Institut für Visualisierung und Interaktive Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Wahrnehmungsorientiertes Volumen-Rendering

Ruben Bauer

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Thomas Ertl
Betreuer/in:	Valentin Bruder M.Sc., Dipl.-Inf. Christoph Schulz, Dr. Steffen Frey
Beginn am:	12. Mai 2018
Beendet am:	12. Oktober 2018

Kurzfassung

Das menschliche Auge ermöglicht dem Menschen seine visuelle Wahrnehmung, welche in foveales und peripheres Sehen unterteilt werden kann. Das foveale Sehen ist detailliert, scharf und farbig und befindet sich im Zentrum des visuellen Wahrnehumgsbereiches. Im Gegensatz dazu ist das periphere Sehen der Bereich außerhalb des Zentrums, welcher unschärfer und weniger farbig wahrgenommen wird. In dieser Arbeit wird speziell diese Eigenschaft des menschlichen Sehapparates im Zusammenhang mit Volumen-Rendering und Eyetracking untersucht. Dies hat den Hintergrund, dass Volumen-Rendering an sich berechnungsintensiv ist und die Anforderungen an die Hardware durch immer höhere Auflösungen und geforderten Bildwiederholungsraten dauerhaft steigen. Dafür werden zwei unterschiedliche Ansätze, die Reduzierung der Strahl- und die Reduzierung der Bildabtastrate, in eine bestehende Volumen-Rendering Anwendung implementiert und untersucht. Damit wird beabsichtigt, die Bildqualität im peripheren Bereich zu senken, so dass die Performanz der Anwendung erhöht wird und gleichzeitig die wahrgenommene Qualität der Darstellung erhalten bleibt. Die Implementierungen der unterschiedlichen Ansätze haben eine deutlich verbesserte Performanz bei einer kaum wahrnehmbaren Veränderung der Bildqualität im peripheren Bereich des Sehens ergeben. Die Analyse der Untersuchungsergebnisse führten zu dem Schluss, dass sich wahrnehmungsorientierte Methoden für Volumen-Rendering besonders gut eignen.

Inhaltsverzeichnis

1 Einleitung

Moderne Bildschirme werden immer größer, haben höhere Pixeldichten und können immer mehr Bilder pro Sekunde anzeigen. Dadurch erhöht sich die Anzahl der zu berechnenden Pixel einer Anwendung pro Sekunde immens, wodurch immer höhere Anforderungen an die Hardware gestellt werden und die Performanz der Anwendungen unter den Grenzen der Hardware leiden. Methoden des wahrnehmungsorientierten Renderings können hier weiterhelfen.

Das visuelle Wahrnehmungssystem des Menschen hat einige Limitierungen, die im wahrnehmungsorientierten Rendering gezielt ausgenutzt werden können. So kann die visuelle Wahrnehmung des Menschen grob in foveales und peripheres Sehen unterteilt werden. Das foveale Sehen ist im Zentrum des Blickpunkts und umfasst nur einen sehr kleinen Bereich in dem man scharf und detailliert sehen kann. Das periphere Sehen hingegen ist unscharf und eher ungenau, wobei die Sehschärfe mit der Distanz zum Zentrum des Blickpunkts immer weiter abnimmt. Im wahrnehmungsorientierten Rendering wird sich unter anderem diese Eigenschaft zu Nutze gemacht, um das Leistungsverhalten von bilderzeugenden Anwendungen zu verbessern. Dafür wird die Bildqualität im peripheren Bereich gesenkt, wodurch die Berechnungsdauer eines Bildes reduziert wird. Da die visuelle Wahrnehmungsmöglichkeiten im peripheren Bereich sowieso abnehmen, wird die reduzierte Bildqualität im besten Fall von dem Nutzer nicht wahrgenommen.

Um wahrnehmungsorientierte Rendering Methoden verwenden zu können ist es notwendig, die visuelle Wahrnehmung des Nutzers zu kennen. Ein Eyetracker ermöglicht es, die aktuelle Blickposition des Nutzers auf dem betrachteten Bildschirm zu messen und der wahrnehmungsorientierten Anwendung zur Verfügung zu stellen. Wahrnehmungsorientierte Anwendungen werden deshalb oft mit zusammen mit einem Eyetracker verwendet.

Wahrnehmungsorientierte Rendering Methoden sind nicht auf Anwendungen beschränkt, die ihre Bilder mit einer Grafikpipeline berechnen. Besonders das Volumenrendering mit Hilfe eines Raycasts, ermöglicht unterschiedliche Ansatzpunkte, um wahrnehmungsorientierte Methoden zu implementieren. Raycasting ist ein Verfahren um Volumendaten abzutasten und auf eine 2D Rastergrafik zu projizieren. Wie beim Raytracing werden beim Raycasting auch Sichtstrahlen verfolgt. Dabei wird in der Regel für jeden Pixel ein Sichtstrahl, ausgehend von einer virtuellen Kamera im Raum, in das Volumen gesendet und in bestimmten Abständen abgetastet. Das Volumen ist aus Voxeln aufgebaut, die dreidimensionale Boxen entsprechen. Wird ein Voxel von einem Sichtstrahl abgetastet, so wird ausgehend von dem Dichtewert des Voxels und einer Transferfunktion ein Farbwert berechnet und dem Sichtstrahl hinzugefügt. Alle abgetasteten Farbwerte eines Sichtstrahls resultieren in einem endgültigen Farbwert für den jeweiligen Pixel. Die Transferfunktion ermöglicht auch eigentlich verdeckte Strukturen innerhalb des Volumens sichtbar zu machen, indem Voxel mit bestimmten Dichtewerten nicht zur Farbwertberechnung beitragen und dadurch ausgeblendet werden. Da das Rendern von Volumen aufgrund der Komplexität der Volumendaten auch sehr berech-

nungsaufwändig sein kann, müsste sich durch die Implementierung von wahrnehmungsorientierten Volumenrendering Methoden, die Performanz und Nutzbarkeit von interaktiven Volumenrenderern deutlich verbessern lassen.

Diese Arbeit befasst sich hauptsächlich mit dieser These. Es soll mit Hilfe eines Eyetrackers die foveale Region auf dem Bildschirm erfasst und zusätzlich eine Möglichkeit geschaffen werden, diese mit der Maus zu simulieren. Ausgehend einer existierenden Volumenrendering-Anwendung, welche für diese Arbeit zur Verfügung gestellt wurde, sollen unterschiedliche Methoden implementiert werden, um abhängig zu der Blick- beziehungsweise Mausposition die Bildqualität des Volumenrenderings im peripheren Bereich des Auges zu reduzieren. Anschließend sollen die unterschiedlichen Methoden anhand ihrer Bildqualität und Performanz und der Wahrnehmung bei der Verwendung von Eyetrackingdaten untersucht und diskutiert werden.

Die Arbeit beschäftigt sich dafür zu Beginn mit einigen Grundlagen. Diese sind unter anderem relevante Aspekte verwandter Arbeiten, des visuellen Sehapparates des Menschen und der Architektur von GPUs. Im Folgenden werden Arbeitspakete, die im Laufe dieser Arbeit entstanden sind und umgesetzt wurden, im Entwurf vorgestellt. Die Arbeitspakete beschäftigten sich unter anderem mit drei Methoden, mit denen die Performanz des Volumenrenderings durch die Verwendung eines Eyetrackers verbessert werden kann. Eine dieser Methode hat die Strahlabtastrate abhängig zum Blickpunkt des Nutzers variiert. Die zwei anderen Methoden veränderten abhängig zur Distanz des Blickpunkts die Anzahl der Strahlen und damit die Auflösung des berechneten Bildes. Die Methode zur Veränderung der Strahlabtastrate konnte dabei mit den anderen beiden Methoden, die die Auflösung des Bildes angepasst haben, kombiniert werden. Die Integration und Implementierung der Arbeitspakete einschließlich der unterschiedlichen Methoden in die bestehende Volumerender-Anwendung wird im anschließenden Kapitel genauer beschrieben. Für die verschiedenen möglichen Kombinationen der Methoden wurden hinsichtlich ihrer Bildqualität und Performanz Messungen erstellt und diese verglichen. Die Resultate sind im Ergebniskapitel vorgestellt und werden dort anschließend diskutiert.

2 Grundlagen

Abschnitt 2.1 dieses Kapitels beschäftigt sich mit verwandten Arbeiten zum Thema Wahrnehmungsorientiertes Volumenrendering. In Abschnitt 2.2 werden Grundlagen des menschlichen Sehapparates behandelt. Hier werden unter anderem die Fähigkeiten und Limitierungen der visuellen Wahrnehmung des Menschen diskutiert. Abschnitt 2.3 behandelt Grundlagen zum Volumenrendering, wie Raycasting und die Funktionsweise einer Transferfunktion. Das Implementieren unterschiedlicher Methoden des Raycastings und die Wahl von geeigneten Transferfunktionen wird Teil dieser Arbeit sein. Anschließend befasst sich Abschnitt 2.4 mit dem Erfassen der visuellen Wahrnehmung mit Hilfe von Eyetracking. Der Raycast wird in der Regel auf einer GPU ausgeführt, dabei sind einige Eigenschaften der GPU Architektur besonders zu beachten. Dies wird in Abschnitt 2.5 beschrieben.

2.1 Related Work

Wahrnehmungsorientiertes Volumenrendering ist kein neues Arbeitsgebiet und war schon Teil verschiedener wissenschaftlicher Arbeiten. Abschnitt 2.1.1 behandelt daher verwandte Arbeiten bezüglich des wahrnehmungsorientierten Volumenrenderings unter dem Aspekt der Performanz, da sich der Hauptteil dieser Arbeit sich auf diesen Aspekt bezieht. Die Ansätze hier beziehen sich meist darauf, dass die Qualität der Darstellung im peripheren Bereich der visuellen Wahrnehmung gesenkt wird und so für die Berechnung eines Bildes weniger Rechenleistung aufgewendet werden muss.

Abschnitt 2.1.2 bezieht sich auf Arbeiten zum wahrnehmungsorientierten Volumenrendering mit dem Ziel, die Qualität der Darstellung zu erhöhen. Dabei werden vor allem Ansätze zur geschickten Anpassung von Parametern einer Transferfunktion vorgestellt, die dem Betrachter ein insgesamt besseres Verständnis der Volumendaten ermöglichen sollen.

2.1.1 Performanz- und Wahrnehmungsorientiertes Volumenrendering

Marc Levoy und Ross Whitaker [LW90] erforschen in ihrer Arbeit, wie Eyetracking-Daten in Rendering-Algorithmen eingesetzt werden könnnen. Ziel ihrer Forschung war es, dem Nutzer einen Arbeitsplatz für Echtzeit-Volumen-Rendering, mit der Illusion eines hochauflösenden Bildes über den gesamten Bildschirm zu präsentieren. Dadurch sollte das durch Eyetracking unterstützte Verfahren einen geringeren Berechnungsaufwand als herkömmliche Volumenrenderer haben. Dafür stellten sie eine Implementierung eines Raycasters für Volumendaten vor, in welcher mit Hilfe der Eyetrackingdaten der Blickfokus auf dem Bildschirm berechnet wurde und die Anzahl der Strahlen und die Samples pro Strahl, abhängig von der Distanz zum Blickfokus, angepasst wurden. Die Implementierung basierte darauf, dass der Detaillgrad der visuellen Wahrnehmung des menschlichen

Auges nur in einem kleinen, zentralen Bereich, der Fovea, am höchsten ist und zu den Rändern des Blickfelds hin stark abnimmt. Ausgehend davon berechneten sie in dem von dem Nutzer fokussierten Bereich das Bild mit der vollen Auflösung und einer hohen Abtastrate der Strahlen. In dem restlichen Bereich reduzierten sie die Auflösung des Bildes und abhängig von dem Abstand eines Pixels zum Blickfokus, die Abtastrate eines Strahls. In ihrer Implementierung verwendeten sie 2D und 3D mip maps, einen Eye Tracker und die *Pixel-Planes 5 rendering engine*, ein hoch paralleles Raster Display System. Um die geringere Abtastung in der Peripherie gut nutzen zu können, wurde aus den 3D Volumendaten eine 3D mip map erstellt. Die 3D mip map wurde durch den Raycasting Algorithmus abgetastet, wodurch anschließend mit trilinearer Interpolation eine 2D mip map erstellt wurde. Die 2D mip map war Grundlage für die Erstellung des endgültigen Bildes. In ihren Ergebnissen konnten sie die Berechnungsaufwand für ein teilweise hochauflösendes Bild im Vergleich zu einem vollständig hochauflösenden Bild, um bis einen Faktor von fünf senken.

Guenter et. al. [GFD+12] untersuchte, ähnlich zu dem Paper von Levoy und Whitaker, wie der Abfall der visuellen Auflösung des Auges außerhalb des visuellen Zentrums ausgenutzt werden kann, um eine beschleunigte Berechnung des Bildes zu ermöglichen. Im Gegensatz zu der davor genannten Arbeit war das darunterliegende System hier kein Raycaster für Volumendaten, sondern eine Grafikpipeline für die Rasterung von 3D Szenen. Das Bild wurde hier aus drei Teilbilder zusammengesetzt, welche jeweils mit unterschiedlichen Auflösungen berechnet wurden und wie Schichten übereinander gelegt wurden. Die drei Schichten waren: innere Schicht, mittlere Schicht und äußere Schicht. Die innere Schicht hatte ungefähr die Größe des fovealen Bereichs auf dem Bildschirm und wurde in der maximalen Auflösung berechnet. Ihr Mittelpunkt war der Blickfokus des Betrachters. Die mittlere Schicht war ein wenig größer als die innere Schicht und wurde mit einer niedrigeren Auflösung berechnet. Sie wurde ebenfalls auf den Blickfokus des Betrachters zentriert. Die äußerste Schicht überdeckte den gesamten Bildbereich und wurde in der niedrigsten Auflösung berechnet. Um die Schichten zusammensetzen zu können, wurden die mittlere und äußere Schicht jeweils zur nativen Auflösung des Bildschirms, die gleiche Auflösung wie die innere Schicht, interpoliert. Scharfe Kanten zwischen den Schichten wurden dadurch vermieden, dass diese sich gegenseitig leicht überdeckten und spezielle *Blend-Masks* verwendet wurden, um die verschiedenen Schichten glatt übereinander zu legen. In ihrer Arbeit nahmen sie sich auch des Problems an, dass das starke Unterabtasten um bis zu dem Faktor sechs in jede Dimension, in der mittleren und äußeren Schicht zu störenden Artefakten führen konnte. Um dem entgegen zu wirken, verwendeten sie drei Antialiasing Techniken: *Hardware Multi-Sample Antialiasing (MSAA)*, *Temporal Reprojection* und *whole frame jitter sampling*. Um die Blickposition zu erfassen nutzten sie den *Tobii Tx 300 Eye Tracker* mit einer Worst-Case Latenz von 10 ms. Das Bild wurde auf einem Computer mit einer *Intel Xeon CPU (E5640 mit 2.67 GHz)* und einer *NVidia GeForce GTX 580 GPU* berechnet. Dargestellt wurde es auf einem 23 Zoll 1920×1080 LCD Monitor mit einer Bildwiederholungsrate von 120 Hz. Mit ihrem System und ihren Techniken erlangten sie eine Performanz-Verbesserung um den Faktor fünf bis sechs auf einem Monitor mit HD Auflösung. Dabei erreichten sie eine Darstellungsqualität, die vergleichbar mit dem Rendern eines hochauflösenden Bildes über den gesamten Bildschirm ist.

2.1.2 Wahrnehmungsorientiertes Volumenrendering zur Qualitätssteigerung

R. Englund und T. Ropinski [ER] untersuchten verschiedene Techniken zur Verbesserung der Wahrnehmung von komplexen volumetrischen Daten. In einer Studie mit über 300 Teilnehmern erforschten sie, wie diese Techniken zur verbesserten Wahrnehmung von Volumen beitragen und verglichen die verschiedenen Ansätze miteinander. Dabei mussten die Teilnehmer jeweils kleine Aufgaben absolvieren, so dass darauf geschlossen werden konnten, wie sehr eine bestimmte Technik den Teilnehmer bei der Erkennung von Form und Tiefe eines Objekts in einem Volumen unterstützt. Um eine bessere direkte Analyse der Volumendaten später ermöglichen zu können, wurden Techniken, die automatisch Renderingparameter angepasst haben, ausgelassen. In ihrer Studie haben sie sechs Techniken ausgewertet, darunter *Direct Volume Raytracing (DVR)* und *Depth Darkening*. In ihrer Auswertung kamen sie unter anderem dazu, dass Techniken, die natürliche Lichteffekte in den Volumendaten simulieren, deutliche Vorteile gegenüber anderen getesteten Ansätzen gezeigt haben.

Anders als zu den untersuchten Techniken von Englund und Ropinski [ER] präsentierten Aidong Lu et. al. [LME06] eine Methode für die automatisierte Parameterauswahl bei der Betrachtung von Volumendaten. Das Ziel welches sie mit dieser Methode verfolgten, war die Vereinfachung der mühsamen Nutzerinteraktionen bei der Auswahl der Parameter, um somit die Nutzbarkeit des Darstellungssystems zu verbessern.

Volumendaten können sehr komplex sein, daher ist es oft schwierig herauszufinden, was der Nutzer in dem Volumen betrachten möchte und was dahingehend hervorgehoben werden soll. Eigenschaften der Volumendaten, wie konstante Größen, Formen und Positionen der Objekte ermöglichen aber Methoden für eine automatische Anpassung der Parameter.

Aidong Lu et. al. nutzten einen Eyetracker, um die Bereiche, die für den Nutzer von Interesse waren, zu bestimmen. Dieser maß die Augenbewegungen und die Blickposition auf dem Bildschirm. Es wurde zwischen zwei hauptsächlichen Augenbewegungen unterschieden: Einer Sakkade und einer Fixation. Eine Sakkade ist eine schnelle Augenbewegung von einem Punkt zu einem anderem. Bei einer Fixation ruht das Auge auf einem Punkt. Punkte die fixiert werden sind für den Nutzer meist von höherem Interesse. Da die Blickposition durch den Eye Tracker nur auf einer 2D-Ebene bestimmt werden konnte, wurde durch ein konstantes Rotieren des Volumenobjektes und der dazu parallelen Aufzeichnung der Augenbewegungen versucht, die 3D Position des fixierten Objektes zu rekonstruieren. Aus den Eyetracking- und Volumendaten bestimmten sie dann mit Hilfe mehrerer Clustering-Methoden Gewichte für die einzelnen Voxel des Volumens und berechneten so einen Wichtigkeitswert für die verschiedenen Objekte innerhalb des Volumens. Entsprechend dieser Ergebnisse wurden die Renderingparameter angepasst, um die für den Nutzer interessantesten Objekte hervorzuheben und anzuzeigen. Aidong Lu et. al. kamen zu dem Schluss, dass die präsentierte Methode den Aufwand für den Nutzer, die Render Parameter anzupassen, signifikant reduzieren kann. Trotzdem konnte ein solcher regelbasierter Ansatz nicht mit der manuellen Einstellung der Renderingparameter mithalten.

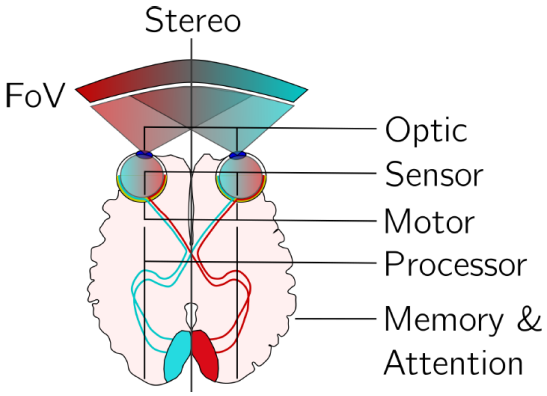

Abbildung 2.1: Modell des visuellen Wahrnehmungsapparates des Menschen aus [**doi:10.1111/cfg.13150**]

2.2 Sehapparat

Wahrnehmungsorientiertes Rendering nutzt gezielt Eigenschaften der visuellen Wahrnehmung des Menschen aus. Dafür ist ein gutes Verständnis des menschlichen visuellen Wahrnehmungsapparates notwendig. M. Weier et. al. [WSR+] befassten sich unter anderem mit den Fähigkeiten und Limitierungen des menschlichen Sehapparates. Die für diese Arbeit relevanten Eigenschaften und Limitierungen werden im Folgenden vorgestellt. Abbildung 2.1 ist ein Modell des visuellen Wahrnehmungsapparates des Menschen. Das Modell unterteilt den Sehapparat des Menschen in Optik, Sensorik, Motorik, Verarbeitung, Speicherung und Aufmerksamkeit. Licht trifft auf die Augen und wird durch die Optik auf die Retina beziehungsweise die Sensorik, weitergeleitet. Hier wird der visuelle Input abgetastet und gefiltert. Dabei entstehen zwei Datenströme welche die Wahrnehmung stereoskopischer Bilder über einen großen Blickwinkel mit unterschiedlichen Auflösungen ermöglichen. Die Retina ist mit dem visuellen Kortex verbunden. Die Signale werden über die visuellen Nerven komprimiert und zum visuellen Kortex transportiert. Dort werden sie von verschiedenen Bereichen im Gehirn verarbeitet. Die Speicherung der Signale und die aktuelle Aufmerksamkeit spielen dabei eine große Rolle.

Das visuelle Wahrnehmungssystem des Menschen hat wesentliche Limitierungen, die bei der Darstellung von Bildern gezielt genutzt werden können. Die Sehschärfe ist ungleich auf der Retina verteilt und nur in in einem kleinen, zentralen Bereich ist die Sehschärfe maximal. Dieser Bereich wird Fovea genannt. Je weiter man sich auf der Retina von der Fovea nach außen hin entfernt, desto mehr nimmt die Sehschärfe ab. Der Bereich außerhalb der Fovea ist der periphere Bereich des Auges.

Das Sichtfeld des visuellen Wahrnehmungsapparates des Menschen ist bei einem gerade gerichteten Blick horizontal bis circa 190 ° und mit Augenrotationen bis zu 290 ° weit. Visuelle Reize werden über das gesamte Sichtfeld wahrgenommen. Abhängig von dem zuständigen Bereich auf der Retina gibt es starke Unterschiede, wie die visuellen Reize verarbeitet werden.

