert und mit dem in einer 3D-Textur gespeicherten Datensatz an das Shader[7]-Programm übergeben. Durch die Subtraktion der beiden Würfelbilder und einer folgenden Normierung erhält man im Anschluss die benötigte Richtung. Da die Richtung pixelweise bestimmt wird und die Integration durch das Volumen ebenfalls pro Pixel erfolgt, eignet sich die parallele Verarbeitung der GPU besonders für das Raycasting-Verfahren und bringt so einen Leistungsvorteil gegenüber Varianten auf der CPU. Für die abschließende Integration verwendet man die Koordinaten des frontface-Würfelbildes als Startpunkt und tastet in vorbestimmten Abständen das Volumen in der erhaltenen Richtung ab. Die dort getroffenen Voxelfarben werden mittels der vorhin vorgestellten vereinfachten Gleichung (Kapitel 2.5) so lange akkumuliert, bis der Abtastpunkt das durch das backface-Bild bestimmte Ende erreicht hat (Abbildung 6). Seit Shadermodel 3.0 kann die Akkumulation mit Hilfe von Schleifenoperationen effizient implementiert werden, wodurch das Raycasting nicht mehr auf mehrfache Renderdurchläufe angewiesen ist.

[7] Shader sind programmierbare Einheiten auf der Grafikhardware. Man unterscheidet zwischen Fragment- und Vertex-Shader. Vertex-Shader manipulieren eingehende Vertex-Punkte, während Fragment-Shader die Bildpunkte verändern bzw. die Pixelfarbe bestimmen können.

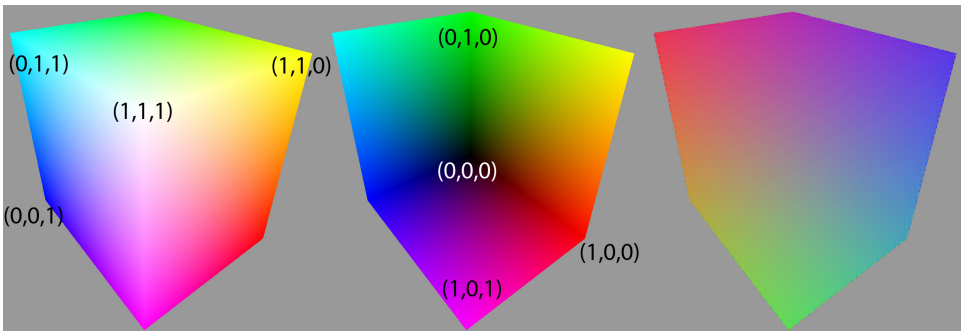

Abbildung 5: Frontface-Bild des Würfels links, Backface-Bild in der Mitte und rechts der Würfel mit der Richtung als Farbe dargestellt. In den Klammern stehen die RGB-Werte bzw. die Raumkoordinaten.

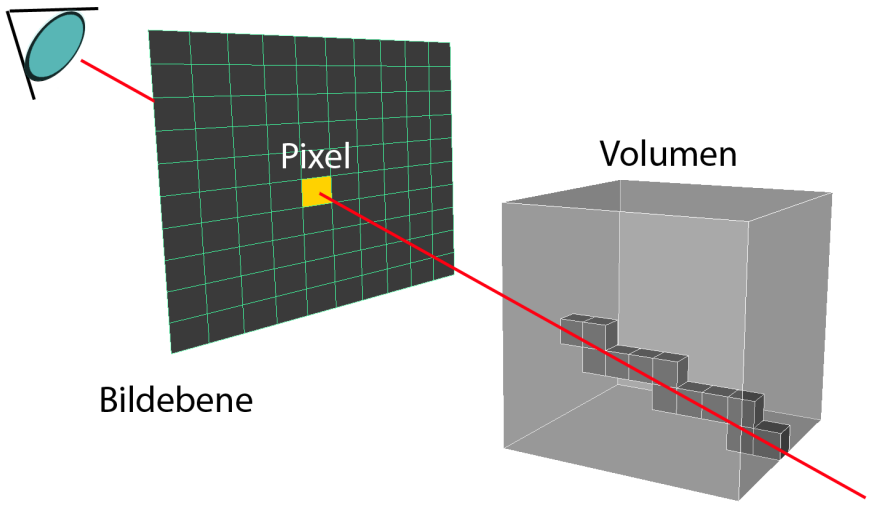

Abbildung 6: Vorgehensweise des Raycasting-Verfahrens: Es wird jeweils ein Strahl für jedes Pixel durch das Volumen verfolgt. Die getroffenen Voxel werden akkumuliert und ergeben dann die Farbe des Pixels.

3 Ray Textures

Ray Textures beschreibt ein Konzept, bei dem durch eingezeichnete Areale in Texturen die im Raycasting-Verfahren verwendeten Strahlen zur Abtastung des Volumens manipuliert werden. Die einzelnen Bereiche werden mittels Mausklick und Mausbewegung auf die Textur aufgetragen, ähnlich wie bei anderen Zeichenprogrammen. Dieses Konzept bietet eine vielfältige Interaktionsmöglichkeit mit dem Volumen und kann je nach Gebrauch bei der Visualisierung oder auch in der Rechenzeitoptimierung nützlich sein. Das Zeichnen auf der Textur bietet viele Freiheiten, sodass einzelne Strahlen oder das gesamte Volumen beeinflusst werden können. Weiterhin können verschiedene Parameter bestimmt werden, was dazu führt, dass die Areale jeweils andere Eigenschaften besitzen, die sich wiederum unterschiedlich auf das Raycasting-Verfahren auswirken.

3.1 Modi

Im folgenden Kapitel werden die in [1] vorgestellten Einstellungsmöglichkeiten besprochen. Allgemein kann man diese in geometrische und Werteeigenschaften unterteilen. Die geometrischen Eigenschaften verändern den verfolgten Strahl, während die Werteeigenschaften den Strahl im Verlauf der Integration kontrollieren. Genauer gesagt, manipulieren sie den Strahl mit Hilfe der ermittelten Werte des betrachteten Voxels. Welche Parameter für die einzelnen Eigenschaften wichtig sind, welchen Einfluss sie auf die Darstellung haben und welchen Vorteil diese Eigenschaften in Verbindung mit den Ray Textures haben, wird im folgenden Kapitel näher erläutert.

3.1.1 Geometrische Eigenschaften

Wie oben schon erwähnt, tastet man beim Raycasting den Sehstrahl in bestimmten Abständen ab. Je höher das Volumen abgetastet wird, wobei der Strahl in kleineren Schritten durchlaufen wird, desto feiner wird der Datensatz dargestellt und umgekehrt. Es ist allerdings nicht immer von Vorteil, eine überhöhte Abtastrate zu verwenden, da die Berechnung dadurch umso länger dauert. Durch die Ray Textures kann man die Berechnung auf wichtige Stellen verlagern und andere irrelevante Bereiche unberücksichtigt lassen.
Im vorangegangenen Kapitel wurde bisher immer davon ausgegangen, dass man durch das gesamte Volumen integriert, was aber nicht zwingend notwendig ist. Es ist durchaus möglich, den verfolgten Strahl auf einen bestimmten Bereich zu beschränken. Dazu werden zunächst ein neuer Einstiegs- und Ausstiegspunkt definiert (in Abbildung 7 als „in“ und „out“ gekennzeichnet), die sich normalerweise am Anfang und am Ende der Boundingbox befinden. Verändert man die Lage des Einstiegspunktes nach hinten, wird der Strahl erst ab dieser Stelle verfolgt. Bei einer Verschiebung des Ausstiegspunktes weiter nach vorne, wird ab dieser Stelle der Abtastvorgang abrupt beendet. Die so erzeugte Darstellung erlaubt einen Einblick

in das Volumen und kann ebenfalls dazu verwendet werden, einzelne Schichtenbilder zu visualisieren. Ein weiterer Vorteil besteht darin, dass durch das Überspringen der vorderen und hinteren Voxel Rechenzeit eingespart wird, da keine Berechnungen am Volumen vorgenommen werden müssen, um die davor- und die dahinterliegenden Daten zu entfernen. Weiterhin ist es möglich, den Strahl in mehrere Intervalle zu unterteilen. Die Farb- bzw. Opazitätsinformationen der Voxel zwischen den Intervallen werden ignoriert und haben dadurch keinen Einfluss auf die resultierende Darstellung.
Beide geometrischen Eigenschaften lassen sich nach Belieben miteinander kombinieren. So kann die Samplerate innerhalb der Intervalle erhöht oder erniedrigt werden. Durch das Markieren verschiedener Bereiche können sogar die Samplerate und die Anzahl/Länge der Intervalle von Strahl zu Strahl unterschiedlich ausfallen, was in Abbildung 7 vereinfacht mit zwei Strahlen dargestellt wird. Es geht aus dem Paper allerdings nicht hervor, wie genau die Samplerate eingestellt wird und wie die Intervalle gesetzt bzw. gesteuert werden und ob diese zur Laufzeit veränderbar sind. Ein Ansatz für eine flexible Steuerung dieser Eigenschaften wird im Kapitel 4 vorgestellt.

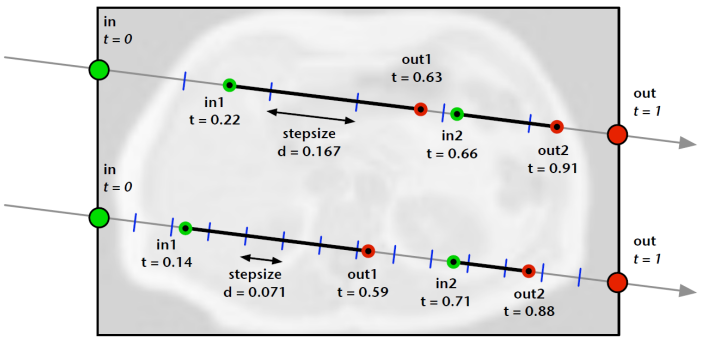

Abbildung 7: [1] Kombination zwischen den geometrischen Eigenschaften, Abtastrate und Intervall vereinfacht dargestellt mit nur zwei Strahlen. Beide Strahlen haben verschiedene Abtastraten, gekennzeichnet durch die vertikalen blauen Balken auf dem Strahl. Zusätzlich besitzen sie zwei Intervalle, die sich wiederum durch ihre Länge unterscheiden.