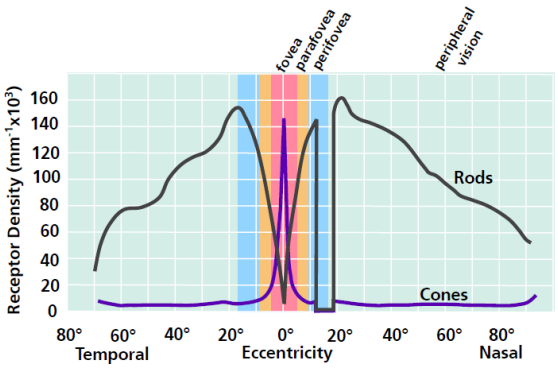

Abbildung 2.2: Verteilung der Photorezeptoren auf der Retina. [**doi:10.1111/cfg.13150**]

Die Retina ist die photosensitive Schicht des Auges und besteht aus zwei Typen von Photorezeptoren, aus Zapfen und Stäbchen. Es sind circa 6×10^6 Zapfen und ungefähr 20 mal so viele Stäbchen auf der Retina verteilt. Stäbchen sind für die Helligkeitswahrnehmung verantwortlich. Zapfen sind für die Farbwahrnehmung zuständig. Man unterscheidet zwischen drei Zapfentypen, die L-Zapfen für lange Wellenlängen, die M-Zapfen für mittlere Wellenlängen und die S-Zapfen für kurze Wellenlängen.

Die Fovea ist der Bereich um circa 5,2 ° um das Zentrum der Retina und besteht fast ausschließlich aus Zapfen. Die Anzahl der Zapfen nimmt nach außen hin stark ab. Der Bereich von circa 5,2 ° bis 9 ° wird als Parafovea bezeichnet. Der Bereich zwischen circa 9 ° bis 17 ° heißt Perifovea. Fovea, Parafovea und Perifovea sind für die zentrale Sicht verantwortlich. Alles außerhalb der zentralen Sicht wird als periphere Sicht bezeichnet. Abbildung 2.2 zeigt die Dichteverteilung der Photorezeptoren auf der Retina. Die höchste Dichte der Stäbchen liegt bei circa 15 ° bis 20 ° um die Fovea herum. Ihre Anzahl verringert sich nach außen hin circa linear. Zapfen und Stäbchen sind sehr unterschiedlich auf der Retina verteilt. Beide folgen aber einem Poisson-Disc Verteilungsmuster.

Die Dichte der Zapfen ist direkt mit der Sehschärfe verknüpft. Daher fällt die Sehschärfe auch nach außen hin stark ab. Bei 6 ° außerhalb des Zentrums beträgt zum Beispiel die Sehschärfe nur noch ein Viertel der maximalen Sehschärfe im Zentrum.

Die Sehschärfe hängt aber auch von der Kontraststärke der visuellen Reize ab. Dabei ist die Kontrastsensitivität von der Anzahl der auf den Reiz reagierenden neuronalen Zellen abhängig, welche ebenfalls nach außen hin stark abnimmt. Die Farbsensitivität ist von der Verteilung von Stäbchen und Zapfen abhängig. Die Zapfen zur Erkennung von grünem und rotem Licht sind vermehrt in der Fovea und eher weniger im peripheren Bereich verteilt. Insgesamt sind lediglich neun Prozent der Zapfen für die Wahrnehmung von blauem Licht verantwortlich. Diese sind eher im peripheren Bereich verteilt als im Zentrum.

Das Auge ist dauerhaft in Bewegung. Sechs externe Muskeln ermöglichen es verschiedene Objekte von Interesse (OvI) zu fokussieren. Die wichtigsten Arten von Augenbewegungen ist die Sakkade, der Vestibular-Okularer Reflex, die *Smooth pursuit eye motion (SPEM)* und *coupled vergence-accommodation motion*. Der Vestibular-Okulare Reflex ist relativ schnell und hat eine Latenz von 7 ms bis 15 ms und ermöglicht dadurch auch während schnellen Kopfbewegungen OvI zu fixieren. Die SPEM ermöglicht die weiche Verfolgung eines sich bewegendes Objektes. Die *coupled vergence-accomoodation motion* ist eine Augenbewegung, die auftritt, wenn erst ein nahes Objekt und danach ein weit entferntes Objekt fokussiert wird.

Für das wahrnehmungsorientierte Rendering wird meist nur zwischen einer Sakkade und einer Fixation unterschieden. Eine Sakkade bezeichnet das schnelle Springen von einem Ovi zu einem anderen. Dabei erreicht das Auge Geschwindigkeiten von bis zu 900 °/s. Die Sehsensitivität ist während einer solchen Sakkade stark geschwächt. Eine Fixation dauert zwischen 100 ms - 1.5 s. Sie tritt meist dann auf, wenn ein OvI genauer betrachtet wird und die Augen darauf ruhen. In natürlichen Szenen treten zwei bis drei Sakkaden pro Sekunde auf, mit jeweils einer durchschnittlichen Fixationszeit von 250 ms. Der räumliche Abstand zwischen den Fixierungen beträgt dabei circa 7 °. Ein Abstand von mehr als 30 ° wird als unangenehm empfunden und hat in der Regel eine Kopfbewegung zur Folge. Auch während einer Fixierung macht das Auge wichtige kleine Bewegungen, sogenannte Tremorbewegungen. Werden diese bewusst unterdrückt, führt dies zu einem schwindenden Bild. Kleinere Augenbewegungen von bis zu 2.5 °/s haben auf die Sehschärfe keinen Effekt.

Dass die Fovea einen sehr wichtigen Teil der visuellen Informationen liefert spiegelt sich auch darin wieder, dass über 30 Prozent des primären Sehverarbeitungsbereichs des Gehirns für die zentralen 5 ° des Sehfeldes, zuständig sind. Der periphere Bereich ist hier benachteiligt, liefert aber trotzdem wichtige kontextuelle Informationen und hilft bei der Wahrnehmung von Kontrasten, Objekten und Tieren. Die Bewegungserkennung ist sowohl in der Fovea als auch im peripheren Bereich ähnlich oder genauso gut.

2.3 Volumenrendering und Transferfunktion

Es ist nicht trivial, 3D Volumendaten auf einem 2D Bildschirm informativ zu präsentieren. In einem Volumen gibt es möglicherweise mehrere für den Betrachter interessante Objekte, welche sich durchaus gegenseitig überdecken können. Daher stellt sich die Frage, wie die Volumendaten auf dem Bildschirm projiziert werden und dabei für den Betrachter auch Informationen an verschiedenen Positionen innerhalb des Volumen sichtbar gemacht werden können. Raycasting ermöglicht in Verbindung mit einer Transferfunktion, ein Volumen auf eine 2D Ebene zu projizieren und dabei einzelne Bereiche des Volumens farbig hervorzuheben oder transparent erscheinen zu lassen.

2.3.1 Raycasting

Raycasting ist ein Verfahren um Volumendaten abzutasten und auf eine 2D Rastergrafik zu projizieren. Wie beim Raytracing werden beim Raycasting auch Sichtstrahlen verfolgt. Die Sichtstrahlen werden dabei ausgehend von einer virtuellen Kamera im Raum ausgesendet und in bestimmten Abständen abgetastet. Die virtuelle Kamera, siehe Abbildung 2.3, ist im Raum an Position $\vec{e}$ positioniert. Dies ist auch das Projektionszentrum, falls es keine orthogonale Projektion ist. Die Position $\vec{e}$ entspricht relativ der Position des Betrachters vor dem Bildschirm. Mit $\vec{u}$, $\vec{v}$ und $\vec{w}$ wird die Ausrichtung der Kamera im Raum eindeutig bestimmt. In einer Entfernung `d` von $\vec{e}$ aus in Richtung $-\vec{w}$ ist die Bildebene 2.4 positioniert. Die Größe der Bildebene wird durch `l`, `r`, `b` und `t` definiert. Abhängig von der Größe des Bildschirms oder der Größe der gewünschten Rastergrafik wird die Bildebene virtuell in die gewünschte Anzahl an Pixel in Breite und Höhe unterteilt. Jeder dieser Teile entspricht nun einem Pixel der Rastergrafik. Nun kann für jeden Pixel ein Strahl, ausgehend von $\vec{e}$, in Richtung des entsprechenden Punktes auf der Bildebene, ausgesendet und verfolgt werden. Trifft ein Strahl Objekte in der Szene, wird dadurch ein Farbwert ermittelt, den der entsprechende

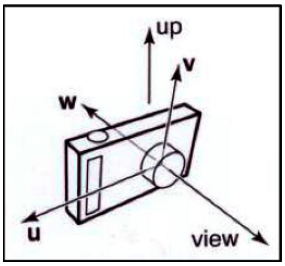

Abbildung 2.3: Illustration einer virtuellen Kamera im Raum [Dr 17]

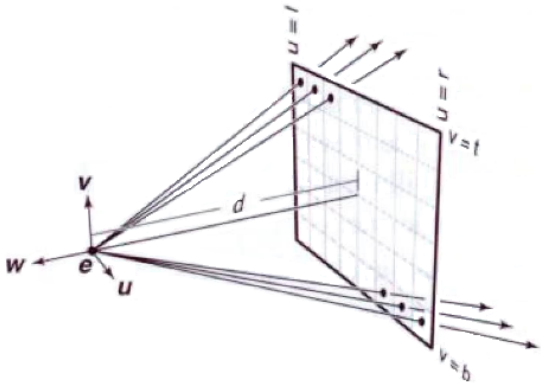

Abbildung 2.4: Illustration der virtuellen Kamera und der Bildebene [Dr 17]

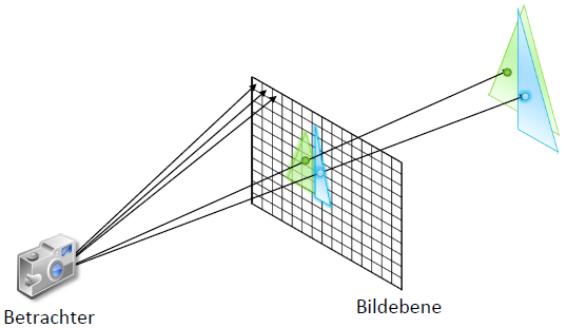

Abbildung 2.5: Illustration der Projektion von Objekten auf die Bildebene mit Hilfe von Raytracing. [Dr 17]

Pixel annehmen kann. Dadurch wird die Szene auf den Bildschirm, wie in Abbildung 2.5 illustriert wird, projiziert.

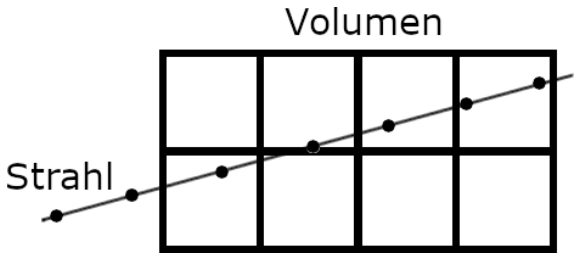

Abbildung 2.6: Illustration des Abtasten eines Strahls in 2D. Jeder ausgefüllte Kreis stellt einen Abtastpunkt dar.

2.3.2 Volumenrendering

Das Volumen ist ein dreidimensionales Bild und besteht aus 3D Voxel. Ein Voxel ist eine Box mit einer Position, einer Ausdehnung in drei Dimensionen und einem Dichtewert. Ein Volumen kann mit Hilfe von Raycasting abgetastet und auf den Bildschirm projiziert werden. Dabei wird für jeden Pixel ein Strahl in die Szene gesendet und in Intervallen abgetastet. Abbildung 2.6 skizziert dies in einer 2D Ansicht. Für jeden Abtastpunkt kann mit Hilfe einer Transferfunktion ein Farbwert ermittelt werden. Zusammen ergeben die abgetasteten Werte einen endgültigen Farbwert, der das Ergebnis des Raycasts für einen Strahl repräsentiert und dem zugehörigen Pixel zugewiesen werden kann.

2.3.3 Transferfunktion

Die Transferfunktion ist ein mächtiges Werkzeug für die Visualisierung von Volumendaten. Mit Hilfe einer Transferfunktion können bestimmte Bereiche des Volumens verschieden stark ausgeblendet oder auch farbig dargestellt werden. Die Transferfunktion bestimmt für einen Voxel, abhängig von seiner Dichte, einen Farb- und Opazitätswert. Dadurch können zum Beispiel nur die Strukturen innerhalb des Volumens, die eine hohen Dichte haben, farbig und kräftig hervorgehoben werden. Strukturen mit einer geringeren Dichte könnten transparenter und weniger farbintensiv dargestellt werden.

2.4 Eyetracking

Für viele wahrnehmungsorientierte Methoden ist es notwendig, den aktuellen Blickpunkt des Betrachters auf dem Bildschirm zu kennen. Ein Eyetrackinggerät ermöglicht die Berechnung der Bickposition des Betrachter auf einem Bildschirm in Realzeit und ist daher auch für interaktive Anwendungen geeignet.

Eyetracking ist das Messen von Augenaktivitäten eines Betrachters. In der Regel schaut der Betrachter dabei auf einen Bildschirm, währenddessen ein Eyetracker seine Augenaktivitäten misst. Aus den Augenaktivitäten können Informationen über den Blickverlauf und die Fixationsdauer bestimmter Punkte auf diesem Bildschirm berechnet werden. Diese Informationen ermöglichen es, Rückschlüsse über das betrachtete Bild zu ziehen, wie zum Beispiel welche Objekte eines Bildes besonders interessant für den Betrachter sind. Eyetracking ermöglicht aber nicht nur das Messen von Daten, um diese im Nachhinein auszuwerten, sondern auch das Nutzen solcher Daten für interaktive Anwendungen. Da in dieser Arbeit ein externer *Tobii Eyetracker* verwendet wurde, beziehen sich folgende Informationen auf *Tobii Eyetracker*. Die Firma *Tobii AB* beschreibt Eyetracking als eine Technologie, die es ermöglicht, ein Gerät durch die natürliche Bewegung der Augen zu steuern [toba]. Dabei werden vier grundlegende Bestandteile des Eyetrackings genannt. In der Regel benötigt man für das Messen von Augenaktivitäten zwei grundlegende Dinge, die auch in Abbildung 2.7 dargestellt sind: Eine Lichtquelle (2) und eine Kamera (3).

Es gibt verschiedene Möglichkeiten, die Augenaktivitäten zu messen. Die am häufigsten genutzte Technik, welche auch von den *Tobii Eyetrackern* genutzt wird, ist *Pupil Centre Corneal Reflection (PCCR)*. Die Lichtquellen sind dabei auf die Augen gerichtet und erzeugen auf ihnen ein spezielles

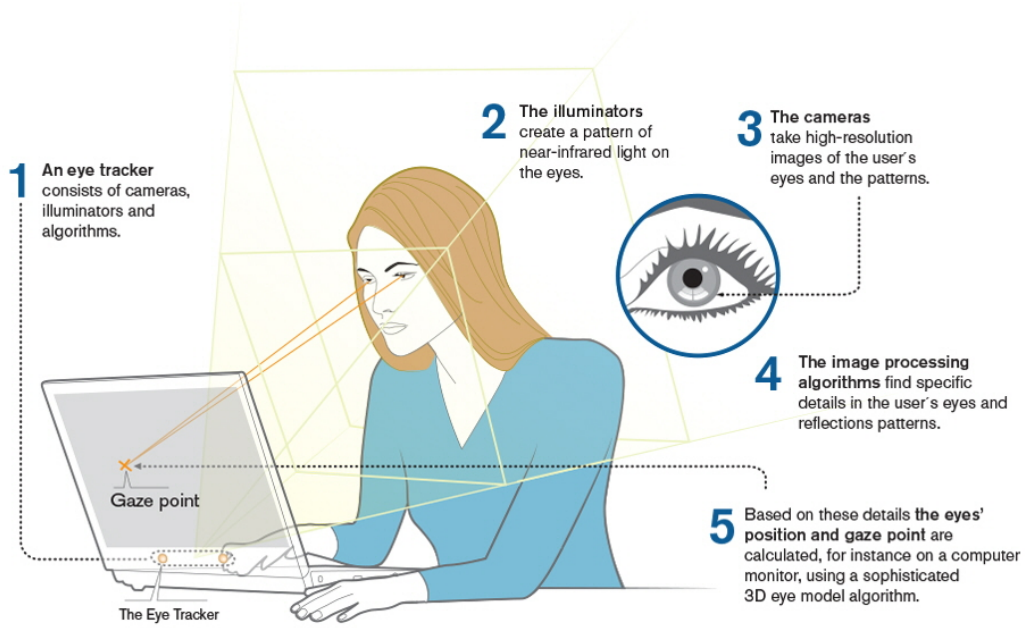

Abbildung 2.7: Illustration der Funktionsweise von Tobii Pro Eyetracker. [tobb]

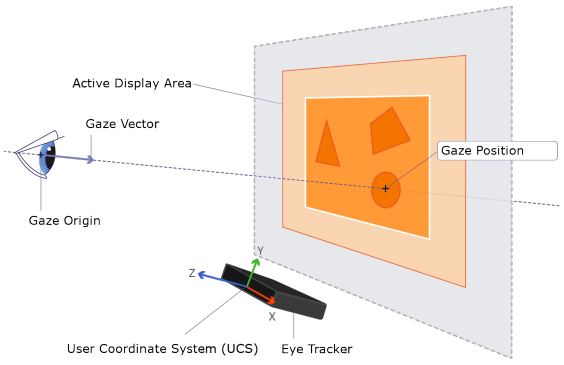

Abbildung 2.8: Nutzer Koordinatensystem [tobc]

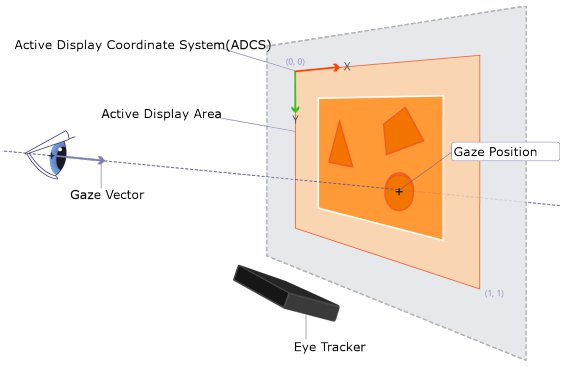

Abbildung 2.9: Aktives Display Koordinatensystem [tobc]

Reflektionsmuster. Die Kamera des Eyetrackers (3) nimmt mit einer hohen Abtastrate Bilder der Augen und ihrer Reflektionsmuster auf. Nun können spezielle Bildverarbeitungsalgorithmen (4) auf die erfassten Daten angewendet und die Reflektionspunkte auf den Bildern bestimmt werden. Anhand dieser Punkte und eines Modells des Auges kann die Blickrichtung der Augen, die Position der Augen im Raum und die Blickpunkte der Augen auf dem Bildschirm berechnet werden (4 + 5).

Die Position eines Auges im Raum wird als *Gaze origin* bezeichnet und wird bei *Tobii Pro Eyetrackern* im Nutzer Koordinatensystem angegeben, siehe Abbildung 2.8. Der Blickpunkt des Auges auf dem Bildschirm ist in der Regel das, was von größtem Interesse ist und wird als *Gaze point* bezeich-

net. Der *Gaze point* wird im *Aktiven Display Koordinatensystem* angegeben (siehe Abbildung 2.9). Dieses Koordinatensystem hat den Ursprung an der Position oben links des aktiven Bildschirmbereichs und der Punkt (1,1) ist an der Position rechts unten des aktiven Bildschirmbereichs.

Die Umrechnung in Bildschirmkoordinaten funktioniert durch die Multiplikation mit der Breite und Höhe des Bildschirms in Pixel. Die Koordinaten sind dann relativ zu dem aktiven Display. Werden mehrere Bildschirme verwendet, muss möglicherweise ein Offset auf die Werte gerechnet werden, um die richtigen Bildschirmkoordinaten für den Bildschirm, auf dem die Anwendung dargestellt wird, zu erhalten.

In Abschnitt 2.2 wurden verschiedene Arten von Augenbewegungen angesprochen. Eine sehr wichtige Rolle spielen dabei Fixationen und Sakkaden. Bei der Analyse der visuellen Wahrnehmung des Menschen ergeben die Daten der Fixationen wichtige Informationen über Eigenschaften des visuellen Stimuli. Bei einer Fixation fixieren die Augen einen gewissen Punkt für eine vergleichsweise lange Zeit (100 ms - 1.5 s) und können dadurch viele Informationen des fixierten Objekts erfassen. Sakkaden hingegen sind kurze (20 ms - 40 ms) und schnelle Augenbewegungen zwischen zwei Fixationen. Für wahrnehmungsorientierte interaktive Anwendungen ist es daher wichtig, eine hohe Abtastrate und niedrige Latenz bei der Erfassung der Augenaktivitäten zu haben. Dies ermöglicht es ohne einer merkbaren Verzögerung das Bild entsprechend des zuletzt gemessenen Blickpunktes darzustellen. So sollte beim wahrnehmungsorientierten Renderings, wobei nur im fovealen Bereich das Bild mit hoher Auflösung dargestellt wird, ein Update circa zwischen 5 ms bis 60 ms nachdem die Augenbewegung gestartet wurde, ausgeführt werden, damit eine Bildveränderung nicht wahrgenommen wird. Diese Zeit ist davon abhängig, wie weit die unscharfe Fläche von der Fovea entfernt ist [WSR+].

Der in dieser Arbeit verwendete *Tobii Pro Eyetracker* hat eine Abtastfrequenz von über 600 Hz. Dies entspricht einem Abtastintervall von 1.67 ms und ist verglichen zu anderen aktuellen Eyetrackern sehr gut. Diese haben oft eine Abtastfrequenz von 60 Hz oder 120 Hz. Geräte mit Abtastraten im Frequenzbereich von 600 Hz und höher sind ausreichen schnell um Sakkaden messen zu können und haben eine auch eine entsprechend niedrige Latenz. Hohe Abtastraten bedeuten aber auch viele Daten pro Sekunde.

Für die Anwendung in dieser Arbeit ist es wichtig, den aktuellsten Blickpunkt des Auges zu kennen. Daher wird der letzte gültige gemessene *gaze point* des Eyetrackers für das Berechnen des nächsten Bildes verwendet. Die Daten, die über die *Tobii Pro SDK* durch den Eyetracker geliefert werden, enthalten dafür einen Validity Code. Dieser Wert gibt an, ob eine Messung mit hoher Wahrscheinlichkeit richtige Daten enthält. So können potentiell fehlerhafte Daten ignoriert werden.

2.5 GPU Architektur

Ein wichtiger Gesichtspunkt bei interaktiven Anwendungen ist die Performanz. Parallelisierung ist ein wichtiger Ansatz, um die Performanz von Anwendungen beziehungsweise ihrer Algorithmen weiter zu verbessern. Gerade Algorithmen in der Bildberechnung und -Verarbeitung eignen sich besonders, um diese zu parallelisieren. In der Bildverarbeitung werden viele gleiche Berechnungen auf unterschiedlichen Daten ausgeführt. *Graphics Processing Units (GPUs)* oder auch Grafikkarten sind für diese Art von Berechnungen optimiert. Im Vergleich zu *Central Processing Units (CPUs)*

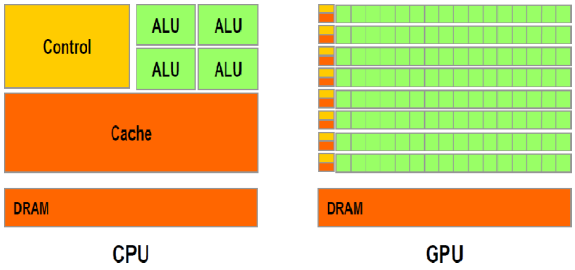

Abbildung 2.10: CPU vs. GPU. []

besitzen GPUs eine deutlich höhere Anzahl an Prozessoren (siehe Abbildung 2.10). Aktuell besitzen Prozessoren meist vier oder acht Rechenkerne und eine Taktfrequenz von circa 4 Ghz. GPUs hingegen haben mehrere tausend Berechnungseinheiten die gleichzeitig Berechnungen durchführen können. Die Taktfrequenz ist jedoch deutlich geringer.

Fast jeder Computer besitzt heutzutage hoch parallele Einheiten, meist eine GPU. Bis vor kurzem waren GPUs meist eng mit grafischen Berechnungen verbunden, um die Berechnung der Farbwerte von Pixeln zu beschleunigen. Heute werden GPUs zur Beschleunigung von fast beliebigen Anwendungen genutzt. Programmierschnittstellen wie *OpenCL* oder *CUDA* ermöglichen die Nutzung von GPUs für allgemeine Berechnungen, daher werden diese heutzutage auch als *General Purpose Computation on Graphics Processing Unit (GPGPU)* bezeichnet und verwendet.

OpenCL

Die *OpenCL (Open Computing Language)* ist ein Standard für das allgemeine Programmieren für parallele CPU- oder GPU-Plattformen. Dabei stellt OpenCL eine Programmierschnittstelle für das Koordinieren paralleler Berechnungen auf unterschiedlichen Plattformen zur Verfügung sowie eine spezielle Programmiersprache für die Programmierung von Programmen, die auf der GPU ausgeführt werden.

Eine OpenCL Anwendung ist die Kombination des Codes eines Programms, das auf dem Host und den OpenCL Devices ausgeführt wird. Der Host ist dabei der Teil der Anwendung, der mit dem OpenCL Kontext über die OpenCL API kommuniziert. Ein Kontext ist die Umgebung, in welche ein OpenCL Kernel ausgeführt wird und weitere für die Ausführung relevante Eigenschaften definiert sind. Der Kernel ist eine Funktion, die in einem OpenCL Programm geschrieben wurde und auf einem OpenCL Device ausgeführt wird. Ein Device entspricht meistens einer GPU oder CPU, die OpenCL implementiert.

OpenCL Plattform Modell

Das Plattform Modell von OpenCL, dargestellt in Abbildung 2.11, ist eine Abstraktion davon, wie OpenCL die Hardware sieht. Die Beziehung zwischen seinen Einheiten und der Hardware ist dabei größtenteils von der verwendeten Hardware und der entsprechenden Implementierung von OpenCL abhängig. Das Modell besteht aus einem Host, welcher mit ein oder mehreren Devices verbunden ist. Diese sind unterteilt in *Compute Units (CUs)*, welche weiter in *Processing Elements (PEs)* unterteilt sind. Berechnungen auf einem solchen Device werden in Processing Elements ausgeführt.

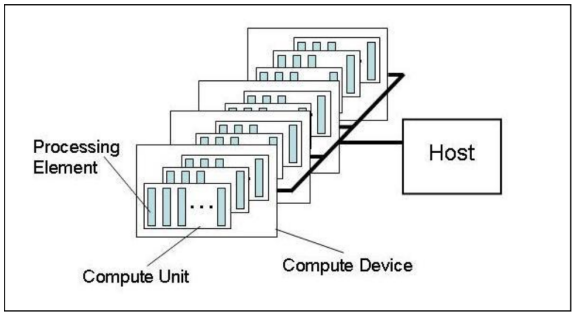

Abbildung 2.11: OpenCL Plattform Modell. Ein Host, mehrere Compute Devices mit je einer oder mehreren Compute Units. [Gro]

Die Host-Anwendung gibt den Start des Kernel-Codes in Auftrag. Das OpenCL Device führt dann den Kernel Code auf den Processing Elements des Devices aus und hat dabei eine große Freiheit darüber, wie die Berechnungen auf die Processing Elements abgebildet werden. Falls die Processing Elements innerhalb einer Compute Unit die gleiche Folge von Befehlen ausführen, bezeichnet man ihren Befehlsfluss als konvergiert, sonst als divergiert.

OpenCL Ausführungsmodell

Der Host kann über Funktionen der OpenCL API mit einem Device über eine *Command-Queue* interagieren. Eine Command-Queue ist mit maximal einem Device verknüpft. Über die Command-Queue können Befehle zum Starten eines Kernels, zum Transferieren von Daten zwischen Host und Device und Befehle zur Synchronisation ausgeführt werden. Ein Befehl durchläuft dabei immer sechs Zustände: *Queued (Eingereiht)*, *Submitted (Übermittelt)*, *Ready (Bereit)*, *Running (Ausführend)*, *Ended (Beendet)* und *Complete (Abgeschlossen)*. Wird ein Kernel für die Ausführung übermittelt, wird für diesen ein Index-Raum definiert. Der Kernel selbst, die zugehörigen Parameter und die Parameter, die seinen Index-Raum definieren, definieren eine Kernel Instanz. Wird eine Kernel Instanz auf dem Device ausgeführt, wird für jeden Punkt in seinem Index-Raum eine Ausführung des Kernels gestartet. Eine solche Ausführung wird Work-Item genannt. Work-Items, die zu einer bestimmten Kernel Instanz gehören, werden von dem Device in Gruppen gehandhabt. Diese Gruppen heißen Work-Groups.

Der Index-Raum ist durch eine *NDRange* definiert, wobei ein Index-Raum immer N-dimensional ist und N die Werte 1, 2 oder 3 annehmen kann. Eine NDRange wird dementsprechend durch drei Integer-Arrays der Länge N definiert: Die Ausdehnung des Index-Raums, ein Offset der Indices an dem sie starten und die Größe der Work-Groups, jeweils in N Dimensionen.

Jedes Work-Item hat ein N-Dimensionales Tupel, welches seine globale ID repräsentiert. Work-Groups erhalten ebenfalls N-Dimensionale Tupel als ID. Die Anzahl von Work-Groups ist abhängig von der definierten Größe von Work-Groups und der Größe des Index-Raums. Work-Items sind je einer Work-Group zugewiesen und haben eine loakle ID innerhalb ihrer Work-Group. Dadurch sind Work-Items eindeutig auf zwei verschiedene Arten definiert: Durch die globale ID beziehungsweise den globalen Index und durch die ID ihrer Work-Group zusammen mit ihrer lokalen ID in dieser Work-Group. Abbildung 2.12 illustriert diesen Sachverhalt bei einem zweidimensionalen Index-Raum.

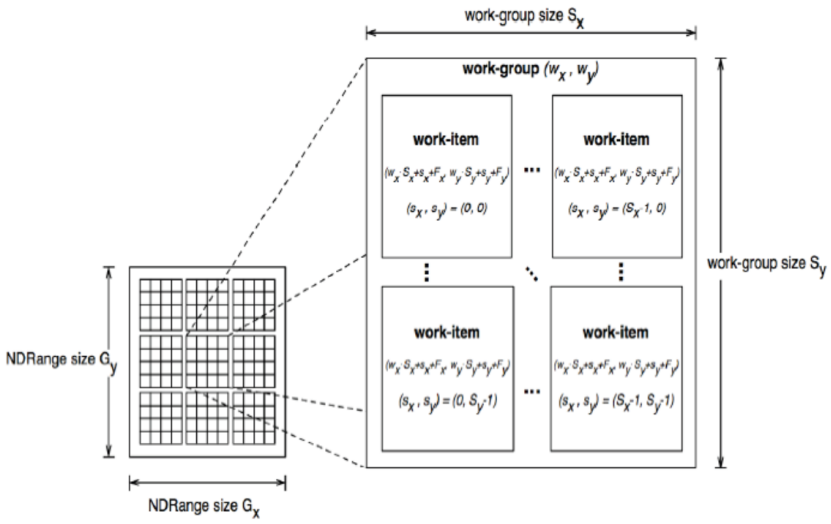

Abbildung 2.12: OpenCL NDRange-Mapping mit 2 Dimensionen. [Gro]

Wird die Ausführung einer Kernel-Instanz gestartet, werden die damit assoziierten Work-Groups in einem Work-Pool platziert und sind somit ausführungsbereit. Das Device entnimmt aus dem Work-Pool nach und nach Work-Groups und führt ein oder mehrere davon gleichzeitig aus. Da die Work-Groups aus dem Work-Pool in jeglicher Reihenfolge ausgeführt werden können, gibt es keinen sicheren Weg, verschiedene Ausführungen von Work-Groups zu synchronisieren.

Hardware Mapping

In der GPU Architektur gibt zwei große Konzepte, *globalen Speicher* und *Streaming Multiprozessoren (SM)*. Der globale Speicher einer GPU entspricht dem RAM einer CPU. Auf diesen kann von den Recheneinheiten der GPU und auch durch die Host-Anwendung auf der CPU zugegriffen werden. Eine GPU hat mehrere Streaming Multiprozessoren. Streaming Multiprozessoren führen die eigentlichen Berechnungen aus und haben jeweils eigene Control Units, Register (lokaler Speicher), Ausführungs-Pipelines und Caches. Ein Streaming Multiprozessor besitzt viele Skalare Recheneinheiten die, ähnlich einer ALU, Berechnungen ausführen.

Im GPU Ausführungsmodell (siehe Abbildung 2.13), werden *Threads* von Skalaren Prozessoren ausgeführt. Ein *Thread Block* ist eine Gruppe von Threads und wird auf einem Streaming Multiprozessor ausgeführt. Dabei werden Threads innerhalb eines Thread Blocks ausschließlich von dem selben Streaming Multiprozessors ausgeführt. Es könnenn aber auch mehrere Thread Blöcke gleichzeitig auf einem Streaming Multiprozessor ausgeführt werden. Dies ist aber durch die verfügbaren Ressourcen limitiert. Abbildung 2.14 stellt den Zusammenhang zwischen Threads und Speicher in CUDA dar.

Ein Thread Block besteht meistens aus 32 Threads und wird in einer NVIDIA Architektur *Warp* und in einer AMD Architektur *Wavefront* genannt. Ein solcher Thread Block wird physisch parallel auf einem Streaming Multiprozessor als *Single Instruction Multiple Thread (SIMT)* Ausführung

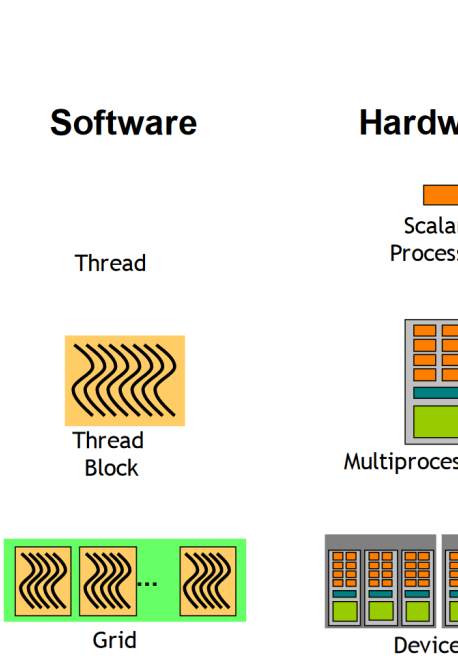

Abbildung 2.13: Ausführungsmodell NVIDIA GPU Architektur. [Lar16]

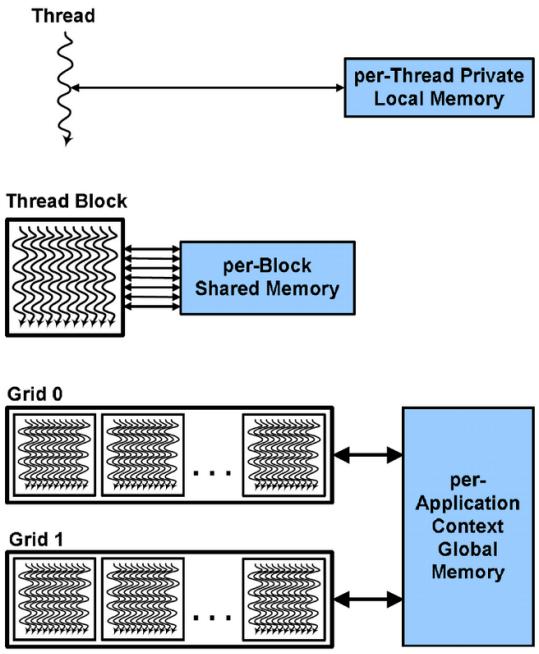

Abbildung 2.14: CUDA Thread- und Speicher Hierarchie. [NVI]

ausgeführt. Die Ausführung eines Thread Blocks wird durch einen Warp-, beziehungsweise Thread-Block Scheduler, geregelt. Da Threads innerhalb eines Thread Blocks differenziert werden können, ermöglicht dies eine *Single Instruction Mutlitple Data (SIMD)* Ausführung. Ein Programm (Kernel) auf der GPU wird als Gitter aus Thread Blöcken gestartet.

OpenCL sieht die Hardware aus abstrakter Sicht und die exakte Beziehung des Plattform Modells zu der Hardware wird von OpenCL nicht festgelegt. Das Plattform Modell besteht aus einem Host und OpenCL Devices mit je mehreren Compute Units die jeweils ein oder mehreren Processing Elements enthalten. In der Realität entspricht der Host in der Regel dem Teil der Anwendung, der auf der CPU ausgeführt wird. Das OpenCL Device entspricht meist einer GPU und eine Compute Unit kann als Streaming Multiprozessor gesehen werden wobei die Processing Elements dementsprechend den Skalaren Prozessoren innerhalb eines Multiprozessors entsprechen.

Nach OpenCL entspricht eine Work-Group einer Menge von verwandten Work-Items, die alle innerhalb der selben Compute Unit arbeiten. Ein Work-Item kann von einem oder mehreren Processing Units als Teil einer Work-Group verarbeitet werden. Sie führen die selbe Kernel Instanz aus und haben gemeinsamen lokalen Speicher und Work Group Funktionen, wie in Abbildung 2.15 dargestellt ist. Daher ist es naheliegend, dass Work-Groups in Thread Blöcke unterteilt werden und alle Thread-Blöcke einer Work-Group auf dem selben Streaming Multiprozessor ausgeführt werden.

In OpenCL können Work-Groups weiter in Sub-Groups unterteilt werden. Eine OpenCL Sub-Group ist eine implementationsabhängige Gruppierung von Work-Items innerhalb einer Work-Group. Diese sind eindimensional und haben immer, mit Ausnahme der letzten Sub-Group, dieselbe Größe. Es ist ebenfalls naheliegend, dass eine solche Sub-Group einem Thread Block entspricht.

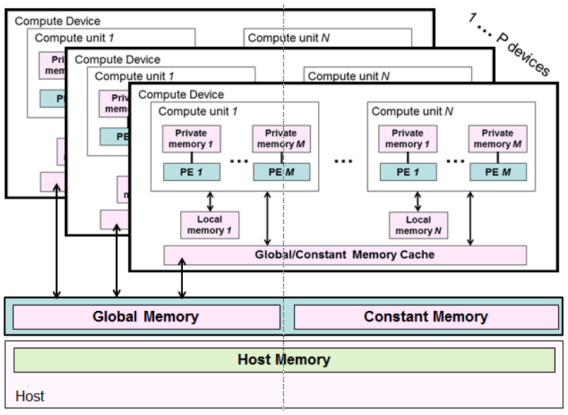

Abbildung 2.15: OpenCL Speicher Modell. [Gro]

Threads innerhalb eines Thread Blocks können also ausschließlich die selben Instruktionen ausführen. Müssen einige Threads innerhalb eines Thread Blocks aufgrund ihrer ID unterschiedliche Statements ausführen, resultiert dies darin, dass unter Umständen der gesamte Block alle Statements ausführt und am Ende der Berechnung die richtigen Ergebnisse auswählt. Dies hat zur Folge, dass ein Thread Block der divergiert deutlich längere Ausführungszeiten hat, als ein konvergierender Thread Block. Bei der Implementierung eines OpenCL Kernels kann eine geschickte Strukturierung des Codes daher das gleiche Ergebnis mit besserer Performanz generieren.

Die Wahl der Work-Group Größe ist ebenfalls wichtig. Da Thread Blöcke aktuell meist eine Größe von 32 Threads besitzen und unter der Annahme, dass Work-Groups in Thread Blöcke unterteilt werden, macht es Sinn, die Größe einer Work-Group als Vielfaches von 32 zu wählen. Ansonsten kann es passieren, dass Thread Blöcke zur Ausführung gestartet werden, bei denen eine große Anzahl an Threads kein Teil der Berechnung sind und Ressourcen dadurch verschwendet werden.

Ein weiteres Problem sind Speicherkonflikte bei der Ausführung eines Kernels. Der Lokale Speicher von Multiprozessoren ist in Speicherbänke unterteilt. Versuchen Threads eines Thread Blocks auf verschiedene Adressen innerhalb der selben Speicherbank zuzugreifen, entstehen Speicher Konflikte, da eine Speicherbank nur eine ihrer Adressen gleichzeitig adressieren kann. Ausführungen eines Thread Blocks mit Speicher Konflikten müssen serialisiert werden und dauern daher deutlich länger. Ein Sonderfall liegt vor, wenn alle Threads eines Blocks auf dieselbe Adresse innerhalb einer Speicherbank lesend zugreifen. In diesem Fall wird der gelesene Wert an alle Threads gleichzeitig übertragen.

3 Entwurf

Die Implementierung dieser Arbeit beruht auf einem Visual Studio Projekt zum Volumen Rendering von Valentin Bruder. Der erste Abschnitt des Entwurfskapitels beinhaltet den Ausgangspunkt der Implementierung dieser Arbeit in Form einer Beschreibung des ursprünglichen Projekts. Dies umfasst die allgemeine Architektur der Anwendung und die Umsetzung des Volumenrenderings durch eine Implementierung eines Raycasters als OpenCL Kernel. Der zweite Abschnitt des Kapitels umfasst Überlegungen für die Erweiterung des Projektes, bezüglich des Ausgangspunktes wie er im ersten Abschnitt beschrieben wurde und dem Ziel dieser Arbeit, sowie die daraus entstandenen Arbeitspakete und die Ansätze für ihre Integration in das bestehende Projekt.

3.1 Projekt

Das Visual Studio Projekt, welches als Ausgangspunkt dient, ist eine auf Qt basierende Anwendung. Qt ist ein vollständiges Cross-Platform Softaware Development Framework, welches in c++ entwickelt wurde, und ermöglicht die einfache Erstellung von Anwendungen mit Benutzeroberflächen und bietet außerdem eine Vielzahl von Bibliotheken, die für eine leichtere und schnellere Entwicklung von Programmen genutzt werden können.

Die Hauptelemente für die Qt Benutzeroberfläche sind Qt Widgets. Widgets können Daten darstellen und Nutzereingaben erkennen. Außerdem stellt ein Widget selbst ein Container für weitere Widgets dar, welche in diesem gruppiert werden. Innerhalb eines Widgets können Elemente platziert werden, welche Informationen, mögliche Operationen oder Nutzereingaben repräsentieren. Für die bequeme Erstellung der grafischen Oberfläche bietet Qt die Anwendung Qt Designer. Der Qt Designer ermöglicht es, Widgets und andere Bausteine der grafischen Oberfläche per Drag und Drop anzuordnen. Die durch den Qt Designer definierte Oberfläche kann abgespeichert werden und wird von Qt verwendet, um eine c++ Header File zu erstellen. Dies ermöglicht es die verschiedenen Elemente der grafischen Oberfläche an die Logik der Anwendung zu binden. [Qt]

Das Projekt ist dementsprechend in c++ geschrieben und die grafische Oberfläche wurde mit Hilfe des Qt Designers erstellt. Die Oberfläche selbst hat eine Menüleiste, die es unter anderem ermöglicht, Volumendaten oder Transferfunktionen zu laden oder auch aktive Transferfunktionen zu speichern sowie das Erstellen eines Screenshots des zuletzt berechneten Bildes. Den Großteil der grafischen Oberfläche wird durch das Volumenrenderwidget ausgefüllt. Das Volumenrenderwidget ist ein Qt OpenGL Widget und kann für das Darstellen von OpenGL Grafiken verwendet werden. Die berechneten Grafiken werden mit Hilfe des Volumenrenderwidgets dargestellt. Neben dem Volumenrenderwidget gibt es noch ein Widget, welches in drei weitere Widgets unterteilt ist, mit denen Parameter für das Volumenrendering gesetzt werden können. Das erste von ihnen ermöglicht das variieren der Abtastrate im Bildraum, also die Anzahl der Strahlen, die ausgesendet werden, sowie das Setzen der allgemeinen Abtastrate der ausgesendeten Strahlen. Außerdem können hier

weitere Rendering Parameter festgelegt werden, wie die Hintergrundfarbe oder ob Voxel beim Abtasten interpoliert werden. Das zweite Widget ist ein Farbenrad, welches für die einfache Auswahl der Farben einzelner Kontrollpunkte der Transferfunktion verwendet werden kann. Das dritte Widget ermöglicht schließlich das Setzen von Kontrollpunkten der Transferfunktion innerhalb eines Diagramms. Die x-Richtung gibt die Dichte an, auf die sich ein Kontrollpunkt bezieht. Die y-Richtung gibt seinen Opazitätswert an. Die Werte zwischen zwei Kontrollpunkten werden entweder linear oder quadratisch interpoliert. Daher gibt es immer mindestens einen Kontrollpunkt für den Dichtewert null und einen Kontrollpunkt für den Dichtewert eins.