3.1.2 Werteeigenschaften

Zu den Werteeigenschaften gehört das Schwellenwertverfahren, das in der Bildverarbeitung genutzt wird, um Objekte oder zusammenhängende Bereiche in einem Bild zu segmentieren. Dazu definiert man zunächst für ein Grauwertbild einen Grenzwert. Dieser wird mit den Pixeln im Bild verglichen, wobei ausschließlich Grauwerte, die größer oder gleich dem Grenzwert sind, zum suchenden Bereich gehören [6]. Es gibt nun eine Möglichkeit, dieses Verfahren in den Raycasting-Kontext zu bringen, der als alternativer Rendermodus genutzt werden kann. Hierbei

wird jeder Strahl so lange abgetastet, bis der erste Wert, der dieses Kriterium erfüllt, gefunden wurde. Diese Methode stellt den Datensatz nur ab einen bestimmten Skalarwert dar; alle anderen Werte fließen nicht in die Berechnung mit ein. Dadurch lassen sich in medizinischen Datensätzen, ähnlich wie in der Bildverarbeitung, einheitliche Strukturen segmentieren, was vor allem bei Knochen gute Resultate liefert. Eine weitere Methode, die auf einem Grenzwert basiert, ist die sogenannte „early-ray termination“. Diese wird beim back-to-front-Sampling genutzt, um das Abtasten des Strahls vorzeitig abzubrechen und dadurch Rechenzeit einzusparen. Der Opazitätswert wird hierzu während der Iterationen mit dem Grenzwert verglichen und sofern eine Überschreitung stattgefunden hat, wird die Schleife zum Abbruch gezwungen. Normalerweise wird so lange iteriert, bis der maximal mögliche Opazitätswert (Alphawert = 1) erreicht worden ist [2]. Allerdings würde auch bei einem kleineren Grenzwert kein signifikanter Unterschied in der Darstellung zu erkennen sein. Durch die Ray Textures ist es möglich, den Grenzwert manuell zur Laufzeit einzustellen und anschließend selbst zu beurteilen, ab welchen Wert die Berechnung gestoppt werden soll.
Zu den Werteeigenschaften zählen außer dem Schwellwertverfahren noch weitere Render-Varianten, wie beispielsweise die „maximum intensity projection (MIP)“ oder die Anwendung verschiedener Transferfunktionen auf den Datensatz. Bei der „maximum intensity projection“ wird jeweils nur der maximal vorkommende Voxelfarbwert pro Strahl angezeigt. Dieses Verfahren wird beispielsweise genutzt, um in medizinischen Datensätzen Blutgefäße hervorzuheben [2].
Die Ray Textures erlauben es auch hier, jeden Bereich auf der Textur individuell zu gestalten, sodass beispielsweise das MIP-Verfahren zur Visualisierung der Blutgefäße verwendet werden kann, während der Rest der Darstellung für die Visualisierung der Knochen über das Schwellwertverfahren genutzt wird. Weiterhin ist auch eine Kombination untereinander möglich, beispielsweise der Gebrauch einer Transferfunktion mit zusätzlichem Einsatz der „early-ray termination“ und ferner die Kombination der beiden Eigenschaftengruppen. Man kann zum Beispiel die Transferfunktion nur in einem bestimmten Intervall definieren oder die Samplerate mit dem Schwellenwert kombinieren, um eine höhere Auflösung zu bekommen. Durch die eingezeichneten Bereiche auf den Texturen und die Eingrenzung der Abtastpunkte durch die Intervalle, ist sogar eine dreidimensionale Manipulation des Volumens möglich.

3.2 Kodierung, Pinselform und -größe

Die Pinselgröße ist ein wichtiger Faktor beim Ray Textures-Konzept, da auf diese Weise entschieden wird, wie viele Strahlen letztendlich während des Raycastings verändert werden. Je größer der Pinsel, desto mehr Strahlen sind davon betroffen und umgekehrt. Dieses Kapitel beschäftigt sich daher mit der Erstellung der Textur und ferner wie diese durch den Pinsel verändert wird und die einzelnen Eigenschaften gespeichert werden.

Die Erstellung der Ray Textures erfolgt über ein zweidimensionales Array auf der CPU, das für jedes Pixel im Bild einen Wert beinhaltet. Je nachdem, auf welche Stelle der Benutzer klickt, wird das Array an diesen Stellen editiert und im Anschluss an die GPU übergeben. Aufgrund der langsamen Kommunikation zwischen GPU und CPU wird das Programm bei jeder Abänderung der Textur durch das erneute Hochladen abgebremst. Damit die Performance bei der Übergabe der Textur an die GPU nicht so stark sinkt, wird weiterhin eine hierarchische Methode verwendet. Hierbei wird die Textur mit Hilfe von MipMaps[8] verkleinert, um die Datenmenge zu reduzieren. Bei einer MipMap der Stufe 5 repräsentiert beispielsweise ein Pixel einen Bereich bzw. eine Pinselgröße von 32x32. Dadurch kann die Pinselgröße entsprechend der MipMap-Stufe variieren. Voraussetzung der MipMap ist allerdings, dass die Dimensionen der Textur einer Zweierpotenz unterliegen müssen, d.h. dass nur Vierecke auf die Textur gezeichnet und richtig dargestellt werden können.
Die Information über die Eigenschaften (siehe Kapitel 3.1), die angewendet werden sollen, müssen ebenfalls in der Textur abgespeichert werden. Dazu werden zwei Alternativen vorgeschlagen. Zum einen kann man für jede Eigenschaft eine Textur erstellen oder man kodiert die Informationen in den einzelnen Farbkanälen. Dies funktioniert aber nur, solange man nicht mehr als vier Eigenschaften (vier da RGBA jeweils ein Kanal) kodieren will, da ansonsten auf eine weitere Textur zurückgegriffen werden muss.

3.3 Mapping

Das Mapping beschäftigt sich damit, wie die Textur aus den Ray Textures auf die dreidimensionale Darstellung projiziert wird. In [1] werden dafür zwei Möglichkeiten vorgestellt, die im folgenden Kapitel näher erläutert werden.

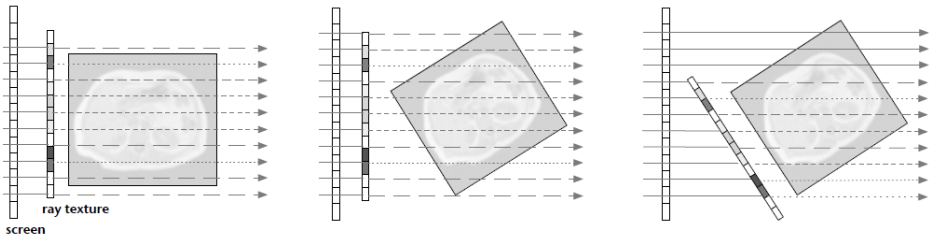

Abbildung 8: [1] Verhalten der Raytextures bei der Verwendung der zwei Mapping-Varianten: links die initiale Ansicht, „View aligned“ in der Mitte und rechts „Volume aligned“. Man beachte, dass die gestrichelten Linien verschiedene Sampleraten repräsentieren.

[8] Bei MipMaps handelt es sich um Texturen, die in verschiedenen Auflösungen vorliegen. MipMaps werden eigentlich dazu genutzt, um den Detailgrad der Textur je nach Sichtbarkeit in einer Szene anzupassen.

3.3.1 View aligned

Unabhängig wie sich die Boundingbox und somit auch die dreidimensionale Visualisierung dreht, bleibt die Textur immer im Vordergrund an einer festen Position. Diese Art von Mapping ähnelt einem Fenster, bei dem die Einstellungen auf jede Ausrichtung des Volumens gleich angewandt werden (Abbildung 8). Die Oberfläche des Würfels wird dabei allerdings nicht beachtet, was dazu führt, dass Strahlen verändert werden können, die man eigentlich nicht verändern möchte. Außerdem kann man mit dieser Methode Areale außerhalb der Boundingbox einzeichnen, die aber keinen Einfluss auf das Rendering haben. Ein intuitiverer Ansatz, bei dem sich die Textur an den Würfel anpasst, wird im Kapitel 4.5.1 vorgestellt.

3.3.2 Volume aligned

Beim Volume aligned wird die Textur sozusagen auf eine Seite der Boundingbox geklebt. Auf diese Weise wandert die Textur mit jeder Drehung des Würfels mit. Die markierten Areale, die zur Veränderung der Darstellung beitragen, drehen sich ebenfalls mit, sodass sie immer nur Einfluss auf die gleiche Stelle nehmen (Abbildung 8). Damit sich die Textur mit dem Würfel mitdreht, werden die Startpunkte aus dem front-face-rendering-Bild genommen und als Texturkoordinaten für die Ray Texture verwendet. Allerdings kann man dadurch nur Bereiche auf eine Seite zeichnen, da die Ray Texture immer an der Vorderseite des Würfels gehängt wird. Sinnvoller wäre es, wenn jede Seite der Boundingbox manipulierbar wäre, damit man jeder Seite andere Eigenschaften zuweisen könnte und so mehr Freiheiten bei der Gestaltung möglich wären. Die Erweiterung, die dafür sorgt, dass auf allen Seiten der Boundingbox gezeichnet werden kann, wird in Kapitel 4.5.2 erläutert.

4 Implementation

In diesem Kapitel wird die Implementation des Raycastings und der „Ray Textures“ im Detail gezeigt. Der Schwerpunkt wird dabei auf die Modi, die Pinselform und -größe und das Mapping gelegt, wobei die entwickelten Erweiterungen noch einmal genauer erläutert werden. Darüber hinaus wird besonders auf die Fragmentshader-Programme[9] eingegangen, da die GPU ausschließlich mit solchen Programmen arbeitet. Zunächst werden aber die Oberfläche und die entwickelten Hilfsmöglichkeiten zur flexibleren Interaktion präsentiert und zusätzlich technische Informationen angesprochen, damit man im Nachhinein die Kodierung der einzelnen Modi und Parameter in der Textur besser nachvollziehen kann.