Das Volumenrenderwidget ist ein OpenGL Widget und für die Darstellung der berechneten Bilder zuständig. Das Qt Framework erlaubt es, das Widget in c++ Code mit Logik zu verknüpfen. Dafür existiert in dem Projekt eine Klasse VolumeRenderWidget, welche von QOpenGLWidget erbt. Die grundsätzliche Funktionalität zum Rendern in diesem Widget wird durch die Methode `paintGL()` ausgeführt. Innerhalb dieser Methode wird der Code geschrieben, der für die Darstellung des Bildes nötig ist. Das Darstellen auf dem Bildschirm beziehungsweise in dem Widget wird durch OpenGL realisiert. OpenGL zeichnet dabei aber lediglich eine durch einen OpenCL Kernel zuvor generierte Textur auf ein Fullscreen Quad. Das Management von OpenCL wird hier durch ein Objekt der Klasse `volumerendercl` geregelt. Innerhalb der `paintGL()` Methode wird eine Methode dieses Objekts zum Starten des OpenCL Kernels für den Raycast des Volumenrenderings aufgerufen. Das `volumerendercl` Objekt regelt die Handhabung der verschiedenen Parameter für den Kernel und die Unterteilung der übergebenen Anzahl an Strahlen in x- und y-Richtung, welche der Anzahl zu startenden Work-Items entspricht, in Work-Groups. Außerdem startet es den Kernel, synchronisiert einen gemeinsamen Stopp und speichert die benötigte Zeit der letzten Ausführung des Kernels. Die berechneten Werte der einzelnen Work-Items können bei der Ausführung des Kernels direkt in die OpenGL Textur geschrieben werden. Daher kann nach der Ausführung der aufgerufenen Methode des `volumerendercl` Objekts die Textur direkt gezeichnet werden. Mit Hilfe von Qt Funktionen und der Information über die Ausführungszeit des Kernels werden anschließend noch ein paar Overlays gezeichnet. Unter Anderem eine Anzeige der ungefähren möglichen Anzahl an Bildern pro Sekunde der letzten Ausführungen, um die Ausführungsdauer des Kernels abschätzen zu können.

Raycaster

Der eigentliche Raycast passiert in einem OpenCL Kernel. Die OpenCL Objekte werden von dem `volumerendercl` Objekt gehandhabt, dessen Methoden innerhalb der `paintGL()` Methode aufgerufen werden. Das `volumerendercl` Objekt regelt auch die Übergabe der Parameter an den Raycast Kernel. Dies sind Parameter wie Volumendaten, Transferfunktionswerte und Strahlabtastrate zum lesen, sowie eine 2D-Textur zum schreiben für die berechneten Farbwerte der einzelnen Work-Items. Jedes Work-Item besitzt eine 2D-ID, die einer Position in der Ausgabetextur zugewiesen bekommt. Ein Work-Item ist für das Abtasten eines Strahls verantwortlich. Ein Strahl hat als Ursprung die Position der Kamera und entsprechend seiner ID, beziehungsweise Texturkoordinaten, wird seine Richtung bestimmt. Abhängig von der Abtastrate des Strahls, berechnet sich die Schrittgröße für das Abtasten. Ausgehend von dem Schnittpunkt des Strahls mit der Bildebene wird nun in einer Schleife der Strahl schrittweise abgetastet. Dabei wird für jeden Schritt die aktuelle Position bestimmt, welche normiert und dann dafür genutzt wird, um in dem 3D-Volumen Objekt den Dichtewert für diese Position zu bestimmen. Mit dem Dichtewert und den Daten der Transferfunktion wird anschließend ein Farbwert

berechnet. Dieser Farbwert wird mit den bisherigen gesammelten Farbwerten des Strahls verrechnet, so dass am Ende der Abtastschleife ein einziger Farbwert für den Strahl existiert. Der Farbwert wird zum Schluss an die entsprechende Texturkoordinate in der Ausgabetextur gespeichert.

Der Raycast Kernel hat außer der grundlegenden Raycast Funktion noch weitere Eigenschaften, die die Performanz der Ausführung und die Qualität des Bildes verbessern. So kann *Empty Space Skipping* aktiviert werden, um größere Bereiche mit rein transparenten Voxeln zu überspringen. Dies wird mit Hilfe eines zuvor gröber Berechneten Volumen ermöglicht. Da das Volumen nur eine begrenzte Auflösung hat aber an einer beliebigen Position ein Wert aus dieser 3D-Textur abgerufen werden kann, muss angegeben werden, wie dieser Wert abhängig von den umliegenden Voxel bestimmt wird. Daher kann hier gewählt werden, dass beim Auslesen des Dichtewerts an einer bestimmten Position des Volumens, dieser interpoliert wird. Außerdem kann eine Orthografische Sicht des Volumens aktiviert werden, indem die Strahlen parallel ausgesendet werden und für solide Oberflächen gibt es die Möglichkeit, den Effekt der *Ambient Occlusion* darzustellen.

3.2 Arbeitspakete und Integration

Ausgehend von der in Abschnitt 3.1 beschriebenen Ausgangslage des Projekts, wurden einige Vorüberlegungen und Arbeitspakete erstellt, welche das Ziel hatten, das Projekt so zu erweitern, dass die Aspekte des wahrnehmungsorientierten Volumenrendering veranschaulicht und umgesetzt werden können. Im folgenden werden die für dieses Ziel entstandenen Vorüberlegungen und daraus erstellte Arbeitspakete aufgeführt und die dazugehörigen Ansätze zur Integration in das bestehende Projekt skizziert. Genauere Angaben zur Implementierung bestimmter Arbeitspakete werden im Kapitel 4 vorgestellt.

3.2.1 Einarbeitung in das Projekt

Das erste Arbeitspaket, welches nicht zu vernachlässigen ist, ist die Einarbeitung in das Projekt beziehungsweise in die bestehende Implementierung. Dies erfordert das Einarbeiten in einige Grundlagen der c++ Programmierung sowie in den grundlegenden Umgang mit dem Qt Framework. Da das Projekt ein Visual Studio Projekt ist und Programmierschnittstellen wie OpenCL oder auch Qt verwendet, ist das Erlangen einer Kenntnisse für das richtige Verlinken der Bibliotheken mit dem Projekt auch Teil dieses Arbeitspakets.

3.2.2 Simulieren der Blickposition

Um die Eigenschaften des visuellen Wahrnehmungssystems des Menschen auszunutzen, dass die Genauigkeit des Auges außerhalb des zentralen Bereichs stark abnimmt, ist es notwendig, die Blickposition beziehungsweise den fokussierten Punkt auf dem Bildschirm zu kennen. Ein Eyetracker kann dies messen und die Daten der Anwendung zur Verfügung stellen. Die Einbindung eines Eyetrackers ist für die ersten Arbeitsschritte, wie das Implementieren erster Versuche und den Raycast Kernel wahrnehmungsorientiert umzuschreiben, nicht notwendig und kann dies sogar behindern. Das Projekt bietet aber einfache Möglichkeiten auf Maus- oder Tastatureingaben zu

reagieren. Daher ist das zweite Arbeitspaket, das Erkennen von Mausbewegungen innerhalb des Volumerenderwidgets und das Abspeichern der letzten erkannten Mausposition in einer globalen Variable, um die Blickposition mit der Maus simulieren zu können.

3.2.3 Reduzierung der Strahlabtastrate im peripheren Bereich

Nun ist es möglich, die Mausposition zu verwenden, um den Blickpunkt auf dem Bildschirm zu simulieren. Dies erlaubt das Erstellen und Testen von wahrnehmungsorientierten Implementierung. Das ursprüngliche Projekt stellt zwei Parameter zur Verfügung, welche auf zwei verschiedene Arten die Ausführungszeit des Kernels beeinflussen. Die Abtastrate der jeweiligen Strahlen und die Anzahl der Strahlen in x- und y-Richtung. Da das Verändern der Anzahl an Strahlen in x- und y- Richtung das gesamte Bild betrifft und dies nicht einfach abhängig von dem Abstand zur Mausposition verändert werden kann, ist die Anpassung der Abtastrate einzelner Strahlen für den Anfang einfacher zu gestalten.

Das dritte Arbeitspaket umfasst die Verwendung der an den Raycast Kernel übergebenen Mausposition, um die Abtastrate der jeweiligen Strahlen, abhängig von der Distanz der Bildposition des Strahls zu der Position des Mauszeigers, zu reduzieren. Die Anpassung des Kernels hat zur Folge, dass für jeden Strahl abhängig seiner Distanz zum Mauszeiger eine eigene Abtastrate berechnet wird. Aufgrund dessen, dass wie im Abschnitt 3.1 die Work-Items den Texel zugeordnet sind und die Work-Groups quadratisch angeordnet sind, soll dies bewirken, dass die Work-Items innerhalb der selben Work-Group eine ähnliche Abtastrate für ihren Strahl berechnen und die gesamte Work-Group früher terminieren kann. Da der Kernel erst beendet wird, wenn alle Work-Groups ihre Arbeit abgeschlossen haben, würde eine schnellere Ausführung einzelner Work-Groups eine insgesamt schnellere Ausführung des Kernels und auch eine bessere Performanz beim Berechnen des Bildes bewirken.

3.2.4 Reduzierung der Strahldichte im peripheren Bereich

Trotz der Reduzierung der Abtastrate der Strahlen, wird weiterhin die gleiche Anzahl an Work-Items gestartet. Dies ermöglicht eine weitere Möglichkeit, die Ausführungszeit des Kernels zu reduzieren, indem die Anzahl der Strahlen und somit auch die Auflösung des berechneten Bildes variiert wird. Weniger zu berechnenden Strahlen bedeutet hier auch weniger benötigte Work-Items und Work-Groups, die ausgeführt werden müssen. Weniger Work-Groups bedeutet dann auch eine geringere Ausführungszeit des Raycast Kernels.

Die erste Überlegung diesbezüglich war es, eine Art virtuelle Linse vor die Bildebene zu setzen, die die Strahlen so auf der Bildebene verteilt, dass an der Mausposition eine höhere Strahldichte existiert und diese mit größerem Abstand zur Mausposition abnimmt. So könnte bei einer geringeren Anzahl an Strahlen, an der Mausposition, die den Blickpunkt simuliert, trotzdem die maximale Auflösung erreicht werden. Die Bildpunkte die dann nicht direkt von einem Strahl abgedeckt werden, müssten interpoliert werden. Dieser Ansatz wurde aber verworfen, da die Implementierung der virtuellen Linse und der anschließenden Interpolation sich als zu aufwändig erwies und es deutlich einfacher umzusetzende Alternativen gibt.

Eine Alternative ist es, zwei Mal das selbe Bild, mit jeweils unterschiedlichen Auflösungen, also einer unterschiedlichen Anzahl an Work-Items, zu berechnen. Daher umfasst das nächste Arbeitspaket das Berechnen des Bildes in zwei verschiedenen Auflösungen und die anschließende Zusammenfügung beider Bilder zu einem. Dabei soll der normal aufgelöste Bereich an der Blickposition sein und der restliche Teil des Bildes hat die niedrigere Auflösung. Wie oben beschrieben soll eine Reduzierung der Auflösung auch die Anzahl der gestarteten Work-Items und Work-Groups reduzieren. Dadurch soll das Bild mit einer geringeren Auflösung deutlich schneller berechnet werden können. Trotzdem muss ein Teil des Bildes in normaler Auflösung berechnet werden. Da aber nur ein kleiner Teil in dieser Auflösung berechnet werden muss, soll durch das frühzeitige Abbrechen von Work-Items, die außerhalb dieses Bereichs liegen, die zweite Bildberechnung ebenfalls beschleunigen. Das Ziel ist, dass die Berechnung zweier Bilder in verschiedenen Auflösungen und das anschließende Interpolieren sowie Zusammenfügen ihrer, eine insgesamt niedrigere Berechnungszeit benötigt, als die Berechnung eines Bildes in normaler Auflösung. Zusätzlich soll dadurch, dass ein kleiner Bereich des Ergebnisbildes an der Blickposition, der leicht größer als die projizierte Fovea ist, die normale Auflösung hat und die Auflösung im peripheren Bereich trotzdem ausreichend ist, die Wahrnehmung geschaffen werden, dass das gesamte Bild mit normaler Auflösung berechnet wurde.

3.2.5 Index-Mapping

Das Ziel eines weiteren Ansatzes ist es, die Auflösung von der Mausposition weg, noch weiter zu reduzieren. Zwischen dem Bereich, der in einer normalen Auflösung und dem Bereich, der in einer niedrigen Auflösung berechnet wird, soll dafür ein weiterer Bereich eingefügt werden, um eine starke Differenz der Bildauflösung an aneinandergrenzenden Bereichen zu verhindern.

Das berechnete Bild besteht dann aus drei Bereichen. Ein äußerer Bereich und zwei innere Bereiche. Die inneren Bereiche sollen die Form von Ellipsen haben, werden also durch Ellipsen beschränkt. Der Mittelpunkt der beiden Ellipsen soll dem Blickpunkt entsprechen, so dass die Auflösung in der Fovea am besten ist und nach außen hin abnimmt. Die drei Bereiche werden jeweils interpoliert und ergeben dann das gesamte Bild.

Dies motivierte das nächste Arbeitspaket. Das Berechnen des Bildes mit einer sehr niedrigen Hintergrundauflösung und einer mittleren und normalen Auflösung innerhalb einer mittelgroßen und kleinen Ellipse um dem Blickpunkt. Die Implementierung sollte diesmal aber nicht aus der Berechnung von drei zusammengefügter Bilder bestehen, sondern aus einer Berechnung eines unterschiedlich stark aufgelösten Bildes und der anschließenden Interpolation der einzelnen Bereiche. Um dies umzusetzen wird sich von der Annahme, dass die ID eines Work-Items die Bildposition des zugehörigen Strahls ist, vollständig getrennt. Hier werden nun die IDs der Work-Items auf von ihrer ID her sehr unterschiedlichen Bildpunkte abgebildet.

3.2.6 Auswahl des Eyetrackers

Für das Ersetzen der Mausposition durch Blickpositionen auf dem Bildschirm, ist es notwendig, diese zu messen und die Daten der Anwendung zur Verfügung zu stellen. Dies ist die Grundlage des nächsten Arbeitspakets. Es umfasst die Auswahl eines geeigneten Eyetrackers, sowie die Einbindung

der Programmierschnittstelle des Eyetrackers in das Projekt. Anschließend sollen die erhaltenen Eyetrackingdaten in Blickpositionen innerhalb des Volumerenderwidgets umgerechnet werden, um schließlich die Mausposition mit dem zuletzt gemessenen Blickpunkt zu ersetzen zu können.

Die Auswahl lag in diesem Fall zwischen einer Eyetracking-Brille und dem an einem Monitor angebrachten Tobii Pro Spectrum Eyetracker. Da die Eyetracking-Brille eine deutlich geringere Abtastrate und Präzision als der Tobii Pro Spectrum, welcher mit bis zu 1200 Hz und hoher Qualität Blickbewegungen erfassen kann, fiel die Entscheidung hier auf den monitorbasierten Eyetracker Tobii Pro Spectrum. Die Tobii Pro SDK erlaubte eine einfache Integration des Eyetrackers in das Projekt. Die Daten des Eyetrackers wurden dem Volumerenderwidget über eine Callback-Funktion zur Verfügung gestellt, welche immer die zuletzt erhaltene Blickposition für die Berechnung des nächsten Bildes liefert. Bevor diese in einer globalen Variable abgespeichert wurde, wurde sie anhand der Validity-Codes auf ihre Gültigkeit überprüft. Die Blickposition wird von dem Eyetracker normiert in $[0, 1]^2$ angegeben und musste daher zuvor in die Bildkoordinaten des Volumerenderwidgets umgerechnet werden. Wird nun der Eyetracker verwendet, wird bei der Berechnung eines Bildes statt der letzten Mausposition die letzte Blickposition verwendet.

3.2.7 Testen der Implementierungen und verschiedener Parameter

Nachdem die zuvor genannten Arbeitspakete umgesetzt wurden, existierten drei verschiedene Möglichkeiten, den Raycast für das Volumenrendering durchzuführen. Der standardmäßige Raycast und zwei Variationen, die aus vorhergegangen Arbeitspaketen entstanden sind. Die erste Variation entstand aus dem Arbeitspaket im Teilabschnitt 3.2.4, wobei das Bild einmal mit einem viertel der Auflösung und einmal nur in einem variablen rechteckigen Bereich um den Mauszeiger herum in normaler Auflösung berechnet wurde. Die beiden Bilder wurden anschließend zu einem Bild zusammengefügt. Die zweite Variation entstand aus dem Arbeitspaket im Teilabschnitt 3.2.5. Hier wird der Raycast Kernel nur einmal gestartet und sich von der Idee, dass die ID eines Work-Items der Bildposition seines Ergebnisses entspricht, komplett gelöst. Die IDs der Work-Items werden auf verschiedene Bildpunkte abgebildet und das Bild besteht letztendlich aus drei Bereichen. Der äußerste Bereich hat die schlechteste Auflösung und umrahmt die inneren Bereiche. Die inneren Bereiche haben die Form von Ellipsen und sind um den Mauszeiger als ihren Mittelpunkt herum positioniert. Der innerste Bereich ist dabei kleiner als der mittlere und hat die normale Auflösung. Der mittlere Bereich hat eine Auflösung, die zwischen dem innersten und dem äußersten Bereich liegt, so dass die gesamte Auflösung des Bildes von der Blickposition her nach außen hin in zwei Stufen abnimmt.

Das nächste Arbeitspaket bestand aus dem Testen der Implementierungen. Dabei sollte mit der Implementierung aus Teilabschnitt 3.2.4, abgekürzt mit *MDC*, und der Implementierung aus Teilabschnitt 3.2.5, abgekürzt mit *DDC*, verschiedene Renderingparameter und Transferfunktionen getestet werden. Die erste Variation (*MDC*) erlaubte es dafür, die Größe des normal aufgelösten Bereiches, welcher die Form eines Quadrates hat, zu verändern. Bei der zweiten Variation (*DDC*), kann die Auflösung der verschiedenen Bereiche sowie jeweils die Größe der inneren Ellipse, die den innersten Bereichs begrenzt und der äußeren Ellipse, die den mittleren Bereich begrenzt, angepasst werden.

3.2.8 Erstellen von Messwerten

Mit den Implementierungen *MDC* und *DDC* sowie ihren bestimmten Parametern ist nun das Ziel, ihre Performanz möglichst reproduzierbar zu messen. Da die Veränderungen selbst hauptsächlich im Kernel existierten, sollten die Ausführungszeiten des Kernels von *MDC* und *DDC* gemessen werden, um die beiden Verfahren untereinander und mit der Standardimplementierung besser vergleichen zu können. Trotzdem beeinflusst der Overhead der Verfahren bei der Ausführung der `paintGL()` Methode die reale Performanz, daher sollte diese auch gemessen werden.

Dies ist der Hintergrund für das nächste Arbeitspaket, welches die Integration von Möglichkeiten, um reproduzierbare Messwerte der unterschiedlichen Verfahren *MDC*, *DDC* und des Standard Raycasts zu erstellen, beinhaltet.

Für die Erstellung der Messwerte wurde eine Mausbewegung über das aktuell angezeigte Bild aufgenommen und diese für die unterschiedlichen Varianten mit jeweils unterschiedlichen Volumen und Transferfunktionen wieder abgespielt. Dabei wurde für jede aufgenommene Mausposition eine Messung vorgenommen und gespeichert. Um nicht unnötig viele Messwerte zu erstellen, wurde das Bild in 10×10 Felder unterteilt und immer nur dann eine Mausposition für die Messung verwendet, falls diese in einem anderen Feld war, als die vorhergegangene. Die Speicherung aller relevanten Parameter einer Messung, inklusive der Mausbewegungen, soll möglichst reproduzierbare Messungen ermöglichen.

3.2.9 Darstellen der Messwerte

Für die gemessenen Messwerte ist es nun notwendig, diese in eine geeignete Darstellung zu bringen. Das Ziel war es hier, die Mauspositionen und die jeweils gemessene Ausführungszeit in einem Diagramm darzustellen. Hier wurde sich für eine Repräsentation mittels einer Heatmap entschieden. Das letzte Arbeitspaket umfasste daher die Aufgabe, die Messwerte je Volumen, verwendeter Transferfunktion und verwendetem Verfahren für die Bildberechnung, in einer verständlichen Form darzustellen. Dies wurde in Form von Heatmaps, Boxplots und einer Tabelle umgesetzt. Für die Heatmap sollte das Bild, welches mit der Standardvariante und einer bestimmten Transferfunktion erstellt wurde, als Hintergrund des Plots dienen. Die entsprechenden Mausposition sollen über das Bild als Punkte mit bestimmten Farben eingezeichnet werden. Die Farbe eines Punktes soll angeben, wie lange der Kernel ausgeführt wurde, um das Bild mit der Mausposition an dieser Stelle zu berechnen. Für ein Volumen und einer bestimmten Transferfunktion soll ein solcher Plot jeweils für die Messwerte von *MDC* und *DDC* erstellt werden.

4 Implementierung

Das Implementierungskapitel enthält eine genauere Beschreibung der Implementierung der Arbeitspakete aus Abschnitt 3.2. Die Beschreibungen sollen einen Überblick geben, wie die Arbeitspakete umgesetzt wurden ohne zu stark ins Detail zu gehen.

4.1 Strukturelle Modifikationen

Das Projekt wurde strukturell modifiziert und für die Integration der Arbeitspakete vorbereitet. So wurde im Volumerenderwidget eine globale Variable erstellt, welche den Wert der aktuellen Raycast Methode beinhaltet. In der paintGL() Methode wurde dann über diese Variable in einem switch-case die jeweilige Methode aufgerufen, die die Berechnung des aktuellen Raycasts durchführt. Da die Raycasts durch einen OpenCL Kernel berechnet werden und nicht direkt gerendert werden, werden die Ergebnisse immer in einer Textur abgespeichert, welche durch OpenGL auf ein Fullscreen-Quad gezeichnet wird. Da die MDC Methode zwei verschiedene Texturen verwendet, wurde dementsprechend der Fragment-Shader durch ein switch-case ergänzt, welches für jede Raycast Methode unterschiedliche Anweisungen ausführen kann.

Die Kernfunktion des Raycasts, die Strahlverfolgung, bleibt für alle Raycast Methoden die gleiche. Ein switch-case, relativ am Anfang des Raycast Kernels, führt je nach aktueller Raycast Methode die entsprechenden Operationen aus. Dadurch konnte für alle Raycast Methoden der gleiche Raycast Kernel verwendet werden, welcher lediglich mit unterschiedlichen Parametern ausgeführt wurde.

Die GUI des Projekts wurde mit einigen Funktionen ergänzt, über die bestimmte Parameter verändert werden konnten. Dies ermöglichte insbesondere die einfache Umstellung der Raycast Methoden zur Laufzeit des Projekts.

4.2 Implementierung der Arbeitspakte

Im folgenden werden die genaueren Umsetzungen der Arbeitspakete vorgestellt, insbesondere die Implementierung des MDC und DDC Raycasts sowie die Anpassung der Strahlabtastrate. Zusätzlich wird beschrieben, wie die Messungen im Projekt vorbereitet wurden, so dass Messwerte erstellt werden konnten.

4.2.1 Simulieren der Blickposition

Da die Einbindung des Eyetrackers für das oberflächliche Testen der Raycast Methoden nicht notwendig ist, wurde die Blickposition vorerst mit der Mausposition simuliert. Das Volumerenderwidget verfügt über eine Callback-Methode, die auf Mausbewegungen reagiert. Diese Methoden wurde sich zu Nutze gemacht, so dass durch einen Aufruf dieser Methode, aufgrund einer Veränderung in der Mausposition, sofort die aktuelle Position des Mauszeigers, bezüglich des Volumerenderwidgets, in einer globalen Variable abgespeichert wurde. Damit jede Raycast Methode diese auch zu Verfügung hat, wurde diese zu Beginn der paintGL() Methode dem OpenCL Raycast Kernel übergeben. (Arbeitspaket aus Abschnitt 3.2.2)

4.2.2 Reduzierung der Strahlabtastrate im peripheren Bereich

Die Reduzierung der Strahlabtastrate wurde im Raycast Kernel implementiert. Nachdem alle Operationen der jeweiligen Raycastmodifikationen ausgeführt wurden, wurde für jedes Work-Item die Distanz der zugehörigen Bildkoordinate des entsprechenden Strahls zu der Mausposition berechnet. Im MDC Raycast lag die Bildkoordinate nur normalisiert vor und wurde dementsprechend vor der Distanzberechnung umgerechnet.

Sei nun $\vec{r}$ ein 2D-Vektor, der die x- und y-Position der Bildkoordinate eines Strahls enthält und $\vec{m}$ ein 2D-Vektor, der die Bildkoordinaten der Mausposition enthält. Die Distanz zwischen der Mausposition und einem Strahl wurde wie folgt berechnet: $d = \texttt{length}(\vec{r} - \vec{m})$. Die Funktion `length()` ist eine *Built-In OpenCL Function* und berechnet die Länge eines Vektors. Mit der Distanz d zwischen einem Strahl und der Mausposition wurde ein Faktor sf berechnet: $sf = 1,0 - d/ib$. Hierbei bezeichnet ib die maximale Distanz zwischen zwei Bildkoordinaten, also die Länge der Diagonale des Bildes. Ist $d < 10$ wird die Strahlabtastrate sr nicht verändert. Ansonsten berechnet sich die neue Strahlabtastrate mit $sr = \texttt{max}(sf * sr, t)$, wobei die Funktion `max()` auch eine *OpenCL Built-In Function* ist. Bei einer zu geringen Abtastrate kann es vorkommen, dass vor allem dünne Strukturen eines Volumens von einigen Strahlen gar nicht oder nur sehr geringfügig abgetastet werden und es so sich kleine bis große Löcher und Artefakte innerhalb dieser Strukturen bilden können. $t = 0,25$ hat sich als guter Schwellwert erwiesen, der auch bei großen Distanzen zwischen einem Strahl und der Mausposition kaum Artefakte generiert. (Arbeitspaket aus Abschnitt 3.2.3)

4.2.3 Reduzierung der Strahldichte im peripheren Bereich

Die Reduzierung der Strahldichte im peripheren Bereich wurde durch zwei verschiedene Methoden umgesetzt, die MDC Methode (Abschnitt 3.2.4) und die DDC Methode (Abschnitt 3.2.5). Im folgenden werden die Implementierungen beider Methoden getrennt voneinander beschrieben.

MDC Implementierung

Die Implementierung des MDC Raycasts wurde in einer eigenen Methode des Volumerenderwidgets umgesetzt, die von der paintGL() Methode aufgerufen wird. Für die Berechnung des Raycasts konnte die bisherigen Funktionen des `volumerendercl` Objektes wieder verwendet werden und wurde zwei Mal hintereinander ausgeführt. Zuerst wurde der Kernel mit einer in x- und y-Richtung halbierter

Abbildung 4.1: Ausschnitt einer Berechnung mit dem MDC Raycast, bei der nur der äußere niedrig aufgelöste Bereich angezeigt wird. Der niedriger aufgelöste Bereich hat nur ein Viertel der normalen Auflösung.

Abbildung 4.2: Ausschnitt einer Berechnung mit dem MDC Raycast, bei der nur der innere, normal aufgelöste Bereich angezeigt wird.

Anzahl an Work-Items gestartet. Da ein Work-Item einen Pixel berechnet, wurde dadurch das Bild in nur einem Viertel der ursprünglichen Auflösung berechnet. Daher wurde für die Ausgabe des Raycasts hier eine neue Textur verwendet, deren Größe ebenfalls nur ein Viertel der ursprünglichen Auflösung ist. In dem jeweiligen Teil des switch-case im Rayast Kernel, der beim MDC Raycast ausgeführt wird, konnte über einen Parameter entschieden werden, ob es die erste oder die zweite Ausführung des Raycasts Kernels innerhalb der MDC Methode ist. Ist es der erste Aufruf, so wurden innerhalb eines festgelegten Quadrats um den Mauszeiger herum alle in diesem Quadrat sich befindenden Work-Items zu Beginn des Kernels discarded. Die Ausmaße des Quadrat wurden dem Kernel als Parameter übergeben. Anschließend führte der Raycast Kernel den normalen Ablauf fort. Wurde nun die berechnete Textur auf das OpenGL Fullscreen-Quad projiziert, so sah das berechnete Bild ungefähr so aus, wie es in Abbildung 4.1 dargestellt ist. Der äußere Bereich des Bildes hat hier also nur ein Viertel der normalen Auflösung. Die Interpolation dieser niedrigen Auflösung auf die normale Auflösung wurde dabei durch den OpenGL Fragment Shader beim Auslesen der Texturdaten geregelt. Der innere Bereich des Bildes entspricht den Work-Items, die frühzeitig im Raycast Kernel discarded wurden, wodurch die Textur an dieser Stelle schwarz ist. Bei der zweiten Ausführung wurde der Kernel mit der normalen Anzahl an Work-Items gestartet. Dafür wurde eine andere Textur verwendet, deren Größe der normalen Auflösung entspricht. Im Gegensatz zur ersten Ausführung wurde nun der äußere Bereich, also alle Work-Items, die sich außerhalb des Quadrat befanden, discarded. Entsprechend sah das Ergebnis dieser Ausführung in etwa so aus, wie in Abbildung 4.2. Hier wurde aber die Größe des Quadrats durch einen positiven Offset ergänzt, wodurch bei der Zusammenfügung der Bilder keine schwarze Umrandung des Quadrat übrig bleibt.

Bei der Implementierung mussten beachtet werden, dass aufgrund dessen, dass bei den verschiedenen Ausführungen verschieden viele Work-Items gestartet wurden, die ID's der Work-Items nicht immer mit ihrer zugehörigen Bildkoordinate übereingestimmt haben. Daher wurden alle Werte, die sich

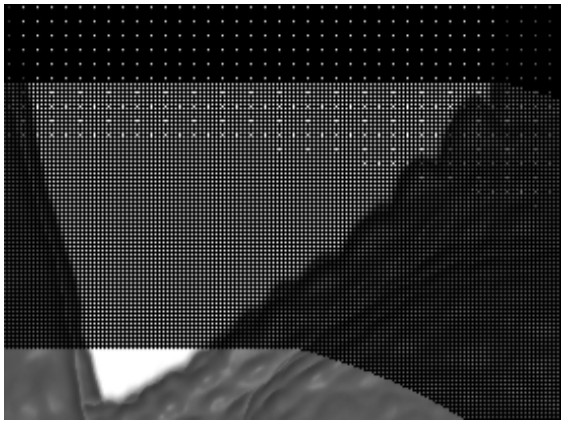

Abbildung 4.3: Ausschnitt einer Berechnung mit dem DDC Raycast ohne einer anschließenden Interpolation der Werte.

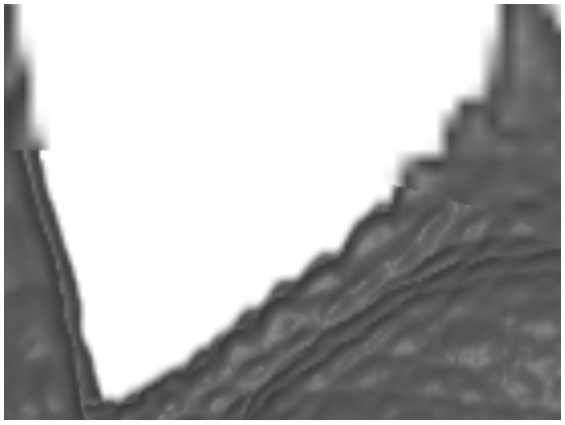

Abbildung 4.4: Ausschnitt einer Berechnung mit dem DDC Raycast mit einer anschließender Interpolation der Werte.

auf Bildpositionen beziehen, normalisiert. Dies waren unter anderem die Mausposition und die Ausmaße des Rechtecks, welche schon im Host-Code der paintGL() Methode normalisiert wurden, sowie die IDs der Work-Items im Kernel.

Das Zusammenfügen der Bilder wurde im OpenGL Fragment Shader durchgeführt. Der Fragment Shader hat dafür die selben Parameter übergeben bekommen, wie der OpenCL Kernel. Eine normalisierte Texturkoordinate wurde dann wie eine normalisierte Work-Item ID behandelt und je nach dem ob sich die Texturkoordinate innerhalb des Quadrats befunden hat oder nicht, wurde der Farbwert in der entsprechenden Textur ausgelesen.

DDC Implementierung

Der DDC Raycast ist die Implementierung des Arbeitspakets aus Abschnitt 3.2.5. Wie für die Implementierung des MDC Raycasts wurde dafür eine eigene Methode geschrieben, die von der paintGL() Methode aufgerufen wird. Auch hier wurden die Funktionen des `volumerendercl` Objektes wiederverwendet und der Raycast über dieses Objekt gestartet. Anders als im ursprünglichen Raycast und MDC Raycast steht der Index eines Work-Items nicht mehr im direkten Zusammenhang mit einer Bildkoordinate, sondern die IDs wurden hier durch eine Funktion auf ganz unterschiedliche Bildkoordinaten zugewiesen. Daher war die Wahl des Verhältnisses der Anzahl der Work-Items in x- und y-Richtung nicht mehr wichtig, sondern ausschlaggebend war nur wie viele Work-Items insgesamt gestartet wurden.

Die DDC Methode sollte drei Bereiche berechnen, die im Anschluss interpoliert werden. Der äußerste Bereich sollte die niedrigste Auflösung haben. Der mittlere Bereich sollte eine dazwischenliegende und der innere Bereich die normale Auflösung besitzen. Der mittlere und innere Bereich sollen dabei durch Ellipsen beschränkt werden und ihr Mittelpunkt liegt auf der Mausposition. Die Auflösungen des mittleren Bereichs und des äußeren Bereichs wurden dafür mit einem Wert angegeben, der die $AnzahlderTexel - 1$ angibt, die je zwei Strahlen in x- und y-Richtung voneinander

entfernt sind. Dieser wert wurde *G-Wert* genannt. Ein G-Wert von 1 würde bedeuten, dass zwei Strahlen ein Texel voneinander entfernt sind. Also zwischen ihnen keine Lücke besteht. Das heißt, ein G-Wert von 1 entspricht der normalen Auflösung.

Die G-Werte der unterschiedlichen Bereiche wurden vor dem Start des Raycasts festgelegt, da diese notwendig sind, um die benötigte Anzahl an Work-Items zu bestimmen. Um die Anzahl der insgesamt benötigten Work-Items abzuschätzen, wurden die Ellipsen als Rechtecke betrachtet. Für die drei Bereiche ergab sich die abgeschätzte Anzahl benötigter Work-Items dann aus $\frac{AnzahlDerPixelDesBereichs}{GWertDesBereichs}$. Für jeden Bereich wurde diese Anzahl anschließend mit einem geringen Offset erhöht, damit später keine Work-Items fehlen. Die Anzahl der Work-Items der Bereiche wurde zusätzlich ohne den Offset dem Raycast Kernel als Parameter übergeben. Die Summe dieser Werte ergab die abgeschätzte gesamte Anzahl an Work-Items, mit welcher der Raycast Kernel gestartet wurde.

Im Raycast Kernel wurden die zweidimensionalen globalen IDs der Work-Items auf eine eindeutige eindimensionale ID abgebildet. Diese waren dem Bereich von 0 bis zur maximalen Anzahl an $gestarteteWorkItems - 1$ zugeordnet. Ist m die Größe der x-Dimension des Kernels und xId und yId die globale ID des Work-Items, so berechnete sich die neue ID mit: $nId = yId * m + xId$. Mit der neuen ID nId und den übergebenen Werten für die maximale Anzahl der Work-Items in den unterschiedlichen Bereichen konnte für jedes Work-Item festgelegt werden, welchem Bereich es zugeordnet werden soll. Da nun der Bereich in welchem sich ein Work-Item befindet, bekannt war, konnte für dieses Work-Item der G-Wert g sowie die ursprüngliche Bereite des Bereiches m bestimmt werden. Das heißt für ein Work-Item, welches im inneren Bereich liegt, galt $g = 1$ und $m = BreiteDerInnerenEllipse$. Ein Work-Item welches im äußeren Bereich lag, hat einen höheren G-Wert zugewiesen bekommen und für m die Breite der Ausgangstextur. Für die Messungen im Abschnitt 5 wurde für den äußeren Bereich ein G-Wert von 7 und für den mittleren Bereich ein G-Wert von 2 verwendet. Mithilfe der bekannten Werte m und g sowie dem Wissen, wie viele Work-Items den unterschiedlichen Bereichen zugeordnet wurden, wurde die nId mit ein paar kleinen Anpassungen wieder zurück in eine zweidimensionale ID überführt, die nun einer Bildkoordinate entspricht. Mit der neuen Work-Item ID kann ein Work-Item die Berechnung eines Strahls, wie bei dem ursprünglichen Raycast ganz normal fortsetzen. Abbildung 4.3 zeigt einen Ausschnitt der berechneten Textur des DDC Raycasts ohne eine anschließende Interpolation der Werte. Man kann erkennen, dass im äußeren Bereich der Abstand zwischen den Strahlen deutlich größer ist, als in dem mittleren und inneren Bereich. Man sieht auch, dass die unterschiedlichen Bereiche sich gegenseitig überlappen. Dies ist für die anschließende Interpolation notwendig, da sonst zwischen den Bereichen schwarze Punkte entstehen können.

Abbildung 4.4 zeigt das in Abbildung 4.3 dargestellte Bild nachdem es interpoliert wurde. Die Interpolation wurde für den DDC Raycast in einem eigenem Kernel implementiert und nicht wie beim MDC Raycast durch den Fragmentshader realisiert. Der Kernel für die Interpolation der Werte wurde für die gesamte Auflösung des Bildes gestartet. Ähnlich wie beim DDC Raycast bestimmt jedes Work-Item den Bereich, in welchem es sich befindet. Dafür ist es hier nicht notwendig, die Indize umzurechnen, da die IDs der Work-Items schon Bildkoordinaten entsprechen und die Mausposition sowie die Ausmaße der Ellipsen dem Kernel übergeben werden. Mit der Information, in welchem Bereich sich ein Work-Item befindet und welche G-Werte und Ausmaße die unterschiedlichen Bereiche haben, wurden für ein Work-Item vier umliegende Werte des zuvor berechneten Raycasts ausfindig gemacht und bipolar interpoliert. Die Ausgabetextur der Interpolation wird durch den Fragmentshader anschließend auf das Full-Screen Quad gerendert.

4.2.4 Erstellen und Darstellen von Messwerten

Für die Umsetzung des Arbeitspakets aus Abschnitt 3.2.8, wurde eine Klasse erstellt, welche für eine Berechnung eines Bildes verschiedene Messwerte speichert. Diese sind unter anderem die gesamte Ausführungszeit des Kernels (bei mehreren Ausführungen innerhalb einer Methode wurden diese addiert), die Ausführungszeit der paintGL() Methode und die Mausposition, die in der letzten Ausführung der paintGL() Methode für die Berechnung des Bildes verwendet wurde. Für jeden Aufruf der `paintGL()` Methode wurde ein solches Objekt erstellt und gespeichert. Die letzte Ausführungszeit eines Kernels konnte mit Hilfe des `volumerendercl` Objektes berechnet werden, welches dafür *OpenCL Profiling* Methoden verwendet hat. Die ungefähre Ausführungszeit der `paintGL()` Methode selbst wurde mit Hilfe eines *Qt QTime* Objektes berechnet.

Um eine Mausbewegung abzuspeichern wurden globale Variablen erstellt, die den aktuellen Zustand des Volumerenderwidgets bezüglich der Messungen angeben. Über Tastatureingaben können diese Variablen verändert werden. Wurde in den Zustand für die Aufzeichnung einer Mausbewegung gewechselt, so hat die Callback-Methode für die Mausbewegungen nicht nur die aktuelle Position in eine globale Variable geschrieben, sondern auch in einen c++ Vektor abgespeichert. Dies passierte aber nur, falls die zuletzt gespeicherte Position sich um mindestens 10 Pixel von der aktuellen gemessenen Position unterscheidet. Während Mausbewegungen aufgenommen wurden, wurden die Ausführung der Raycasts blockiert. Nachdem eine Mausbewegung aufgenommen wurde, konnten die gesammelten Daten gespeichert und wieder geladen werden. Falls es gespeicherte Mausbewegungen gab, konnte über eine Tastatureingabe in den Replay-Zustand gewechselt werden. In diesem Zustand wurden alle Tastatur- und Mauseingaben, bis auf eine Taste, um das Replay vorzeitig abzubrechen, ignoriert. Wurde in den Replay-Zustand gewechselt, wurde die paintGL() Methode für jede gespeicherte Mausposition in dem c++ Vektor einmal ausgeführt und die dabei gesammelten Messwerte in lesbarer Form in eine `.txt`-Datei geschrieben.

Mit Hilfe eines Python Scripts konnten die gespeicherten Messwerte geladen und verwertet werden. Für die Erstellung der Heatmaps und Boxplots wurde *pyplot* aus der Python Bibliothek *matplotlib* verwendet.

5 Ergebnisse und Diskussion

In den Ergebnissen werden die verschiedenen Raycast Methoden auf Bildqualität und Performanz untersucht. Anschließend folgt eine Diskussion bezüglich der verschiedenen Ergebnissen.

5.1 Ergebnisse

Der erste Abschnitt beinhaltet Ergebnisse der Implementierungen des Standard Raycasts, *MDC* (Abschnitt 3.2.4) und *DDC* (Abschnitt 3.2.5) sowie der Implementierung der Variierung der Strahlabtastrate (Abschnitt 3.2.3). Dabei liegt der Fokus auf die Beschreibung der wahrgenommenen Bildqualität der verschiedenen Methoden. Der zweite Abschnitt beinhaltet gemessene Performanzleistungen der jeweiligen Implementierungen bei der Verwendung unterschiedlicher Transferfunktionen und Volumendaten. Die verschiedenen Methoden werden hier im Hinblick auf die benötigten Ausführungszeiten bei Berechnungen mit verschiedenen Volumen verglichen.

5.1.1 Bildqualität

Für die Bildqualität werden Screenshots von Berechnungen mit unterschiedlichen Raycast Methoden vorgestellt, bewertet und verglichen. Die Bewertung erfolgt anhand der wahrgenommenen Bildqualität und wie diese sich verändert, wenn die Bildabtastrate und die Strahlabtastrate variiert werden. Es wird von drei Methoden die Bildqualität untersucht, wobei diese jeweils Ergebnisse mit unterschiedlichen Bildabtastraten produzieren. Diese sind: Standard-, *MDC*- und *DDC*-Methode. Die Bildqualität dieser Methoden wird einmal mit konstanter Strahlabtastrate und einmal mit variierte Strahlabtastrate vorgestellt. Dafür werden zwei Volumen verwendet, wobei die Ergebnisse des ersten Volumens für die unterschiedlichen Methoden getrennt-, und die Ergebnisse mit dem zweiten Volumen im Anschluss für einen besseren Vergleich zusammen vorgestellt werden.

Die Ergebnisse in diesem Abschnitt wurden alle mit einer Auflösung von 2263×1306 Pixel berechnet. Alle Abbildungen, die das selbe Volumen zeigen, wurden mit der gleichen Transferfunktion und aus der gleichen Perspektive berechnet. Die Unterschiede bei der Berechnung ergeben sich ausschließlich aus einer veränderten Bild- oder Strahlabtastrate.

Der Begriff *maximale Bildabtastrate* bezieht sich auf die Bildabtastrate mit einem Wert von circa 1 und bedeutet, dass für jeden Pixel des Bildes ein Strahl berechnet wurde, der den Farbwert des Pixels angibt. Eine Bildabtastrate von $\frac{1}{2}$ bedeutet, dass in x- und y-Richtung nur für jeden zweiten Pixel ein Strahl berechnet wurde. Eine Bildabtastrate von $\frac{1}{7}$ bedeutet, dass in x- und y-Richtung nur für jeden siebten Pixel ein Strahl berechnet wurde. Die Bildabtastrate kann theoretisch aber auch auf einen Wert größer als $1,0$ gesetzt werden. In allen Berechnungen, außer bei der Methode

MDC, wurde die maximale Bildabtastrate $1,0$ gewählt. Aufgrund der Implementierung und der verwendeten Auflösung für die Berechnungen ist die Bildabtastrate für die Methode *MDC* nur $0,96$. Dies wird bei der Bewertung aber wie eine Bildabtastrate von $1,0$ behandelt.

Die Strahlabtastrate gibt an, in welchem Abstand ein Strahl abgetastet wird. Die Abtastdistanz wird mit $\frac{1 \times Voxellaenge}{maximaleStrahlabtastrate}$ berechnet. Die Voxellänge ergibt sich aus der Skalierung der Voxel, welche abhängig von den verwendeten Volumen ist. In allen Berechnungen ist die maximale Strahlabtastrate $1,5$ gewählt worden.

Wurde eine Abbildung mit variierter Strahlabtastrate berechnet, so bedeutet dies, dass die Strahlabtastrate an der Mausposition den maximalen Wert der Strahlabtastrate hat. Mit zunehmender Distanz zur Mausposition nimmt die Strahlabtastrate dann bis auf einen Wert von $\frac{maximaleStrahlabtastrate}{4}$ ab.