4.1 Grafische Benutzeroberfläche

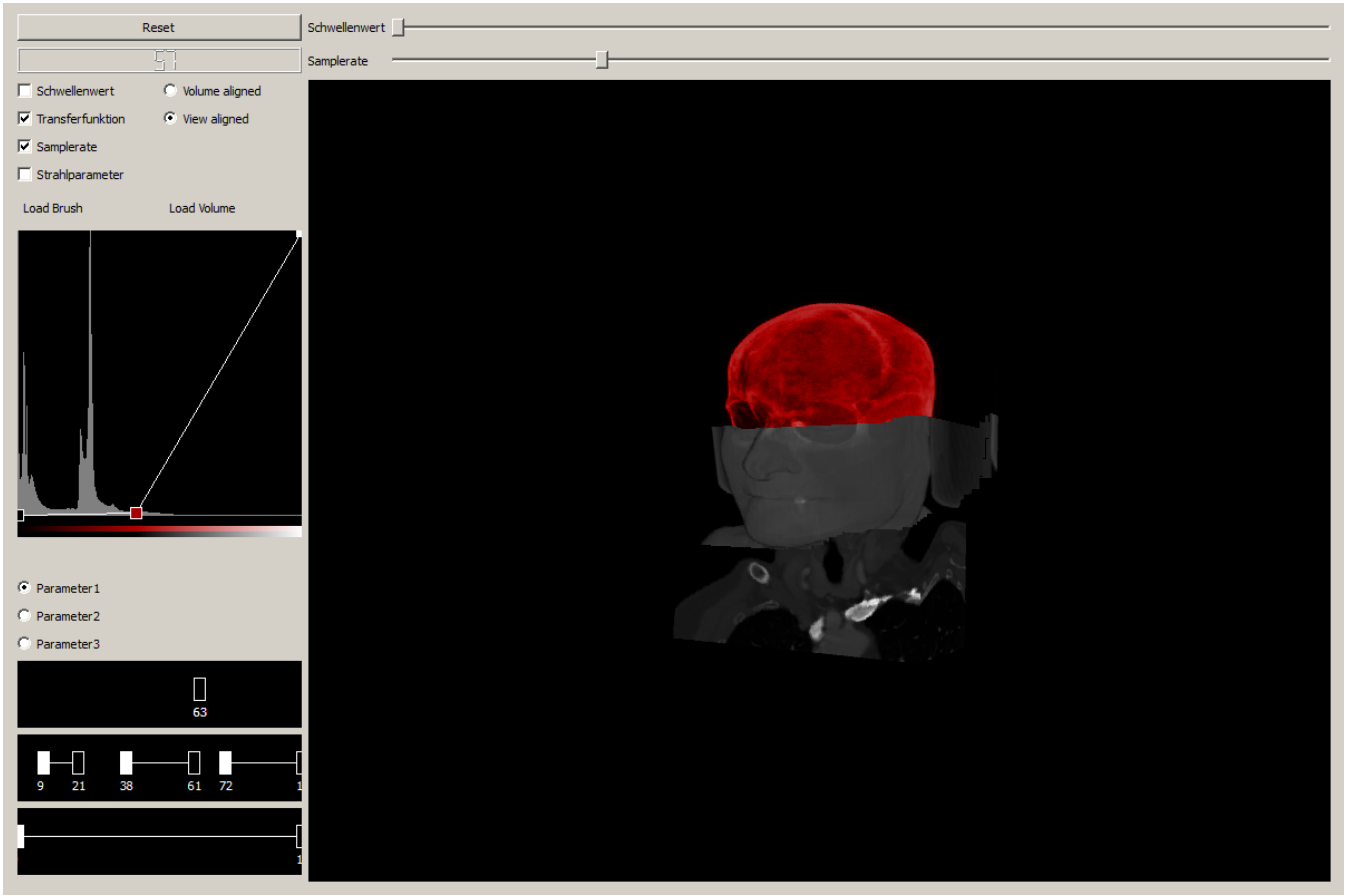

Abbildung 9: Die Benutzeroberfläche.

[9] Als Shaderhochsprache wurde in dieser Arbeit GLSL verwendet.

Oberhalb der dreidimensionalen Darstellung ist das Einstellen des Schwellenwertes und der Samplerate mittels der beiden Schieberegler möglich. Damit man den Überblick über den aktuellen Wert behält, wird dieser unter dem Reset-Knopf angezeigt. Die oben erwähnten Modi Samplerate, Schwellenwert (wobei hier der aktuelle Wert aus dem Schieberegler übernommen wird), Transferfunktion und Intervall können über die Checkboxen selektiert und durch das gegenseitige Anwählen kombiniert werden. Es wird immer der aktuelle Stand der Modi und der Schieberegler beim Zeichnen an den Shader übergeben. Mit den rechts davon liegenden Radiobuttons kann zwischen den zwei Mapping-Varianten „Volume aligned“ und „View aligned“ umgeschaltet werden. Klickt man auf die darunter positionierten Buttons „Load Volume“ oder „Load Brush“, öffnet sich ein entsprechendes Dialog-Fenster, in dem man neue Pinsel oder Volumen auswählen und laden kann. Das Laden der Volumen beschränkt sich allerdings auf das Einlesen medizinischer Datensätze, wobei die repräsentativsten Dateiformate DICOM[10] und RAW unterstützt werden. Da es sich bei dem DICOM-Format um eine eher komplexere Dateistruktur handelt, wurde auf den Dateilader aus dem „Cascada-Framework“[11] zurückgegriffen, der das Volumen und weitere wichtige, beispielsweise statistische Informationen über eine Schnittstelle zur Verfügung stellt.
Unterhalb dieser beiden Buttons befindet sich die Komponente zur Bestimmung der Transferfunktion. Die grauen Balken im Hintergrund repräsentieren das Histogramm, während die Transferfunktion durch die Verbindung der Knoten definiert wird. Es lassen sich mehrere Knoten hinzufügen und verschieben, wobei die Position auf der Y-Achse den Alphawert und die Position auf der X-Achse die Farbe bestimmt. Durch die anschließende Interpolation der Werte von Knoten zu Knoten entsteht eine Look-up-Tabelle, die daraufhin im Shader verwendet wird.
Der untere Bereich dient zur Einstellung der Intervalle, wobei über die Radiobuttons zwischen drei Intervallreglern gewechselt werden kann. Aufgabe der Knoten ist die Definition der Intervalle. Dazu wird die prozentuale Position auf dem Strahl unterhalb jedes Knotens angezeigt. Die Farbe der Knoten gibt Aufschluss darüber, ob es sich um einen Eintritts- (weiß) oder um einen Austrittspunkt (schwarz) handelt. Ähnlich wie bei normalen Schiebereglern lassen sich die Knoten waagerecht nach links und rechts bewegen, wobei darauf geachtet wird, dass ein Eintrittspunkt nicht den Austrittspunkt überschreitet und umgekehrt, da die Verbindung zweier Knoten (jeweils ein Eintritts- und Austrittspunkt) den Bereich beschreibt, auf dem der Strahl abgetastet wird. Darüber hinaus lässt die Verbindung darauf schließen, welche Voxel für die Berechnung der Darstellung genutzt werden. Des weiteren ist es auch möglich, das Intervall auf dem Strahl auf eine Stelle zu beschränken, indem man den Eintritts- und Austrittspunktes gleichsetzt, was dazu führt, dass einzelne Schnittbilder angezeigt werden. Weiterhin können mehrere Intervalle pro Intervallregler eingefügt werden, allerdings beschränkt sich die Anzahl auf acht Intervalle,

[10] DICOM steht für „Digital Imaging and Communications in Medicine“ und ist ein offener Standard zur Speicherung radiologischer Daten in der Medizin. [7]

[11] „Cascada“ ist ein an der Universität Koblenz entwickeltes GPU-basiertes System zur Visualisierung und Bearbeitung medizinischer Daten.

da die Informationen über die Knotenpositionen in einer 4x4 Matrix an den Shader geschickt werden.

4.2 Raycasting-Shader

```
1  uniform sampler2D back;
2  uniform sampler2D front;
3  uniform sampler3D volumen;
4  uniform vec2 winSize;
5  vec3 direction;
6
7  void main(){
8    vec2 fragCoord = vec2(gl_FragCoord.x/winSize.x, gl_FragCoord.y/winSize.y);
9    vec3 rayStart = texture2D(front,fragCoord).rgb;
10   vec3 rayEnd = texture2D(back,fragCoord).rgb;
11
12   vec3 vector =  rayEnd - rayStart;
13   float maxLength = length(vector);
14
15   float stepSize = 0.001;
16   vec3 step = direction * stepSize;
17   direction = normalize(vector);
18   vec3 value;
19
20   vec4 dst = vec4(0.0,0.0,0.0,0.0);
21   vec4 src;
22   vec3 position = vec3(0.0,0.0,0.0);
23
24   for(int i=0; i<(1/stepSize); i++){
25
26     if( length(position) >= maxLength || dst.a >=1.0){
27             dst.a = 1.0;
28             break;
29     }
30
31     value = texture3D(tex3,position + rayStart).rgb;
32     src = vec4(value.r,value.r,value.r,value.r);
33
34     dst.rgb += (1.0 - dst.a) * src.a * src.rgb;
35     dst.a += (1 - dst.a) * src.a;
36
37     position += step;
38
39   }
40   gl_FragColor = dst
41 }
```

In Zeile 1-4 stehen zunächst die übergebenen Texturen: die Vorderseite der Boundingbox, die Rückseite und der Volumendatensatz in einer 3D-Textur. Die Größe des Fensters wird zusätzlich vom Programm an den Shader übergeben. An dieser Stelle muss noch einmal detailierter auf die GPU eingegangen werden. Da die Grafikhardware darauf ausgelegt ist, Geometrie (Dreiecke, Polygone etc.) darzustellen

und zu berechnen, muss in der üblichen GPGPU-Vorgehensweise[12] ein Viereck über den gesamten Viewport gezeichnet werden, damit der Fragmentshader für jeden Pixel angewandt wird. Anhand dieses Vierecks lässt sich die Pixelposition in Bildschirmkoordinaten durch den Befehl gl_FragCoord bestimmen. Der Zugriff auf die übergebenen Texturen (Zeile 9-10) findet allerdings im Bereich zwischen (0,0) (unten links) und (1,1) (oben rechts) statt. Daher muss noch durch die Gesamtgröße des Fensters in X- und Y-Richtung geteilt werden. Durch die so gewonnene Texturposition lässt sich die Farbe in den Würfelbildern bestimmen, wodurch man den Anfangs- und Endvektor der Strahlrichtung erhält. In Zeile 14 wird die Richtung in „direction“ gesichert, die vorher mittels der Subtraktion der beiden Vektoren (Zeile 12) und der anschließenden Normierung (Zeile 17) berechnet wird. Die Initialisierung der Abtastrate, hier mit einem Wert von 0.001, erfolgt in Zeile 15. Die Samlplerate wird beim Raycasting statisch festgelegt und gilt somit für das gesamte Volumen. Davon abhängig ist die Anzahl der Schleifendurchläufe, die dafür sorgen muss, dass jeder Strahl immer komplett abgetastet wird. Dabei orientiert man sich an den Diagonalen der Boundingbox, an denen die Strecke des Strahls am längsten ist. „1/stepSize“ besagt demnach, dass an den Diagonalen in diesem Fall 1000 Iterationen nötig sind, um den gesamten Strahl abzulaufen. Jedoch ist es nicht sinnvoll, jeden Strahl mit 1000 Iterationen abzutasten, da die Strahlen an den anderen Stellen kürzer sind und somit das Volumen überschritten und die Strahlen in einem undefinierten Bereich weiterverfolgt werden würden. Zusätzlich hätte das Auswirkungen auf die Rechenzeit des Programms, die dadurch schnell an ihre Grenzen stoßen würde. Daher wird in Zeile 26 die Länge der aktuellen Position auf dem Strahl mit der maximalen Länge, die sich aus den beiden Würfelbildern ergibt, verglichen. Kommt es zu einer Überschreitung, wird die Schleife zum vorzeitigen Abbruch gezwungen. Als weiteres Abbruchkriterium in derselben Zeile dient die „early-ray termination“, die immer dann den Abbruch garantiert, wenn der Alphakanal bereits den maximalen Wert (eins) erreicht hat. Eine weitere Berechnung würde in diesem Fall keinen Unterschied in der Darstellung machen. Anschließend wird in „src“ (Zeile 32) die Farbe des aktuellen Voxels gespeichert. Sollte der Strahl nicht genau ein Voxel treffen, wird über die Grafikhardware trilinear interpoliert, d.h. dass aus den umliegenden Voxel der Farbwert bestimmt wird. Als Transferfunktion dient hier eine einfache Darstellung, bei der jeder Skalarwert als RGBA-Wert interpretiert und danach mit dem optischen Modell in front-to-back Richtung verrechnet wird. Die hier verwendete Formel unterscheidet sich geringfügig vom Volumenrenderingintegral, das in Kapitel 2.5 vorgestellt wurde, da eine Opazitätskorrektur [2] mit in die Berechnung einfließt (Zeile 34-35). Der Unterschied besteht darin, dass bei dieser Variante die Darstellung opaker und realitätsnäher wirkt, während die Formel ohne Berücksichtigung der Opazität eine röntgenartige Darstellung erzeugt. Die Idee des Emission-Absorptions-Modells

[12] Mit GPGPU (General Purpose Computation on Graphics Processing Unit) bezeichnet man die Verwendung der Grafikhardware für Berechnungen, die über eine Darstellung der Geometrie hinausgehen.

bleibt davon aber unberührt. Das Ergebnis der Iterationen steht in der Variablen „dst“, deren Wert nach der Schleife für den Pixel eingetragen wird.