Das erste Volumen ist ein ct-scan eines Bonsai Baums. Das Volumen hat eine Auflösung von $256 \times 256 \times 256$ Voxel. Die Slice-Dicke der einzelnen Voxel ist $1,0 \times 1,0 \times 1,0$ und gibt die Skalierung der einzelnen Voxel an. Das heißt, dass die Voxel in x-, y- und z-Richtung hier eine einheitliche Skalierung haben.

Standard Raycast

Abbildung 5.1 zeigt eine Berechnung des Volumens *Bonsai* mit dem Standard Raycast. Die Bildabtastrate ist bei der Berechnung des Standard Raycasts im ganzen Bild homogen und hat den Wert 1. Die Strahlabtastrate ist für das ganze Bild konstant und hat den Wert $1,5$. Das heißt, dass für jeden Pixel des Bildes ein Work-Item gestartet wurde, um den Raycast eines Strahls zu berechnen. Der jeweilige Pixel erhält die berechnete Farbe des Work-Items.

Durch die Interpolation der Voxel wird das Volumen mit weichen Konturen und Flächen gezeichnet. Die Transferfunktion hebt die Voxel mit etwas höherer Dichte farblich hervor. Dadurch sind farblich einheitliche und zusammenhängende Strukturen erkennbar, unter anderem die Blätter, Äste und den Stamm. Die Voxel, die nicht zu dem Bonsai Objekt gehören, zum Beispiel Leerräume, haben teilweise auch Dichtewerte größer null. Daher wurde die Transferfunktion so gewählt, dass diese ebenfalls vollständig ausgeblendet sind. Insgesamt werden die Flächen aber transparent gezeichnet, so dass auch eigentlich verdeckte Strukturen zu sehen sind.

Wie in den Grundlagen im Abschnitt 2.2 zum visuellen Wahrnehmungssystem beschrieben wurde, kann man feststellen, dass wenn ein bestimmter Punkt in der Abbildung 5.1 fokussiert wird, die vom Blickpunkt entfernten Bereiche unscharf erscheinen. Im Standard Raycast werden auch diese mit normaler Bildabtastrate gezeichnet, obwohl dies aufgrund der Limitierungen des visuellen Wahrnehmungssystems, eigentlich nicht notwendig ist.

In Abbildung 5.2 wurde das Volumen auch mit dem Standard Raycast verwendet aber die Strahlabtastrate variiert. Mit zunehmender Distanz zur Mausposition, welche als schwarzes Kreuz eingezeichnet wurde, nimmt die Strahlabtastrate ab. Dies ist selbst an weit von der Mausposition entfernten Objekten, an denen die Strahlabtastrate am niedrigsten ist, kaum wahrnehmbar. Wird die Mausposition selbst fokussiert, ist kein Unterschied feststellbar.

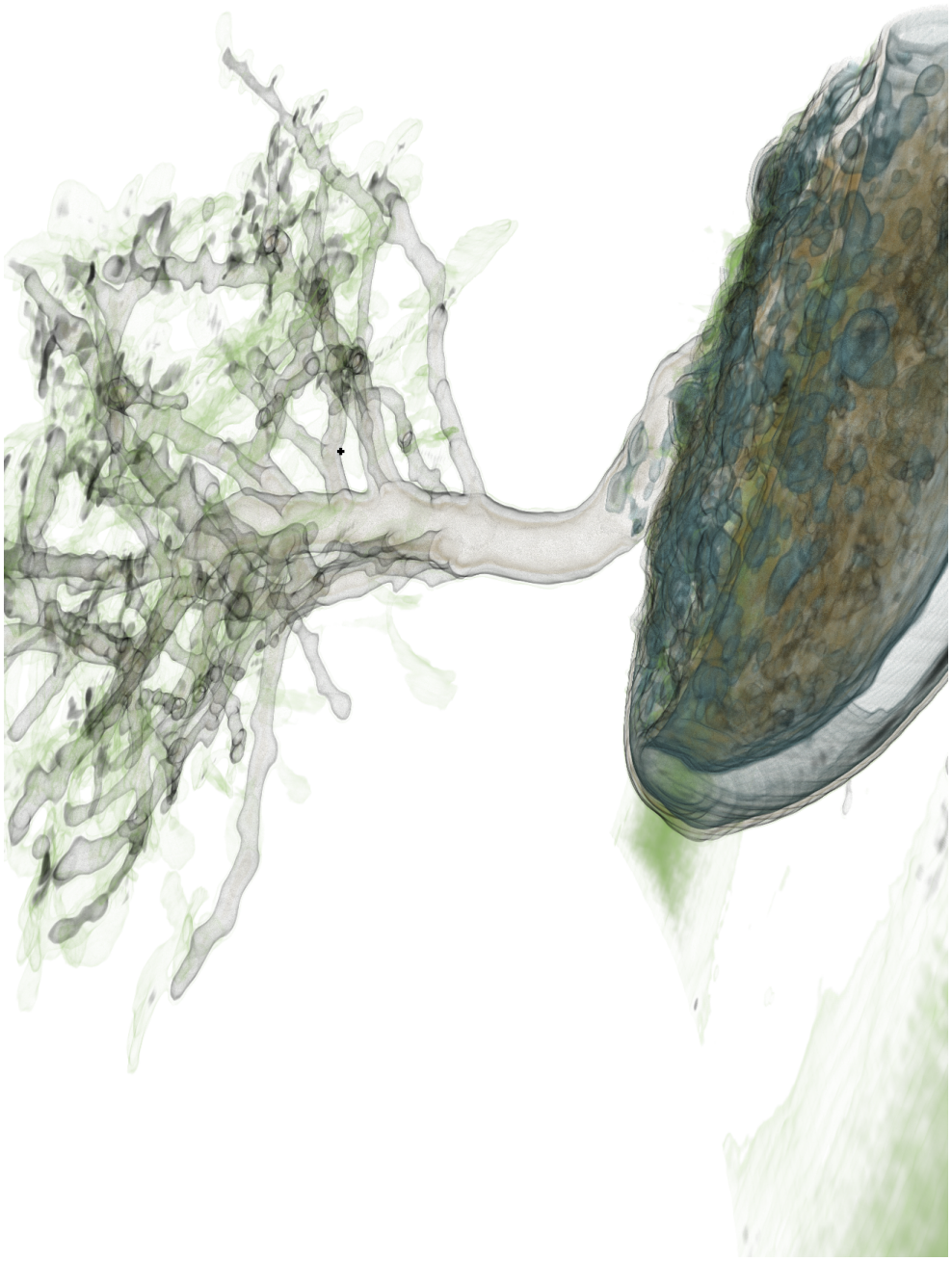

Abbildung 5.1: Volumen Bonsai mit ursprünglichem Raycast berechnet. Das Bild hat eine Bildabtastrate von 1 und eine Strahlabtastrate von 1,5.

Abbildung 5.2: Volumen Bonsai mit modifizierten Standard Raycast berechnet. Die Bildabtastrate beträgt 1. Die Strahlabtastrate hat an der Mausposition einen Wert von $1,5$ und nimmt, abhängig von der Distanz zur Mausposition, bis zu einem Wert von $\frac{1,5}{4}$ ab.

MDC Raycast

Der *MDC* Raycast ist das Ergebnis des Arbeitspakets, aus dem Abschnitt 3.2.4. Die Abbildung 5.3 zeigt das Volumen *Bonsai*, welches mit dem *MDC* Raycast erstellt wurde. Das Bild wurde dabei aus der selben Perspektive und mit der gleichen Transferfunktion berechnet, wie dies auch in Abbildung 5.1 der Fall war.

Die maximale Bildabtastrate in Abbildung 5.3 hat einen Wert von $0,96$ statt einem Wert von $1,0$. Damit wird einer leichten Verschiebung des Bildes, aufgrund der zwei unterschiedlichen Bildabtastraten, entgegengewirkt. Die Anpassung der Bildabtastrate ist abhängig von der Auflösung des Bildes.

In diesem Fall, beim *MDC* Raycast, wurden zwei Bilder berechnet, welche anschließend passend zusammengefügt wurden. Das erste Bild wurde mit nur einem Viertel der Auflösung berechnet, also der halben Bildabtastrate, und anschließend auf die ursprüngliche Auflösung interpoliert. Das zweite Bild wurde mit der ursprünglichen Auflösung berechnet, dafür aber nur ein rechteckiger Ausschnitt um die Mausposition. Anschließend wurde das zweite Bild an der Mausposition in das auf die normale Auflösung interpolierte erste Bild eingefügt.

An den Konturen der Abbildung 5.3 ist ein leichter Abfall der Auflösung des Bildes zu erkennen. Die Konturen, welche in Abbildung 5.1 noch weich gezeichnet wurden, haben außerhalb des Rechtecks um die Mausposition nun Artefakte der Unterabtastung, wie zum Beispiel leichte Treppenstufen. Innerhalb des Rechtecks sind die Konturen immer noch weich gezeichnet und es ist dort kein Unterschied zu Abbildung 5.1 zu sehen.

Die Größe des Rechtecks beträgt 350 Pixel in x- und y-Richtung und ist groß genug, dass die Abnahme der Auflösung im äußeren Bereich, bei Fokussierung der Mausposition, nicht wahrnehmbar ist. Wird bei aktivem Eyetracking anstelle der Mausposition die Blickposition verwendet, so ist der aktuell betrachtete Bereich immer hoch aufgelöst. Ein Unterschied zwischen den beiden Bereichen fällt während einer Fokussierung kaum auf. Wenn der Fokus langsam über das Bild wandert, kann man einen Unterschied zwischen den Auflösungen erkennen. An dem Übergang des normal aufgelösten Bereichs zu dem Bereich mit der halben Auflösung kommt es während einer Bewegung an den Konturen des Volumen zu leichten Veränderungen. Diese können die Aufmerksamkeit auf sich ziehen und daher auffallen.

Obwohl der Bereich außerhalb des Rechtecks mit nur einem Viertel der Auflösung innerhalb des Rechtecks berechnet wurde, ist die Bildqualität in diesem Bereich noch recht gut. Da die Sehschärfe mit zunehmenden Winkel von der Fovea weg immer weiter abnimmt, könnte die Bildabtastrate außerhalb des Rechtecks vermutlich weiter gesenkt werden, ohne dass dies die Bildqualität bei der Verwendung eines Eyetrackers beeinträchtigt.

Abbildung 5.4 zeigt die selbe Berechnung wie Abbildung 5.3 aber mit einer variierten Strahlabtastrate. Anders als in Abbildung 5.2, in welcher ebenfalls die Strahlabtastrate auf die gleiche Weise über das Bild hinweg variiert wurde, macht sich in Abbildung 5.4 an zum Beispiel den Ästen, die geringere Strahlabtastrate bemerkbar. Die Flächen der Äste erscheinen ein bisschen löchrig, da Voxel teilweise übersprungen werden. Wird die Mausposition fokussiert, kann man dies, aufgrund der Distanz zum Blickpunkt, nicht wahrnehmen.

Abbildung 5.3: Volumen Bonsai. Das Bild wurde mit dem *MDC* Raycast berechnet. Ein kleiner Teil des Bildes, in Form eines Rechtecks, hat eine Bildabtastrate von $1,0$. Das restliche Bild hat nur eine Bildabtastrate von $0,5$. Die Strahlabtastrate beträgt für das ganze Bild 1.5.

Abbildung 5.4: Volumen Bonsai. Das Bild wurde mit dem *MDC* Raycast berechnet. Ein kleiner Teil des Bildes, in Form eines Rechtecks, hat eine Bildabtastrate von $1,0$. Das restliche Bild hat nur eine Bildabtastrate von $0,5$. Die Strahlabtastrate hat an der Mausposition einen Wert von $1,5$ und nimmt, abhängig von der Distanz zur Mausposition, bis zu einem Wert von $\frac{1,5}{4}$ ab.

DDC Raycast

Der *DDC* Raycast ist das Ergebnis des Arbeitspakets aus dem Abschnitt 3.2.5. Abbildung 5.5 zeigt das Volumen *Bonsai*, welches mit dem *DDC* Raycast erstellt wurde. Das Bild wurde aus der selben Perspektive und mit der gleichen Transferfunktion berechnet, wie in den Abbildungen 5.1 und 5.3. Die maximale Bildabtastrate ist 1. Diese nimmt in zwei Stufen nach außen hin ab.

Beim *DDC* Raycast wurde nur ein Aufruf des Raycasts gestartet und die Work-Items und ihre entsprechenden Strahlen auf verschiedene Bildpunkte abgebildet. Das gesamte Bild wurde in einem zweiten Kernel Aufruf interpoliert. Innerhalb eines kleinen Bereichs in der Form einer Ellipse um den Blickpunkt herum hat das Bild eine Bildabtastrate von 1, dass heißt jeder Pixel entspricht einem Farbwert der Auswertung eines Strahls. Außerhalb von diesem Bereich aber innerhalb einer zweiten größeren und umschließenden Ellipse hat das Bild in x- und y-Richtung eine Bildabtastrate von $\frac{1}{2}$. Das heißt, dass in x- und y-Richtung nur für jedes zweiten Pixel ein Strahl ausgewertet wird. In dem äußersten Bereich beträgt die Bildabtastrate in x- und y-Richtung $\frac{1}{7}$. Es wird also in jedem 7×7 Pixel-Feld nur ein Strahl ausgewertet. Die die Farbwerte der Pixel, die selbst nicht durch einen Strahl bestimmt wurden, wurden durch vier umliegende Farbwerte von Strahlen bilinear interpoliert.

In Abbildung 5.5 ist deutlich zu sehen, dass der Großteil des Bildes eine geringe Auflösung hat. In dem am niedrigsten abgetasteten Bereich sind an den Konturen, insbesondere die der Äste, deutliche Merkmale der Unterabtastung zu sehen. In Abbildung 5.1 und 5.3 verlaufen die Konturen der Äste noch diagonal und sind relativ weich gezeichnet, hier haben sie starke Treppeneffekte. Manche Konturen sind auch teilweise unterbrochen.

In dem mittleren Bereich ist die Bildqualität recht gut und das Bild scharf zu erkennen. Trotzdem hat das Bild hier nur ein Viertel der Auflösung. Den Übergang zwischen dem mittleren und inneren Bereich, welcher die normale Auflösung hat, ist kaum zu erkennen. Aufgrund des starken Abfalls der Bildabtastrate vom mittleren zum äußeren Bereich, fällt dieser Übergang stärker auf.

Bei aktivem Eyetracking wird anstelle der Mausposition die aktuelle Blickposition verwendet. Der Fokus liegt bei aktivem Eyetracking nun im Zentrum des inneren Bereichs, dieser wurde hier mit der Mausposition simuliert und ist als schwarzes Kreuz eingezeichnet. Der innere Bereich ist groß genug, so dass der Abfall der Bildabtastrate zum mittleren Bereich nicht auffällt. Zusammen ergeben der innere und mittlere Bereich den Teil des Bildes, in welchem das Bild scharf zu sehen ist. Dieser ist ausreichend groß, dass der Bereich an und um dem Blickpunkt immer scharf wahrgenommen wird. Anders als in Abbildung 5.3, ist es in Abbildung 5.5 auch wenn der innere und mittlere Bereich auf den Blickpunkt zentriert sind, auffallend, dass der äußere Bereich mit einer deutlich niedrigeren Bildabtastrate berechnet wurde. Die grundlegenden Konturen des Bildes können aber im äußeren Bereich noch erkannt werden wodurch das Identifizieren von interessanten Objekten immer noch möglich ist.

Abbildung 5.6 zeigt die selbe Berechnung wie Abbildung 5.5 aber mit einer variierten Strahlabtastrate, wie sie in Abbildung 5.2 und 5.4 auch der Fall war. Wie in Abbildung 5.4 erscheinen Teile, der von der Mausposition weiter entfernten Objekte, löchrig. Der Unterschied zwischen variierter und nicht-variierter Strahlabtastrate fällt bei der DDC Raycast Methode ein wenig stärker auf, als bei der MDC Raycast Methode. Trotzdem ist beeinflusst die viel niedrigere Auflösung im äußeren Bereich

die Bildqualität viel maßgebender, als die reduziert Strahlabtastrate, so dass diese kaum mehr die Wahrnehmung beeinträchtigt. Dies ist noch weniger der Fall, wenn die Mausposition fokussiert wird.

Abbildung 5.5: Volumen Bonsai mit *DDC* Raycast berechnet. Die Bildabtastrate nimmt nach außen hin in zwei Schritten ab. An der Mausposition hat sie den höchsten Wert von $1,0$. Etwas weiter außen einen Wert von $0,5$ und noch weiter außen hat sie den niedrigsten Wert von $\frac{1}{7}$. Die Strahlabtastrate beträgt für das ganze Bild 1.5.

Abbildung 5.6: Volumen Bonsai mit *DDC* Raycast berechnet. Die Bildabtastrate nimmt nach außen hin in zwei Schritten ab. An der Mausposition hat sie den höchsten Wert von $1,0$, weiter außen einen Wert von $0,5$, dann einen Wert von $\frac{1}{7}$. Die Strahlabtastrate hat an der Mausposition einen Wert von $1,5$ und nimmt, abhängig von der Distanz zur Mausposition, bis zu einem Wert von $\frac{1,5}{4}$ ab.

Direkter Vergleich

Abbildung 5.7 zeigt eine Berechnung des zweiten Volumens, welche mit dem Standard Raycast und ohne einer variierten Strahlabtastrate berechnet wurde. Anders als das erste Volumen, welches ein ct-scan ist, ist dieses der 1353-te Zeitschritt aus der Simulation einer *shock wave formation in core-collapse supernova* von John Blondin, NCSU. Es hat eine Auflösung von $432 \times 432 \times 432$ Voxel, welche ebenfalls in x-, y- und z-Richtung eine Skalierung von $1,0$ haben. Die Dichte der Voxel gibt die physikalische Entropie an der Position in der Simulation an.

In Abbildung 5.7 sind drei Kästchen eingezeichnet und nummeriert. Für jede der Berechnungsmethoden wurde das Volumen mit der gleichen Transferfunktion und Perspektive beziehungsweise Kameraausrichtung berechnet. Für jede der Berechnungen sind die drei Ausschnitte vergrößert worden und nebeneinander der Reihe nach aufgeführt. Die Mausposition für die jeweiligen Berechnungen war dabei immer an der selben Position und ist jeweils in Ausschnitt 2 eingezeichnet.

Ausschnitt 1 liegt etwas weiter weg von der Mausposition aber circa am Übergang des mittleren Bereichs zum äußeren Bereich des *DDC* Raycasts. Die Strahlabtastrate wäre entsprechend in Ausschnitt 1 ein wenig geringer, als an der Mausposition, falls eine Methode mit variierte Strahlabtastrate verwendet wurde. Ausschnitt 2 enthält die Mausposition und ist dementsprechend jeweils im inneren Bereich mit einer Bildabtastrate von 1. Die Strahlabtastrate ist hier ebenfalls fast maximal bei $1,5$. Ausschnitt 3 liegt relativ zur Auflösung des Bildes weit von der Mausposition entfernt. Die Bildabtastrate beträgt hier für den Standard Raycast 1, für den *MDC* Raycast $\frac{1}{2}$ und für den *DDC* Raycast $\frac{1}{7}$. Wird eine Methode mit variierte Strahlabtastrate verwendet, so hat diese in Ausschnitt 3 fast den minimalen Wert von $\frac{1,5}{4}$.

Da die Mausposition sich innerhalb des zweiten Ausschnitts befindet, ist die Strahlabtastrate und die Bildabtastrate hier immer fast maximal, so dass Ausschnitt 2 in den verschiedenen Methoden fast identisch ist. Auch ist hier auffallend, dass obwohl, dass die Strahlabtastrate in Ausschnitt 3 bei der Verwendung des selben Raycast mit einmal konstanter Strahlabtastrate und variierter Strahlabtastrate, sehr unterschiedlich ist, dies sich bei diesem Bild aber kaum auswirkt und nicht wahrgenommen wird. Womöglich liegt das daran, dass durch die Transferfunktion in diesem Volumen eher größere und dickere Strukturen hervorgehoben wurden und daher selbst bei einer deutlich geringeren Strahlabtastrate diese Strukturen genügend oft abgetastet wurden.

Die eigentlichen Unterschiede sieht man bei der Verwendung verschiedener Raycasts besonders in Ausschnitt 1. So ist mit dem Standard Raycast der gesamte Ausschnitt 1 scharf zu sehen (siehe Abbildung 5.9), während bei dem *MDC* Raycast an den Konturen am Rand des Volumens zu sehen ist, dass diese leicht unschärfer sind (siehe Abbildung 5.11). Ausschnitt 3 unterscheidet sich hier zwischen dem Standard Raycast und dem *MDC* Raycast kaum.

Ein wesentlicher Unterschied ist sowohl in Ausschnitt 1 als auch in Ausschnitt 3 zwischen dem *DDC* Raycast und dem *MDC* sowie dem Standard Raycast wahrnehmbar. Man kann sehen, dass der äußere Bereich des *DDC* Raycast (siehe Abbildung 5.13), deutlich unschärfer ist, als der äußere Bereich des *MDC* Raycast (siehe Abbildung 5.11).

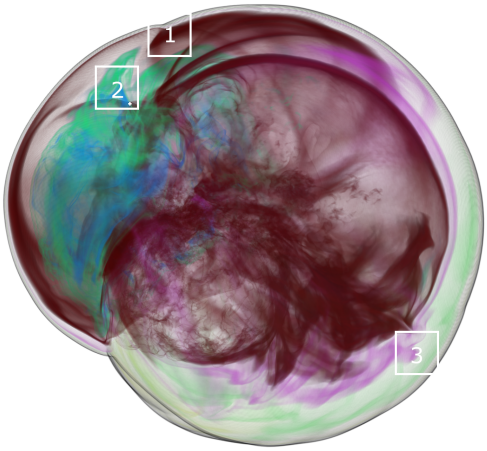

Abbildung 5.7: Ein Zeitschritt der Supernova mit dem Standard Raycast berechnet. Die Bildabtastrate beträgt für das ganze Bild 1. Die Strahlabtastrate beträgt für das ganze Bild 1,5. die Mausposition und drei nummerierte Quadrate sind eingezeichnet, die Positionen der Quadrate werden im folgenden Vergleich verwendet.

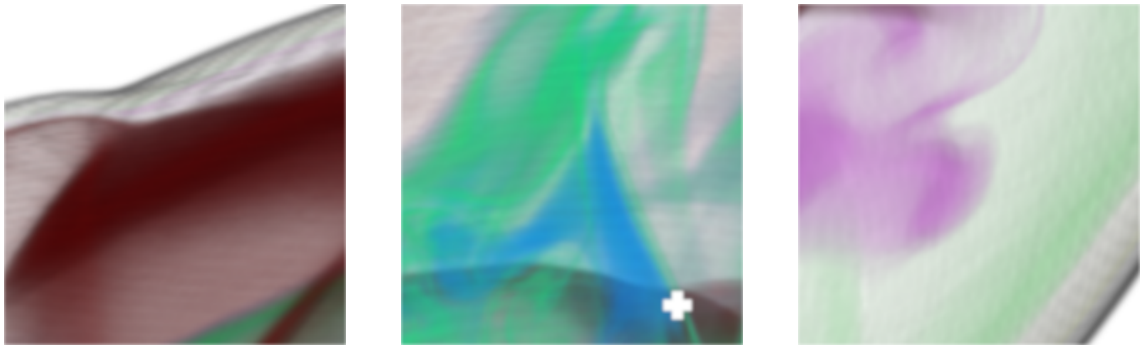

Abbildung 5.8: Supernova mit Standard Raycast und ohne variierter Strahlabtastrate berechnet.

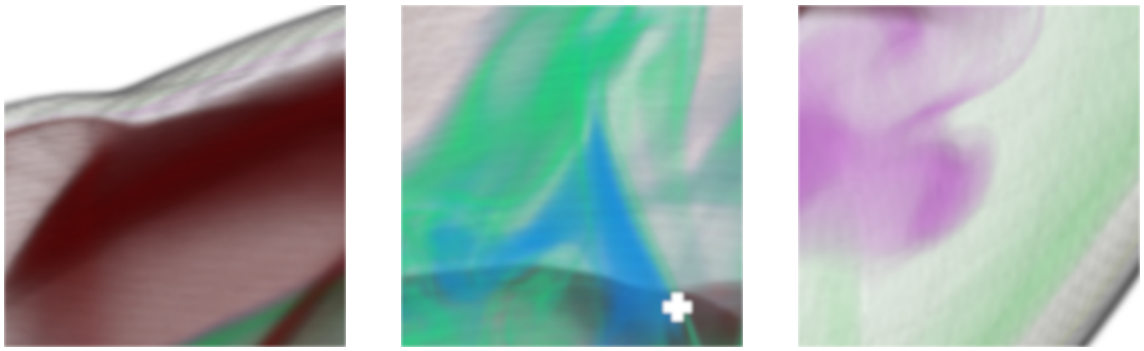

Abbildung 5.9: Supernova mit Standard Raycast und variierter Strahlabtastrate berechnet.

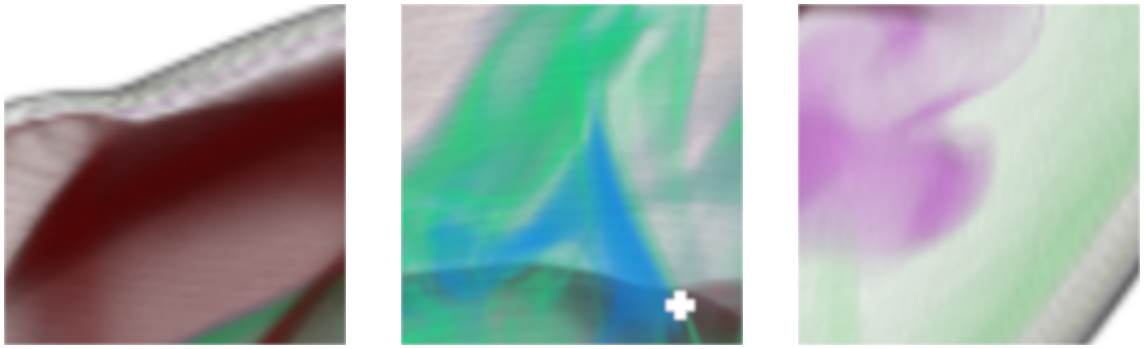

Abbildung 5.10: Supernova mit MDC Raycast und ohne variierter Strahlabtastrate berechnet.

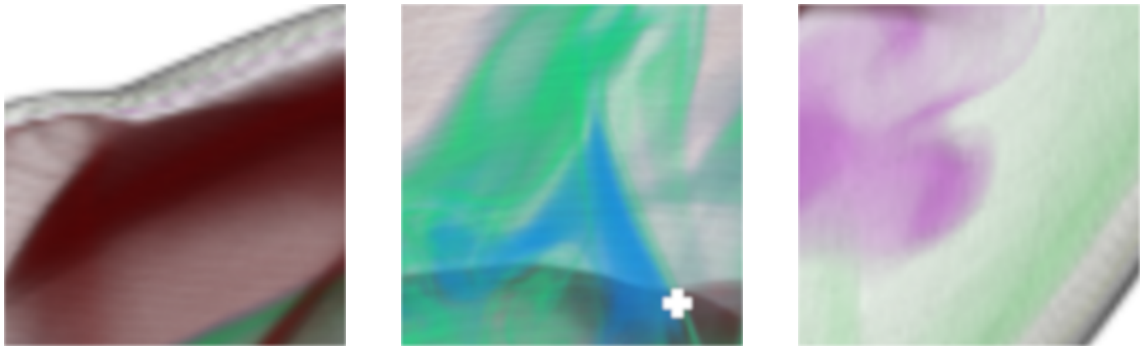

Abbildung 5.11: Supernova mit MDC Raycast und variierter Strahlabtastrate berechnet.

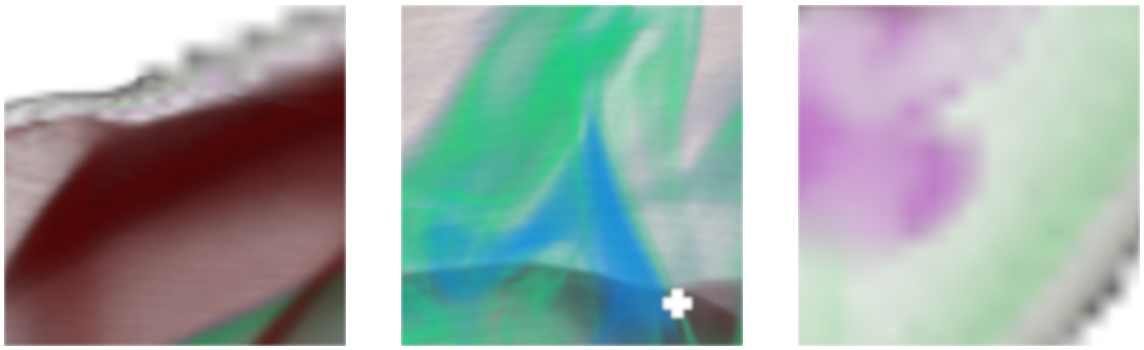

Abbildung 5.12: Supernova mit DDC Raycast und ohne variierter Strahlabtastrate berechnet.

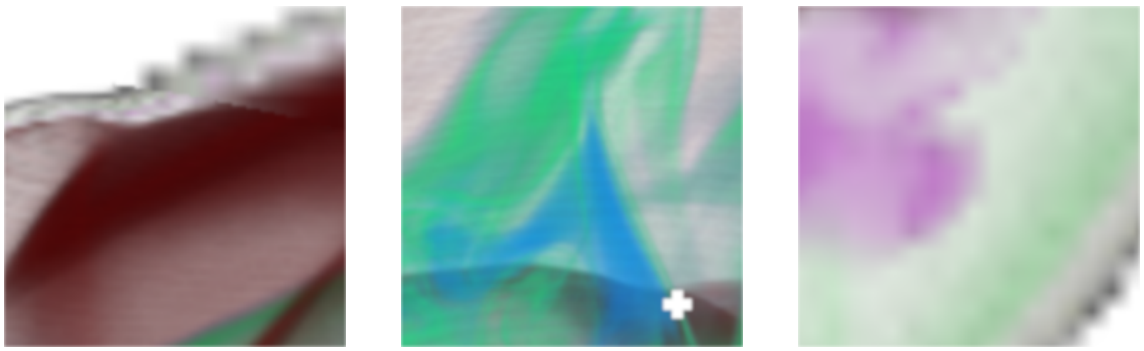

Abbildung 5.13: Supernova mit DDC Raycast und ohne variierter Strahlabtastrate berechnet.

5.1.2 Performanz

Für die Messung der Performanz der Implementierungen wurden zwei Messwerte bei einer Berechnung eines Bildes genommen. Die Ausführungszeit des Kernels innerhalb einer Ausführung der *paintGL()*-Methode und die Ausführungszeit der *paintGL()*-Methode selbst. Dies wurde deshalb so gewählt, da die Implementierungen nicht ausschließlich innerhalb des Kernels durchgeführt wurden, sondern auch außerhalb der Kernelaufrufe Programmcode geschrieben wurde und der Start einer Kernelausführung sowie die synchronisierte Beendigung eine gewisse Zeit brauchen. Die Ausführungszeit der *paintGL()*-Methode ist letztendlich der Wert, welcher die reale spürbare Performanz für den Nutzer angibt.

Für die Messungen der Performanz wurde kein Eyetracking verwendet, da dies lediglich die Mausposition mit der Blickposition ersetzt und keinen Einfluss auf die Ausführungszeit des Kernels und nur einen konstanten Einfluss auf die Ausführungszeit der *paintGL()*-Methode hat. Auch sollten, bei den Messungen mit den verschiedenen Methoden, möglichst die selben Mauspositionen verwendet werden, um vergleichbare Ergebnisse zu erhalten. Die Verwendung eines Eyetrackers würde es fast unmöglich machen, die selben Blickpunkte in einem gewissen Zeitraum zu fokussieren.

Für die Messergebnisse selbst wurde ein spiegelverkehrtes *S* mit dem Mauszeiger auf einem Bild geformt und dabei die Mauspositionen abgespeichert. Insgesamt wurden hier 584 Mauspositionen aufgezeichnet. Es wurden anschließend verschiedene Volumendaten mit jeweils drei verschiedenen Transferfunktionen für die drei verschiedenen Implementierungen, den Standard-, *MDC*- und *DDC*-Raycast sowie jeweils einmal mit konstanter und einmal mit variierter Strahlabtastrate für die Messungen verwendet. Bei jeder dieser Messung wurde dabei für jede der Mauspositionen des gespeicherten Mausverlaufs ein Bild berechnet und zu dieser Berechnung die Mausposition, die Ausführungszeit des Kernels und die Ausführungszeit der *paintGL()*-Methode gespeichert.

Die Hardware des Computers, mit welchem die Messungen durchgeführt wurden, besteht unter anderem aus einem *Intel Core i5 4670k bei* $3,40$ *GHz* CPU und einer *GIGABYTE AMD Radeon (TM) R9 390 Series* GPU. Die Auflösung der zu berechnenden Bilder betrug immer 2263×1306 Pixel.

Die Messungen wurden mit den Volumen *Bonsai*, *Supernova* und *Chameleon* durchgeführt. Das Volumen *Bonsai* hat eine Auflösung von $256 \times 256 \times 256$ Voxel, wobei die Voxel eine Slice-Dicke von $1,0 \times 1,0 \times 1,0$ haben. Das Volumen *Supernova* hat eine Auflösung von $432 \times 432 \times 432$ Voxel, wobei die Voxel eine Slice-Dicke von $1,0 \times 1,0 \times 1,0$ haben. Das Volumen *Chameleon* hat eine Auflösung von $1024 \times 1024 \times 1024$ Voxel, wobei die Voxel eine Slice-Dicke von $0,09228515625 \times 0,09228515625 \times 0,105$ haben.

Für einen anschaulichen Vergleich der Methoden wurde für die Messungen des Volumen Chameleon Heatmaps erstellt (Abbildung 5.14). Die gemessene Ausführungszeiten des Kernels für die unterschiedlichen Mauspositionen wurden für die Erstellung der Heatmaps verwendet. Da die Differenz zwischen Ausführungszeit des Kernels und der Ausführungszeit der *paintGL()*-Methode unabhängig von dem zu berechnenden Volumen und der verwendeten Transferfunktion ist, wurde diese hier nicht einbezogen.

Abbildung 5.14 zeigt, dass die Ausführungszeit des Kernels für den Standard Raycast ohne variierter Strahlabtastrate für alle Mauspositionen im Bereich von 80 ms liegt und damit im Vergleich zu den anderen Methoden am höchsten ist. Betrachtet man nun die Standard Variante mit variierter Strahlabtastrate, so kann man eine allgemeine Verbesserung feststellen. Die Ausführungszeit des Kernels liegt bei dieser Methode bei circa 65 ms und variiert je nach Mausposition leicht. So ist

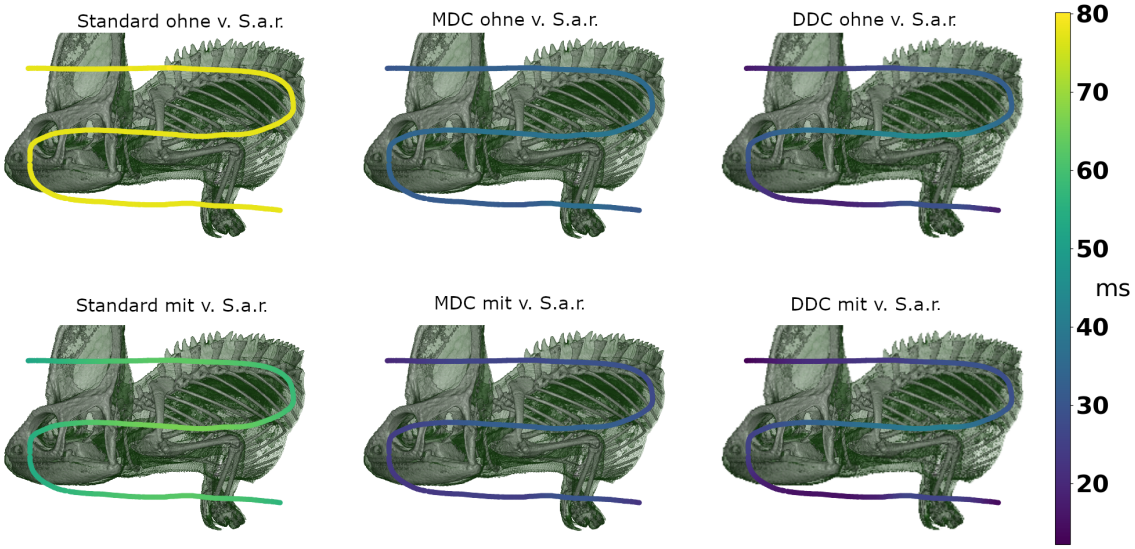

Abbildung 5.14: Heatmaps der verschiedenen Verfahren für das Volumen Chameleon. Für verschiedene Mauspositionen wurde eine Berechnung durchgeführt. Die jeweilige Ausführungszeit des Kernels für diese Berechnung wurde entsprechend des Farbbalkens farblich eingezeichnet. Der Farbbalken gibt die Werte in ms an. *v. S. a. r.* steht für *variierter Strahlabtastrate*

die Ausführungszeit für eine Mausposition am oberen und unterem Rande des Bildes ein wenig geringer, als wenn diese sich in der Mitte des Bildes befindet. Die Heatmap zur Methode mit dem MDC Raycast und ohne variierter Strahlabtastrate zeigt zu beiden Standard Raycast Methoden eine Verbesserung. Die Ausführungszeit des Kernels liegt hier bei circa 40 ms und ist im mittleren Bereich des Bildes minimal höher als im oberen oder unteren Bereich. Wird der MDC Raycast mit einer variierten Strahlabtastrate kombiniert, so ist die Ausführungszeit des Kernels im oberen und unteren Bereich des Bildes sichtbar schneller, als in der Variante ohne einer variierten Strahlabtastrate. Im mittleren Bereich des Bildes liegt sie nun bei circa 35 ms und in den äußeren Bereichen des Bildes liegt sie bei circa 20 ms. Die Heatmap zum DDC Raycast zeigt, dass mit dem DDC Raycast je nach Mausposition die niedrigsten Ausführungszeit des Kernels erreicht wurde. Auch ohne einer variierten Strahlabtastrate zeigt der DDC Raycast verhältnismäßig ein ähnliches Muster, wie der Standard und MDC Raycast mit variierter Strahlabtastrate. Die Ausführungszeit des Kernels betrug hier im äußeren Bild ebenfalls geringere Werte als im mittigen Bereich des Bildes. Im äußeren Bereich betrug sie circa 20 ms und im mittleren circa 45 ms. Es ist auffallend, dass die Differenz zwischen der Ausführungszeit des Kernels im oberen und unteren Bereich und der Ausführungszeit im mittleren Bereich des Bildes hier größer als bei den anderen Verfahren ist. In der Heatmap zum DDC Raycast mit variierter Strahlabtastrate ist zu erkennen, dass die äußeren Bereiche des Bildes, besonders oben links und unten rechts, nochmal niedrigere Ausführungszeiten des haben, als im DDC Raycast ohne variierter Strahlabtastrate. In den äußeren Bereichen fällt diese auf bis zu 15 ms und im mittleren Bereich auf leicht über 40 ms. Vergleicht man die Heatmaps des DDC und MDC Raycasts miteinander, so fällt auf, dass der DDC Raycast in den äußeren Bereichen des Bildes niedrigere Ausführungszeiten als der MDC Raycast erreicht, dafür die Ausführungszeiten des DDC Raycasts im mittleren Bereich des Bildes leicht höher ist, als die des MDC Raycasts.

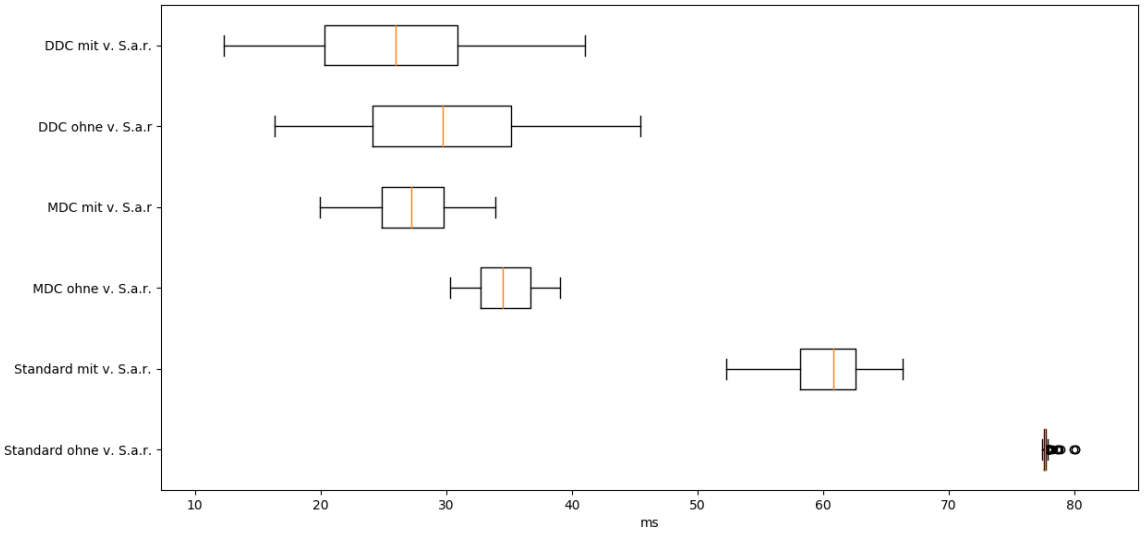

Abbildung 5.15: Boxplots der Ausführungszeiten des Kernels der verschiedenen Verfahren bei der Verwendung des Volumen Chameleon. Die Box reicht von den unteren Viertel bis zu den oberen Viertel der Werte und der Median ist orange eingezeichnet.

Das die Ausführungszeiten im äußeren Bereich des Bildes geringer sind wird zwei unterschiedliche Gründe haben. Erstens wird, wenn sich die Mausposition weiter weg von dem Volumen befindet, ein Großteil des Volumens beziehungsweise des Bildes mit einer geringeren Strahlabtastrate abgetastet, wodurch und einige Work-Items und Work-Groups früher terminieren können. Zweitens ist die Bildabtastrate in der Nähe der Mausposition am höchsten. Ist diese am Rande des Volumens, wie es in Abbildung 5.14 auch der Fall ist, so fällt ein großer Anteil der Strahlen auf Bereiche, in denen nur wenige Voxel abgetastet werden müssen und dadurch wieder einige Work-Items und Work-Groups früher terminieren. Das die MDC Methode ohne einer variierten Strahlabtastrate auch in den äußeren Bereichen eine relativ gleiche Ausführungszeit hat, wie in den inneren Bereichen des Volumens, wird daran liegen, dass der hoch aufgelöste innere Bereich des MDC Raycasts deutlich kompakter und ein wenig kleiner ist, als der des DDC Raycasts. Dadurch fallen auch weniger Strahlen des hoch aufgelösten Bereichs außerhalb des Volumens, auch wenn sich die Mausposition am Rande des Volumens befindet.

Die Ausführungszeiten des Kernels der unterschiedlichen Methoden, wie sie in den Heatmaps (Abbildung 5.14) farbig dargestellt wurden, sind in Abbildung 5.15 als Boxplots dargestellt. Wie an den Heatmaps schon ersichtlich war, sind die Ausführungszeiten des Standard Raycast ohne variierter Strahlabtastrate am höchsten und relativ gleich. Die Messwerte sind konzentriert um die 77,5 ms. Die Ausführungszeiten für den Standard Raycast mit variierter Strahlabtastrate sind ein wenig geringer und sind untereinander stärker verteilt. Für den MDC Raycast sieht es ähnlich aus. Ohne variierter Strahlabtastrate sind die Werte ein wenig höher und dafür kompakter. Mit einer variierten Strahlabtastrate sind sind die Werte niedriger aber weiter verteilt. Anders wie bei den vorherigen Raycast Methoden sind die Ausführungszeiten des DDC Raycast mit variierter Strahlabtastrate ein wenig kompakter, als ohne variierter Strahlabtastrate. Mit variierter Strahlabtastrate sind die Werte insgesamt aber trotzdem niedriger.

	Bonsai				Supernova				Chameleon			
	Ø K. in ms	Varianz	Ø Oh. in ms	Md. in s	Ø K. in ms	Varianz	Ø Oh. in ms	Md. in s	Ø K. in ms	Varianz	Ø Oh. in ms	Md. in s
St. o. v. S.a.r.	36,930	0,003	9,825	27,305	82,126	0,002	10,954	54,359	77,710	0,046	10,089	51,276
St. m. v. S.a.r.	29,848	2,637	9,695	23,093	63,458	19,373	10,779	43,354	60,340	11,290	10,366	41,291
MDC. o. v. S.a.r.	14,204	4,241	12,249	15,449	30,741	13,171	13,245	25,688	34,780	5,881	14,923	29,025
MDC. m. v. S.a.r.	11,699	5,474	11,888	13,775	24,903	21,605	12,685	21,951	27,550	11,045	12,528	23,408
DDC. o. v. S.a.r.	11,029	24,428	16,708	16,198	23,047	40,341	19,049	24,584	30,120	62,086	20,295	29,450
DDC. m. v. S.a.r.	9,851	22,366	16,279	15,260	20,407	38,809	18,132	22,507	26,310	58,257	19,816	26,935

Tabelle 5.1: Die Ergebnisse der verschiedenen Verfahren mit den Volumen Bonsai, Supernova und Chameleon. Für jedes Verfahren und Volumen ist die durchschnittliche Ausführungszeit des Kernel (K.) und die Varianz von dieser sowie der durchschnittliche Overhead (Oh.) und die summierte Ausführungszeit der *paintGL()*-Methode (Md.) für die verschiedenen Mauspositionen angegeben.