4.3 Pinsel-Shader

```
1  uniform sampler2D rayTextureOld;
2  uniform sampler2D mask;
3  uniform vec2 pos;
4  uniform vec2 maskSize;
5  uniform vec2 winSize;
6  uniform vec4 modi;
7  uniform float schwellenwert;
8  uniform float samplerate;
9  uniform vec3 rayPar;
10
11 void main(){
12         vec2 fragCoord = vec2(gl_FragCoord.x /winSize.x,gl_FragCoord.y / winSize.y);
13         vec4 rayTexture = texture2D(rayTextureOld,fragCoord);
14         vec2 fraghlp = vec2(gl_FragCoord.x, winSize.y - gl_FragCoord.y);
15
16         float xP = pos.x + maskSize.x/2.0;
17         float xM = pos.x - maskSize.x/2.0;
18         float yP = pos.y + maskSize.y/2.0;
19         float yM = pos.y - maskSize.y/2.0;
20
21         if((fraghlp.x < xP && fraghlp.y < yP) && (fraghlp.x > xM && fraghlp.y > yM)){
22
23                 float xValue = (fraghlp.x - pos.x + maskSize.x/2.0)/maskSize.x;
24                 float yValue = (fraghlp.y - pos.y + maskSize.y/2.0)/maskSize.y;
25                 vec4 maskValue = texture2D(mask, vec2(xValue,yValue));
26
27                 float m1=0.0; float m2=0.0; float m3=0.0; float m4=0.0;
28
29                 if(modi.r == 1.0)
30                         m1 = samplerate;
31                 if(modi.g == 1.0)
32                         m2 = schwellenwert;
33                 if(modi.b == 1.0)
34                         m3 = 1.0;
35                 if(modi.a == 1.0){
36                         if(rayPar.r == 1.0)
37                                 m4 = 0.1;
38                         if(rayPar.g == 1.0)
39                                 m4 = 0.5;
40                         if(rayPar.b == 1.0)
41                                 m4 = 0.7;
42                 }
43
44                 if(maskValue.r > 0.0 && maskValue.g > 0.0 && maskValue.b > 0.0){
45                         gl_FragColor = maskValue * vec4(m1,m2,m3,m4);
46                 }else
47                         gl_FragColor =  gl_Color + rayTexture;
48         }else
49                 gl_FragColor =  gl_Color + rayTexture;
50 }
```

Die übergebenen „uniform-Variablen“ aus dem Hauptprogramm teilen dem Shader den aktuellen Zustand der einzelnen Parameter mit sowie die Position an der

gezeichnet werden soll und die Form des Pinsels. Bevor aber die einzelnen Parameter in der Textur kodiert werden, müssen einige Berechnungen durchgeführt werden, um den entsprechenden Wert aus der Pinseltextur und die aktuelle Position im Bildschirm zu bestimmen.
Die übergebene Position ist der Mittelpunkt, an der der Pinsel angewandt werden soll. Um jedoch den gesamten Pinselbereich zu definieren, werden dafür die Koordinaten für zwei Punkte ermittelt, die diesen Bereich durch ein Viereck abdecken. Dies geschieht, indem man die Hälfte der Pinselgröße in X- und Y-Richtung vom Mittelpunkt aus nach rechts hinzugefügt und nach links abzieht (Zeile 16-19). Durch die unterschiedliche Interpretation der Bildschirmkoordinaten muss zusätzlich beachtet werden, dass die Y-Koordinate der übergebenen Position gespiegelt werden muss, damit sie mit der Y-Koordinate aus dem „gl_FragCoord“-Befehl übereinstimmt (Zeile 14). Ist der Bereich für den Pinsel definiert, sorgt die Abfrage in Zeile 21 dafür, dass nur im Bereich des Pinsels weitere Berechnungen stattfinden. Hier muss nun für jede Stelle die Position in der Pinseltextur ermittelt werden, die, wie vorher bereits erwähnt, bei den hier verwendeten Texturen zwischen (0,0) und (1,1) liegt. Dazu wird der Abstand zwischen der aktuellen Pixelposition und dem Mittelpunkt berechnet, wobei dieser Wert durch die Pinselgröße dividiert wird, wodurch man in den Texturwertebereich gelangt. Auf diese Weise erhält man den Wert in der Pinseltextur, der im Anschluss in der „maskValue“-Variablen gespeichert wird (Zeile 25).
In Zeile 29-42 werden die einzelnen Modi ausgewertet und die entsprechenden Werte in den Variablen m1-m4 gespeichert. Bei der Samplerate und dem Schwellenwert beziehen sich diese auf den aktuellen Stand der Schieberegler, während bei der Transferfunktion nur entscheidend ist, ob diese benutzt werden soll (m3 gleich eins) oder nicht (m3 gleich null). Des Weiteren muss bei der Kodierung der Intervalle noch gewährleistet werden, dass zwischen den drei Intervallreglern unterschieden werden kann, was über eine Kodierung innerhalb des Farbwertes geschieht. Hierfür wird in „rayParam“ nachgeschaut, welcher Intervallregler gerade aktiv ist und anhand dieser Information wird ein Wert (0.1, 0.5 und 0.7) für die drei verschiedenen Regler in m4 gespeichert.
Da der Pinselbereich immer als Viereck definiert wird, aber nicht nur Vierecke als Pinselform darstellbar sein sollen, wird der Pinsel als eine Art Schablone benutzt. Die Idee besteht darin, dass Werte, die gleich null sind, ignoriert und alle Werte ungleich null dargestellt werden. Dadurch, dass nicht alle Werte abgebildet werden, kann der Pinsel eine beliebige Form annehmen, sogar eine die Löcher beinhalten.
Bisher wurde nur auf die Erstellung eines Pinselabdruckes eingegangen. Um zu verstehen, wie der komplette Mal-Ablauf funktioniert, muss noch einmal näher auf die Vorgehensweise des Shaders eingegangen werden. Der Pinsel-Shader wird für jedes erneute Zeichnen aufgerufen. Damit die bestimmten Areale aus den älteren Durchgängen nicht verloren gehen, erhält der Shader bei jedem Aufruf ebenfalls eine Textur (rayTexture), die diese Informationen enthält. Diese ältere Textur wird immer dann als Pixelwert eingetragen, wenn der Pinsel nicht angewandt werden soll (Zeile 49) oder der „maskValue“-Wert null entspricht (Zeile 47). Ist der

„maskValue“-Wert nicht null, wird er mit einem Vektor multipliziert, der aus den ausgewerteten Modi m1-m4 besteht. Dieser Vorgang entspricht der Kodierung in den vier Farbkanälen, die anschließend als Pixelfarbe ausgegeben wird. Sofern der Wert in „maskValue“ eins beträgt, werden die kodierten Werte vollständig dargestellt. Durch die Multiplikation sind aber auch Grauwerte und Farben in der Pinseltextur erlaubt. Die Variablen m1-m4 werden dann je nach Wert in den einzelnen Kanälen verändert.

4.4 Modi

Durch die Kombinationsmöglichkeiten der Modi vermischen sich einige Programmteile, wodurch eine Erklärung erschwert werden würde. Daher wird zunächst auf die Implementation der einzelnen Modi eingegangen und zuletzt das komplette Shaderprogramm erläutert. Weiterhin werden ausschließlich die Änderungen zum normalen Raycasting aufgezeigt.

4.4.1 Samplerate-Shader

```
1  ...
2  void main(){
3          ...
4          vec4 rayTexture = texture2D(rayTexture2,fragCoord);
5          float stepSize;
6
7          if(rayTexture.r > 0.0){
8                  stepSize = (1.0/(1000.0*rayTexture.r));
9          }else{
10                 stepSize = 0.01;
11         }
12         step = direction * stepSize;
13         for(int i = 0; i< (1/stepSize); i++){
14         ...
15         }
16         ...
17 }
```

Der Rot-Kanal der Ray Texture gibt Aufschluss darüber, welche Strahlen durch die Samplerate verändert werden sollen. Zunächst wird daher der Wert im Rot-Kanal pro Pixel abgefragt, wobei Werte größer null eine Veränderung der Samplerate zur Folge haben. Zusätzlich ist im Rot-Kanal die Information enthalten, um wieviel die Abtastrate verstellt werden soll. Dazu muss allerdings der Wertebereich noch umgerechnet und invertiert werden, da der kodierte Farbwert im Bereich zwischen 0.0 und 1.0 liegt und der Wert eins die höchstmögliche Abtastrate darstellt. Durch diese Umrechnung ergeben sich anschließend eine minimale Samplerate von 1 und eine maximale Samplerate von 0.001, welche sowohl eine Verminderung als auch eine Erhöhung der Abtastrate zur Laufzeit erlauben.
Für alle Pixel, bei denen die Abtastrate nicht verändert werden soll, ist ein Standardwert von 0.1 vordefiniert (Zeile 10). Dieser Wert wurde bewusst niedrig gewählt, damit beim Start des Programms ein grober Umriss zu erkennen ist und so

im Nachhinein eine Weiterverarbeitung mit dem Objekt vereinfacht wird. Somit wird dem Benutzer überlassen, welche Stellen er für interessant hält und feiner darstellen möchte, ohne von Anfang an die Performance zu mindern.

4.4.2 Schwellenwert und MIP-Shader

```
1  void main(){
2          vec4 rayTexture = texture2D(rayTexture2,fragCoord);
3          float max = 0.0;
4
5          for(int i = 0; i< (1/stepSize); i++){
6                  if( length(position) >= maxLength || dst.r >=1.0){
7                          //Nur Bei Schwellenwert
8                          if(src < rayTexture.g && length(position) >= maxLength){
9                                          src = 0.0;
10                         }
11                         break;
12                 }
13                 ...
14                 src = texture3D(tex3,position + rayStart);
15
16                 //Für Schwellenwert
17                 if(src > rayTexture.g && rayTexture.g > 0.0){
18                         break;
19                 }
20                 //Für MIP
21                 if(rayTexture.g > 0.0){
22                         if(src > max) max = src;
23                 }
24         }
25         //Für Schwellenwert
26         gl_FragColor = vec4(src,src,src,src);
27         //Für MIP
28         gl_FragColor = vec4(max,max,max,max);
29 }
```

Das Schwellenwert- und das MIP-Verfahren werden in diesem Kapitel gemeinsam vorgestellt. Das MIP-Verfahren kann aufgrund der Kodierung in nur vier Kanälen nicht mit Hilfe der grafischen Benutzeroberfläche eingestellt werden. Da beide Darstellungsarten nicht auf dem optischen Modell aufbauen, verläuft die Implementation auf eine ähnliche Art und Weise, wodurch sich das Shader-Programm leicht umschreiben lässt. Aus diesem Grund kann man das MIP-Verfahren als Alternative zum Schwellenwertverfahren einsetzen.
Die Ray Texture beinhaltet im Grün-Kanal den Grenzwert für das Schwellenwertverfahren, der mit dem aktuellen Voxelwert (src) verglichen wird. Sobald der erste Voxelwert den Grenzwert überschreitet, bricht die Schleife ab und der aktuelle Voxelfarbwert wird als Pixelfarbe ausgegeben. Gleichermaßen verhält es sich mit dem MIP-Verfahren. Der einzige Unterschied besteht darin, dass der Wert im Grün-Kanal keinen Einfluss auf die Schleifendurchgänge hat und somit diese immer vollständig durchlaufen werden. Während der Iterationen wird der maximale Voxelwert ermittelt und anschließend ausgegeben.

Beim Schwellenwert muss zusätzlich noch ein Sonderfall abgefangen werden. Ansonsten kann es vorkommen, dass an Stellen, an denen der Schwellenwert nicht überschritten wird, die Voxel am Rand der Boundingbox angezeigt werden (Zeile 8 und 9).

4.4.3 Transferfunktions-Shader

```
1  ...
2  void main(){
3          ...
4          vec4 rayTexture = texture2D(rayTexture2,fragCoord);
5          vec4 src;
6          for(int i = 0; i< (1/stepSize); i++){
7                  ...
8                  src = texture3D(tex3,position + rayStart);
9                  if(rayTexture.b > 0.0){
10                         dst.rgb += (1.0 - dst.a) *
11                                 texture1D(transferFunction,src.a).a * src.rgb;
12                         dst.a += (1 - dst.a) * texture1D(transferFunction,src.a).a;
13                 }else{
14                         dst.rgb += (1.0 - dst.a) * src.a * src.rgb;
15                         dst.a += (1 - dst.a) * src.a;
16                 }
17         }
18         if(rayTexture.b > 0.0){
19                 gl_FragColor = vec4(texture1D(transferFunction,dst.r).rgb,1.0);
20         }else{
21                 gl_FragColor = dst;
22         }
23 }
```

Anhand des Blau-Kanals der RayTexture wird entschieden, für welche Strahlen die über die Komponente bestimmte Transferfunktion gelten soll. Im Laufe der Iterationen wird nur der Alphakanal über die Transferfunktion bestimmt. Das liegt daran, dass es verschiedene Möglichkeiten gibt, die Transferfunktion in das optische Modell einzubinden. Dabei unterscheidet man zwischen Pre- und Post-Interpolative Darstellung [2]. Bei der Pre-Interpolativen Darstellung fließt die Farbe der Transferfunktion vor dem optischen Modell in die Berechnung ein, d.h. dass zuerst jedem Voxel die neue Farbe der Transferfunktion zugewiesen wird. Bei der anderen Variante erfolgt die Zuweisung der Farbe erst nach dem optischen Modell, wobei zunächst mit den ursprünglichen Skalarwerten gerechnet wird. Durch eine Pre-Interpolative Berechnung werden allerdings Spitzen in der Transferfunktion (Abbildung 10) verpasst, was wiederum zu schlechteren Ergebnissen führt. Daher wird hier die Post-Interpolative-Variante genutzt und der Farbwert mittels der Transferfunktionstextur erst nach der Schleife bestimmt (Zeile 19). Für alle anderen Pixel wird in den beiden „else“-Zweigen das normale Raycasting durchgeführt (Zeile 14,15 und 21).

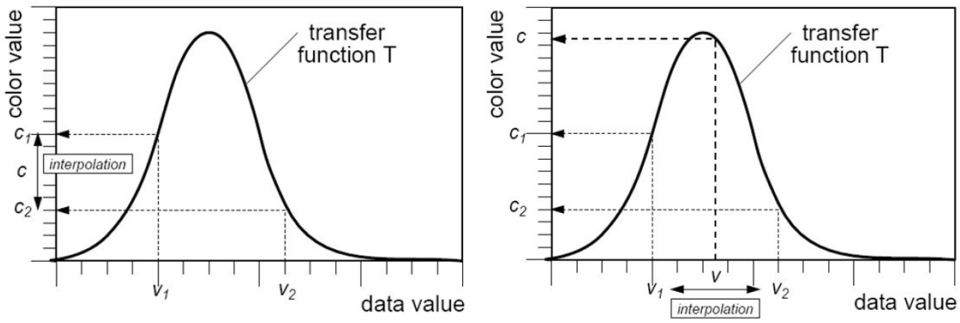

Abbildung 10: [8] Links: Pre-Interpolativ: Hier wird jedem Voxel zuerst die Farbe zugewiesen und während des optischen Modells der Farbwert interpoliert.
Rechts: Beim Post-Interpolativen-Verfahren werden zuerst die ursprünglichen Werte interpoliert und erst danach der Farbwert bestimmt.

4.4.4 Intervall-Shader

```
uniform mat4 samples1;
uniform mat4 samples2;
uniform mat4 samples3;
...
void main(){
 vec4 rayTexture = texture2D(rayTexture2,fragCoord);
 mat4 samples;
 bool intervall = false;
 ...
 if(rayTexture.a > 0.0){
  if(rayTexture.a == 0.1){
   rayTexture = samples1;
  }
  if(rayTexture.a == 0.5){
   samples = samples2;
  }
  if(rayTexture.a == 0.7){
   samples = samples3;
  }
   intervall =true;
 }
 float hlp=0;
 int index1 =0;
 int indes2 =0;

 for(int i = 0; i< (1/stepSize); i++){
   if(intervall){
     if(samples[indes2][index1] > -1){
       if((index1+1)%2 !=0){
         hlp = samples[indes2][index1];
         position = direction * samples[indes2][index1];
         index1++;
         if(index1 >3){
           index1=0;
           indes2++;
         }
```

```
37          }else{
38             if(samples[indes2][index1] < hlp){
39               index1++;
40               if(index1 >3){
41                 index1=0;
42                 indes2++;
43             }
44           }
45         }
46       }else{
47         break;
48       }
49    }
50    ...
51    hlp += 0.01;
52    step = direction * 0.01;
53
54  }
55  ...
56 }
```

Ganz oben stehen zunächst die übergebenen Werte der Intervallregler in jeweils einer 4x4 Matrix. Vor der Schleife entscheidet der Wert in der Ray Texture, welcher Intervallregler angewandt werden soll, und somit welche Intervalle für den Strahl gelten. Zusätzlich wird noch eine „boolsche“-Variable gesetzt, damit auch in der Schleife erkannt wird, dass nicht der komplette Strahl abgetastet werden soll. Die Variable „sample“ bekommt die Intervalle zugewiesen, mit denen man anschließend weiter arbeitet. Wurde durch die „boolsche“-Variable signalisiert, dass die Intervallgrenzen genutzt werden sollen, wird vorerst festgelegt, ob es sich um einen Eintritts- oder um einen Austrittspunkt handelt. Dies geschieht in Zeile 29, in der abgefragt wird, ob der Index, der auf den aktuellen Intervallwert verweist, gerade oder ungerade ist.
Ist der Index gerade, handelt es sich um einen Eintrittspunkt und die Position auf dem Strahl wird auf den Intervallwert gesetzt, sodass alle Voxel, die davorliegen, übersprungen werden. Im Anschluss wird noch eine Hilfsvariable „hlp“ eingeführt, die sich die prozentuale Position auf dem Strahl merkt. Außerdem muss der Index erhöht werden, damit man auf die nächste Intervallgrenze zugreifen kann.
Ein ungerader Index weist auf einen Ausstiegspunkt hin. Hierbei ist zu beachten, dass der Index erst erhöht werden darf, wenn die Hilfsvariable „hlp“ so lange in der Schleife um die Abtastrate erhöht wird, bis sie den Ausstiegspunkt erreicht bzw. überschritten hat. Die Voxel werden währenddessen regulär abgetastet und die Werte mittels optischen Modells akkumuliert. Im Anschluss wird der Index in einen Einstiegspunkt abgeändert.
Sofern nicht alle acht Intervalle gesetzt worden sind, muss sichergestellt werden, dass nach dem letzten gültigen Ausstiegspunkt nicht weiter abgetastet wird. Als Markierung dafür, dass die aktuelle Intervallgrenze ungültig ist, dient die Zahl -1 (Zeile 28).

4.4.5 Render-Shader

Der endgültige Shader besteht aus allen hier gezeigten Shader-Programmen. Der Render-Shader kann nicht in mehrere Shader unterteilt werden, da auch die unterschiedlichen Modi miteinander kombiniert werden sollen. Zusätzlich müssen noch ein paar Abfragen hinzugefügt und ein Randfall abgefangen werden. Wird das Schwellenwertverfahren in Verbindung mit den Intervallen genutzt, muss beim letzten Ausstiegspunkt der aktuelle Voxelwert mit dem Grenzwert verglichen und gegebenfalls auf null gesetzt werden. Dies ist notwendig, da nach dem Ausstiegspunkt alle weiteren Schleifendurchgänge unterbunden werden, weil der aktuelle Wert immer noch gespeichert ist, unabhängig davon, ob dieser den Grenzwert überschritten hat. Je nachdem, welche Farbe für den Pixel bei der Darstellung eingetragen wird, muss noch einmal zwischen den einzelnen Kanälen unterschieden werden, da abhängig von der Darstellungsart nicht immer „dst“, sondern „src“ oder „max“ beim Schwellenwert- oder MIP- Verfahren ausgegeben werden muss. Darüber hinaus muss bei diesen beiden Verfahren ein weiterer Fall hinzugefügt werden, damit eine Kombination mit der Transferfunktion möglich ist.

4.5 Mapping

In diesem Kapitel werden die Weiterentwicklungen im Bereich des Mappings vorgestellt, die zum einen die Anpassung der Textur an die Boundingbox-Oberfläche beim „View aligned“ beinhaltet und zum anderen die Erweiterung beim „Volume aligned“, sodass alle sechs Seiten zum Zeichnen benutzt werden können.

4.5.1 View aligned

Der Pinsel-Shader erstellt die Ray Texture durch eine Ebene über den gesamten „viewport“. Legt man nun diese Textur ohne weitere Berechnungen vor die Boundingbox, wird die Oberfläche des Würfels nicht beachtet. Dadurch können Bereiche eingezeichnet werden, die sich außerhalb der Geometrie befinden. Um dieses Problem zu vermeiden und um die Oberfläche der Boundingbox zu berücksichtigen, wird in dieser Arbeit eine projektive Textur verwendet. Projektive Texturen kann man sich wie einen Dia-Projektor vorstellen, bei dem das Bild auf alle Objekte die sich im Blickfeld befinden, projiziert wird (Abbildung 11).
Technisch gesehen muss man normalerweise jedem Eckpunkt eine Position in der Textur zuweisen, was jedoch bei der Boundingbox nur schwer zu bewerkstelligen ist. Diese besteht nämlich aus sechs unabhängigen Seiten und nach jeder Drehung müssten alle Eckpunkte neue Texturkoordinaten erhalten, die sich der Sicht des Betrachters anpassen. Durch die projektiven Texturen werden die Texturkoordinaten automatisch für die gesamte Geometrie in der Szene berechnet, sodass die Textur in diesem Fall auf den gesamten Würfel projiziert wird. Setzt man den Projektor an dieselbe Stelle, an der sich der Augpunkt befindet, passt sich die Textur der Boundingbox dem Blickfeld des Betrachters an. Diese Projektion wird in einem zusätzlichen Render-Durchgang in einer Textur gespeichert und anschließend als neue Ray Texture an den Render-Shader übergeben.

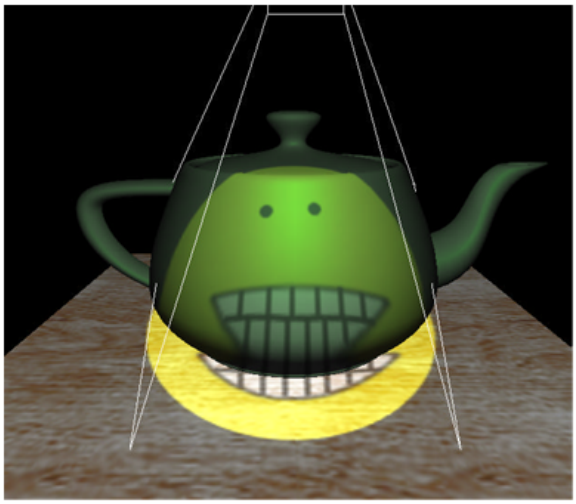

Abbildung 11: [9] Beispiel einer projektiven Textur.

4.5.2 Volume aligned

Damit man beim „Volume aligned“ auf alle sechs Seiten der Boundingbox zeichnen kann, wird zunächst jeder dieser Seiten eine Textur zugewiesen. Sobald der Benutzer anfängt zu zeichnen, muss festgestellt werden, auf welcher der Seiten die Interaktion stattfindet. Dazu wird über die Mausposition die Farbe der Boundingbox abgefragt, die sich an derselben Stelle befindet. Da die Farben durch die Interpolation der Eckpunkte nur einmal vorkommen, lassen sie sich dadurch einer Seite eindeutig zuordnen. Anhand dieser Information wird anschließend die richtige Textur ausgewählt und an den Pinsel-Shader übergeben, dem noch mitgeteilt werden muss, an welcher genauen Stelle der Pinsel eingezeichnet werden soll. Dabei ist hilfreich, dass die Farben auch gleichzeitig Koordinaten beschreiben und eine der Koordinaten sich nicht verändert, so bleibt beispielsweise die X-Koordinate unverändert, wenn die Seite auf der YZ-Ebene liegt. Die Vektoren liegen im Bereich von (0,0,0) bis (1,1,1), wobei die konstante Koordinate außer Acht gelassen wird. Somit ergibt sich eine Position von (0,0) bis (1,1) auf der Textur, auf die der Pinsel eingezeichnet werden soll. In einem Zwischenschritt wird anschließend ein Bild der Boundingbox mit allen sechs Texturen aus der Sicht des Betrachters gerendert. Dieses Bild wird dann in einer Textur gespeichert und als Ray Texture an den Render-Shader geschickt, der diese dann auf das Raycasting anwendet.

5 Diskussion

5.1 Modi

Die Ergebnisse aus der Anwendung des Abtastrate- und Schwellenwert-Modus sollen nun anhand eines Beispiels vorgestellt werden. Beim Konzept der Ray Textures werden Bereiche markiert, die für den Anwender von besonderem Interesse sind. Das Volumen wird zunächst in einer groben Darstellungsstufe gerendert, wodurch nur ein grober Umriss des Volumens zu erkennen ist. Dieser dient als Orientierungshilfe, anhand derer das gewünschte Gebiet schneller gefunden werden kann, ohne sonderlich viel Rechenzeit in die Visualisierung zu investieren. Abbildung 12 zeigt eine genaue Darstellung eines Gesichts, wobei in diesem Fall die Abtastrate im Gesichtsbereich durch die Interaktion des Benutzers um das 100fache erhöht wurde. Auf diese Weise entsteht in diesem Bereich eine klarere Struktur und mehr Details sind zu sehen. Beim Schwellenwert-Verfahren werden nun die Knochen an derselben Stelle segmentiert. Man erkennt die zusammenhängenden Bereiche und auch die Knochen wurden isoliert, allerdings ist die Darstellung nicht sonderlich detailgetreu (Abbildung 12).

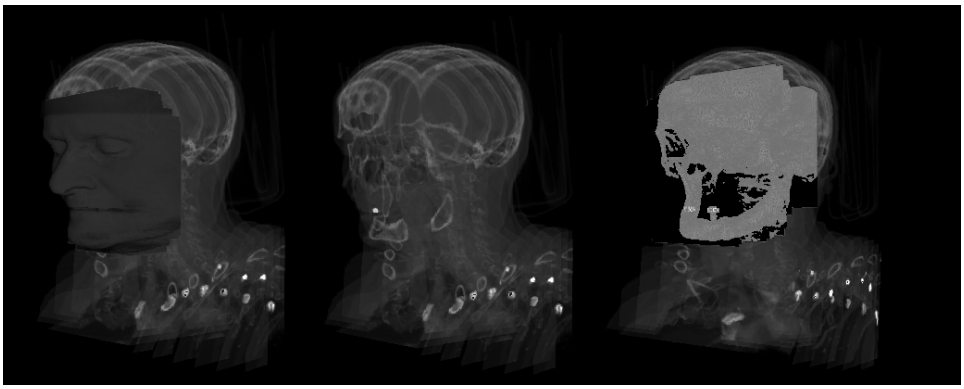

Abbildung 12: Links: das Gesicht mit erhöhter Samplerate. Mitte: der initiale Umriss des Volumendatensatzes. Rechts: Gesicht mit Schwellenwert-Darstellung .

Ein wichtiger Punkt bei diesen beiden Modi ist die Einstellung der jeweiligen Parameter. Für die Regulierung der Abtastrate und des Schwellenwertes wurden daher mehrere Varianten in Betracht gezogen. Es besteht die Möglichkeit, dass sich die Parameter der jeweiligen Eigenschaften durch mehrmaliges Malen auf dieselbe Stelle um einen bestimmten Wert erhöhen, wodurch man auf keine weiteren Eingabehilfen angewiesen ist. Allerdings stellt sich dieses Verfahren als sehr mühsam heraus, falls man sich dazu entschließt, das Volumen mit einer sehr hohen Abtastrate oder einem hohen Schwellenwert darzustellen, da der Benutzer häufig auf denselben Bereich malen müsste. Der Veränderungswert muss zudem sinnvoll gewählt werden, da davon abhängig ist, wie groß die Sprünge zwischen den einzelnen Stufen ausfallen. Wählt man einen zu hohen Wert, kann die Einstellung der Parameter zu ungenau werden, wohingegen ein zu niedrigerer Wert dafür

sorgt, dass der Benutzer häufiger auf den Bereich malen muss. Außerdem kann die Samplerate nur umständlich, beispielsweise über eine Taste gesenkt werden, wobei man auf Eingabehilfen zurückgreifen müsste. Die Bestimmung der Abtastrate und des Schwellenwertes in diesem Programm erfolgt, wie oben bereits erwähnt, über einen Schieberegler. Der Vorteil des Schiebereglers besteht darin, dass die Werte lückenlos und vor allem schnell einstellbar sind. Wünschenswert wäre es, wenn sich die Darstellung nach dem Auftragen und bei erneuter Versstellung des Reglers aktualisieren würde, sodass man mehr Kontrolle über die Auswirkung der Regler hätte und das Volumen so besser anpassen könnte. Dies ist aber aufgrund der Kodierung in der Textur beim Einzeichnen mehrerer Bereiche problematisch, da die hier vorgestellte Implementierung den aktuellen Wert des Reglers für alle Areale übernehmen würde. Dadurch könnten unterschiedliche Bereiche nicht die gleichen Eigenschaften bei anderen Einstellungen besitzen, was jedoch bei der Segmentierung mehrerer Objekte über den Schwellenwert oder bei der Verminderung der Abtastrate an uninteressanten Stellen von Vorteil wäre. Eine Erweiterung des Programms ist durchaus denkbar, damit der letzte eingezeichnete Bereich sich mit den Schiebereglern aktualisiert. Ferner könnte man eine Struktur implementieren, bei der zusammengehörige Bereiche in der Textur angeklickt werden könnten, was eine Korrektur der Parameter erleichtern würde.
Die Intervalle sind sehr nützlich, um einen Einblick in das Volumen zu bekommen. Durch die Verwendung der Intervallregler können zur Laufzeit die verschiedenen Schichtenbilder stufenlos angezeigt werden, wodurch das normale Vorgehen eines Radiologen unterstützt werden kann. Die Intervalle können nicht nur dazu verwendet werden, die üblichen Aufnahmen (seitlich vorne und oben) anzuzeigen, sondern sie sorgen auch dafür, dass die Schichtenbilder in jeder Ausrichtung des Volumens dargestellt werden (Abbildung 13). Aus technischer Sicht werden hierbei jedoch zwei Probleme aufgeworfen. Zum einen sind durch die Übergabe der Intervalle an den Shader durch eine 4x4 Matrix nur acht Intervalle auf einem Strahl möglich. Die Matrix kann selbstverständlich durch eine Textur erweitert werden, allerdings ist die Handhabung bei mehr Intervallen umständlich und ab einer bestimmten Anzahl lassen sich die Intervallknoten nicht mehr vernünftig über den Intervallregler steuern. Zum anderen stehen drei Intervallregler zur Verfügung. Dies bietet mehr Freiheiten, da drei verschiedene Intervalle auf einer Textur unabhängig verändert werden können oder in Verbindung mit der „Volume aligned“-Mapping Methode jeweils eine Seite einem anderen Intervallregler unterliegen kann. Sofern man jedoch noch mehr Intervalle steuern möchte, kommt es zu Problemen mit der Realisierung. Man könnte sogar soweit gehen und für jeden Strahl einen eigenen Intervallregler einrichten, was bei einer Auflösung von 512x512 nicht mehr vernünftig darzustellen wäre, da die gleiche Anzahl an Komponenten hinzugefügt werden müssten. Das gleiche Problem ergibt sich auch bei der Transferfunktion. Es wäre hier sogar ein wesentlicher Vorteil, wenn auf mehreren Bereichen verschiedene Transferfunktionen angewandt werden könnten, um so jeweils andere Köperpartien zu veranschaulichen. Jedoch erweist sich auch hier die Durchführung als schwierig.

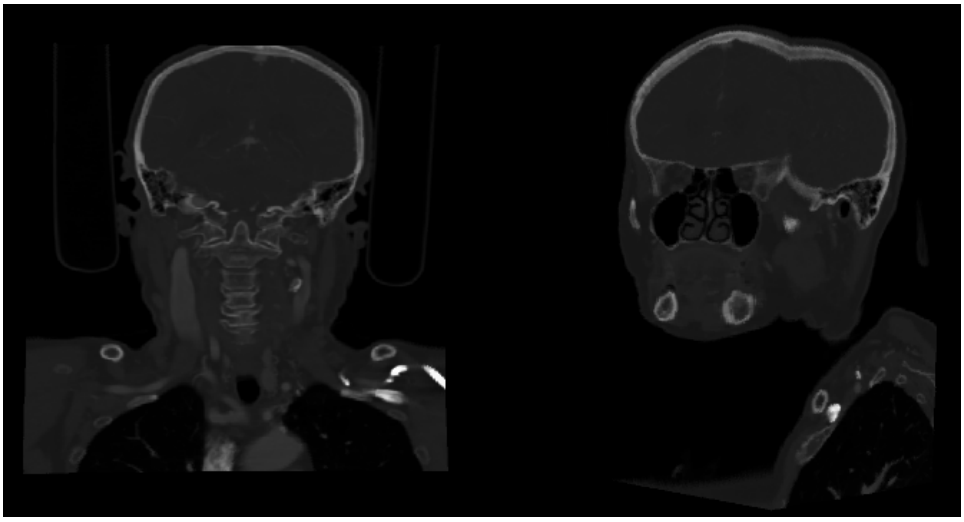

Abbildung 13: Links: Schichtenbild von vorne. Rechts: Interpolation des Schichtenbildes nach Drehung der Boundingbox.

Bei den Modi gibt es Fälle, bei denen sich eine Kombination untereinander lohnt, wobei bei anderen wiederum eine Verbindung eher unangebracht ist. Die Erhöhung der Abtastrate an den interessanten Stellen zum Beispiel ist immer angemessen und sollte daher generell eingestellt sein. Auch die Verwendung der Transferfunktion mit einem ausgewählten Intervall hat seine Vorteile, da das Gebiet durch das Intervall zusätzlich eingegrenzt wird (Abbildung 14). Andererseits ergibt die Darstellung mit dem MIP-Verfahren bei einem zu kleinen Intervall keinen Unterschied in der Darstellung. Ein weiteres Beispiel für eine unpassende Kombination ist der Einsatz einer Transferfunktion während des Schwellenwert-Verfahrens. Hier wird die Transferfunktion lediglich dazu benutzt, um der Visualisierung eine neue Farbe zu geben, die meist einheitlich ausfällt, da diese Darstellungs-Variante nicht auf dem optischen Modell beruht (Abbildung 15).

5.2 Mapping

Beim „View aligned“-Mapping passen sich die Ray Textures der Sicht des Betrachters an, wobei das Volumen unabhängig von der Orientierung der Boundingbox immer einheitlich verändert wird. Das „Volume aligned“-Mapping hingegen wird eingesetzt, um die eingezeichneten Bereiche mit dem Volumen zu drehen oder jede Seite anders zu gestalten, um beispielsweise die Visualisierung des Volumens durch andere Darstellungsarten zu berechnen (Abbildung 16).
Durch die neu implementierte Projektion beim „View aligned“ passt sich die Textur an die Boundingbox an, was für den Benutzer intuitiver ist (Abbildung 17). Allerdings nimmt die Genauigkeit der Interaktion an den Seiten dadurch ab, weil der Pinsel an den Seiten nicht mit mehr mit der Mausposition übereinstimmt. Dies

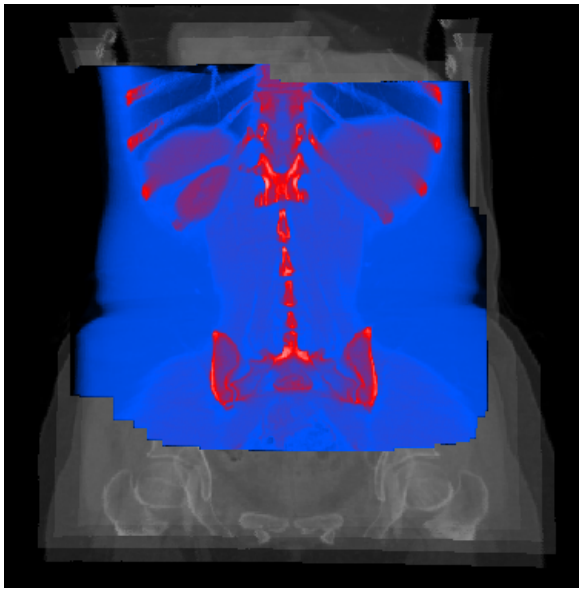

Abbildung 14: Transferfunktion in einem bestimmten Intervall.

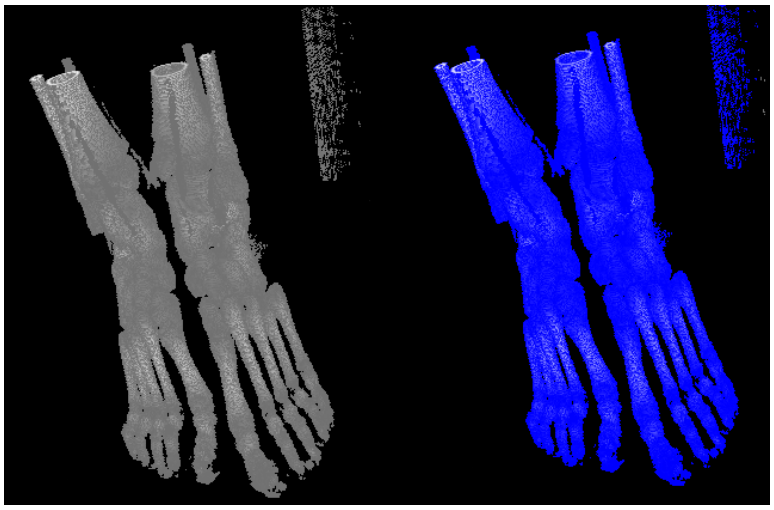

Abbildung 15: Links: Darstellung über das Schwellenwert-Verfahren. Rechts: Darstellung über das Schwellenwert-Verfahren in Kombination mit einer Transferfunktion.

kann man zwar korrigieren, indem man den Projektor neu positioniert, allerdings muss beim Zoomen dieser wieder neu angepasst werden, da der Abstand zwischen Projektor und Boundingbox nur in unregelmäßigen Verhältnissen der Mausposition entspricht. Eine weitere Unstimmigkeit besteht darin, dass sich die Pinselgröße beim Wechseln zwischen „Volume aligned“ und „View aligned“ verändert. Dies ist auf die verschiedenen Vorgehensweisen beider Mapping-Methoden zurückzufüh-

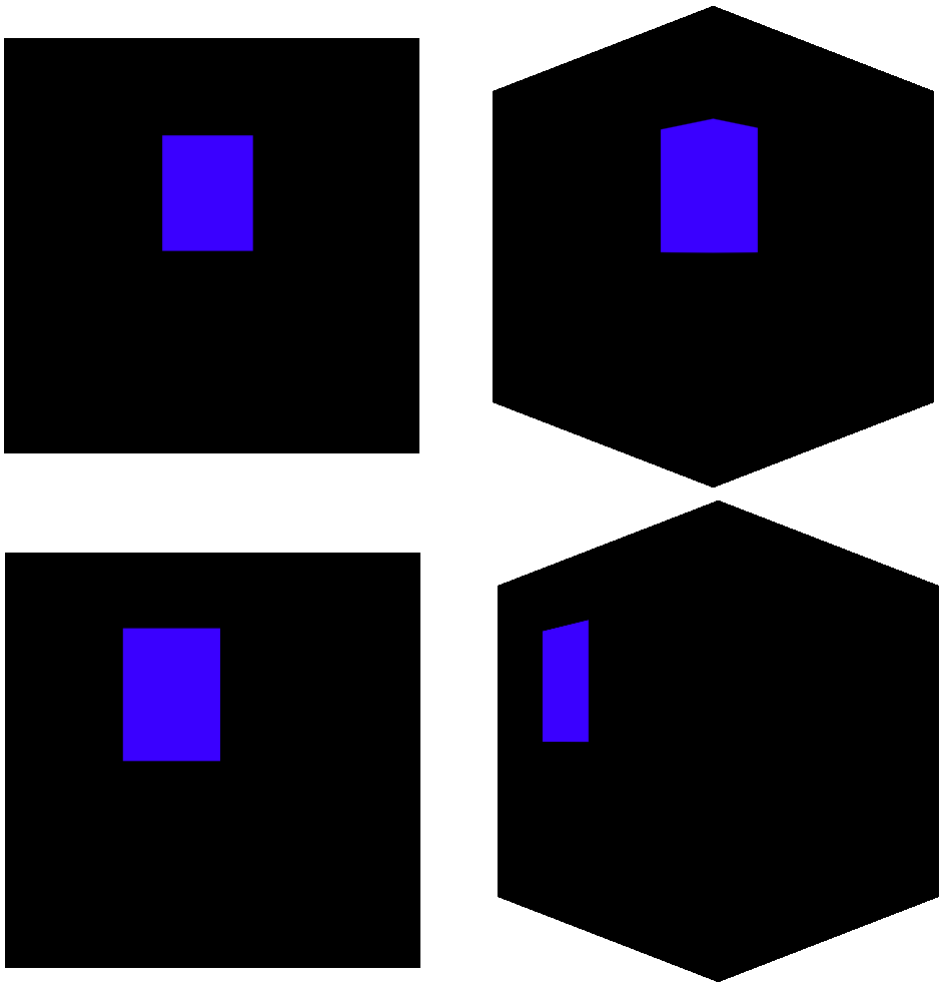

Abbildung 16: Vergleich beider Mapping-Varianten nach einer 45° Drehung. Oben: Beim „View aligned“ bleibt die Ray Texture immer im Vordergrund. Unten: Beim „Volume aligned“ dreht sich die Ray Texture mit der Boundingbox.

ren. Beim „Volume aligned“ wird die Textur nur auf eine Seite der Boundingbox gehaftet, während beim „View aligned“ eine Textur auf den kompletten Würfel projiziert wird. Dies ist zwar nicht weiter bedenklich, bedeutet aber aufgrund der Implementation, dass ein neuer Pinsel geladen werden muss, der sich dem Modus anpasst. Weiterhin entsteht beim „Volume aligned“ in Datensätzen, bei denen sich das Objekt weiter weg von der Boundingbox befindet, ein Fenster-Effekt, ähnlich wie beim „View aligned“. Durch die Kombination mit einem Clipping[13]-Verfahren könnte der Effekt aber vermindert werden, wodurch weitere Interaktionsmöglichkeiten entstehen.

[13]Mit Clipping bezeichnet man in der Volumenvisualisierung Verfahren, die das Volumen beschneiden. Dies geschieht meist über die Seiten der Boundingbox. Nähere Informationen erhält man unter [2].

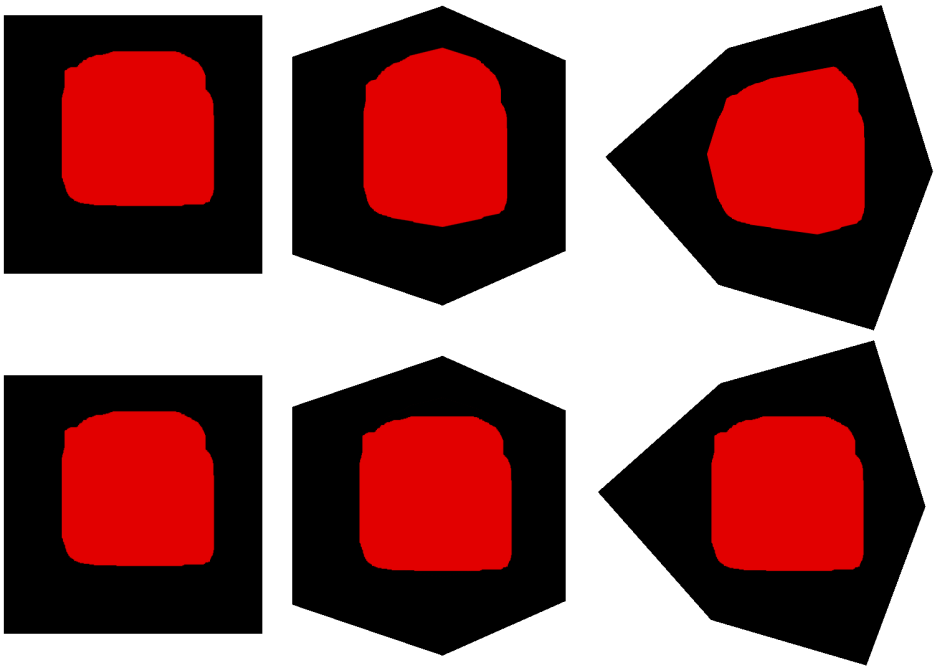

Abbildung 17: Oben: Mapping der Ray Texture mit projektiver Textur. Unten: Mapping der Ray Texture ohne projektive Textur.

Bei Formen, die nicht symmetrisch sind, kann es vorkommen, dass der Pinsel je nach Rotation seitlich oder auf dem Kopf steht. Weiterhin kommt es zu Schwierigkeiten, wenn man beim „Volume aligned“ über eine Kante zeichnen möchte, da der Pinselabdruck nur teilweise auf einer Seite dargestellt wird.

5.3 Pinselform- und größe

Die Pinselgröße gibt darüber Aufschluss, wie viele Strahlen im Raycasting manipuliert werden sollen. Daher ist es sinnvoll, den Gebrauch verschiedener Pinselgrößen zur Verfügung zu stellen. Durch "Load Brush"gibt es auch keine Einschränkungen beim Laden verschiedener Pinselgrößen, wobei der Nachteil an dieser Implementation darin besteht, dass für jeden neuen Pinsel eine neue Textur geladen werden muss. Lohnenswert wäre es daher, die Regulierung der Größe über einen Regler zu handhaben, ohne ständig im erscheinenden Dialog-Fenster einen neuen Pinsel auswählen zu müssen.
Auch die Form der Pinsel ist beliebig wählbar (Abbildung 18), so könnten beispielsweise Pinselformen mit bekannten anatomischen Formen geladen und auf das Volumen aufgetragen werden, sodass man das Interessengebiet von vornherein sinnvoll eingrenzen kann. Durch das Malprinzip kann der Bereich im Nachhinein noch angepasst werden.

Um verschiedene Pinselformen zu ermöglichen, wird die Pinseltextur als Schablone verwendet, bei dem die Pinselwerte angeben, was dargestellt wird. Auf diese Weise werden aber auch Grau- und Farbwertpinsel zugelassen. Diese Pinsel (Abbildung 19) erweisen sich jedoch nicht als sinnvoll, weil die Eigenschaften in den Farbkanälen mit den Pinselwerten multipliziert werden, was dazu führt, dass der Wert des Reglers beim Schwellenwert und bei der Samplerate erneut verändert wird. Weiterhin macht die Änderung des Wertes bei der Transferfunktion keinen Unterschied.

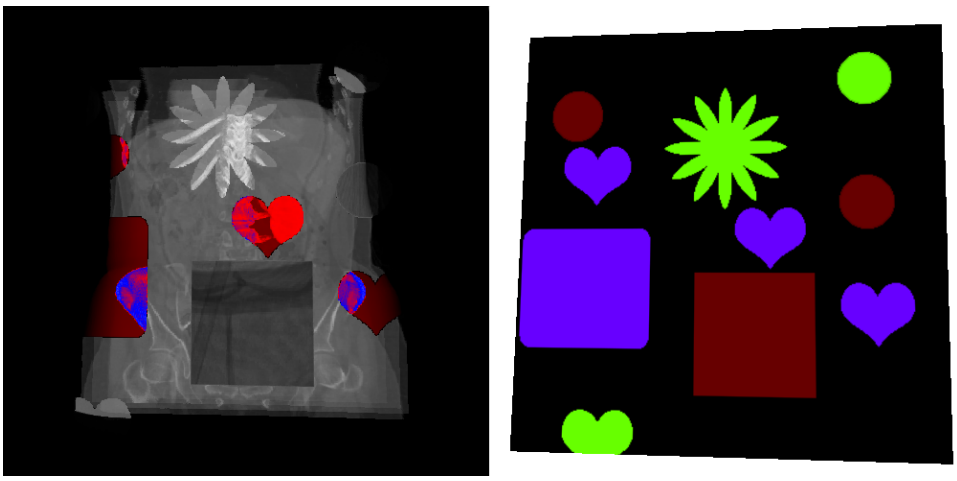

Abbildung 18: Beispiele für verschiedene Pinselformen. Die Farben visualisieren die Kodierung der Modi.

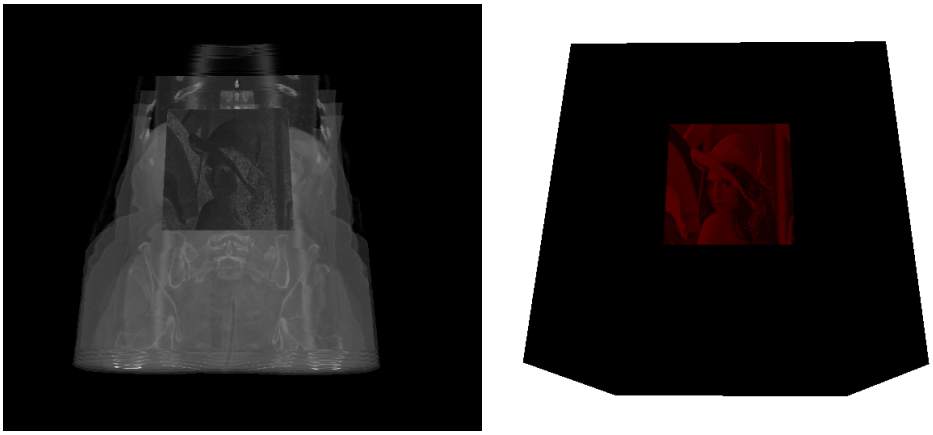

Abbildung 19: Verwendung eines Farbwertpinsels.

6 Ausblick

Es gibt noch einige Möglichkeiten, das Konzept zu erweitern und auch die Schwächen in der Implementation in Bezug auf die Interaktion zu verbessern. Das umständliche Wechseln der verschiedenen Pinselgrößen ließe sich beispielsweise durch MipMaps lösen. Durch einen Regler könnten daraufhin verschiedene MipMap-Stufen ausgewählt werden, die das Größenverhältnis des Pinsels umstellen. Weiterhin würden sich weitere Optionen bei der Markierung der Bereiche anbieten. Programme wie zum Beispiel Photoshop bieten Werkzeuge (Lassowerkzeug oder Auswahlrechteck-Werkzeug) an, bei dem die Größe und Form durch die Interaktion des Benutzers bestimmt werden können.
Wie in Kapitel 5 bereits erwähnt, kann eine Korrektur im Nachhinein durch eine Interaktion mit der Textur erleichtert werden. Zusammengehörende Bereiche könnten dadurch angewählt werden, ohne das Areal erneut zeichnen zu müssen, wobei nur die Parameter neu eingestellt werden müssten. Dadurch könnten auch zwei Transferfunktionen verwendet werden, indem man die Komponenten mit den einzelnen Bereichen auf der Ray Texture synchronisiert. Darüber hinaus könnten neue Darstellungsarten eingebunden werden, die andere Lichtgleichungen einsetzen, oder man könnte, zusätzlich zu dem hier verwendeten Volumenrenderingintegral, die Berechnung von Streuung und Schatten hinzuziehen. Die Berechnung wird zwar nicht weniger aufwändig, aber durch das Einbeziehen der Ray Texture könnte diese nur an interessanten Stellen erfolgen. Denkbar wäre auch eine Übertragung des Konzeptes auf andere Gebiete der Volumenvisualisierung, sodass die Bereiche auf den Texturen nicht nur die im Raycasting verwendeten Strahlen verändern, sondern auch direkten Einfluss auf das Volumen nehmen. Dabei wäre beispielsweise eine Abart des Clippings vorstellbar, die aber nicht auf die Seiten der Boundingbox angewiesen ist.

7 Fazit

Das Konzept der Ray Textures bietet viele Möglichkeiten, ein Volumendatensatz zur Laufzeit zu verändern. Ein großer Vorteil besteht darin, dass man es sowohl in der Rechenzeitoptimierung als auch in der Darstellung verwenden kann. Bei letzterem bieten vor allem die verschiedenen Modi und deren Kombination viele Freiheiten bei der Manipulation des Volumens. Da die Nutzung des Konzeptes einem Zeichenprogramm entspricht, stellt dieses eine einfache und intuitive Steuerung zur Verfügung, mit der man genau einzelne Strahlen verändern kann. Daher ist das Konzept eine nützliche Zusatzfunktion für ein Volumenvisualisierungs-Programm. Das Programm, das in dieser Arbeit entwickelt wurde, hat das Konzept bis zu einem gewissen Punkt implementiert, wobei aber noch einige Erweiterungen möglich sind. Zum einen können noch weitere Modi hinzugefügt werden, zum anderen wäre es sinnvoll, die in der Diskussion besprochenen Schwachstellen zu beheben und die im Kapitel 6 genannten Punkte zu implementieren.

Literatur

[1] Raspe, Matthias and Müller, Stefan: Controlling GPU-based Volume Rendering using Ray Textures. In Skala, Vaclav, ed.: International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision. (2 2008) 277–283

[2] Hadwiger, Markus and Kniss, Joe M. and Rezk-Salama, Christof and Weiskopf, Daniel and Engel, Klaus: Real-time Volume Graphics. A. K. Peters, Ltd., Natick, MA, USA (2006)

[3] Raspe, Matthias: Vorlesung Computergraphik3: Visualisierung und Volumenrendering I + II (2008)

[4] Lorensen, William E. and Cline, Harvey E.: Marching cubes: A high resolution 3D surface construction algorithm. SIGGRAPH Comput. Graph. **21**(4) (1987) 163–169

[5] Pfister, Hanspeter: Moderne Volumenvisualisierung. it - Information Technology **46**(3) (2004) 117–122

[6] Wikipedia: Schwellwertverfahren — Wikipedia, Die freie Enzyklopädie (2009) `http://de.wikipedia.org/w/index.php?title=Schwellwertverfahren&oldid=63281424`[Online; Stand 12. August 2009].

[7] Wikipedia: Digital Imaging and Communications in Medicine — Wikipedia, Die freie Enzyklopädie (2009) `http://de.wikipedia.org/w/index.php?title=Digital_Imaging_and_Communications_in_Medicine&oldid=60953823` [Online; Stand 15. August 2009].

[8] Happe, Markus and Kenter, Tobias: Advanced Topics in Computer Graphics:Transferfunktionen (2007) `http://www.cs.uni-paderborn.de/fileadmin/Informatik/AG-Domik/teaching/lectures/ws0607_advanced_topics/downloads/Transferfunktionen.pdf` [Online; Stand 15. August 2009].

[9] Everitt, Cass: Projective Texture Mapping (2001) `http://developer.nvidia.com/object/Projective_Texture_Mapping.html` [Online; Stand 15. August 2009].