Wie schon in Abbildung 5.14 und 5.15 die Verhältnisse der Ausführungszeiten der verschiedenen Methoden dargestellt wurden, wird dies in Tabelle 5.1 durch die genauen Werte bestätigt. In Tabelle 5.1 sind für die unterschiedlichen Verfahren und die Volumen Bonsai, Supernova und Chameleon die durchschnittliche Kernelzeit, dessen Varianz sowie der durchschnittliche Overhead der *paintGL()*-Methode ohne die Ausführungszeit des Kernels dargestellt. Zusätzlich ist auch die summierte Ausführungszeit der *paintGL()*-Methode für die insgesamt 584 verschiedenen Berechnungen jeder Messung angegeben.

Betrachtet man den Overhead der verschiedenen Methoden, so sieht man, dass der Standard Raycast für die unterschiedlichen Volumen ähnlich ist und mit circa 10 s den geringsten Overhead hat. Die Implementierung des Standard Raycasts verwendet auch nur einen Kernel Aufruf. Der Overhead des MDC Raycasts ist ein wenig höher als der des Standard Raycasts und beträgt für das Bonsai Volumen circa 12 s, für das Supernova Volumen circa 13 s und für das Chameleon Volumen circa 14 s. Der Overhead des DDC Raycasts ist am höchsten und beträgt für den Bonsai circa 16 s, für die Supernova circa 19 s und für das Chameleon circa 20 s. Sowohl der MDC Raycast als auch der DDC Raycast verwenden jeweils zwei Kernel Aufrufe aber der DDC Raycast berechnet vor der Ausführung des Kernels noch einige Werte, die für diesen Raycast benötigt werden.

Vergleicht man die verschiedenen Volumen, so wird das Bonsai Volumen deutlich schneller als das Supernova oder Chameleon Volumen berechnet. Dies wird hauptsächlich an der geringeren Auflösung des Volumens und damit einer deutlich geringeren Anzahl an Voxeln liegen. Im Widerspruch dazu benötigt aber die Messung für das Supernova Volumens etwas länger, als die Messung für das Chameleon Volumen. Die Wahl der Transferfunktion in Verbindung mit dem in allen Raycasts vorimplementierten *Empty-Space-Skipping (ESS)* könnte hier eine Rolle spielen. Werden in einem Volumen die Voxel nicht vollständig transparent gemacht, sondern haben immer noch einen gewissen Opazitätswert, so werden diese nicht durch das ESS übersprungen, was zu einer längeren Berechnung führt. Ein weiterer Grund ist kann die Perspektive auf die Volumen sein. Das Chameleon Volumen wird von der Seite betrachtet und hat aufgrund der nicht einheitlichen Slice-Dicken eine geringere Tiefe, wenn man es von der Seite betrachtet. Daher terminieren die Strahlen von der Seite früher. Das Supernova Volumen hingegen ist relativ rund und hat eine einheitlich Slice-Dicke.

5.2 Diskussion

Betrachtet man rückblickend auf die Ergebnisse die unterschiedlichen Bildabtastraten, so ist es offensichtlich, dass der Standard Raycast von den unterschiedlichen Methoden her, die beste Bildqualität erzeugt. Trotzdem kann durch das Ausnutzen der Limitierungen des visuellen Wahrnehmungssystems des Menschen bei der Verwendung eines Eyetrackers annähernd der Effekt erzeugt werden, dass die Raycast Methoden MDC und DDC das Bild in voller Auflösung berechnen, obwohl dies nicht der Fall. Da die Auflösung beim MDC Raycast im äußeren Bereich im Vergleich zum DDC Raycast immer noch deutlich höher ist, ist dieser Effekt für den MDC Raycast viel realistischer und kaum wahrnehmbar während beim DDC Raycast noch Artefakte des am niedrigsten aufgelösten Bereichs wahrnehmbar sind. Um die Bildqualität des DDC Raycasts weiter zu verbessern, könnte die Bildabtastrate des äußersten Bereichs angehoben und oder der Radius des mittleren Bereichs vergrößert werden, so dass die reduzierte Auflösung noch weniger auffällt. Für den MDC Raycast hingegen wäre es denkbar, die Auflösung des äußeren Bereichs selbst weiter zu verringern oder einen zusätzlichen Bereich einzufügen, der nur noch ein Achtel der maximalen Auflösung hat.

Betrachtet man rückblickend auf die Ergebnisse die unterschiedlichen Varianten, mit variierter und ohne variierter Strahlabtastrate, so gibt es hier bei den Bildern kaum wahrnehmbare Unterschiede. Im Gegensatz zu den statischen Bildern wurde aber bei aktivem Eyetracking bei den Varianten mit variierter Strahabtastrate, Artefakte durch die veränderte Strahlabtastrate im Bild wahrgenommen, wenn sich die Augen über das Bild bewegt haben. Während einer Fixation verblassen diese Artefakte und sind wie bei einem statischen Bild kaum mehr wahrnehmbar. Die Ergebnisse scheinen anzudeuten, dass die Strahlabtastrate im äußeren Bereich noch weiter reduziert werden könnte, da diese bisher nur auf ein Minimum von ein Viertel der normalen Strahlabtastrate nach außen hin fallen kann. In weiteren Tests hat das Reduzieren des Limits auf ein Fünftel schon deutlich Artefakte der Unterabtastung ergeben, die dafür aber auch nur an den vom Blickpunkt entferntesten Stellen aufgetreten sind und daher nicht so aufgefallen sind. Weiter wäre es denkbar, die Funktion, die die Strahlabtastrate abhängig von der Distanz zur Mausposition anpasst, abzuändern, so dass diese schon bei einer geringeren Entfernung stärker abnimmt und früher das Limit erreicht. Da selbst bei nur einem Viertel der maximalen Strahlabtastrate im äußeren Bereich kaum Artefakte wahrnehmbar sind, sollte dies ohne eine wahrnehmbare Reduzierung der Bildqualität möglich sein.

Hinsichtlich der Ausführungszeit des Kernels zeigen die Ergebnisse deutlich, dass der Standard Raycast ohne variierter Strahlabtastrate am schlechtesten abschneidet. Der MDC Raycast ohne variierter Strahlabtastrate ist hier besser und hat aber eine höhere Varianz. Der DDC Raycast ebenfalls ohne variierter Strahlabtastrate erreicht deutlich geringere Kernelausführungszeiten als der MDC Raycast und hat einen niedrigeren Median und Durchschnitt. Dafür erreicht der DDC Raycast im Vergleich zum MDC Raycast aber auch deutlich höhere Kernelausführungszeiten und hat daher auch eine deutlich höhere Varianz. Wird zusätzlich zu den verschiedenen Methoden eine variierte Strahlabtastrate verwendet, so verringert sich bei allen Methoden die Kernelausführungszeiten und bis auf den DDC Raycast, bei dem sich die Varianz dadurch ein wenig verringert hat, hat sich beim Standard und MDC Raycast die Varianz dadurch erhöht.

Hinsichtlich der wahrgenommenen Performanz, also der Ausführungszeiten inklusive des Overheads, schneidet der MDC Raycast trotz einer höheren durchschnittlichen Kernelausführungszeit aufgrund des geringeren Overheads besser ab, als der DDC Raycast. Da zusätzlich die Varianz des MDC Raycasts geringer ist und dies daher auch bei der Verwendung eines Eyetrackers zu weniger Verzögerungen führt, als beim DDC Raycasts sowie der MDC Raycast bei Mauspositionen, die

sich innerhalb des Volumens befinden, sich besser verhält, ist aus Sicht der Performanz der MDC Raycast mit variierter Strahlabtastrate die beste Wahl. Da zusätzlich der MDC Raycast höherwertige Bilder als der DDC Raycast generiert, trotzdem der Verlust der Bildqualität bei der Verwendung eines Eyetrackers im Vergleich zum Standard Raycast nicht wahrnehmbar ist, schneidet der MDC Raycast mit variierter Strahlabtastrate in Verbindung mit einem Eyetracker von den verschiedenen Methoden her insgesamt am besten ab.

Dass der MDC Raycast von der Performanz her besser abschneidet als der DDC Raycast liegt an unterschiedlichen Faktoren. So ist wie genannt, der Overhead ein wichtiger Faktor, der die wahrnehmbare Performanz des DDC Raycasts im Vergleich zum MDC Raycast beeinträchtigt. Ein weiterer Faktor ist die Umsetzung des Index-Mappings. Aufgrund dessen, dass dieses versucht wurde variabel zu halten, wurden viele Berechnungen für das Abbilden der Indizes durchgeführt, bevor der eigentliche Raycast beginnt. Die Work-Items werden hier anhand ihrer globalen ID auf völlig unterschiedliche Bildkoordinaten abgebildet. Dies passiert unabhängig davon, in welcher Work-Group sie sich befinden, wodurch unter Umständen sehr unterschiedliche Bereiche durch eine Work-Group berechnet werden und die gesamte Work-Group immer auf das langsamste Work-Item warten muss. Eine Änderung der Implementierung dahingehend, dass die Work-Items nicht anhand der globalen ID sondern anhand der Work-Group ID mit der gesamten Work-Group auf einen zusammen liegenden Block von Bildkoordinaten abgebildet werden, würde die Ausführungszeit vieler Work-Groups verbessern. Ein anderer Faktor ist die Art und Weise, wie die NDRange des Kernels festgelegt wird. Eine Anpassung dieser, dass sie in mindestens eine Dimension ein Vielfaches der Work-Group Größe ist, würde die Anzahl zu startender Work-Groups reduzieren und die Ausführungszeit des Kernels weiter verbessern. Die hohe Varianz des DDC Raycasts liegt vermutlich also vor allem an der Art der Implementierung. Eine Optimierung dieser Implementierung würde sowohl die Ausführungszeit des Kernels, als auch den Overhead reduzieren, so dass der DDC Raycast insgesamt eine bessere Performanz als der MDC Raycast hat.

6 Fazit

Die Einarbeitung in das Projekt war aufwändig und es hat daher eine gewisse Zeit gebraucht, bis sich ein Überblick über die Struktur des Projektes gebildet werden konnte. Dies ist im Hinblick auf ein umfangreiches Projekt nicht verwunderlich, wurde aber dadurch erschwert, dass unter anderem die Einarbeitung in ungewohnte Programmiersprachen, Programmierumgebungen und Programmierbibliotheken erforderlich waren. Die Auseinandersetzung mit diesen hat aber viele neue Einblicke in bisher unbekannte Bereiche ermöglicht, die im Laufe dieser Arbeit sich als sehr hilfreich präsentiert haben. So haben sich vor allem Einblicke in die GPU Architektur als sehr interessant und hilfreich für die Implementierung der Raycasts, besonders des DDC Raycasts, erwiesen. Zum Beispiel wurde aufgrund dieser Einblicke im Verlauf der Arbeit die Implementierung des DDC Raycasts strukturell verändert, wodurch sich die Performanz weiter verbesserte. Trotzdem, wie in der Diskussion (Abschnitt 5.2) dargestellt, muss bei der Programmierung für die GPU bestimmte Aspekte beachtet werden. Die Auseinandersetzung mit den Grundlagen und Limitierungen des Sehapparates (Abschnitt 2.2) war für das Verständnis des Hauptziels dieser Arbeit und für die Umsetzung der Implementierungen wichtig und hat die Art und Weise, wie die Implementierungen entworfen wurden, beeinflusst. Allerdings wurden die Parameter, wie die unterschiedlichen Auflösungen und die unterschiedlichen Ausmaße der Bereiche in den verschiedenen Raycastmethoden, nach der eigenen Wahrnehmung eingestellt. Diese Parameter könnten für weitere Arbeiten genauer untersucht werden, um optimale Werte zu finden. Zum Beispiel könnte untersucht werden, wie es sich auswirkt, wenn die Größe der inneren Ellipse des DDC Raycasts auf die genaue Größe der Fovea auf dem Bildschirm angepasst und wie es wahrgenommen wird, wenn die innere Ellipse kleiner oder größer dargestellt wird.

Im Verlauf der Arbeit wurde klar, dass die Umsetzung des Volumenrenderings durch einen Raycast sehr gut möglich ist und auch viele Erweiterungen und Modifikationen des Volumenrenderings, wie das Anwenden einer Tranferfunktion und insbesondere wahrnehmungsorientierte Methoden, zum Beispiel das Reduzierung der Strahlabtastrate im peripheren Bereich, umgesetzt werden können. Wahrnehmungsorientiertes Rendering ist aber nicht auf das Volumenrendering und Raycasts beschränkt. Die Referenzliteratur in Abschnitt 2.1.1 zeigte diesbezüglich Arbeiten, die wahrnehmungsorientierte Methoden in Anwendungen anwenden, die eine Grafikpipeline zur Berechnung verwenden. Die vorgestellten Methoden MDC und DDC können prinzipiell auch in Anwendungen, die eine Grafikpipeline verwenden, umgesetzt werden.

Die Anfertigung von Arbeitspaketen hat die Strukturierung der Arbeit erleichtert und erwies sich auch für die Implementierungen der Arbeit als hilfreich, da diese unter anderem gewisse Vorgaben repräsentierten und eine ungefähre Einschätzung des Zeitaufwandes ermöglichten.

Die Arbeit hat gezeigt, dass wahrnehmungsorientierte Methoden durchaus einen Performanzgewinn erbringen können ohne die Bildqualität wahrnehmbar zu beeinträchtigen. Alleine die Verwendung einer varrierten Strahlabtastrate für den Standard Raycast hat selbst bei genauer Betrachtung eines statischen Bildes einer Berechnung kaum wahrnehmbare Veränderungen der Bildqualität nach sich

gezogen und trotzdem die Ausführungszeit spürbar verbessert. Die Anpassungen der Bildabtastrate wurde jeweils durch die Raycastmethoden MDC und DDC umgesetzt und hat jeweils einen noch größeren Performanzgewinn ermöglicht. Dabei hat sich die Implementierung des MDC Raycasts als vergleichsweise einfach erwiesen und war mit wenigen Änderungen des ursprünglichen Raycasts implementierbar. Erstaunlicherweise lieferte der MDC Raycast im späteren Vergleich mit dem DDC Raycast sowohl eine bessere Bildqualität, als auch eine bessere Performanz. Dieser ist aufgrund der schnellen Ausführung und geringen Varianz für die Verwendung eines Eyetrackers gut war. Trotzdem gibt es weitere Möglichkeiten, diesen zu verbessern. Zum Beispiel kann die Bildabtastrate im äußeren Bereich weiter verringert werden oder auch noch ein dritter Bereich eingefügt werden, solange der Overhead dadurch die Ausführungszeit der Berechnung eines Bildes nicht zu sehr beeinträchtigt. Der DDC Raycast war deutlich zeitaufwändiger zu implementieren. Die Indizes mussten vergleichsweise aufwändig auf Bildkoordinaten abgebildet werden und anschließend war es notwendig, diese zu interpolieren. Dafür erreichte der DDC Raycast bessere Spitzenwerte, ist aber von der Gesamtperformanz her aufgrund des Overheads und der deutlich größeren Varianz der Ausführungszeiten schlechter als der MDC Raycast. Auch die Bildqualität ist im äußeren Bereich deutlich geringer. Aufgrund der Limitationen des visuellen Wahrnehmungssystems, wird dies bei der Verwendung eines Eyetrackers nur geringfügig wahrgenommen. Auch der DDC Raycast ist wie im Diskussionsabschnitt (Abschnitt 5.2) schon erwähnt wurde, ausbaufähig und kann vermutlich durch eine bessere Implementierung deutlich effizienter werden. Im Vergleich zur Referenzliteratur, in welcher durch die Implementierung von wahrnehmungsorientierten Verfahren Peformanzgewinnungen um den Faktor 6 möglich waren, wurde hier lediglich ein Faktor um 2 erzielt. Dies liegt unter anderem daran, dass die Raycastmethoden, vor allem der DDC Raycast, in der Implementierung möglichst variabel gehalten wurde. Eine Festlegung auf bestimmte Parameter würde es ermöglichen, die Implementierung des DDC Raycasts anders und effizienter umzusetzen.

Bei der Arbeit mit dem Eyetracker ist aufgefallen, dass es durch Tremor-Bewegungen des Auges oder durch Messungenauigkeiten des Eyetrackers, zu vielen kleinen Differenzen in den gemessenen Blickpositionen kam. Da das Bild bei aktivem Eyetracking auf diese Differenzen reagiert und sich dadurch verändert hat, ziehen diese Änderungen teilweise die Aufmerksamkeit auf sich und wirken störend. Daher sollten die Eyetracking Daten vor der Verwendung gefiltert oder geglättet werden.

Die Umsetzung der Messmethoden in dem Projekt und besonders die Erstellung leicht verständlicher grafischer Repräsentationen hat viel Zeit benötigt. Schlussendlich konnten die Messmethoden aber so implementiert werden, dass diese reproduzierbar sind. Für die Messungen wurden die Eyetrackingdaten, die die Blickposition liefern, durch eine Mausbewegung simuliert und damit Messungen für die verschiedenen Raycastmethoden und für verschiedene Volumen erstellt. Die Darstellung durch Heatmaps veranschaulichte die Vor- und Nachteile der Methoden DDC und MDC indem die benötigte Ausführungszeiten für unterschiedliche Mauspositionen für das Volumen dargestellt wurden. Die Darstellung der Messwerte durch Boxplots ermöglichte einen noch genaueren und verständlicheren Vergleich.

Abschließend ist zu sagen, dass wahrnehmungsorientiertes Volumenrendering hohes Potential hat. Die Bildschirme und Pixeldichten werden immer größer, vor allem auch in VR-Anwendungen, da diese selten in Verbindung mit High-End Hardware berechnet werden und eine große und hoch aufgelöste Bildfläche bieten. Dadurch werden Berechnungen für Bilder dieser Auflösungen immer aufwendiger und Methoden, die die Performanz ohne Einbußen in der Bildqualität verbessern, mehr benötigt denn je.

Literaturverzeichnis

[] *CPU vs. GPU*. URL: http://www.keremcaliskan.com/wp-content/uploads/2011/01/CPU-GPU-Structures1.png (besucht am 08. 10. 2018) (zitiert auf S. 19).

[Dr 17] D. G. R. Dr. Michael Krone. *Vorlesungsfolien in Computergraphik, Raytracing*. Institut für Visualisierung und Interaktive Systeme der Universität Stuttgart, 2016 / 2017 (zitiert auf S. 15).

[ER] R. Englund, T. Ropinski. „Quantitative and Qualitative Analysis of the Perception of Semi-Transparent Structures in Direct Volume Rendering“. In: *Computer Graphics Forum* 37.6 (), S. 174–187. DOI: 10.1111/cgf.13320. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13320. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13320 (zitiert auf S. 11).

[GFD+12] *Foveated 3D Graphics*. ACM SIGGRAPH Asia, Nov. 2012. URL: https://www.microsoft.com/en-us/research/publication/foveated-3d-graphics/ (zitiert auf S. 10).

[Gro] K. O. W. Group. *The OpenCL Specification*. Hrsg. von L. Howes. URL: https://www.khronos.org/registry/OpenCL/specs/opencl-2.1.pdf (besucht am 08. 10. 2018) (zitiert auf S. 20, 21, 23).

[Lar16] J. Larkin. *GPU Fundamentals*. 14. Nov. 2016. URL: http://www.icl.utk.edu/~luszczek/teaching/courses/fall2016/cosc462/pdf/GPU_Fundamentals.pdf (besucht am 08. 10. 2018) (zitiert auf S. 22).

[LME06] A. Lu, R. Maciejewski, D. S. Ebert. „Volume Composition Using Eye Tracking Data“. In: *Proceedings of the Eighth Joint Eurographics / IEEE VGTC Conference on Visualization*. EUROVIS'06. Lisbon, Portugal: Eurographics Association, 2006, S. 115–122. ISBN: 3-905673-31-2. DOI: 10.2312/VisSym/EuroVis06/115-122. URL: http://dx.doi.org/10.2312/VisSym/EuroVis06/115-122 (zitiert auf S. 11).

[LW90] M. Levoy, R. Whitaker. „Gaze-directed Volume Rendering“. In: *Proceedings of the 1990 Symposium on Interactive 3D Graphics*. I3D '90. Snowbird, Utah, USA: ACM, 1990, S. 217–223. ISBN: 0-89791-351-5. DOI: 10.1145/91385.91449. URL: http://doi.acm.org/10.1145/91385.91449 (zitiert auf S. 9).

[NVI] NVIDIA. *NVIDIA's Next Generation CUDA Compute Architecture Fermi*. NVIDIA. URL: https://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf (besucht am 08. 10. 2018) (zitiert auf S. 22).

[Qt] Qt, Hrsg. *Qt Documentation*. URL: http://doc.qt.io/ (besucht am 10. 10. 2018) (zitiert auf S. 25).

[toba] tobii. *What is Eyetracking?* Tobii AB. URL: https://www.tobii.com/tech/technology/what-is-eye-tracking/ (besucht am 12. 09. 2018) (zitiert auf S. 16).

[tobb] tobiipro. *How do tobii eye trackers work*. Tobii AB. URL: https://www.tobiipro.com/learn-and-support/learn/eye-tracking-essentials/how-do-tobii-eye-trackers-work/ (besucht am 12. 09. 2018) (zitiert auf S. 17).

[tobc] tobiisdk. *Eyetracking Common Concepts, Tobii Pro SDK*. Tobii AB. URL: http://developer.tobiipro.com/commonconcepts.html (zitiert auf S. 17).

[WSR+] M. Weier, M. Stengel, T. Roth, P. Didyk, E. Eisemann, M. Eisemann, S. Grogorick, A. Hinkenjann, E. Kruijff, M. Magnor, K. Myszkowski, P. Slusallek. „Perception-driven Accelerated Rendering“. In: *Computer Graphics Forum* 36.2 (), S. 611–643. DOI: 10.1111/cgf.13150. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13150. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13150 (zitiert auf S. 12, 18).

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift