UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Webbasierte und GPU-unterstützte medizinische Visualisierung

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Computervisualistik

vorgelegt von

Jens Eraßmy

Erstgutachter:	Prof. Dr. Stefan Müller Institut für Computervisualistik, AG Computergrafik
Zweitgutachter:	Dipl.-Inf. Diana Röttger Institut für Computervisualistik, AG Computergrafik

Koblenz, im Dezember 2011

Erklärung

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

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

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

Inhaltsverzeichnis

1 Einleitung

Diese Arbeit beschreibt einen Ansatz zur webbasierten und GPU-unterstützten medizinischen Visualisierung. Der Schwerpunkt liegt auf der client-seitigen Ausführung von direktem Volumenrendering mittels WebGL und der Übertragung von medizinischen Datensätzen von Server zu Client. Im Folgenden wird die Motivation der Arbeit begründet. Nach einem Überblick über die Strukturierung folgt die Erklärung der Zielsetzung.

1.1 Motivation

Die Motivation dieser Arbeit liegt vor allem in den neuesten Entwicklungen von Webtechnologien begründet. Bisher war es nicht möglich hardwarebeschleunigte 3D-Grafiken direkt im Webbrowser darzustellen. Erst seit der Entwicklung der 3D-Grafik-Programmierschnittstelle WebGL und der Implementierung dieser in bekannte Webbrowser besteht die Möglichkeit, die Grafikkarte über den Browser direkt anzusprechen. Im Hinblick auf die dreidimensionale Visualisierung medizinischer Bilddaten besteht damit also die Grundlage, GPU-unterstütztes Volumenrendering im Browserkontext ohne den Einsatz zusätzlicher Software zu realisieren. Auf dieser Basis und mit Hilfe geeigneter Webtechnologien kann ein webbasiertes Volumenrendering-System entwickelt werden.

Eine solches System erfordert keine Installation zusätzlicher Software. Einzige Voraussetzung ist der Einsatz eines Webbrowsers mit WebGL-Unterstützung. Im Gegensatz zu Softwarelösungen kann das Visualisierungssystem als Webanwendung zentral gewartet und aktualisiert werden. Die permanente Erreichbarkeit der Webanwendung fördert zudem die kollaborative Zusammenarbeit an Visualisierungsprojekten. Mehrere Anwender können gemeinsam an einem Visualisierungsprozess über das Web teilnehmen.
Durch immer präzisere Bildakquisationsverfahren nimmt auch die Menge an zu verarbeitenden Daten stetig zu. Diese Problematik ist in besonderem Maße für ein webbasiertes Visualisierungssystem zu berücksichtigen, da die Bandbreitenbeschränkung des Internets den limitierenden Faktor darstellt. So besteht die Herausforderung vor allem darin, in Hinblick auf Interaktivität und die Größe der Datensätze ein effizientes System zu entwickeln.

1.2 Struktur der Arbeit

Die Arbeit ist wie folgt strukturiert: Kapitel 2 beschreibt die theoretischen Grundlagen. Diese umfassen Erläuterungen zur medizinischen Visualisierung im Allgemeinen und beschreiben die für diese Arbeit relevanten Vi-

sualisierungsmethoden. Des Weiteren wird hier ein Überblick über den technologischen Stand von Webapplikationen und 3D-Grafik im Internet gegeben. In Kapitel 3 werden verwandte Arbeiten vorgestellt und diskutiert. Kapitel 4 erläutert die technischen Grundlagen und die Umsetzung sowie die Implemtierung des Systems. Im Anschluss werden in Kapitel 5 die Ergebnisse dargestellt und in Kapitel 6 ausgewertet. Auf dieser Basis soll ein Ausblick auf mögliche Weiterentwicklungen gegeben werden. Abschließend wird in Kapitel 7 das Fazit der Arbeit dargestellt.

1.3 Zielsetzung

Im Rahmen dieser Arbeit soll ein webbasiertes Volumenrendering-System mit Schwerpunkt auf direktem Volumenrendering durch den Einsatz der GPU entwickelt werden. Dabei soll der Einsatz von WebGL für das direkte client-seitige Volumenrendering im Hinblick auf Möglichkeiten und etwaige Einschränkungen untersucht werden. Des weiteren ist eine Umsetzung erforderlich, die dem Benutzer, auch für große Datensätze, keine zu langen Wartezeiten bis zur Darstellung zumutet. Hierzu soll ein inkrementelles Konzept entwickelt werden. Mit Hilfe eines Transferfunktion-Editors soll eine grundlegende Explorationsmethode realisiert werden. Die Umsetzung wird abschließend auf Interaktivität und Effizienz in Bezug auf die Genauigkeit der Darstellung ausgewertet.

2 Grundlagen

Dieser Abschnitt beschreibt die Grundlagen der einzelnen Bestandteile möglicher webbasierter medizinischer Visualisierungssysteme. Diese umfassen zum einen Theorien und Methoden der medizinischen Visualisierung. Zum anderen werden die Anforderungen und Anwendungsgebiete eines webbasierten Visualisierungssystems erläutert. Darüber hinaus wird zusammenfassend auf den Bereich der medizinischen Visualisierung im Webkontext eingegangen.

2.1 Medizinische Visualisierung

Die medizinische Visualisierung ist seit Ende der achtziger Jahre ein spezielles Forschungsgebiet aus der wissenschaftlichen Visualisierung. Sie hat ihre Wurzeln zum einen in der wissenschaftlichen Illustration, wie beispielsweise in den anatomischen Zeichnungen von da Vinci (siehe Abbildung 1). Zum anderen ist ihr Einfluss in der Computergrafik, im speziellen in der dreidimensionalen Geometrie und ihren Algorithmen begründet. Ein weiteres Einflussgebiet stellt die Bildverarbeitung dar. Sie ist essentiell im Anwendungsfeld der mathematischen Analyse und Vorverarbeitung

von Bilddaten.
Das ungerichtete Explorieren der Daten, das Testen von Annahmen in

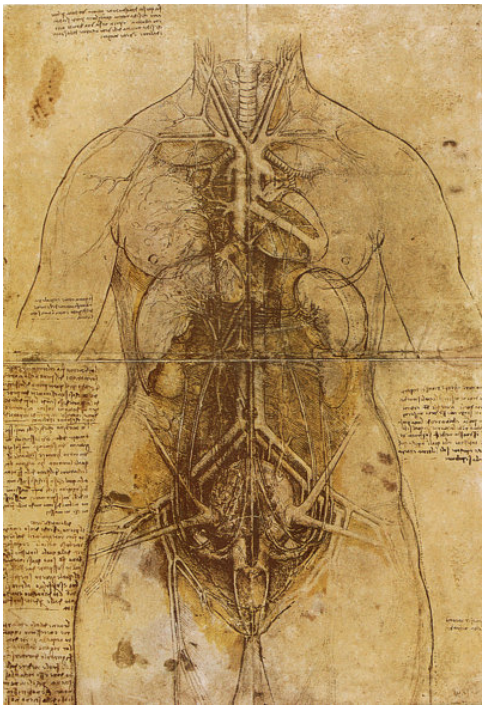

Abbildung 1: Anatomische Zeichnung von Leonardo da Vinci

Bezug auf Messungen und die Präsentation der Ergebnisse sind grundlegende Ziele und Anwendungsfelder der wissenschaftlichen Visualisierung. Viele Anwendungsfelder der medizinischen Visualisierung greifen auf diese allgemeinen Ziele zurück:

Bildungszwecke Mit Hilfe von Visualisierungstechniken entstehen Systeme zum Zweck der Anatomie- und Operationslehre. Dreidimensionale Visualisierungen können für Lernzwecke beispielsweise mit zusätzlichen Informationen, wie Bezeichungen für selektierte Strukturen, erweitert werden. Für das operative Training werden im Gegensatz zu anatomischen Lehranwendungen deformierbare Modelle eingesetzt.
Für den Bildungseinsatz werden oftmals nicht-klinische Daten verwendet. Durch eine mögliche höhere Strahlendosis als beim lebenden Organismus können höhere Auflösungen und ein gutes Signal-Rausch-Verhältnis erreicht werden. Die Visualisierung erziehlt dadurch eine sehr genaue Darstellung der Anatomie.

Diagnose Dreidimensionale, interaktive Visualisierungen unterstützen zweidimensionale medizinische Darstellungen durch einen direkten räumlichen

Überblick über die Morphologie.

Behandlungsplanung Interaktive 3D-Visualisierungen verbessern die Planung der Behandlungsmethoden. Räumliche Beziehung zwischen lebenskritischen Strukturen und pathologischen Verletzungen können in dreidimensionalen Darstellungen besser ausgewertet werden.

Intra-operative Unterstützung Bilddaten, die während einer Operation entstehen, werden in eine vor der Operation erzeugte Visualisierung integriert. Die Visualisierung medizinischer Bilddaten ist ein Prozess von der Datenerfassung durch Simulationen oder Messungen über Berechnungen bis hin zur interaktiven Exploration einer dreidimensionalen Darstellung. Im Folgenden werden die Bestandteile dieses Prozesses im erläutert.

2.1.1 Bildakquisition

Bei der Bildakquisition oder Bilderfassung handelt es sich in der Regel um Verfahren aus der Bildgebung. Eingesetzte Techniken sind unter anderem die Computertomographie (CT), die Magnetresonanztomographie (MRT) und die Positronen-Emissions-Tomographie (PET). Durch Anwendung dieser Methoden entstehen aufeinanderfolgende zweidimensionale Schnittbilder einer Körperregion. Dieses Ergebnis kann als Volumendatensatz interpretiert werden.

2.1.2 Medizinische Bilddaten

Im Kontext der medizinischen Visualisierung werden die medizinischen Bilddaten in der Regel als Volumedaten aufgefasst. Die Struktur medizinischer Volumendaten basiert auf einem diskreten uniformen Raster. Der Volumendatensatz ist ein Stapel von individuellen Schnittbildern. Jedes dieser Schnittbilder spiegelt eine dünne Schicht des Körpers wieder. Die Pixel eines Bildes sind als zweidimensionales Raster angeordnet, wobei der Abstand der Pixel konstant bleibt. Die Volumendaten entstehen also durch Kombination einzelner Bilder zu einer 3D Repräsentation. Grundlage dieser Repräsentation ist ein dreidimensionales Raster (siehe Abbildung 2). Die Datenelemente eines Volumens werden als Voxel (Volumenelement) bezeichnet. Die zusätzliche Dimension z des Volumens wird auch als Tiefe verstanden.
Die Bezeichung für den Abstand eines Voxels in jede Richtung ist das sogenannte *Voxel Spacing*. In der Medizin liegen in der Regel Datensätze mit einem anisotropen Raster vor. Im Gegensatz zu isotrop strukturierten Daten ist hier der Abstand der Schnittbilder ungleich dem Abstand der Pixel pro Bild. Im Allgemeinen liegen Datensätze mit einem größeren Abstand in z- als in x- und y-Richtung vor. Oft ist es erforderlich Werte innerhalb

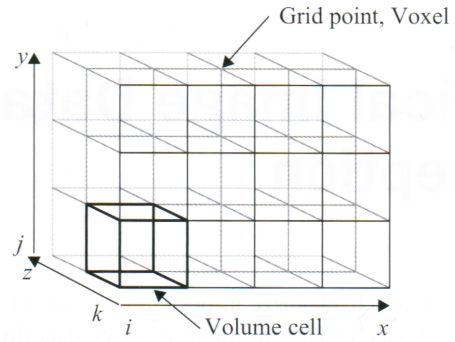

Abbildung 2: Datenstruktur eines Volumendatensatzes. Die Voxel sind auf einem dreidimensionalen Raster definiert. Acht Voxel können zu einer Voxel-Zelle zusammengefasst werden.

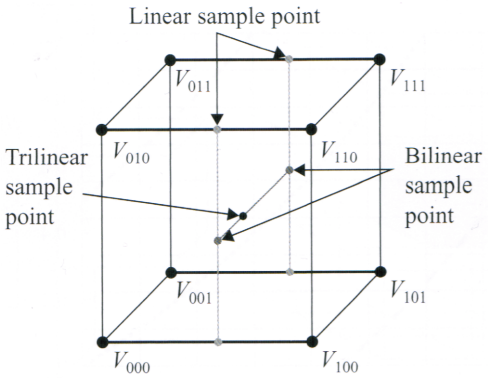

Abbildung 3: Die beiden gegenüberliegenden Seiten der Voxelzelle werden bilinear interpoliert. Zusätzlich dazu wird zwischen den beiden bilinear interpolierten Punkten eine weitere Interpolation durchgeführt. Man erhält den trilinear interpolierten Wert an einer bestimmten Position innerhalb der Voxelzelle.

einer Volumenzelle zu berechnen. Zu diesem Zweck wird unter anderem die trilineare Interpolation eingesetzt (siehe Abbildung 3).

DICOM Medizinische Bilddaten werden zusammen mit weiteren Informationen gespeichert, die für die Identifizierung notwendig sind. Diese Informationen sind im DICOM-Standard (Digital Imaging and Communications in Medicine) definiert. Der Standard ist weit verbreitet und findet in vielen medizinischen Applikationen Anwendung. Weiterhin ist DICOM ein Industrie-Standard für den Austausch radiologischer Bilder und weiterer Informationen zwischen Computern und medizinischen Geräten.

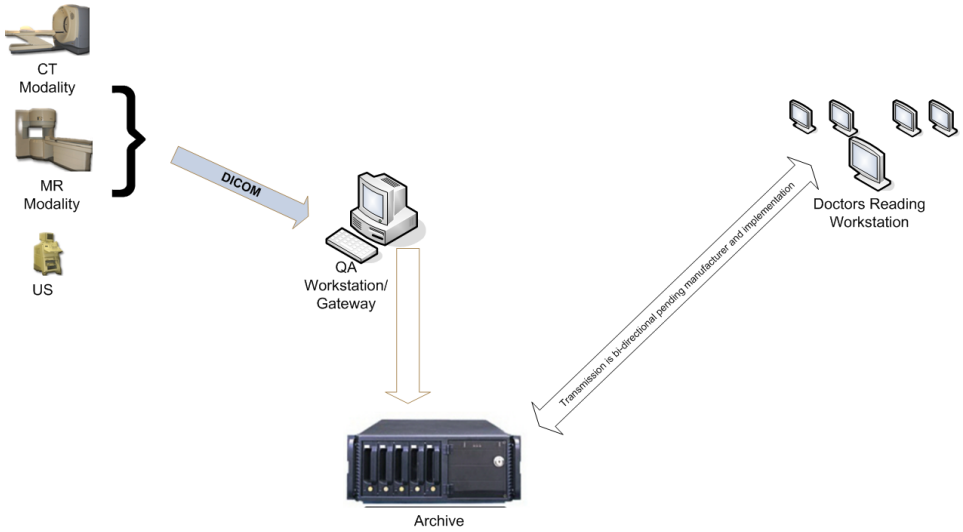

Abbildung 4: Picture Archiving and Communication System

Die Bilddaten aus Messungen wie der CT oder MRT ergeben eine Serie von DICOM-Dateien. Jede von ihnen repräsentiert ein Schnittbild. Für die Zusammensetzung des Volumendatensatzes müssen die Dateien zunächst identifiziert werden. Dies ist über identische Akquisationsparameter, wie eindeutige Kennungen, auszumachen.

PACS PACS (Picture Archiving and Communication System) ist eine Technologie für die Speicherung und den Zugriff medizinischer Bilddaten. Die PACS Architektur basiert auf vier Hauptkomponenten (siehe Abbildung 4). Zuerst werden über ein bildgebendes Verfahren die medizinischen Daten erfasst. Diese werden dann einer Qualitätskontrolle unterzogen und bei korrekter Untersuchung an ein Archiv für die Datensicherung weitergeleitet. Über ein gesichertes Netzwerk erhält nun eine Workstation Zugriff auf die Daten. PACS setzt für die Ablage und den Transfer der Daten das DICOM-Format ein.

2.1.3 Analyse und Vorverarbeiung medizinischer Bilddaten

Die medizinische Bild-Analyse umfasst computerunterstützte Methoden zur Extrahierung notwendiger klinischer Informationen von Datensätzen. Im Vorfeld gehören hierzu auch Vorverarbeiungsschritte und die Filterung der Bilddaten. Eine Verbesserung der Bildqualität ist erforderlich, da es aufgrund technischer Gegebenheiten bei der Bilderfassung zu Rauschen und Artefakten kommen kann. Des Weiteren kann der relevante Bildteil mittels einer Region of Interest definiert werden, um unnötige Berechnungen außerhalb des Interessenbereichs zu vermeiden.

Kern der Analyse ist die Segementierung. Sie dient der Identifizierung anatomischer oder pathologischer Strukturen. Die Segmentierung ist ein sehr zeitaufwendiger Prozess, der in einer geometrischen Beschreibung einer Struktur resultiert. In der Regel sind hier noch Nachverarbeitungsschritte notwendig, um eine glatte und fehlerfreie Oberfläche zu erhalten. Mit Hilfe der Segmentierung können extrahierte Gewebesegmente klassifiziert werden.

2.1.4 Volumenvisualisierung

Die Volumenvisualisierung beschäftigt sich mit der Erzeugung einer visuellen Repräsentation der Volumendaten. Die Kernfrage hierbei lautet: Wie können alle Informationen des Volumens in einer Darstellung repräsentiert werden? Dazu muss prinzipiell jedes einzelne Voxel des Datensatzes in den Prozess einer zweidimensionalen Projektion einfließen.

Für die Darstellung existieren zwei grundlegend verschiedene Ansätze: indirektes und direktes Volumenrendering. Die Idee des **indirekten Volumenrenderings** ist, von einem Teil der Volumendaten erst eine geometrische Repräsentation zu erzeugen und diese dann auf die Bildebene zu projizieren.
Da verschiedene Intensitätswerte unterschiedliche Strukturen identifizieren, kann dieser Unterschied als Abgrenzung zu anderer angrenzender Materie interpretiert werden. Ziel des sogenannten oberflächenbasierten Volumenrenderings ist es, die Voxel mit gleichem Intensitätswert miteinander zu verknüpfen und somit Geometrie zu erstellen. Im Visualisierungsprozess wird dazu ein Iso-Wert festgelegt, der die zur Geometrie beitragenden Voxel definiert. Mit Hilfe des *Marching Cubes* Algorithmus [LC87] wird dann eine polygonale Oberfläche der Struktur erzeugt. Die Ergebnis-Geometrie wird auch Iso-Oberfläche genannt.

Das **direkte Volumenrendering** verzichtet hingegen auf den Zwischenschritt der Geometrieerzeugung. Der Grundgedanke ist die unmittelbare Darstellung der Bilddaten. Das direkte Volumenrendering wird in Kapitel 2.2 ausführlich behandelt, da es einen Schwerpunkt dieser Arbeit darstellt.

2.1.5 Exploration der Visualisierung

Die Exploration eines Datensatzes findet im Rahmen der interaktiven Möglichkeiten für den Benutzer statt. Zu diesem Zweck sollte ein medizinisches Visualisierungssystem unterstützende Hilfestellungen bei der Navigation und Selektion geben. Außerdem stellt der Benutzer die Anforderung, Datensätze mit ähnlichen Gegebenheiten zum visuellen Vergleich heranzuziehen und darzustellen. Für die optische Darstellung müssen Parameter

individuell justiert werden können. Letzendlich führt die Exploration zu einer Interpretation und Klassifizierung der medizinischen Bilddaten.

2.2 Volumenrendering

Die folgenden Erklärungen basieren auf den Ausführungen von M. Hadwiger u.a. (vgl. [HKRs+06] S.1ff). Volumenrendering umfasst eine Vielzahl an Techniken zur Bildgenerierung eines dreidimensionalen skalaren Datensatzes. Beispiele hierfür sind unter anderem medizinische Datensätze aus der Computertomographie (CT) oder der Magnet-Resonanz-Tomographie (MRT). Weitere Datensätze stammen aus der numerischen Strömungsmechanik (englisch: computational fluid dynamics, CFD), der Geologie und Seismologie oder beschreiben abstrakte mathematische Daten und sonstige dreidimensionale skalare Abbildungen.
Die traditionelle Computergrafik beschäftigt sich in der Regel mit der Erzeugung von Oberflächen. Visuelle Eigenschaften wie Farbe, Material und Lichtreflektierung werden über Algorithmen definiert. Während der Lichttransport hierbei nur an Punkten auf der Oberfläche berechnet wird, ist es bei der grafischen Darstellung von Volumen maßgeblich, dass die Lichtinteraktion im Inneren nachvollzogen werden kann.
Das Kernproblem des Volumenrenderings ist also die Interaktion zwischen Licht und Medium im Inneren eines Volumens. Die Interaktion bezieht sich auf die Absorption, die Streuung und Emittierung des Lichts in Abhängigkeit zum jeweiligen auftretenden Medium. Beim photorealistischem Volumenrendering liegt der Darstellung von realistischen gasförmigen Materialien eine physikalische Simulation zu Grunde, die unter anderem auf einer physikalisch korrekten Beschreibung des an der Lichtinteraktion teilnehmenden Stoffes basiert. Im Gegensatz dazu kommt es bei der Visualisierung von dreidimensionalen Datensätzen darauf an, visuelle Informationen aus einem dreidimensionalen Skalarfeld zu extrahieren. Das Medium bleibt also bei der Berechnung konstant. Folgende Abbildung beschreibt die Funktion vom 3D Raum zu einem skalaren Wert:

$$\phi : R^3 \rightarrow R \tag{1}$$

Dieses 3D-Skalarfeld wird aus diskreten Messungen oder Simulationen gewonnen und stellt somit ebenfalls ein diskretes Raster dar. Die Aufgabe der direkten Volumevisualisierung besteht also darin, dieses skalare Feld auf physikalische Eigenschaften zu übertragen, welche die Lichtinteraktion am jeweiligen Raumpunkt beschreiben. Es wird also der Weg des Lichtes durch das teilnehmende Medium betrachtet. Im Allgemeinen spielen für das Volumenrendering folgende Interaktionen eine Rolle.

- Emission: Gasförmiges Material emittiert Licht und erhöht somit die Strahlungsenergie.

- Absorption: Material absorbiert Licht indem es Strahlungsenergie in Wärme umwandelt. Die Lichtenergie wird also reduziert.
- Streuung: Licht kann durch das beteiligte Material gestreut werden und somit ändert sich auch die Richtung der Lichtausbreitung.

Die exakte Übertragung des physikalisch korrekten Modells ist bei der direkten Volumen Visualisierung nicht unbedingt notwendig. Das sehr komplexe Modell wird hier vereinfacht, indem der Einfluss durch unterschiedliche Wellenlängen, Diffraktion und Lichtstreuung vernachlässigt wird. Außerdem spielen Refraktion und Reflektion beim Volumenrendering zunächst keine Rolle. Letztendlich bleiben die Lichtemission und -absorption als maßgebende Parameter erhalten und bilden die Grundlage des physikalischen Standardmodells für Volumenrendering, das Density-Emitter-Modell. Dieses Modell beschreibt jeden Partikel des Volumens als Lichtquelle (Emission). Während man einem Strahl durch das Volumen folgt, verliert die jeweilige Lichtquelle durch Absorption an Leuchtkraft. Ziel ist es letzendlich die Strahlungsenergie zu bestimmen, die beim Betrachter ankommt. Mit Hilfe der Volumenrendering Gleichung in ihrer Differentialform kann die Strahlungsengergie entlang des Strahls berechnet werden.

Emission Betrachtet man ausschließlich das physikalische Modell der Lichtemission, so wird das Licht weder abgeschwächt noch gestreut. Zur Veranschaulichung kann man sich eine glühende transparente Gaswolke vorstellen. Jeder Partikel dieser Wolke kann als Lichtquelle mit Hilfe der Funktion

$$\frac{dI}{ds} = Q(s) \tag{2}$$

interpretiert werden, wenn man von einer konstanten Wellenlänge ausgeht. Hierbei spezifiert s die Strahlungsrichtung. Die Lichtintensität I(s) an einem Punkt s entlang eines Strahls S kann durch Integration der Differentialgleichung (2)

$$I(s) = I_{s_0} + \int_{s_0}^{s} Q(t)dt \tag{3}$$

bestimmt werden. Hierbei bestimmt s_0 den Eintrittspunkt in das Volumen und I_{s_0} die initiale Lichtintensität. Diese kann auch als Hintergrundbeleuchtung interpretiert werden.

Absorption Das physikalische Modell der Lichtabsorption steht unter der Annahme, dass die Partikel des Volumens das Licht ausschließlich absorbieren und somit die Lichtintensität kontinuierlich schwächen. Die Absorption an einer Position s entlang eines Strahls S wird durch folgende Differentialgleichung definiert:

$$\frac{dI}{ds} = -\tau(s) \cdot I(s) \tag{4}$$

$\tau(s)$ beschreibt das Absorptionsmaß, welches die Intensität des Lichts entlang des Strahls innerhalb des Volumens herabsetzt. Dieses Maß entspricht ebenfalls der Transparenz des Wertes an der Stelle s. Dieser Transparenzwert wird durch eine Transferfunktion festgelegt. Durch Integration der Differentialgleichung 4 erhält man das folgenden Integral:

$$I(s) = I_{s_0} \cdot e^{(-\int_{s_0}^{s} \tau(t)dt)} \tag{5}$$

Der zweite Faktor $e^{(-\int_{s_0}^{s} \tau(t)dt)}$ beschreibt die Abschwächung der initialen Lichtintensität I_{s_0} an einem Punkt s entlang des Strahls S.

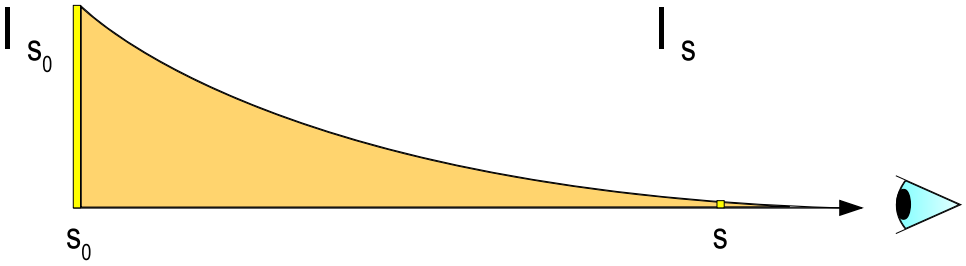

Abbildung 5: Modell der Absorption

Volumenrendering-Gleichung Durch die Kombination der Terme aus Emission (2) und Absorption (4)

$$\frac{dI}{ds} = Q(s) - \tau(s) \cdot I(s) \tag{6}$$

entsteht wiederum durch Integration das Volumenrendering-Integral:

$$I(s) = I_{s_0} \cdot e^{(-\int_{s_0}^{s} \tau(t)dt)} + \int_{s_0}^{s} Q(p) \cdot e^{(-\int_{p}^{s} \tau(t)dt)} dp \tag{7}$$

I(s) ist also die Strahldichte, die durch Emission und Absorbtion innerhalb des Volumens beim Betrachter ankommt. Der erste Term definiert hier das Hintergrundlicht, das in das Volumen eindringt. Der zweite Term hingegen beschreibt das Integral, welches die Lichtquellen entlang des Strahls und dessen Abschwächung durch das teilnehmende Medium beinhaltet. Zur Veranschaulichung siehe Abbildung 6.
Das Volumenrendering-Integral erfordert zur Berechnung eine numerische Lösung mit Hilfe der Riemann Summe und einer konstanten Schrittweite Δs entlang des Strahls. Durch diese Diskretisierung erhält man:

$$I(s) = I_0 \prod_{k=0}^{n-1} t_k + \sum_{k=0}^{n-1} Q(k \cdot \Delta s) \cdot \Delta s \prod_{j=k+1}^{n-1} t_j \tag{8}$$

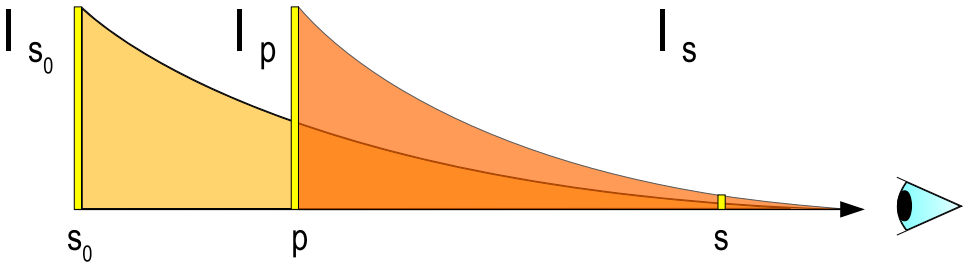

Abbildung 6: Modell des Volumenrendering-Integrals. Die Volumenrendering-Gleichung aus (6) kann durch Integration entlang der Strahlrichtung vom Startpunkt s_0 bis zum Endpunkt s gelöst werden.

Dabei beschreibt $\prod_{k=0}^{n-1} t_k$ die vereinfachte und diskretisierte Form der Abschwächung (siehe Gleichung 5) mit $n-1$ als letzter Abtastung und t_k als Wert der Transparenz an der diskreten Position k. Die diskrete Volumenrendering Gleichung beschreibt, wie ein Strahl s mit dem initialen Wert aus $I_0 \prod_{k=0}^{n-1} t_k$ ein Volumen traversiert und dabei an den diskreten Stellen k die entsprechenden Werte akkumuliert. Die jeweiligen Werte stammen aus der lokalen Quelle $Q(k \cdot \Delta s)$, abgeschwächt durch die Transparenz, die bis zur Stelle t_j akkumuliert wurde.
Die Volumerendering-Pipeline beschreibt anschließend die Auswertung der Volumen Rendering Gleichung.

2.2.1 Volumenrendering-Pipeline

Die Volumenrendering-Pipeline beschreibt den Prozess der zweidimensionalen Bilddarstellung von Volumendaten unter Anwendung der Volumenrendering-Gleichung. Einen Überblick über die individuellen Operationen stellt Abbildung 7 dar und soll im Folgenden erläutert werden.

Sampling Das Sampling beschreibt die Abtastungen entlang eines Strahls durch das Volumen an bestimmten Positionen. Die Werte an diesen Stellen werden akkumuliert. Die Auswahl dieser Positionen ist nicht willkürlich. Aufgrund des *Sampling Theorems* sollte eine Samplingrate gewählt werden, die mindestens zweimal höher ist als die Auflösung des Datensatzes. So können größere Artefakte vermieden werden. Die Sampling Positionen sind die Basis für die Evaluation des Volumenrendering-Integrals.

Filterung Die Positionen der Abtastungen unterscheiden sich in der Regel von dem zugrundeliegenden Raster des Volumens. Aus diesem Grund ist eine Interpolation des Datensatzes notwendig. Für ein uniformes Raster findet in der Regel eine trilineare Interpolation statt.

Klassifikation Die Klassifikation überträgt lokale Eigenschaften des Datensatzes auf optische Werte (Farbe und Transparenz). So hilft die Klassifizierung unterschiedliche Charakteristika eines Volumens zu visualisieren und somit zu identifizieren. In der Regel werden diese optischen Eigenschaften mit Hilfe einer Transferfunktion konfiguriert.

Shading Zur besseren visuellen Erscheinung der Darstellung kann zusätzlich ein Beleuchtungsmodell in den Emissions-Teil der Volumenrendering-Gleichung integriert werden.

Compositing Beim Compositing werden die abgetasteten und zuvor berechneten Werte entlang des Strahls akkumuliert und zum Endbild zusammengefügt. Das Compositing stellt die Grundlage für die iterative Berechnung des diskretisierten Volumenrendering-Integrals dar. Die Idee ist das Volumenrendering-Integral in einfachere Operationen umzuformen.
Das Compositing kann auf zwei Arten vollzogen werden:

- Der Strahl wird vom Augpunkt in das Volumen hinein verfolgt (*front-to-back*).
- Der Strahl wird von der Rückseite des Volumens aus traversiert (*back-to-front*).

Die Berechnung des Compositings hängt also von der Traversierungsrichtung ab. An dieser Stelle soll exemplarisch das Front-to-Back-Compositing aufgezeigt werden:

$$\begin{aligned} C_{dst} &\leftarrow C_{dst} + (1 - \alpha_{dst}) * C_{src} \\ \alpha_{dst} &\leftarrow \alpha_{dst}(1 - \alpha_{dst}) * \alpha_{src} \end{aligned} \tag{9}$$

Die mit $_{dst}$ markierten Parameter beschreiben den akkumulierten Farb- beziehungsweise Transparenzwert. Auf den Farbwert wird die aktuell ausgelesene Farbe C_{src} mit der Gewichtung der umgekehrten akkumulierten Transparenz $(1 - \alpha_{dst})$ addiert. Für die Transparenz erfolgt der Schritt entsprechend. Diese Zuweisungen werden bei der Traversierung des Strahls mehrfach ausgeführt.

2.2.2 Methoden

Nach dem Prinzip des direkten Volumenrenderings wurden verschiedene Methoden entwickelt. Techniken die in dieser Arbeit erwähnt oder umgesetzt werden, sollen in diesem Abschnitt kurz erläutert werden.

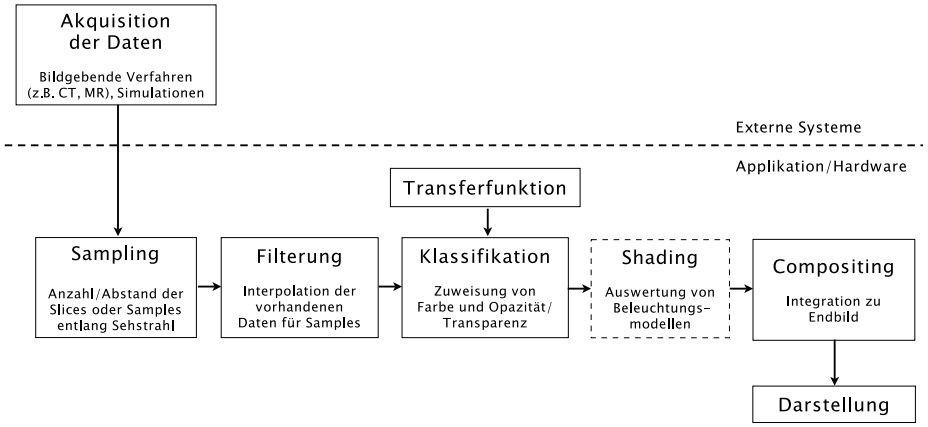

Abbildung 7: Volumenrendering-Pipeline

Texturbasiertes Volumenrendering Das texturbasierte Volumenrendering ist ein sogenanntes *objekt-basiertes* Verfahren. Bei dieser Methode werden die Volumendaten in einzelnen Texturen gespeichert. Die Schnittbilder werden als 2D-Texturen auf sogenannte Proxy-Geometrie übertragen. Diese repräsentiert einen Stapel von Polygonen, die im 3D-Raum entsprechend der räumlichen Ausdehnung des Datensatzes parallel zueinander angeordnet sind. Für die Darstellung werden diese Schnittbilder über eine Kompositionsfunktion auf die Bildebene projiziert (vgl. Abbildung 8). Dieser Ansatz wird direkt von der Grafikkarte unterstützt, da auschließlich Texturen und Blending eingesetzt werden. Aus diesem Grund ist die Methode äußerst performant, bleibt aber auf die Rasterisierung beschränkt. Deshalb ist sie unflexibel und kann nicht durch Algorithmen oder Beschleunigungsmethoden erweitert werden. Die interaktive Rotation der Darstellung erfordert eine Umsortierung der Polygone und Texturdaten, um von jeder Betrachterposition aus eine korrekte Darstellung zu erhalten. Dazu müssen die Texturdaten als Stapel in jeder axialen Raumrichtung vorliegen. Dies erfordert eine dreimal so große Auslastung des Grafikspeichers. Zwischen der Umschaltung der Texturstapel entsteht störendes visuelles Flimmern.
Im Gegensatz dazu kann eine dreidimensionale Textur eingesetzt werden. Bei Änderungen der Kameraposition, wird die Proxy-Geometrie hierbei neu erzeugt, so dass diese entlang der akutellen Sicht ausgerichtet ist (vgl. Abbildung 9). Das Mapping der Texturdaten erfolgt dann entsprechend der Geometrieposition. Dieser Vorgehensweise eliminiert die Nachteile des objekt-orienterten Ansatzes und liefert eine bessere visuelle Qualität, da die Hardware auf der 3D-Textur eine trilineare Interpolation ausführt.

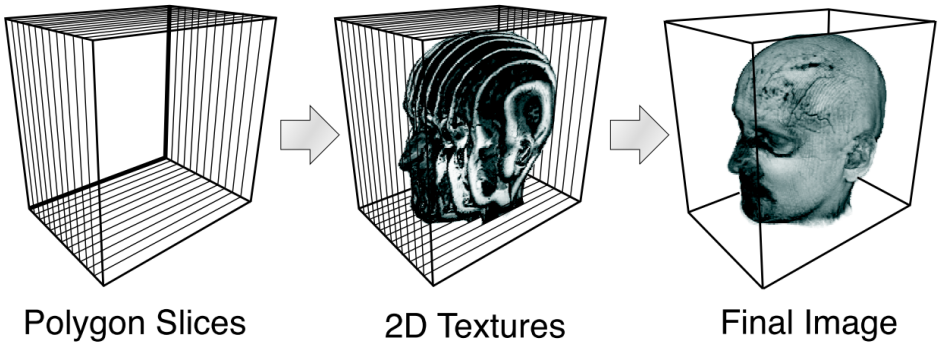

Abbildung 8: Objekt-orientiertes Volumenrendering. Die Polygone werden entlang der Achsen des Objekt-Koordinatensystems ausgerichtet.

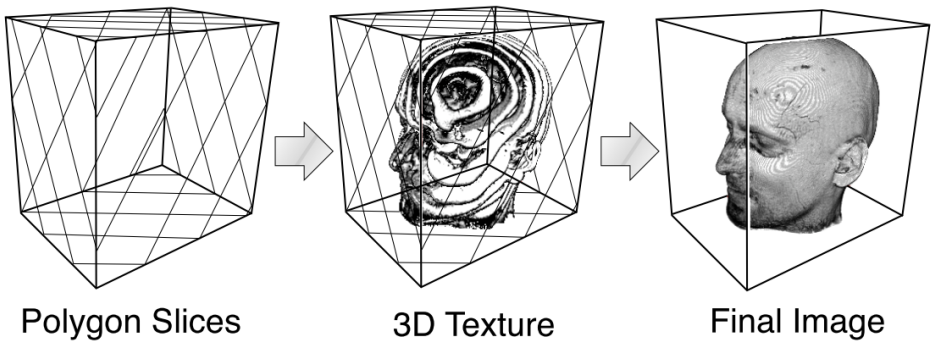

Abbildung 9: Sicht-orientiertes Volumenrendering. Die Geometrie wird entlang der Betrachterposition organisiert.

Volumen-Raycasting Die Idee des Raycastings ist es, von der Kameraposition aus Strahlen durch die Bildebene in das Volumen zu senden. Daher wird es auch als *bild-basierte* Methode bezeichnet. Für jedes Pixel der Bildebene wird ein Strahl verschickt. Dabei wird entlang eines Strahls das Volumen traversiert und an diskreten Positionen abgetastet. Das Volumenrendering-Integral wird so direkt ausgewertet.
Vorteil dieses Verfahrens ist die Flexibilität des Algorithmus. Dadurch, dass die Strahlen unabhängig voneinander evaluiert werden, existieren viele Optimierungsmöglichkeiten, wie *Early-Ray-Termination*, *Adaptives Sampling* und *Empty Space Leaping*. Ein weiterer Vorteil ist die Anwendung des Verfahrens auf tetraeder-förmigen Rasterdaten. Mit der rasanten Weiterentwicklung der Grafik-Hardware wurde diese Methode auch auf der GPU anwendbar und spielt heute eine wichtige Rolle in der Volumenvisualisierung.

An dieser Stelle folgt eine kurze Beschreibung des Volumen-Raycasting-Algorithmus:

1. **Startkonfiguration des Strahls:** Der Eintrittspunkt und die Strahlrichtung müssen in Bezug auf die Kameraposition und das Pixel der Bildebene definiert werden. Die Koordinaten für die Startposition ist der Schnittpunkt des Strahls mit der begrenzenden Geometrie des Datensatzes.

2. **Traversierungsschleife:** In einer Schleife wird der Strahl an diskreten Positionen abgetastet. Jeder Durchgang besteht aus folgendem Ablauf.
   (a) **Datenzugriff:** An der aktuellen Position wird der entsprechende Wert aus dem Datensatz ausgelesen. An dieser Stelle kann über eine Transferfunktion Farbe und Opazität zugewiesen werden.
   (b) **Compositing:** Farbe und Opazität werden über das Front-to-Back-Compositing akkumuliert.
   (c) **Fortschritt:** Die nächste Sampling-Position entlang des Strahls wird berechnet
   (d) **Traversierungsabbruch:** Sobald der Strahl das Volumen verlassen hat ist die Traversierung beendet. Hierzu muss innerhalb der Schleife das Abbruchkriterium getestet werden.

GPU Volume Raycasting Da beim Volumen-Raycasting das Volumenrendering-Integral für jeden Strahl individuell ausgewertet wird, ist eine Parallelisierung des Prozesses möglich. Daher eignet sich eine Applikation auf

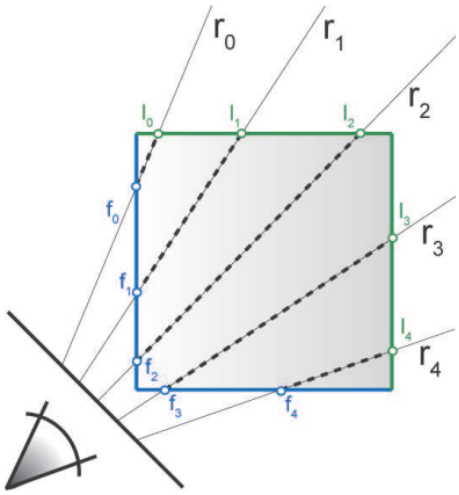

Abbildung 10: Für jedes Pixel der Bildebene wird ein Strahl verfolgt. Der Strahl wird an diskrete Stellen abgetastet um das Volumenrendering-Integral auszuwerten.

der parallelen Prozessstruktur der Grafik-Hardware. Die programmierbare Grafikpipeline ermöglicht dies mit Hilfe von Shadern. Da das Raycasting ein bildbasiertes Verfahren ist, lässt es sich mit Hilfe des Fragment-Shaders umsetzen. Nahezu alle Operationen des Algorithmus können seit Shader Version 3.0 als Single-Pass-Methode über ein Fragment-Programm realisiert werden.
Für die Berechnung der Strahlrichtung existieren zwei Varianten. Zum einen werden für die Berechnung der Startkonfiguration die Vorder- und Rückseite (*Front-* und *Backfaces*) der *Bounding-Box*, die das Volumen umfasst, separat in eine Textur gerendert. Dabei werden die Positionen, wie in Abbildung 11 dargestellt, als RGB-Farbwert kodiert in einer Textur gespeichert. Im Shader kann für jedes Fragement die Strahlrichtung mit Hilfe der beiden Texturen berechnet werden, indem der Farbvektor der Frontfaces von dem der Backfaces subtrahiert wird. Die Länge des Ergebnisvektors wird für das Abbruchkriterium der Traversierung gespeichert. Das Ergebnis der Subtraktion wird normalisiert.
Die andere Version des GPU-Volumen-Raycastings kommt ohne das Rendern der Front- und Backfaces in eine RGBA-Textur aus. Stattdessen wird die Richtung des Strahls direkt über die Kameraposition bestimmt. Diese wird im Vertex-Shader mit Hilfe der inversen Model-View-Matrix in Bezug zum lokalen Koordinatensystem des Volumens berechnet. Anschließend wird die Position der Kamera zusammen mit der jeweiligen Vertex-Koordinate der Bounding-Box an den Fragment-Shader als *varying* übergeben. Im Fragment-Programm wird über die interpolierte Vertex-Position

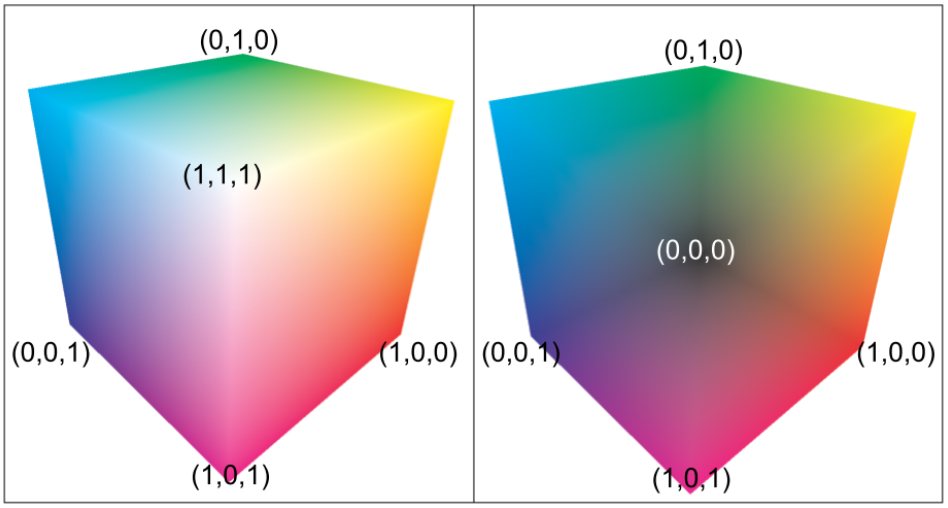

Abbildung 11: Vorder- (links) und Rückseite (rechts) der Bounding Box als RGBA-Texturen kodiert.

und die Kamerakoordinaten der Richtungsvektor berechnet. Die Bounding-Box wird über die minimale und maximale Position der Volumenausdehnung festgelegt. Die Traversierung wird nur fortgesetzt, wenn sich die aktuelle Sampling-Position innerhalb dieses Wertebereichs befindet.

2.2.3 Transferfunktionen

Innerhalb eines Volumendatensatzes repräsentieren abstrakte skalare Werte räumliche und physikalische Unterschiede. Aus diesem Grund existiert keine natürliche Möglichkeit die Absorptions- und Emissionskoeffizienten bei der Abtastung zu identifizieren. Im Gegensatz dazu muss der Benutzer entscheiden, welche optischen Eigenschaften unterschiedlichen Strukturen zugewiesen werden. Diese Zuweisung von optischen Eigenschaften, wie Farbe und Transparenz, wird auch *Transferfunktion* genannt. Der Prozess der Findung einer geeigneten Transferfunktion für den konkreten Anwendungsfall, auch als *Klassifikation* bezeichnet, ist nicht trivial. Transferfunktionen werden verwendet um bestimmte Muster zu identifizieren und diese zu bestimmten skalaren Wertebereichen der Eingangsmenge zuzuordnen. Die Anwendung einer Transferfunktion zur Klassifizierung der Daten ist somit kein einmaliges Übertragen von RGB- und α-Werten auf die Skalare, sondern ein komplexer Prozess. Der Lösungsraum der möglichen Konfigurationen ist sehr groß. Die bewusste Einstellung einer Transferfunktion erfordert genaues Wissen über die zugrunde liegenden Daten und deren räumliche Struktur. Aus diesem Grund ist es währenddessen notwendig, dass der Benutzer direktes visuelles Feedback über seine aktuellen Einstellungen erhält, indem die Transferfunktion in Echtzeit auf die Volumenvi-

sualisierung angewendet wird. Oftmals werden für spezifische klinische Fragestellungen auch vorkonfigurierte Transferfunktionen (*Presets*) eingesetzt.
Neben der eindimensionalen Farb- beziehungsweise Opazitätswertzuweisung beschäftigt sich die Forschung mit mehrdimensionalen Transferfunktionen. Diese beziehen beispielsweise zusätzlich zum Skalarwert den Gradientenbetrag ein, um Gewebeunterschiede deutlicher hervorzuheben und so auch feinere Strukturen an Grenzflächen zu visualisieren.
Außer den Absorptions- und Emissionskoeffizienten können auch andere optische Eigenschaften aus dem skalaren Wert abgeleitet werden. Diese umfassen unter anderem Reflektion und Transluzenz. Im Rahmen dieser Arbeit soll auf diese aber nicht weiter eingegangen werden.

2.3 3D im Web

Die Idee dreidimensionale Inhalte im Web darzustellen existiert bereits seit Veröffentlichung der ersten 2D HTML Seiten. Die Möglichkeit räumlichen Inhalte in Echtzeit zu visualisieren und zu manipulieren schien neue Möglichkeiten für viele Anwendungen zu eröffnen. Seit Mitte der Neunziger wurden immer wieder vielversprechende neue Modelle zur 3D-Integration in den Browser vorgeschlagen. Trotzdem wird bisher keine einzige Technologie weitläufig verwendet. Die Gründe dafür sind vielfältig und hängen mit den überwiegenden Nachteilen der jeweiligen Vorschläge und Methoden zusammen, welche die Akzeptanz bei den Benutzern behindert. Darunter fallen beispielsweise die Installation zusätzlicher Plugins oder Software. Auch auf Entwickler-Seite sind Defizite, wie mangelnde Schnittstellen und die geringe Ausführungsgeschwindigkeit, Gründe für die geringe Verbreitung. Es fehlte bisher an Vereinheitlichungen und Standards an denen sich die Programmierer orientieren konnten.
Auf dem Stand heutiger Technik bieten die vielerorts verfügbare Breitbandanbindung und die zunehmende Verbreitung von Smartphones und Tablets Grundvorraussetzungen für neue Herangehensweisen an die Entwicklungen von 3D im Web. In heutigen Geräten ist 3D-Hardware oft integriert.

Im Folgenden soll nun eine Übersicht über die bisher vorgeschlagenen Modelle gegeben werden. Die meisten Systeme folgen dabei dem pluginbasierten Ansatz. Daneben existieren Versuche, das 3D-Rendering-System direkt in die Browser-Architektur zu integrieren. Dazu werden Ansätze aufgezeigt, die 3D-Darstellungen mit Hilfe zweidimensionaler Techniken emulieren. Die folgenden Ausführungen beziehen sich auf [BEJZ09].

2.3.1 Plugin-basierte Ansätze

Plugin-basierte Ansätze haben im Allgemeinen den Nachteil, dass das Plugin nicht standardmäßig auf dem System installiert ist. Es wird auf Seite des Benutzers also vorausgesetzt dieses nachträglich einzurichten. Des Weitern muss sich der Anwender mit unerwünschten Nebeneffekten, wie Sicherheitsfragen und eventuellen Inkompatibiltäten mit dem System, beschäftigen. Dabei ist das Plugin weitgehend abgekapselt vom Browsersystem und gängigen Webtechnologien, so dass die Flexibiltät durch fehlende Schnittstellen eingeschränkt ist.

Adobe Flash Der bekannteste Vertreter von Browser-Plugins ist wohl Flash von Adobe Systems. Flash ist eine Multimedia-Plattform zur Darstellung von Animationen, Videos und interaktiven Inhalten. Seit der Version 10 bietet Flash Unterstützung für 3D-Transformationen und Objekte an. Das 3D-Engine basiert jedoch auf der CPU und nicht auf Hardwarebeschleunigung durch die Grafikkarte.
Am 4. Oktober 2011 hat Adobe die Version 11 des Flash Players veröffentlicht. Dieser integriert die sogenannte Stage3D-API. Über diese Schnittstelle lässt sich hardwarebeschleunigte interaktive 3D-Grafik erzeugen.

Silverlight Als Alternative zu Flash hat Microsoft Silverlight veröffentlicht, welches auf dem .NET-Framework basiert. Das Prinzip ist ähnlich. Silverlight ermöglicht ebenfalls interaktive Vektorgrafiken und Animationen, sowie Audio-Wiedergabe. Version 3 bietet lediglich die Option, Darstellungen aus dem Zweidimensionalen in einen dreidimensionale Raum zu transformieren. Silverlight unterstützt dabei allerdings keine hardwarebeschleunigten 3D-Grafiken.

Java Mit Hilfe von Java-Applets ist es möglich, hardwarebeschleunigte grafische Inhalte im Webkontext zu visualisieren. 1997 wurde mit *Java3D* ein erster 3D-fähiger Ansatz des Java-Entwicklers Sun Microsystems veröffentlicht. Die Java3D Bibliothek ist ein Szenengraphsystem zur Darstellung und Manipulation dreidimensionaler Grafiken. Die Entwicklung wurde aber zugunsten des Low-Level API *JOGL* eingestellt. Ziel von JOGL war es, ein javabasiertes Werkzeug für die Spielentwicklung umzusetzen. JOGL bietet dabei Zugriff auf alle Funktionalitäten von OpenGL und OpenGL ES.

O3D O3D oder Open 3D wurde 2009 von Goolge Inc. eingeführt. Das System bestand ursprünglich aus zwei Teilen. Der erste Teil stellt ein Browser-Plugin dar, welches in C++ implementiert ist. Es liefert Geometrie und Shader Abstraktionen, die letztendlich entsprechend der vorhandenen Schnittstelle in DirectX oder OpenGL Befehle umgesetzt werden. Der zweite Teil

ist eine in JavaScript implementierte Programmierschnittstelle. Diese entspricht der Struktur eines Szenengraphen, wie Java3D oder OpenSG.
Unter dem Kürzel o3d hat Google die Weiterentwicklung von O3D herausgegeben. Diese ist nicht länger plugin-basiert, sondern setzt auf WebGL auf.

X3D X3D (Extensible 3D) ist ein auf XML basierender ISO Standard zur Beschreibung von 3D-Modellen im Browserkontext. Entwickelt wurde X3D durch das web3D Consortium. Im Jahr 2001 hat das World Wide Web Consortium (W3C) X3D zum offziellen Standard für 3D-Inhalte im Internet erklärt. Als Nachfolger von *VRML* lassen sich auch in X3D interaktive 3D-Anwendungen in Echtzeit realisieren. Gegenüber VRML bietet X3D weitaus mehr Schnittstellen und Komponenten. Des Weiteren arbeitet X3D mit Profilen, welche die notwendigen Komponenten für die jeweiligen Anwendungsbereiche mit sich bringen. Für die Betrachtung von X3D-Dateien ist jedoch ein Browser-Plugin notwendig.

2.3.2 Ansätze ohne Plugins

Es existieren Entwicklungen, die sich nur an den Gegebenheiten des Browsers und den vorhandenen Webtechnologien orientieren. Hier wird auf den Einsatz zusätzlicher Software verzichtet.

JavaScript Die Sprache JavaScript ist mittlerweile Bestandteil von Webbrowsern. Durch die stetige Weiterentwicklung von JavaScript und der JavaScript-VM (Virtuelle Maschine) wächst die Beliebtheit dieser Skript-Sprache zunehmend. Durch die enorme Verbesserung der Ausführungsgeschwindigkeit von JavaScript in den letzten Jahren, konnten anspruchsvolle Applikationen realisiert werden. Unter anderem wurden Versuche ausgeführt, Grundideen der 3D-Grafik-Pipeline mit 2D-Elementen nachzubauen. Apple hat beispielsweise 3D-CSS-Transformationen in der Webkit-Engine integriert. Diese Ansätze beziehen sich allerdings nur auf zweidimensionale Elemente.

Hardwarebeschleungigung Erste Ansätze hardwarebeschleungigter 3D-Grafik im Browser wurden von Mozilla (Canvas3D) und Opera (3D-Context) umgesetzt. Der Grundgedanke ist hierbei, über ein spezielles Canvas-Element mit Hilfe von JavaScript direkt OpenGL-Befehle aufzurufen. Basierend auf diesem Konzept wurden erste Bibliotheken entwickelt, die dem Entwickler die Programmierung erleichtern sollten. Das Hauptproblem ist jedoch der Unterschied der Ausführungsgeschwindigkeit von JavaScript gegenüber nativen Programmiersprachen. Dieser macht sich bei komplexen Berechnungen, wie zum Beispiel Kollisionserkennung, bemerkbar. Trotzdem bie-

ten die Entwicklungen immer schnellerer JavaScript-Engines Grund zum Optimismus. Das Canvas3D-Element ist der Ursprung von WebGL.

2.3.3 WebGL

Die Darstellung von interaktiven 3D-Inhalten mit Hilfe von OpenGL war im Browser bisher ausschließlich über externe Erweiterungen, wie Java-Applets und Browser-Plugins möglich. Seit der Veröffentlichung von HTML5 und WebGL besteht nun die Möglichkeit, ohne Umwege 3D-Inhalte im Browser darzustellen.
Die Version 5 der Hypertext Markup Language (HTML) bringt für die grafische Darstellung vielfältige Neuerungen. So bietet die Auszeichnungssprache Funktionen für dynamische 2D- und 3D-Grafiken, die zuvor nur mit Hilfe von Plugins, wie beispielsweise Adobe Flash, eingesetzt werden konnten.
WebGL steht für Web Graphics Library und ist ein plattformunabhängiger, lizenzfreier Web-Standard. Die Bibliothek erweitert die Möglichkeiten von JavaScript, um interaktive 3D-Inhalte im Browserkontext darzustellen. Die Spezifikation der API basiert auf OpenGL ES 2.0, weshalb sich WebGL als stark shaderbasierte Schnittstelle darstellt. Die Spezifikation wurde als Version 1.0 am 3. März 2011 veröffentlicht. Der WebGL Kontext ist über das Canvas-Element von HTML 5 zugänglich.
Die meisten Browser-Hersteller haben WebGL bereits intgeriert. Darunter befinden sich Apple, Google, Mozilla und Opera, die auch Mitglieder der *WebGL Working Group* [1] sind.

2.4 Webbasierte Visualisierung

Im folgenden Abschnitt beziehen sich die Ausführungen auf [BW00]. Die Fortschritte der Visualisierung im Webkontext beruhen prinzipiell auf zwei Grundlagen. Zum einen bietet das Web die Möglichkeit, mehreren Benutzern eine Anwendung zur kollaborativen Interaktion zur Verfügung zu stellen. Auf eine Applikation kann also von einer Gruppe von Personen, die sich zu einem Zeitpunkt an verschieden Orten aufhalten, zugegriffen werden. Diese interagieren gemeinsam mit der Applikation.
Die andere Domäne umfasst die Verteilung von Rechenlast über das Netz. Bei dieser *verteilten Visualisierung* muss weiter differenziert werden.
Die verteilte Visualisierung umfasst zum einen den Fall, dass aufwändige Programme in kleinere Prozesse aufgeteilt werden. Dabei können sehr rechenintensive und komplexe Aufgaben auf Supercomputer ausgelagert werden. Anspruchsvolle wissenschaftliche Berechnungen können somit effizient auf externen Geräten ausgeführt werden. Die Darstellung des Ergebnisses erfolgt dann client-seitig. In diesem Zusammenhang spielt auch

[1] http://www.khronos.org/webgl/

der Begriff *Modular Visualization Environment (MVE)* eine Rolle. MVE beschreibt die Aufspaltung des Systems in einzelne, für bestimmte Aufgaben des Visualissierungssystems konzipierte Module. Im Rahmen der Anwendung innerhalb eines Netzwerkes besteht die Möglichkeit der transparenten Ausführung von Modulen auf entferntliegenden Rechnern. Die einzelnen Module sind dabei kompatibel zu verschiedenen Plattformen und können, je nach Aufwand der Aufgabe, auf entsprechenden Servern mit den notwendigen Resourcen ausgelagert werden.
Eine andere Option ist die webbasierte Verteilung von Aufgaben zwischen Client und Server. Im Kern unterscheidet man hier zwischen client- und server-seitiger Visualisierung. Hierzu soll Abbildung 12 zunächst eine Übersicht über die verschiedenen Wege zur webbasierten Visualisierung geben.

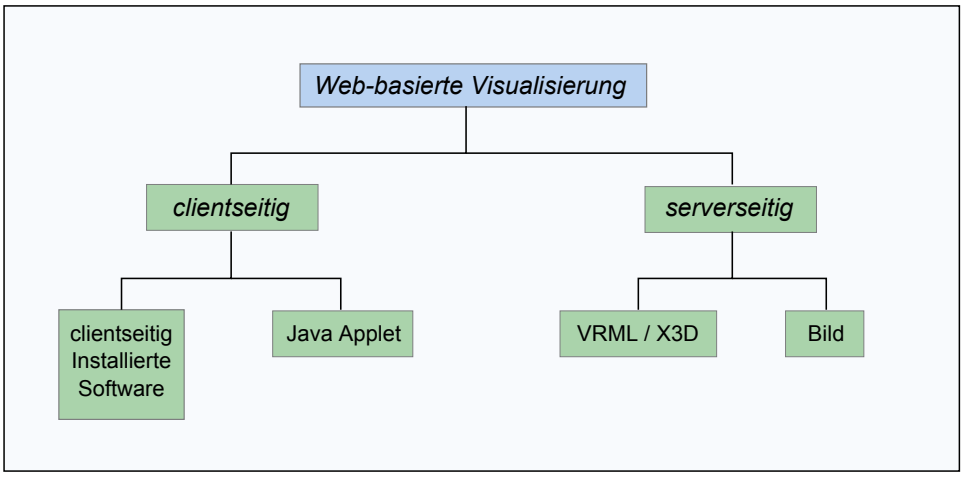

Abbildung 12: Ausprägungen der webbasierten Visualisierung als Baumgrafik.

2.4.1 Client-seitige Visualisierung

Im Sinne der client-seitigen Visualisierung findet die Ausführung auf dem Rechner des Clients statt. Hier gibt es verschiedene Ansätze, die im Folgenden aufgezeigt werden.

Visualisierungssoftware Es wird vorausgesetzt das beim Client bereits die notwendige Visualisierungssoftware, zum Beispiel als Browseranwendung oder externe Applikation, verfügbar ist. Die Anwendung ist nun so konfiguriert, dass sie über den Browser zugänglich ist. Sobald die Website Daten eines bestimmten MIME-Typ enthält wird die Anwendung aktiviert. Diese führt dann den Visualisierungsprozess aus. Zusätzlich zu den Daten kann auch ein Skript mit einem MIME-Typ zur Verfügung gestellt werden,

das den Prozess beschreibt, durch den die Daten visualisiert werden sollen. Dieses Skript kann zum Beispiel von einer MVE-Applikation verstanden werden, welche dann eine vorkonfigurierte Visualisierung ausführen kann.

Java-Applets Mit Hilfe eines Java-Applets kann neben Daten und Skript auch die Applikation übertragen werden. Somit ist es für den Client nicht weiter erforderlich, dass eine Visualisierungssoftware bereits installiert ist. Java-Applets unterliegen allerdings einer Sicherheitsvorkehrung, die es nicht erlaubt Daten vom Dateisystem des Clients oder anderen Quellen als dem Applet-Host zu beziehen. Daher müssen die Daten über eine Uploadfunktion vom Client auf den Server übertragen werden. Eine andere Möglichkeit ist die Einrichtung eines Dateiservers auf dem Server, der das Applet bereitstellt. Dieser läuft als Java-Service und unterliegt nicht den Einschränkungen eines Applets. Daten können so über eine URL von einem beliebigen Repository bezogen werden.

2.4.2 Server-seitige Visualisierung

Dem gegenüber stehen server-seitige Herangehensweisen. Die Visualisierung wird primär auf einem externen Server ausgeführt, wobei der Client die Benutzeroberfläche und das Ergebnis über den Browser darstellt. Die Rechenlast liegt also beim Server. Demzufolge muss dieser die notwendige Software und Rechenkapazität bereitstellen. Auch bei diesem Ansatz kann zwischen unterschiedlichen Modellen differenziert werden.

VRML und X3D Der Benutzer definiert Daten und Visualisierungsanweisungen und überträgt sie über ein Formular oder Java-Applet an die Visualisierungssoftware (zum Beispiel das VTK) auf dem Server. Diese berechnet ein Oberflächenmodell und sendet dieses dem Client als Antwort zurück. Der Browser empfängt das Modell in einem entsprechenden Format, zum Beispiel VRML oder X3D und kann mit Hilfe eines Plugins die 3D-Darstellung anzeigen. Dabei bleibt ein gewisser Grad an Interaktivität erhalten. Der Benutzer kann mit dem Modell im Rahmen der Möglichkeiten des Format und des Plugins interagieren.

Bildserien Eine weitere Option ist die Ausgliederung der Projektion. In diesem Fall erstellt der Server direkt die zweidimensionalen Bilder und überträgt diese an den Browser. Zusätzliche Plugins sind also nicht notwendig, die Interaktivität geht dabei jedoch verloren.

2.5 Medizinische Visualisierung im Webkontext

Die webbasierte medizinische Visualisierung muss sich mit zusätzlichen Angelegenheiten beschäftigen. Im einzelnen spielen Sicherheit, die große Datenmenge, die Privatsphäre des Patienten, Skalierbarkeit der Applikation, Interaktion und Genauigkeit der Darstellung eine entscheidende Rolle im webbasierten Visualisierungsprozess.
Medizinische Datensätze weisen häufig eine sehr große Datenmenge auf. Durch immer präziser werdende Bildakquisitionsverfahren wird diese auch weiterhin steigen. Diese Daten werden vermehrt zentral und vernetzt in sogenannten Repositories gespeichert. Für die Visualisierung der Daten muss entschieden werden, ob diese lokal stattfinden soll. Das bedeutet, dass eventuell große Datenmengen auf den Rechner des Anwenders übertragen werden müssen oder eine entfernte Routine, die Zugriff auf das jeweilige Repository besitzt, den Visualisierungsprozess durchführt.
Webbasierte Anwendungen sind in ihrer Effektivität hauptsächlich über die Bandbreitenbeschränkung des Internets limitiert. Von daher ist es eine Herausforderung ein webbasiertes interaktives Visualisierungssystem zu realisieren, welches die Ergebnisse einer serverseitigen Routine darstellt.

3 Verwandte Arbeiten

Im Rahmen der oben erwähnten Technologien wurden verschiedene Arbeiten und Frameworks veröffentlicht. Diese, die nahen Bezug zu dieser Arbeit haben sollen in diesem Teil beschrieben und diskutiert werden.

Der Ansatz aus [BA01] beschreibt den Versuch texturbasierte Volumenvisualisierung in VRML oder X3D zu integrieren. Die Motivation der Arbeit bestand darin, den bisherigen Standard VRML97 um Volumenrendering-Techniken zu erweitern, da bisher nur gewöhnliche Geometrie damit dargestellt werden konnte. Diese Option bietet bis dahin nur die Möglichkeit Isoflächen als Polygonnetze [EWE99] in VRML zu visualisieren.
Durch die Erweiterung von VRML um einen zusätzlichen Knoten konnte Texture Slicing mit Hilfe von 2D und 3D Texturen eingesetzt werden.
Das vorgestellte Framework lieferte erste Möglichkeiten zur texturbasierten Volumenvisualisierung im Webbrowser. Mit Hilfe eines VRML oder X3D Plugins ist somit auch eine interaktive Darstellung von Volumendaten möglich. Die Methode hat jedoch den Nachteil, dass große Datenmengen zunächst durch den Client heruntergeladen werden müssen. Voraussetzung für das Texture Slicing ist, dass sich die gesamten Daten im Grafikspeicher befinden. Auch liefert der texturbasierte Ansatz nicht für jedes Anwendungsfeld eine ausreichende Bildqualität.

Die service-basierte Visualisierung auf Basis von Webservices [BM07] beschreibt ein System, dass die serverseitigen Kapazitäten zur Bildsynthese verwendet. Dieser Ansatz ist eine Ausprägung der serverseitigen Visualisierung. Nach der Bildgenerierung werden die Bilddaten in Form von X3D-Modellen zum Client geschickt. Für die Definition des Visualisierungsprozesses wird dazu vom Client ein Skript mitgesendet.
Motiviert war diese Arbeit durch den Wunsch nach virtueller Zusammenarbeit über das Internet bei anspruchsvollen Visualisierungsaufgaben. Aufgrund dessen wurde ein System entwickelt, welches die Daten zentral verwaltet und den Benutzern Zugriff auf die Visualisierungsroutine erlaubt. Diesbezüglich können durch den Anwender Projekte erstellt werden beziehungsweise verbindet sich dieser mit einem existierenden Visualisierungsprojekt. Medizinische Daten, die über das System verschickt werden unterlaufen vorher einem Anonymisierungsprozess. Persönliche Informationen, die der Identifikation der Daten dienen, werden entfernt. Die Kommunikation funktioniert ausschließlich über temporäre IDs der Projekte. Zusätzlich werden gesicherte Verbindungen für den Datenaustausch verwendet.
Als Beispielprozess wurde eine Segmentierungsfunktion mit Hilfe eines 2D-Joint-Histogrammes implementiert. Dazu wurden Skalarwerte und Gradientenwerte miteinander verzahnt. Über Primitive können einzelne Zonen in dem zweidimensionalen Histogramm ausgewählt werden, welche eine bestimmte Segmentierung des Datensatzes identifizieren. Mit Hilfe der Segmentierungsdaten wird über den Marching-Cubes-Algorithmus eine X3D-Szene erzeugt, welche mehrere Geometrieobjekte entsprechend der segmentierten Anteile als X3D-Objekte beinhaltet.
Das System verteilt die Last der Visualisierungsroutinen auf externe Services. Da jeder Visualisierungsprozess, abhängig von seiner Komplexität, eine gewisse Latenzzeit aufweist, sind bei kollaborativen Anwendung durch mehrere verbundene Clients längere Verzögerungen zu erwarten. Durch die starke serverseitige Beanspruchung ist dieser Ansatz ungeeigent für interaktive Visualisierungen. Ein weiterer Nachteil ist die ausschließliche Darstellung von Isoflächen mit Hilfe von X3D. Die Notwendigkeit eines Browserplugins spricht nicht für eine breite Akzeptanz des Modells.

In [WLC10] wird ein Ansatz vorgestellt, der interaktive medizinische Visualisierung im Webkontext mit Hilfe von Java-Applets und dem VTK realisiert. Clientseitig wird dem Benutzer ein Applet zur Verfügung gestellt, das die VTK-Bibliothek für die Visualisierungsfunktionalität einbindet. Das VTK liefert Elemente für die Benutzeroberfläche, sowie zusätzliche interaktive Funktionen. Die serverseitige Architektur besteht aus einer Applikations- und einer Service-Ebene. Auf dem Applikationsserver ist ein Servlet für die Koordinierung zwischen clientseitiger Visualisierung und serverseitiger Datenverarbeitung zuständig. Der Webservice hingegen ist für die Ausführung der notwendigen Berechnungen vorgesehen, die der Benutzer im

Rahmen der Visualisierungsroutine definieren kann. Hierzu integriert der Service auch Bibliotheken, wie das VTK und das *Multimod Application Framework (MAF)* [2]. Die Daten liegen in dem System zunächst beim Client vor, der diese mittels einer Uploadfunktion des Applets über eine FTP-Verbindung auf den Server überträgt. Dafür wurden spezielle XML-Pakete definiert, die neben den Daten auch Anweisungen für die Datenverarbeitung enthalten. Diese Parameter werden vom Benutzer über die Benutzeroberfläche konfiguriert. Da das VTK eine C++-basierte Bibliothek ist und der Austausch, sowie die Serialisierung und Deserialisierung von Java- und C++- Objekten des VTK nicht miteinander kompatibel sind, wird XML als Kommunikationsmedium verwendet. Die Daten müssen nicht zwangsläufig für jede Operation von Client und Server verschickt werden. Einfache Operationen, die keine Veränderung des Datensatzes erforden kann der Client lokal ausführen.
Der Ansatz zeigt eine Möglichkeit, wie interaktive Visualisierung im Web realisierbar ist. Grundvoraussetzung ist allerdings der Einsatz von Java-Applets. Diese bringen jedoch den Vorteil mit sich, dass auch clientseitig vorhandene Bibliotheken, wie das VTK, zur Visualisierung eingesetzt werden können. Grundsätzliche Bedingung dafür ist allerdings, dass diese auf Java basieren oder zumindest ein Java Interface anbieten[3]. Somit wurde ein Modell entwickelt, welches sich sowohl clientseitiger als auch serverseitiger Vorteile bedient. Einzige Einschränkungen bringt der Einsatz von Java-Applets mit sich, denen wie Java-Programmen nicht unbegrenzt Arbeitsspeicher zur Verfügung steht. Bei der Verarbeitung großer Datensätze kann es hier zu Speicherengpässen kommen. Der sogenannte *Java Heap Space* kann vom Anwender erhöht werden, was jedoch zusätzliche Konfiguration durch den Benutzer erfordert.

In direktem Bezug zu dieser Arbeit stehen aktuelle Ansätze, die WebGL als Grundlage für die interaktive Visualisierung verwenden. In [CRP11] wird ein Projekt vorgestellt, dessen Ziel es ist eine GPU-unterstützte medizinische Visualsierungsplattform für das Web zu entwickeln. Der Browser erhält vom Server eine WebGL-Anwendung, über die direkt mit der GPU kommuniziert wird. Der Server liefert segmentierte medizinische Bilddaten als komplexe Polygonnetze, die direkt im Browser interaktiv visualisiert werden können. Die Geometriedaten werden dabei iterativ vom Server nachgeladen. Die Interaktion beschränkt sich zur Zeit auf Kamerakontrolle und grundlegende Mausinteraktionen. Die Architektur des Systems ist ähnlich zu [WLC10] aufgebaut. Es existiert ein Anwendungsserver, der

[2] Open Source Bibliothek für die Entwicklung bio-medizinischer Applikationen (`https://www.biomedtown.org/biomed_town/MAF/`)

[3] Das in C++ geschriebene VTK unterstützt Java über einen Wrapper. Diese stellt mit Hilfe des *Java Native Interface* (JNI) eine Java-Schnittstelle bereit um auf die nativen Funktionen zuzugreifen

für die Datenverarbeitung der Visualisierungsroutine zuständig ist. Dazu integriert das Back-End Bibliotheken wie das VTK und ITK, mit denen unter anderem komplexe Segmentierungsalgorithmen bereitgestellt werden. Die Brücke zwischen Anwendungsserver und Client bildet ein Webserver. Dieser reagiert auf Benutzeranfragen und leitet für bestimmte Berechnungen die Kontrolle an den Anwendungsserver weiter. Der Webbrowser dient als Client und ist direkt mit dem Webserver verbunden. Mit Hilfe von AJAX werden Anfragen an den Webserver gestellt, um asynchrones Laden von Bilddaten oder Geomterie zu ermöglichen.
Das Projekt bedient sich der neuen Webtechnologien HTML5 und WebGL. Dadurch bietet es den Vorteil, auf zusätzliche Browsererweiterungen und Java-Applets verzichten zu können. Das Ergebnis zeigt, dass damit das Rendern großer Polygonnetze in Echtzeit möglich ist. Die aktuelle Arbeit stellt den Einsatz von direktem Volumenrendering in Aussicht.

An dieser Stelle setzt [CSK+11] an und präsentiert ein direktes Volumenrendering-System für das Web. Das Framework untersucht die Möglichkeiten von WebGL zur Anwendung des Volume-Raycasting-Verfahrens. Mit Hilfe von Shaderprogrammen ist die Programmierung der Grafikpipeline möglich, daher kann die gesamte Visualsierungsprozess lokal auf der GPU des Clients stattfinden. Das Verfahren wurde für die Visualisierung medizinischer Bilddaten und Radarbilder entworfen. Letztere liefern ebenso Volumedaten, die jedoch kein uniformes Raster bilden, sondern kugelförmig strukturiert sind.
Da WebGL zur Zeit keine 3D-Texturen unterstützt, werden die Bilddaten in eine 2D-Textur geschrieben. Dazu werden die einzelnen Schnittbilder des Datensatzes zu einem Bild zusammengefügt. Dieser Prozess ist allerdings nicht Teil der Visualisierungspipeline, sondern wird in einem Vorverarbeitungsschritt durchgeführt. Zusätzlich wird zwischen den Schnitten interpoliert, so dass keine qualitativen Nachteile entstehen. Das Resultat ist zur Veranschaulichung in Abbildung 13 aufgezeigt. Da die meisten Handheld-Geräte gegenüber einem PC einen geringeren Grafikspeicher besitzen, wird die Textur in diesem Fall in ihrer Größe entsprechend angepasst.
Im Shader wird über einen Algorithmus die erforderliche 3D-Koordinate zurückgerechnet. Dazu müssen zusätzliche Angaben, wie die Anzahl der Reihen, die Bilder pro Reihe und die Gesamtzahl der Schnittbilder dem Shader übergeben werden.
Des Weiteren unterstützt das Framework bisher nur Datensätze mit einer Tiefe von bis zu 8 Bit. In der Regel weisen medizinische Datensätz eine höhere Auflösung auf (von bis zu 16 Bit). Die derzeitigen Webtechnologien unterstützen aber nur Bilder mit geringerer Tiefe. Aus diesem Grund werden die Daten auf 8 Bit reduziert. Es wird die Möglichkeit angegeben

mit Hilfe eines zusätzlich Farbkanals die fehlenden Bits zu ergänzen und im Shader zusammenzurechnen. Die Implementierung steht aber noch aus.
Für die Darstellung wird beim Raycasting eine Lookup-Tabelle als weitere Textur für die Integration einer Transferfunktion eingebunden. Der Ansatz

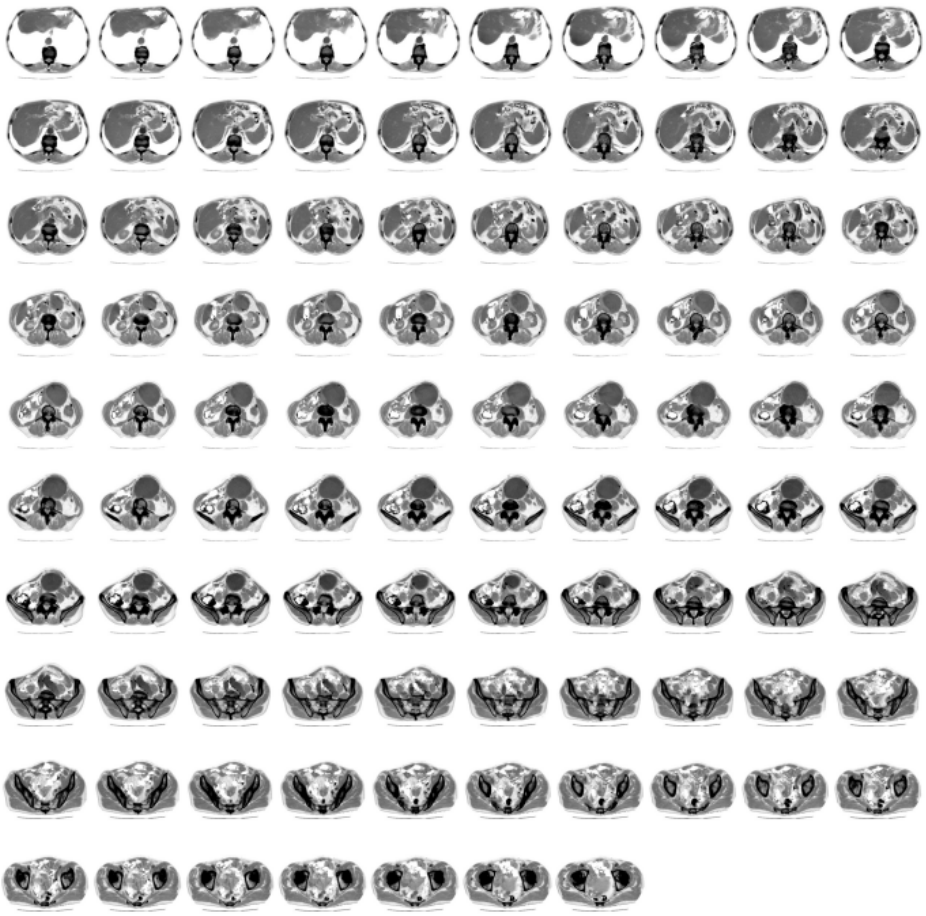

Abbildung 13: Aorta Datensatz in Matrix-Form. Mit Hilfe dieser Strukturierung kann ein Volumen dem Shader als 2D-Textur bereitgestellt werden.

ist vollständig client-basiert. Die Ergebnisse zeigen, dass WebGL die notwendige Leistung für die interaktive direkte Volumenvisualisierung liefert. Die Funktionalität ist hauptsächlich über den Shader implementiert, so dass die Geschwindigkeit vergleichbar mit der von nativen Anwendungen ist. Da die rechenintensiven Aufgaben also beim Client ausgeführt werden, wird die Anwendung für kollaborative Zwecke in Aussicht gestellt. Bedingung hierfür ist allerdings, dass diese die notwendigen Hardwarevoraussetzung mitbringen. Probleme machen jedoch noch aktuelle Browserimplementationen von WebGL, die die Ausführung

sehr komplexer Shader prinzipiell nicht erlauben. Dies verhindert beispielsweise die Ausführung des Raycasting-Algorithmus mit sehr kleiner Schrittweite.

3.1 Fazit

An dieser Stelle soll noch einmal zusammenfassend auf den aktuellen Stand der Technik eingegangen werden und die Ergebnisse bewertet werden.

Die plugin-basierten Ansätze sind aufgrund neuer Technologien nicht mehr zeitgemäß. Die 3D-Formate X3D und VRML konnten sich zu ihrer Zeit nicht richtig durchsetzen und fanden keine weite Verbreitung. Zudem ist der Einsatz dieser Techniken für die medizinische Visualisierung unflexibel und nur über Workarounds möglich. Da kein Zugriff auf die Low-Level-Grafikprogrammierschnittstelle existiert, ist beispielsweise Volumen-Raycasting schwer realisierbar. Der Einsatz von Java-Applets erlaubt die Verwendung nativer Bibliotheken. So können bereits etablierte Toolkits, wie das VTK, direkt integriert werden. Der zur Verfügung stehende Arbeitsspeicher ist für Applets allerdings beschränkt. Dies behindert die Verarbeitung großer Datensätze.
Der Weg über die serverseitige Visualisierung bringt den Vorteil mit sich, dass komplette Datensätze für den Visualisierungsprozess nicht zwangsläufig übertragen werden müssen. Stattdessen erhält der Client nur das Ergebnis der Routine, beispiesweise die Geometrie einer segmentierten Struktur. Bei gleichzeitiger Verwendung durch mehrere Benutzer kann das Serversystem stark beansprucht werden und bietet dadurch keine Interaktion in Echtzeit mehr.

WebGL liefert für die medizinische Visualisierung im Webkontext neue Möglichkeiten. Die auf OpenGL ES 2.0 basierte Schnittstelle stellt die notwendige Flexibiltät für client-seitige GPU-basierte Visualisierungsalgorithmen zur Verfügung. So können komplexe Polygonnetze können in Echtzeit dargestellt werden und texturbasierte Verfahren sowie auch Volumen-Raycasting sind realisierbar. Das WebGL API bringt allerdings nicht den vollen Funktionsumfang von OpenGL mit sich. Dies ist damit begründet, das WebGL-Applikationen geringe Hardware-Anforderungen voraussetzen um auf vielen Geräten lauffähig zu sein. Für die direkte Volumenvisualisierung hat dies enorme Auswirkungen. So werden bisweilen keine 3D-Texturen unterstützt, was den Umweg über zweidimensionale Texturen erfordert. Außerdem sind die Texturkomponenten auf ein Tiefe von 8 Bit beschränkt.

4 Umsetzung

Im Hinblick auf die erwähnten Ansätze soll in dieser Arbeit ein webbasiertes Volumenrendering-System mit neuen Webtechnologien entwickelt werden. Intention dieser Arbeit ist es, die Erkenntnisse und Ideen der oben genannten Arbeiten zu integrieren und nach Möglichkeit weiterzuentwickeln.
In diesem Umfang soll ein erweiterbares und objektorientiertes Framework entwickelt werden, dass die Umsetzung der folgenden Aspekte behandelt:

1. Volumen-Raycasting mit WebGL zur direkten Darstellung von Volumendaten in Echtzeit
2. Server-seitige Bereitstellung der Datensätze und Berechnung des Histogramms der Grauwertverteilung
3. Inkrementelles Laden der Texturdaten bei gleichzeitiger direkter Visualisierung
4. Integration einer interaktiv konfigurierbaren Transferfunktion

Für die technische Umsetzung soll das Google Web Toolkit und die Google AppEngine als Infrastruktur verwendet werden. Mit diesen Technologien sind skalierbare Webanwendungen realisierbar, die mit der objektorientierten Programmiersprache Java entwickelt werden können.

4.1 Grundlagen für die Entwicklung

Die praktische Umsetzung der Arbeit erfordert den Einsatz verschiedener internetbasierter Technologien. Zunächst folgen Erläuterungen über die Funktionalität von Webanwendungen im Allgemeinen. Im Anschluss daran wird aufgrund dieser Basis auf die verwendeten Webtechnologien, das Google Web Toolkit und die Google App Engine, eingegangen. Die entsprechenden Ausführungen basieren auf den Online-Dokumentationen (vgl. [Goo11b] und [Goo11a]).

4.1.1 Webanwendungen

Durch technologische Fortschritte der Breitbandanbindung in Sachen Geschwindkeit und Verfügbarkeit wurde es in den letzten Jahren zunehmend beliebter, rechenintensive und komplexe Programme über das Internet zur Verfügung zu stellen. Webseiten werden im Gegensatz zu früheren sehr statischen Darstellungen von Informationen zunehmend dynamischer. In diesem Sinne stehen Webandwendungen oder Webapplikationen für ein exemplarisches Anwendungsfeld. Eine Webapplikation ist ein Programm,

das mit Hilfe eines Webbrowser über eine URL erreichbar ist. Die eigentliche Anwendung erfolgt auf einem Webserver. Die Darstellung wird durch den Client (Webbrowser) realisiert, über den die Interaktion mit dem Benutzer erfolgt.
Die Funktionsweise einer Webanwendung sieht dabei wie folgt aus: Zu Beginn ruft der Benutzer die URL der Webanwendung, beispielsweise über einen Webbrowser, auf. Diese Anfrage, auch HTTP-Request genannt, wird an den Webserver (*Host*) übertragen, der die Anfrage an ein zuständiges Programm weiterleitet. Dieses Programm ist für die Verarbeitung der Eingangsdaten des Requests verantwortlich und generiert entsprechende Elemente, die vom Browser interpretiert werden können. Dazu gehören komplette Websiten als HTML-Quellcode, aber auch Inhalte, wie Bilder, PDF-Dokumente oder Flash-Programme. Die Ergebnisdaten werden als Antwort (HTTP-Response) an den Browser zurückgeschickt und von diesem dargestellt. Dieser Zyklus (Request Cyle) wiederholt sich, sobald der Benutzer eine neue Anfrage an die Anwendung stellt. Zur Definition der Anfrageparameter dienen dabei Formulareingaben, die Parameter der URL und Cookies, die an den Webserver geschickt werden.
Bei der Verwendung einer Webapplikation entstehen Daten, die sinnvollerweise für die Dauer der Anwendung gespeichert werden sollten. Hier ist die Rede von sogenannten persistenten Daten oder Sitzungsdaten. Solche Daten fallen zum Beispiel für Anwendungen an, die eine Benutzeranmeldung erforden. Die Zeitspanne zischen Anmeldung und Abmeldung von der Anwendung wird als Sitzung (Session) bezeichnet. Da es bei der Verbindung von Client und Server über zustandslose Protokolle, wie HTTP, zu keiner stehenden Verbindung kommt und zudem keine Identifikationsdaten übertragen werden, müssen Benutzer auf anderem Wege identifiziert werden. Aus diesem Grund wird beim Zugriff vom Client auf einen Webserver ein Session-ID übertragen. Über diese eindeutige Kennung können die Sitzungsdaten einem Benutzer zugeordnet werden. Die Sitzungsdaten werden dazu in Datenbanken oder Dateien temporär gespeichert. Für längere Sessions von bis zu mehreren Tagen werden auch HTTP-Cookies verwendet, diese speichern die Session-ID clientseitig ab. Abbildung 14 stellt die Client-Server-Kommunikation zusammenfassend dar. Das Web hat sich also von der statischen Darstellungen von HTML-Seiten weitestgehend verabschiedet. Heute sind die meisten Webseiten dynamisch gestaltet. Mit der Einführung von CSS wurde ein erster Schritt erreicht, um den Inhalt von der Darstellung zu trennen. Heutzutage basieren viele Internetauftritte auf ausschließlich dynamischen Webseiten, die mit JavaScript und AJAX umgesetzt sind. Damit ist es grundsätzlich möglich Webanwendungen zu entwickeln, die in Bezug auf Interaktivtät nahe an die Leistung von Desktopapplikationen herankommen. Grundlage für die voranschreitende Technik sind vor allem die Entwicklungen von XHTML und HTML5 Gemeinschaften, aber auch Browsersystemen, wie Mozilla und WebKit.

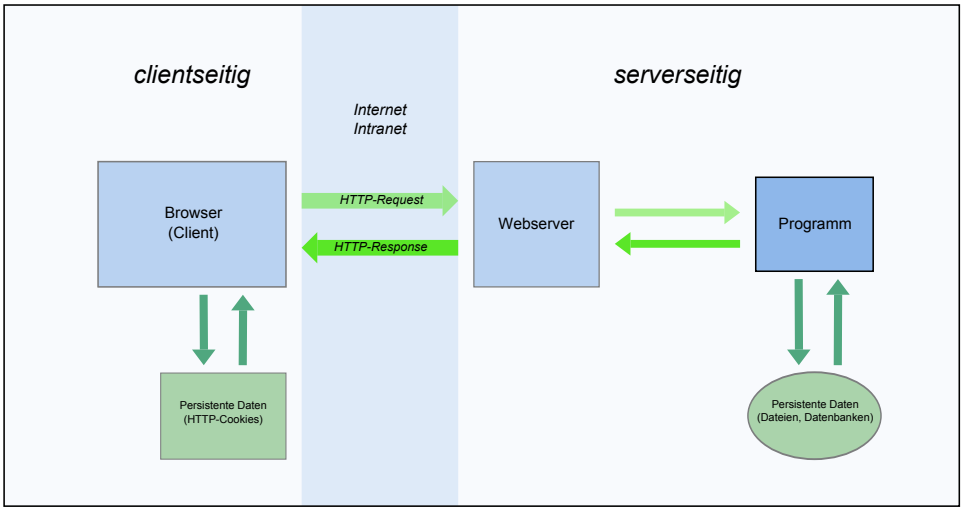

Abbildung 14: Die Kommunikation zwischen Client- und Serverseite. Der Request-Cycle beschreibt die grundlegende Funktionsweise einer Webanwendung.

4.1.2 Google Web Toolkit

Das *Google Web Toolkit*[4] *(GWT)* ist eine Sammlung von Werkzeugen zur Entwicklung von Webapplikationen. Das Toolkit erlaubt die Entwicklung komplexer JavaScript-Front-Ends mit Hilfe der objekt-orientierten Programmiersprache Java. Das GWT integriert dazu einen Cross-Compiler, der eine Java-Anwendung in eine unabhängige JavaScript-Version übersetzt. Dem Entwickler wird somit erlaubt, seine gesamte Anwendung sowohl server- als auch client-seitig mit Java zu realisieren. Dazu liefert das GWT verschiedene Bibliotheken, die bis auf wenige native Methoden, ebenfalls in Java geschrieben sind. Das Google Web Toolkit wurde am 17. Mai 2006 unter der Apache License 2.0 veröffentlicht und ist daher als Open Source Projekt verfügbar. Seit dem 8. September 2011 steht es in der aktuellen Version 2.4.0 zur Verfügung.
Des Weiteren bringt das Toolkit Unterstützung für Ajax (Asynchronous JavaScript and XML) mit sich. Darunter fallen beispielsweise asynchrone Remote Procedure Calls (RPC) zur Kommuniktion zwischen Client und Server, History Management, Bookmaring, Internationalisierung und Cross-Browser Portabilität.

Das Entwickeln und Debuggen funktioniert aufgrund weniger Abhängigkeiten mit beliebigen Java-Entwicklungs-Werkzeugen. Beim *Deployen*, dem Einbinden der Anwendung in die Server Laufzeitumgebung, kann der JavaScript-Quelltext durch Quelltextverschleierung (*Obfuscating*) verdeckt werden. Außerdem findet eine starke Optimierung der kompilierten

[4] `http://code.google.com/intl/de-DE/webtoolkit/`

JavaScript-Dateien statt.

Eine weitere Besonderheit des GWTs ist, dass GWT Anwendungen in zwei verschiedenen Modi lauffähig sind:

- Zum einen existiert der Entwicklungs-Modus (*Development mode*). Die Applikation läuft als Java Bytecode innerhalb der Java Virtual Machine (JVM). In der Regel wird dieser Modus verwendet, solange sich die Anwendung noch in der Entwicklungsphase befindet. Der Entwicklungs-Modus unterstützt Debugging sowohl auf Client- als auch auf Server-Seite. Außerdem kann der Code zur Laufzeit verändert werden, ohne die Programmausführung zu unterbrechen.
- Sobald die Anwendung fehlerfrei und lauffähig ist, kann sie deployt werden. Die Anwendung läuft in diesem Fall als reine HTML und JavaScript-Applikation auf einem Server. Dieser Modus wird als Produktions-Modus bezeichent (*Production mode*).

An dieser Stelle sollen die für die Umsetzung relevanten Aspekte der Google Web Toolkits kurz aufgezeigt werden:

Java-to-JavaScript Compiler Übersetzt die Programmiersprache Java in die Programmiersprache JavaScript. Bei der Kompilierung werden für die verschiedenen unterstützten Browsertypen entsprechende Versionen erstellt. So kann bei einer Anfrage vom Client die Version für den entsprechenden Browser zurückgesendet werden. Mit Hilfe eines *bootstrapping*-Prozesses wird zur Laufzeit die richtige Version geladen. Im GWT wird diese Technik als „deferred binding“ bezeichnet.

Development Mode Der Development Mode muss durch das *GWT Developer Plugin* vom verwendeteten Browser unterstützt werden, um nativen Java Code ausführen zu können.

JRE Emulation Library Diese Bibliothek beinhaltet die JavaScript Implementierungen der meisten Klassen aus der Standard Java-Bibliothek. Außerdem stellt sie große Teile aus java.lang und einzelne gebräuchliche Klassen aus dem java.util Paket zur Verfügung. Die Funktionalität dieser Klassen kann für Entwicklung vollständig verwendet werden. Neben diesen Standard-Bibliotheken existieren Erweiterungsmöglichkeiten für das GWT. So bietet Google selbst weitere Bibliotheken an. Zudem können auch Bibliotheken von anderen Entwicklern eingebunden werden.

Web UI Class Library Bietet eine Vielzahl an Klassen und Schnittstellen zur Erzeugung von Oberflächenelementen, sogenannten *Widgets*.

UIBinder Mit Hilfe des UIBinder-Frameworks kann das Layout der Benutzeroberfläche der GWT-Applikation als HTML-Seite deklariert werden. Die Applikation kann auf die Elemente der Seite über UIBinding zugreifen, um die Oberfläche dynamisch zu erweitern und zusätzliche Funktionalität zuzuweisen.

4.1.3 Client-Server Kommunikation im Google Web Toolkit

Das GWT bietet ein abstraktes **Remote Procedure Call** (RPC) Framework zur Vereinfachung der Client-Server-Kommunikation. Über die RPC API können Objekte zwischen Client und Server über das Hypertext Transfer Protocol (HTTP) ausgetauscht werden. Dabei wird der auszuführende serverseitige Code auch als *Service* bezeichnet. RPC Aufrufe laufen dabei asynchron ab. Das bedeutet, dass der Client nicht erst auf eine Antwort warten muss, sondern in dieser Zeit seinen eigenen Programmablauf fortsetzen kann. Nach dem der Service die Anfrage bearbeitet hat wird umgehend die Antwort zurück an den Client geschickt. Dazu muss der Client eine Callback-Methode definieren, die dem Service als Objekt mitgeliefert wird. Das Diagramm in Abbildung 15 liefert einen Überblick über die Implementierung der Remote-Procedure-Calls. Die Implementierung eines Service-

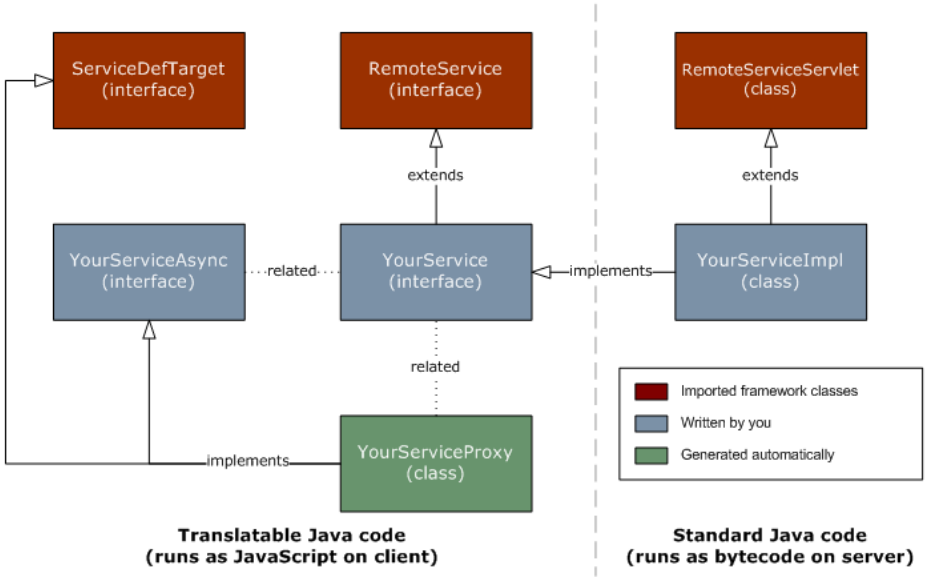

Abbildung 15: Aufbau der Komponenten des Remote-Procedure-Calls.

Aufrufs nach dem Muster der Remote-Procedure-Calls umfasst die folgenden Komponenten:

Ein Java-Interface, welches von dem Interface `com.google.gwt.user.client.rpc.RemoteService` abgeleitet

werden muss, stellt das Bindeglied für die Kommunikation dar. Dieses Interface wird sowohl vom Client als auch vom Servercode verwendet. Zur Definition des Interfaces bedarf es nur einer zusätzlichen Annotation (`@RemoteServiceRelativePath("nameOfService")`), die den Namen des Services definiert (siehe Listing 1). Diese Annotation verbindet den Service mit einem Standard-Pfad relativ zur Basis-URL der Moduls. Des Weiteren handelt es sich hierbei um eine gewöhnliche Java Schnittstelle, die beliebig viele Methodendeklarationen beinhalten darf. Da das Interface dem Client zugänglich sein muss, wird es im Client Paket platziert.

Listing 1: Beispieldefinition eines synchronen Interfaces

```
@RemoteServiceRelativePath("myservice")
public interface MyService extends RemoteService {
        public String getMyResponse();
}
```

Das GWT kann nur serialisierbare Datentypen über einen Remote-Procedure-Call austauschen. Aus diesem Grund dürfen in den Methodendeklarationen des RemoteService Interfaces auschließlich serialisierbare Datentypen verwendet werden. Jeder Datentyp, der nicht von Natur aus serialisierbar ist, wie beispielsweise primitive Datentypen, Wrapper Klassen, die Klassen `java.lang.String` und `java.util.Date` oder Enumerationen müssen dazu das Interfaces `java.io.Serializable` implementieren. Für die Implementierung des serverseitigen RPC-Mechanismus werden Servlets eingesetzt. Servlets sind Bestandteil der Java Enterprise Edition (Java EE) Spezifikation und werden für dynamische Website Generierung eingesetzt. Servlets werden dabei in der Regel nur für die HTTP-Kommunikation verwendet.
Ein **Servlet** ist eine Erweiterung eines sogenannten Web-Containers, wie beispielsweise Tomcat oder Jetty. Dieser ist für die Entgegennahme einer Anfrage (*Request*) verantwortlich und leitet diese an das zuständige Servlet weiter. Dazu ist es notwendig, dass der Servlet-Container, das entsprechende Servlet bekanntmacht. Dies geschieht über die Konfiguration der web.xml Datei. In dieser müssen das Servlet und das Mapping auf dem jeweiligen Pfad zum Service angegeben werden. Dies ist exemplarisch in Listing 2 dargestellt.

Listing 2: Konfiguration des Servlets in der web.xml

```
<servlet>
 <servlet-name>applicationServlet</servlet-name>
 <servlet-class>de.uniko.iwm.gwt.server.rpc.ApplicationServiceImpl
     </servlet-class>
</servlet>
```

```
<servlet-mapping>
 <servlet-name>applicationServlet</servlet-name>
 <url-pattern>/gwtchannelapiexample/applicationService</url-
     pattern>
</servlet-mapping>
```

Der Request wird vom Container in ein Request-Objekt umgewandelt (*Marshalling*) um es dem Servlet zugänglich zu machen. Das Servlet erhält so Informationen über die Herkunft und den Inhalt der Anfrage, ohne Kenntniss über den Transfer zu benötigen.

Innerhalb des GWTs muss ein Servlet die abstrakte Klasse `com.google.gwt.user.server.rpc.RemoteServiceServlet` erweitern und das vom Benutzer erstellte Remote-Service-Interface implementieren. Die Klasse RemoteServiceServlet übernimmt dabei die komplexen Aufgaben des Servlets. Darunter fallen die Annahme eines Requests, diesen zu deserialisieren, zu parsen und wiederum die Antwort zu serialisieren um diese zum Client zurückzusenden. Der Entwickler muss nun lediglich die Methoden seines Service-Interfaces implementieren.

Zu klären ist darüber hinaus, wie die Client-Seite mit dem Service kommuniziert. Um dem Client den Service zugänglich zu machen wird eine asynchrone Version des Interfaces benötigt. Da der Client, wie bereits erwähnt in seiner Ausführung nicht warten soll bis die Antwort zurückgeschickt wird, muss ein sogenanntes Callback-Objekt erstellt werden. Die Deklaration dieses Objekts wird bei der Methodendeklaration des asynchronen Interfaces angegeben (siehe Listing 3). Hier beschreibt der Generic Type Parameter den Rückgabetyp entsprechend des synchronen Interfaces. Anders als bei diesem muss der Rückgabetyp hier immer `void` sein.

Listing 3: Beispieldefinition eines asynchronen Interfaces

```
public interface MyServiceAsync {
        void getMyResponse(AsyncCallback<String> callback);
}
```

Die noch ausstehende clientseitige Implementierung des erstellten Interfaces wird vom GWT übernommen. Das vorher beschriebene *Deferred Binding* findet auch an dieser Stelle wieder seinen Einsatz. Da der clientseitige Code in verschiedenen Browsern ausgeführt werden kann und jeder Typ eine andere Laufzeitumgebung mit sich bringt, müsste der Entwickler für jeden Browsertyp eine entsprechende Implementierung bereitstellen. Stattdessen wird mit Hilfe des Deferred Binding ein Proxy-Objekt erzeugt. Dazu liefert das GWT die statische Methode GWT.create() welches als Parameter die Definition des Service Interfaces erhält (siehe Listing 4). Diese Methode gibt eine Client-Instanz zum entsprechenden Service zurück. Über diese können die Funktionen des Services aufgerufen werden.

Listing 4: Erzeugen eines Client Proxy Objekts.

```
MyServiceAsync myService = GWT.create(MyService.class);
```

Für die Verarbeitung der Serverantwort auf der Clientseite stellt das GWT das bereits erwähnte Interface `AsyncCallback` bereit, welches vom Entwickler implementiert werden muss. Dazu liefert das Interface zwei Methodendeklarationen. Zum einen muss die Methode `onSuccess` implementiert werden, für den Fall, dass der Server die Anfrage erfolgreich beantworten konnte. Zum anderen die konträre Methode `onFailure`, welche aufgerufen wird, wenn serverseitig ein Fehler bei der Ausführung aufgetreten ist. In diesem Fall wird ein Throwable-Objekt zurückgeliefert welches die Exception beschreibt. Listing 5 zeigt den Aufruf eines Services inklusive einer Beispielimplementation eines Callback-Objekts.

Listing 5: Beispiel für den Aufruf eines Services. Im Falle einer erfolgreichen Antwort wird das Ergebnis als String mit Hilfe eines Label Widgets ausgegeben. Im Fehlerfall wird die Exception innerhalb einer Alert Box angezeigt.

```
myService.getMyResponse(new AsyncCallback<String>() {
        @Override
        public void onSuccess(String result) {
                Label label = new Label(result);
                RootPanel.get().add(label);
        }

        @Override
        public void onFailure(Throwable caught) {
                Window.alert(caught.getMessage());
        }
});
```

Struktur eines GWT-Projekts Die wichtigsten Bestandteile eines GWT-Projekts lassen sich über folgende Komponenten beschreiben. Zur Veranschaulichung zeigt Abbildung 16 den Aufbau eines Beispielprojekts.

- Der **Quellcode** wird in die Pakete *client*, *server* und *shared* aufgeteilt. Das *client*-Paket beinhaltet client-seitigen Code, der in JavaScript übersetzt wird. Server-seitiger Code wird nicht in JavaScript transformiert und muss daher im *server*-Paket abgelegt werden. Des Weiteren müssen Java-Klassen, die auf beiden Seiten verwendet werden (z.B. Entitäten), im Paket *shared* vorhanden sein. Innerhalb dieser Pakete ist die Struktur beliebig.
- Die **Modul-Definition** ist in der Datei mit dem Suffix *gwt.xml* im Wurzel-Paket beschrieben. In der Modul-Definition wird die Java-Klasse angegeben, die den Eintrittspunkt der Applikation definiert.

Außerdem werden zusätzlich verwendete GWT-Bibliotheken und die Pfade zum Quellcode eingetragen, der übersetzt werden soll.

- Das **Web Archiv** (*war*-Verzeichnis) beinhaltet die vollständige Webanwendung nach der Java-Servlet-Spezifikation. Es beinhaltet die HTML-Host-Seite mit dazugehöriger CSS-Datei und ein *WEB-INF*-Verzeichnis. Hierin liegen die notwendigen Bibliotheken sowie der Deployment-Descriptor vor.
- Im **Deployment-Descriptor** *web.xml* wird der Name der Host-Seite gesetzt. Vor allem aber ist hier das Definieren der Servlets relevant.
- Die **HTML Host-Seite** im *war*-Verzeichnis ist der Einstiegspunkt beim Zugriff auf eine GWT-Webanwendung. Die als JavaScript kompilierte Version der Applikation wird über ein SCRIPT-Tag eingebuden:

Listing 6: Einbinden der GWT-Applikation in die Host-Seite

```
<script type="text/javascript" language="javascript" src="
    helloworld/helloworld.nocache.js"></script>
```

Zudem können dieser Seite Widgets über das *id*-Attribut von HTML-Elementen hinzugefügt werden. Die Host-Seite kann auch dynamisch mit Hilfe eines Servlets oder als JSP-Seite (*Java Server Pages*) generiert werden.

4.1.4 Entwicklung von GWT-Applikationen nach dem MVP-Pattern

Das Model-View-Presenter-Pattern (MVP) ist eine Ableitung des Model-View-Controller-Entwurfsmusters, das dazu gedacht ist das Datenmodell und die Logik programmiertechnisch von der Ansicht zu trennen. Es wurde 2004 von Martin Fowler neu formuliert und findet nach seiner Definition heute Anwendung.
Das Modell (*Model*) umfasst die Anwendungslogik und muss die Erreichbarkeit seiner Funktionalität gewährleisten. Die Ansicht (*View*) beschreibt ausschließlich, wie die Daten dargestellt werden sollen. Sie kennt das Modell nicht und beinhaltet keine steuernde Logik. Die Kommunikation zwischen Modell und Ansicht funktioniert über den Präsentator (*Presenter*), welcher beide steuert. Dazu bieten Modell und Ansicht Schnittstellen an, über die der Präsentator die beiden Teile miteinander verknüpft. Dadurch wird die Austauschbarkeit und Wiederverwendbarkeit einzelner Komponenten unterstützt. Die explizite Trennung einzelner Komponenten begünstigt die gleichzeitige Entwicklung durch mehrere Benutzer. Außerdem ist der Einsatz der Musters dazu gedacht, die Durchführbarkeit von Tests zu ermöglichen, ohne die Benutzeroberfläche einzubeziehen.
Chris Ramsdale motiviert in seinem Artikel *Large scale application develop-*

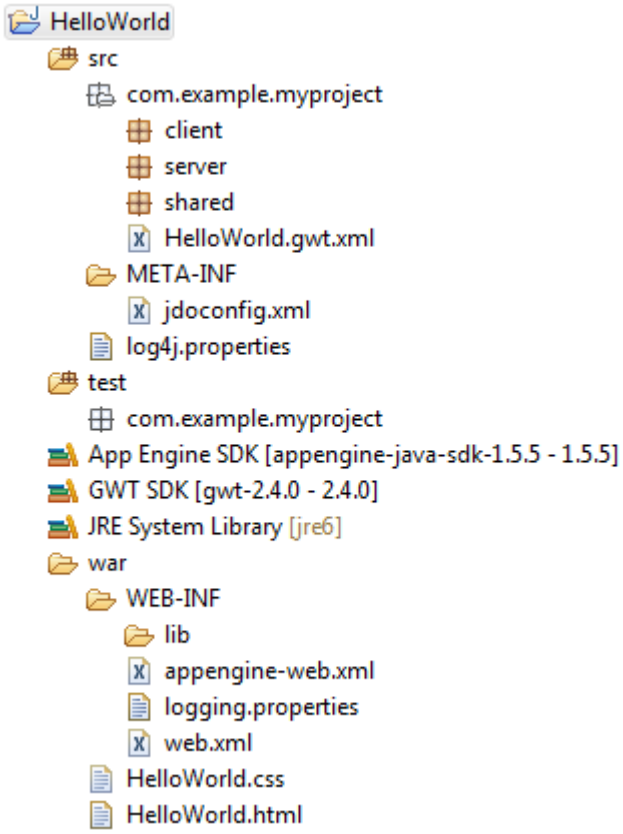

Abbildung 16: Standard-Verzeichnisstruktur und Paket-Hierarchie eines GWT-Projekts in Eclipse.

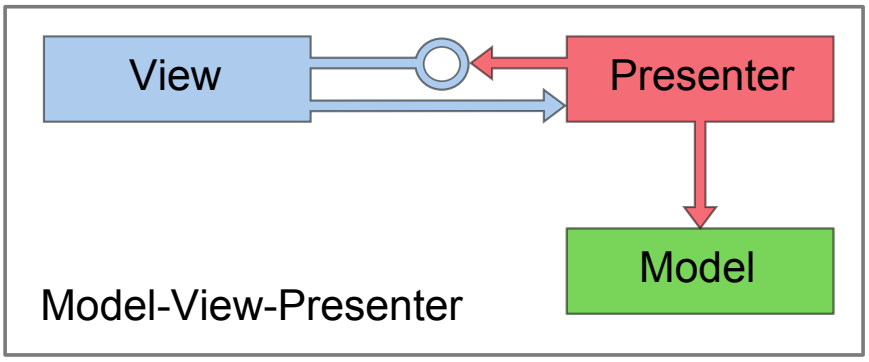

Abbildung 17: Die Struktur des Model-View-Presenter Entwurfmusters. Es verdeutlicht die Zusammenhänge der einzelnen Komponenten und insbesondere die Unkenntnis von Sicht und Model zueinander. Die Kommunikation findet über den Präsentator statt.

ment and MVP [Ram10] den Einsatz des Model-View-Presenter-Entwurfsmusters in GWT-Applikationen. Er begründet die Verwendung vor allem mit der Minimierung der Browserabhängigkeit beim Testen der Anwendung. Tests können so mit der Java-Laufzeitumgebung durchgeführt werden und benötigen keinen Browser. Außerdem kann so nach dem Prinzip vorgegangen werden, die Programmierung der Ansicht so einfach wie möglich zu gestalten. Diese umfasst also lediglich das Layout und GUI-Komponenten wie Buttons, Tabellen und Textfelder. Auch der Einsatz des UIBindings wird dadurch motiviert. Mit dessen Hilfe kann eine deklarative Beschreibung der Oberfläche in der Sprache XML erstellt werden. Der Präsentator ist zuständig für die Datensynchronisation mit dem Server über Remote-Procedure-Calls. Jede Ansicht hat ihren eigenen Präsentator. Dieser behandelt auch die Ereignisse, die von der Ansicht ausgelöst worden sind.
Zusätzlich wird im Rahmen einer GWT-Anwendung ein Controller eingesetzt, der *AppController*. Dieser ist für die Logik verantwortlich, die das gesamte System betrifft und nicht nur spezifisch für einen Präsentator behandelt werden muss. Der AppController verwaltet die Kommunikation zwischen einzelnen Ansichten. Dazu wird ein sogenannter HandlerManager als Event-Bus eingesetzt. Ändert sich beispielsweise nach der Ausführung eines Remote-Procedure-Calls der Status der Anwendung, wird über den Event-Bus der AppController informiert, der daraufhin die Anzeige einer anderen Ansicht auslöst. Des Weiteren verwaltet der AppController auch das *History Management*. Über das History Management werden für den Status der Anwendung Marker gesetzt. So kann auch über die Steuerelemente des Webbrowsers zwischen den Ansichten vor und zurück navigiert werden.

4.1.5 Google App Engine

Bei der Google App Engine (GAE) handelt es sich um eine Plattform sowie ein *Software Development Kit (SDK)* für die Entwicklung und das Hosten von Webanwendungen. Dazu wird die Serverinfrastruktur von Google eingesetzt. Die GAE zählt damit zu den sogenannten *Platform as a Service (PaaS)*-Diensten. Zusammen mit dem GWT bietet sie somit eine komplette, java-basierte Lösung für AJAX Webapplikationen. Die App Engine bietet für die in ihrem Kontext laufenden Anwendungen automatische und fast unbegrenzte Skalierbarkeit, sobald die Nachfrage der Anwendung steigt. In diesem Fall werden mehr Resourcen alloziiert. Bis zu einem gewissen Level an verwendeten Resourcen bleibt die Nutzung der App Engine kostenlos. Zur Zeit werden die Programmiersprachen Pyhton, Java und die von Google entwickelte Programmiersprache Go von der Laufzeitumgebung der App Engine unterstützt. In Zukunft soll die GAE aber programmiersprachenunabhängig werden.

Die App Engine hat jedoch diverse Einschränkungen. Einige von ihnen werden im Folgenden aufgezeigt:

- Java-Anwendungen können nur einen Teil der Klassen aus der JRE Standard Edition verwenden. Dazu existiert die sogenannte *JRE Class White List*.
- Java-Applikationen dürfen keine eigenen Threads erzeugen.
- Verschlüsselte Verbindungen über SSL/HTTPS werden nur von appspot.com-Domänen unterstützt.
- Ein Prozess, der auf dem Server gestartet wird, um eine Anfrage zu beantworten, darf nicht länger als 30 Sekunden dauern.

4.1.6 Services und Technologien der Google App Engine

Die Google App Engine bietet diverse Services und Technologien an, die für die Entwicklung von Webanwendungen essentiell sind. Auf die für diese Arbeit relevanten Technologien wird im Folgenden eingegangen:

Datastore Mit dem Datenspeicher (englische Bezeichnung: *Datastore*) liefert Google ein Kernmodul der App Engine. Der Datastore stellt einen skalierbaren Speicher für Webanwendungen dar und beschreibt einen Objektdatenspeicher ohne Schema. Der Datenspeicher ist keine relationale Datenbank, stattdessen gleicht der GAE-Datenspeicher mehr einer Objekt-Datenbank, die eine große, auf verschiedene Systeme verteilte und sortierte Hashmap darstellt. Der Datastore unterstützt ein Anfragemodul und atomare Transaktionen.
Das Java SDK bietet Implementierungen der Schnittstellen des GAE Datenspeichers. Dazu gehören Java Data Objects API (JDO), das Java Persistence API (JPA) und ein Low-Level-Datenspeicher-API. Diese beinhalten Mechanismen zur Definition von Klassen für Datenobjekte und zur Durchführung von Abfragen. Neben diesen APIs existieren noch weitere Frameworks, die mehr Programmierkomfort bieten, wie beispielsweise Objectify, TWiG und Slim3. Da Objectify für diese Arbeit relevant ist, soll darauf später genauer eingegangen werden.
Eine Schwerpunkt liegt auf der Lese und Abfragegeschwindigkeit. Bei einer Abfrage soll sich die Zeit der Abfrage linear zur Anzahl der Ergebnisse verhalten. Google erreicht dies durch Indizierung und Einschränkugen der Abfragemöglichkeiten.

Im Folgenden werden Kernbegriffe und ihre Funktionalität genauer erläutert:

Entität ist die Bezeichnung für ein Datenobjekt im GAE-Datenspeicher. Eine Entität kann eine oder mehrere **Eigenschaften** besitzen. Eine Eigenschaft ist ein benannter Wert eines oder mehrerer Datentypen. Als zulässige Datentypen gelten unter anderem Zeichenfolgen, Ganzzahlen, Gleitkommazahlen, Datumswerte, Binärdaten sowie Verweise auf andere gültige Entitäten. Eine Entität muss immer einen **Schlüssel** zur Identifikation besitzen. Dieser setzt sich zum Beispiel aus dem **Typ** der Entität und einer Entitäts-ID zusammen, die vom Datenspeicher oder der Applikation zugewiesen wird. Der Typ wird außerdem dazu verwendet Entitäten zu kategorisieren, um die Abfrage zu erleichtern. Eine Entität, die im Datastore gespeichert werden soll, darf ein Datengröße von maximal 1MB besitzen.
Eine **Abfrage** lässt sich mit allen Entitäten eines bestimmten Typs realisieren. Die Anwendung kann Entitäten entweder über den Schlüssel oder Eigenschaften einer Entität abfragen. Das Ergebnis kann keine oder mehere Entitäten liefern. Die mögliche Ergebnisliste kann sortiert oder gefiltert werden. Eine Filterung der Abfrage bringt zudem Vorteile im Hinblick auf Speicherverbrauch, Lauf- und CPU-Zeit.
Die bereits angesprochene Indizierung wird bei jeder Abfrage eingesetzt. Der **Index** stellt dabei eine Tabelle dar, die mögliche Ergebnisse der Abfrage in sortierter Reihenfolge enthält. Die Inidizes werden erstellt und aktualisiert sobald Datenobjekte erzeugt oder modifiziert werden. Der Index wird über eine Konfigurationsdatei definiert und ist von Hand optimierbar. Bei einer Abfrage ruft der Datenspeicher das Resultat direkt aus den entsprechenden Indizes ab.
Eine **Transaktion** kapselt eine oder mehrere Operationen mit dem Datenspeicher. Das Erstellen, Aktualisieren oder Löschen von Entitäten wird also innerhalb einer Transaktion ausgeführt. Die Transaktion stellt die Speicherung sicher. Sie ist erfolgreich, wenn alle Operationen innerhalb der Transaktion korrekt durchgeführt wurden. Im Falle eines Fehlers werden keine Änderungen vorgenommen und bisher durchgeführte Operationen widerrufen (*Rollback*). Eine Transaktion verhindert außerdem, dass mehrere Operationen, die an einer Entiät durchgeführt werden, durch andere Prozesse gestört werden.

Objectify Das zuvor erwähnte Framework Objectify [5] erleichtert die Handhabung des Datenspeichers. Das Low-Level Datastore API bietet die vier grundlegenden Operationen *get*, *put*, *delete* und *query* an, die jedoch umständlich zu bedienen sind. Außerdem speichert die Google App Engine nur spezifische Entitäten statt normalen *POJO* (Plain Old Java Objects)-Klassen ab. Schlüssel sind darüber hinaus nicht typisiert und fehleranfällig. Objectify nimmt sich diesen Problemen an und liefert dem Entwickler ein intuitiveres Framework. Des Weiteren gehören dazu typsichere Schlüssel,

[5] http://code.google.com/p/objectify-appengine/

ein benutzerfreundliches Abfrage-Interface mit Hilfe von Java Generics, ein leichter verständliches Transaktionsmodell und die Möglichkeit, Entitäten ohne sogenannte *Data Transfer Objects* zu verwenden. Die grundlegende Funktionsweise von Objectify ist in Listing 6 aufgezeigt.

Listing 7: Beispiel für die Verwendung für Objectify

```
class Car {
    @Id
    String name;
    String color;
}

Objectify objectify = ObjectifyService.begin();
objectify.put(new Car("911GT3", "black"));

Car car = objectify.get(Car.class, "911GT3");
objectify.delete(car);
```

Blobstore Der Blobstore ist im Gegensatz zum Datenspeicher für das Speichern und Bereitstellen großer binärer Datenobjekte zuständig. Diese Datenobjekte werden als sogenannte BLOBs (*Binary Large Data Objects*) bezeichnet. Die maximale Größe eines BLOBs liegt bei 2 Gigabyte. Ein Blob wird über einen Datei-Upload erstellt. Dabei werden die binären Daten über eine HTTP-POST-Anfrage zum Beispiel mit Hilfe eines Formulars erstellt. Aus dem Inhalt der Anfrage erstellt der Blobstore einen Blob.

Im Detail funktioniert das wie folgt. Dem Benutzer steht ein Datei-Upload-Feld zur Verfügung. Dieses ist über ein Formular mit einer Aktions-URL verbunden. Diese URL beinhaltet den Pfad zum Blobstore und kann zusätzliche Attribute beinhalten. Über die URL wird die Datei direkt in den Blobstore geladen. Daraufhin wird die Datei als Blob gespeichert und eine Referenz (Blob-Schlüssel) auf das Datenobjekt generiert. Die Anfrage wird anschließend so umgeschrieben, dass sie den Blob-Schlüssel und den Pfad zum Service der Applikation beinhaltet. Innerhalb dieses Services wird dann die get-Methode aufgerufen, um weitere Aktionen auszuführen. Darunter fällt beispielsweise das Eintragen des Blob-Schlüssels und zusätzlicher Informationen, wie das Erstelldatum und der MIME-Type in ein Datenspeicher-Objekt. Auch das Auswerten zusätzlicher Attributwerte des ursprünglichen Formulars werden hier behandelt.

Zudem gilt für ein Datenobjekt im Blobstore, dass es nicht veränderbar ist. Nur das Erstellen und Löschen sind erlaubte Operationen. Über den Blob-Schlüssel kann eine Anfrage an den Blob gestellt werden. Das vollständige Bereitstellen eines Blobstore-Wertes kann stückweise und nichtsequentiell erfolgen.

Authentifizierung Im Rahmen einer GWT-Applikation ist eine Benutzerverwaltung möglich. Diese kann über das Google-Konto, eine eigene Kontoverwaltung der Applikation oder über OpenID realisiert werden. Die Anwendung kann feststellen, ob ein aktueller Nutzer angemeldet ist und dadurch entsprechende Optionen für diesen freischalten.
Exemplarisch soll hier die Authentifizierung über ein Google Konto beschrieben werden. Für die Anmeldung findet ein Servlet Einsatz, welches Zugriff auf das Google Nutzerservice-API hat. Bei der Anfrage an das Servlet ermöglicht das Standard-Servlet-API den Test, ob ein Benutzer bereits angemeldet ist. Mit Hilfe der Nutzerservice-Schnittstelle kann dann entsprechend eine Anfrage gestellt werden, um eine Login- oder Logout-URL zu erhalten. Der jeweiligen Methode wird die aktuelle URL der Applikation mitgegeben, um später wieder zum aktuellen Ausführungspunkt der Applikation zurückzukehren. Die Login- beziehungsweise Logout-URL leitet den Nutzer zur Google-Login-Seite oder meldet ihn direkt ab. Nach erfolgreicher Anmeldung oder Abmeldung erfolgt eine Zurückleitung zur Applikation.
Die Anwendung hat bei erfolgreicher Anmeldung Zugriff auf die e-Mail-Adresse, die den Anwender identifiziert. Außerdem kann festgestellt werden, ob der authentifizierte Benutzer Administratorrechte besitzt.

Channel API Im Hinblick auf eine mögliche kollaborative Anwendung soll an dieser Stelle die Funktionsweise der *Channel API* der AppEngine beschrieben werden.

Die Channel API ist das Mittel um zwischen der Applikation und der Servern von Google eine persistente Verbindung aufzubauen. Über diese Verbindung können Nachrichten vom Server zum Client in Echtzeit übertragen werden, ohne dass der Client eine explizite Anfrage stellen muss (*Pollling*). Mit dieser Technik ist es möglich, den Anwender direkt über serverseitige Änderungen zu informieren oder Benutzereingaben und -interaktionen direkt an andere Benutzer zu senden (*Broadcast*). So können Anwendungen entwickelt werden, die gleichzeitig von mehreren Benutzern interaktiv bedienbar sind. Beispiele dafür sind neben kollaborativen Anwendungen auch Multiplayer-Spiele und Chaträume.

Für die Kommunikation mit dem Client erzeugt der Server einen spezifischen Kanal. Dazu schickt der Server dem Client ein *Token* zur Identifizierung des Kanals, über das sich der Client mit dem Kanal verbinden kann. Der Client kann über diesen Kanal nun unvorhersehbare Ereignisse vom Server empfangen. Der umgekehrte Weg von Client zu Server funktioniert weiterhin über gewöhnliche HTTP-Anfragen. Die Kommunikation zwischen mehreren Clients ist in Abb. 18 visualisiert.

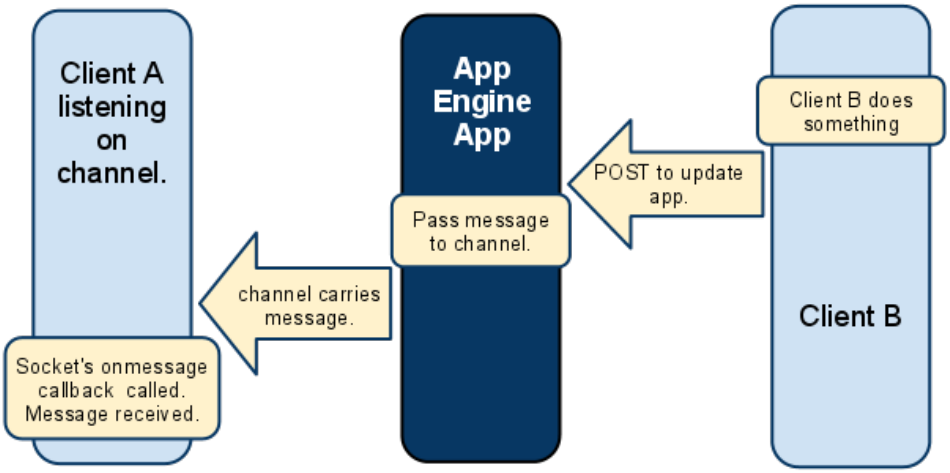

Abbildung 18: Client B sendet eine Nachricht zur Applikation auf dem Server (AppEngine App). Der Server verarbeitet die Nachricht und sendet über den Kanal das Ergebnis zu Client A. Client A empfängt die Nachricht und kann sie verwenden.

4.1.7 Integration von WebGL im GWT

Wie bereits angesprochen ist das GWT erweiterbar. In diesem Zusammenhang wurden verschiedene GWT-Module für WebGL entwickelt. Diese sogenannten *Wrapper* ummanteln das JavaScript-Interface WebGL und ermöglichen die WebGL-Programmierung mit der objekt-orientierten Sprache Java. Für die Entwicklung wurde das quelloffene gwt-g3d-Projekt[6] eingesetzt. Außerdem bietet das Modul ein Shader-System sowie eine Bibliothek zur Berechnung von Vektoren und Matrizen.

4.1.8 WebGL und direktes Volumenrendering. Einschränkungen und Workarounds

Vor der Beschreibung des Systems und der Implementierung sollen in diesem Abschnitt noch die technischen Aspekte von WebGL in Bezug auf direktes Volumenrendering mittels Raycasting aufgezeigt werden.

Programmierbare Grafikpipeline Da WebGL auf OpenGL ES 2.0 basiert, besteht die Option der programmierbaren Grafikpipeline. Da das Volumenraycasting ein bildbasiertes Verfahren ist kann der Hauptteil über ein Fragmentshader-Programm realisiert werden.

Volumendaten Die Größe des zu Datensatzes hängt direkt mit einer effizienten Ausführung der Visualisierungsalgorithmen, der Ladezeit und der

[6] http://code.google.com/p/gwt-g3d/

Ausnutzung des Grafikkartenspeichers zusammen. Da WebGL keine 3D-Texturen unterstützt, muss der Datensatz in eine zweidimensionale Repräsentation überführt werden.

Browser-Support und Shaderkomplexität Unter Linux und Mac OS X ist OpenGL die primäre 3D-Schnittstelle. Im Gegensatz dazu kommt auf Windows-Systemen DirectX zum Einsatz. Bisher konnte keine Darstellung mit WebGL erfolgen, solange der Nutzer keinen OpenGL-Treiber installiert hat. Aus diesem Grund hat Google das *Angle*-Projekt entwickelt. Angle übersetzt die API-Aufrufe von WebGL in DirectX-Befehle für DirectX 9.0c. Solange der verwendete Browser die Angle-Technik implementiert, können WebGL-Inhalte damit dargestellt werden.
Die meisten Browser-Hersteller haben die Verwendung von Angle standardmäßig aktiviert. So findet vor der Ausführung, selbst bei vorhandenem OpenGL-Treiber, eine Übersetzung in DirectX statt. Dies bringt folgende Nachteile mit sich: Zum einen müssen in diesem Rahmen auch Shaderprogramme DirectX-konform kompiliert werden. Diese Übersetzung dauert bei komplexen Programmen sehr lange. Zum anderen können zu komplexe Programme nicht kompiliert werden. Diese Probleme tauchen auch bei der Übersetzung des Raycasting-Shaders mit hoher Abtastungsrate auf. Hierfür existiert bisher lediglich die Möglichkeit, auf eine hohe Darstellungsqualität zu verzichten beziehungsweise den Browser für die native Verwendung von OpenGL zu konfigurieren. Dies lässt sich über das Setzen bestimmter Startparameter oder ein Browser-internes Setup realisieren.

4.2 Implementierung

Unter Berücksichtigung der oben genannten Voraussetzungen und den Einsatz der erläuterten Techniken soll an dieser Stelle auf die Implementierung des Systems eingegangen werden.

4.2.1 Struktur des Systems

Dieser Abschnitt behandelt den Aufbau des Systems. Die grundsätzliche Ausprägung der Webanwendung ist in Abbildung 19 visualisiert. Auf Client-Seite wird ein Webbrowser mit WebGL-Unterstützung für die Visualisierung medizinischer Bilddaten eingesetzt. Der Server sendet das grafische Benutzer-Interface in Folge einer HTTP-Anfrage zum Client. Der Anwender hat nun die Möglichkeit, über die Benutzeroberfläche mit der Applikation zu kommunizieren. Diese Kommunikation umfasst den Upload und die Anfrage von medizinischen Datensätzen. Dazu wird auf Server-Seite die App Engine Infrastruktur eingesetzt. Medizinische Datensätze werden als Blob im Blobstore gespeichert. Der Datenspeicher hingegen referenziert

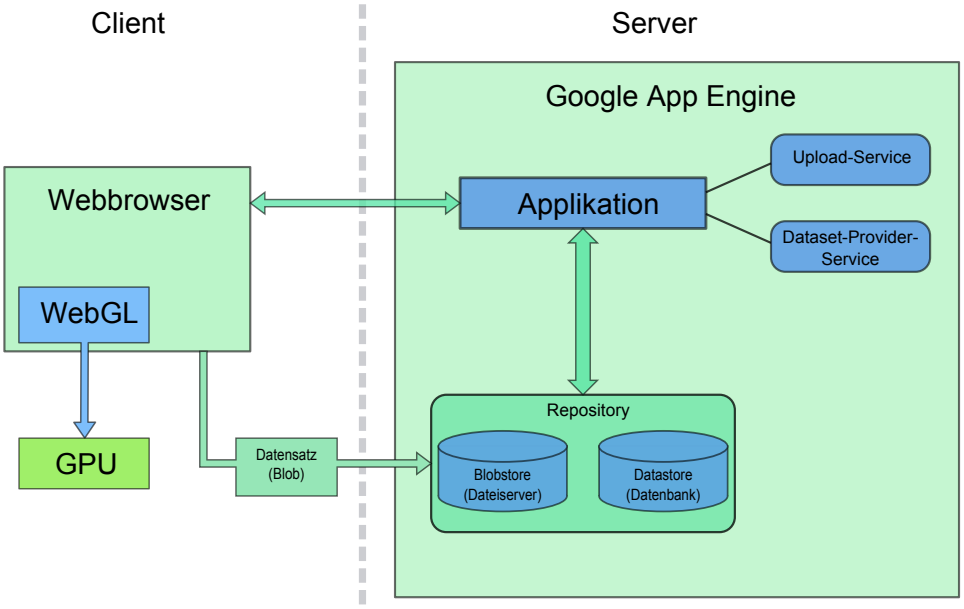

Abbildung 19: Übersicht des entwickelten Systems

diese Daten und trägt weitere beschreibende Informationen als Datenobjekt ein. Bei der Anfrage des Clients wird der Volumendatensatz über den DatasetProvider schrittweise zum Client übertragen und dort visualisiert. Die Implementierung und die Funktionsweise der einzelnen Bausteine soll in den nächsten Abschnitten näher erläutert werden.

4.2.2 Volumenrendering

Für die visuelle Darstellung der Volumendaten wurden verschiedene Verfahren eingesetzt. Diese wurden in einzelnen Render-Modulen umgesetzt (siehe dazu das Klassendiagramm Abbildung 20). Im Prozess der Entwicklung wurde zunächst mit einfachem Textur-Mapping der Zugriff auf die korrekten Texel in der zweidimensionalen Textur überprüft. Für diesen Zweck wurde das SliceRender-Modul umgesetzt, um durch die Transversal-, Sagital- und Frontalebene des Datensatzes zu navigieren.
Im Anschluss daran konnte das GPU-basierte Raycasting realisiert werden. Dazu wurden die beiden unter 2.2.2 vorgestellten Varianten umgesetzt. Für das Sampling von Datensätzen, die Werte einer Tiefe von mehr als 8 Bit repräsentieren, existiert zur Zeit nur die Möglichkeit, den 16-Bit-Wert aus zwei 8-Bit Komponenten im Shader zurückzurechen. Der folgende Pseudocode beschreibt diesen Prozess:

$$topBitsValue \leftarrow topBits * (255/256);$$

$$result16Bit \leftarrow topBitsValue + bottomBits * (1/256);$$

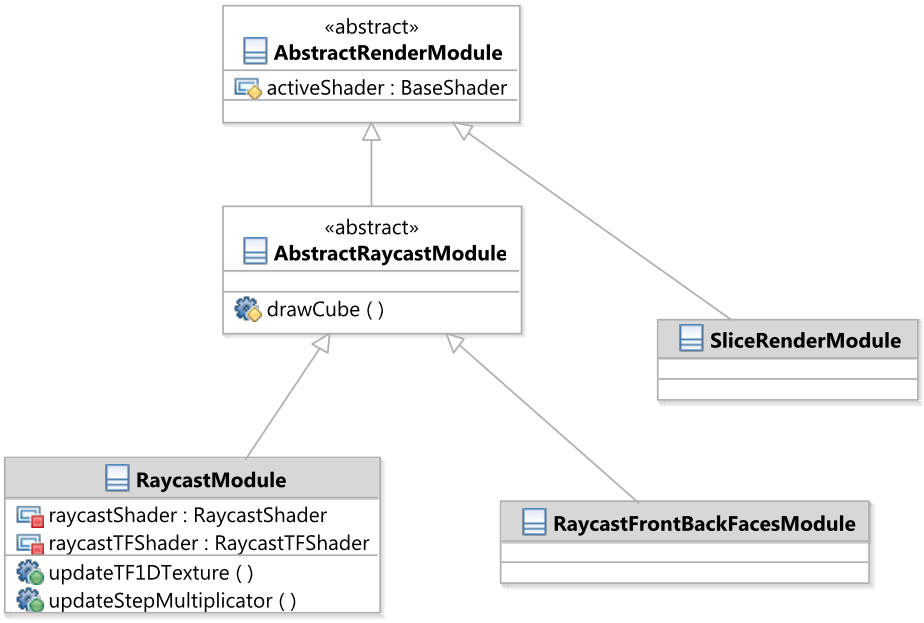

Abbildung 20: Klassendiagramm des Volumenrendering-Systems

Die Repräsentation von Volumendaten als 2D-Textur wird im Folgenden beschrieben.

4.2.3 Komprimierung und Bereitstellung der Datensätze

Die im Browser unterstützten Bildformate sind überschaubar. Die verlustfreie Komprimierung liefert jedoch nur das PNG-Format. Außerdem kann eine PNG-Datei mittels WebGL als Textur geladen werden. Aus diesem Grund ist es für den Einsatz medizinischer Bilddaten im Webkontext relevant. Unter diesem Aspekt wurde ein Java-Programm entwickelt, dass die Umwandlung der Volumendaten in PNGs durchführt. Das Java-Programm sollte später als Service eingesetzt werden, um diese Berechnungen extern durchzuführen. Im Rahmen der Entwicklung mit der App Engine kam es hierbei jedoch zu Problemen. Die Erzeugung von PNG-Bildern aus Rohdaten ist ein anspruchsvoller Prozess. Java greift für diese Komprimierung auf Resourcen des Betriebssystems zurück. Die App Engine stellt diese Mittel nicht bereit. Stattdessen existiert ein zusätzlicher Dienst zur Erzeugung und Manipulation von Bilddaten, der allerdings im Rahmen der Entwicklung nicht eingesetzt wurde.
Der erste Ansatz befasst sich mit der Umstrukturierung des Datensatzes als RGBA-Textur nach der Idee von [Ras09]. Aufgrund der Komplexität dieses Ansatzes wurde auf die vollständige Umsetzung zugunsten anderer Aspekte der Entwicklung allerdings verzichtet. Trotzdem soll an dieser

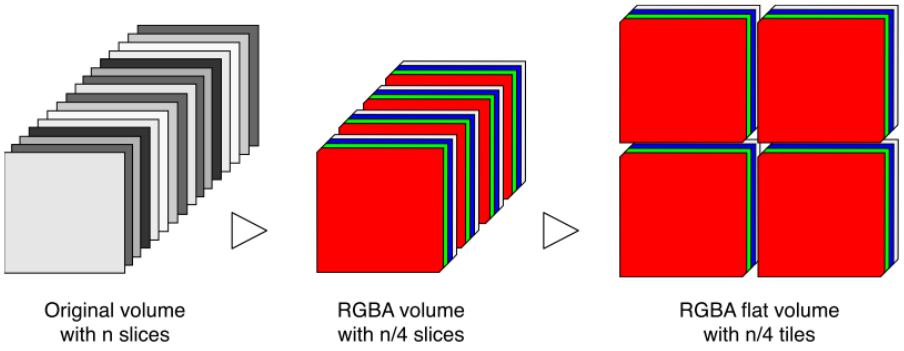

Abbildung 21: Das skalare Volumen wird in RGBA Kanäle gepackt und anschließend in eine zwei-dimensionale RGBA Textur umgeschichtet.

Parametername	Wert
Texturgröße (x,y)	(768,768)
Anzahl der RGBA-Schnittbilder (x,y)	(3,3)
Anzahl der Teil-Texturen (x,y)	(3,3)

Tabelle 1: Parameterangaben für den Texturzugriff

Stelle dieser Versuch beschrieben werden, da ein möglicher Performance-Zuwachs bei der trilinearen Interpolation auf 2D-Texturdaten zu untersuchen wäre. In diesem Fall können mit einem Texturzugriff vier aufeinanderfolgende Werte aus den RGBA-Komponenten ausgelesen werden, zwischen denen direkt interpoliert werden kann.

Exkurs: Aufbereitung der Datensätze als RGBA-Flat3D-Textur Innerhalb einer RGBA-Flat3D-Textur repräsentiert jeder Kanal einen skalaren Wert von vier aufeinanderfolgenden Schnittbildern (siehe Abbildung 21). Für die spätere Übertragung des Datensatzes findet neben der RGBA-Umsortierung, eine Unterteilung des Datensatzes in einzelne Teil-Texturen statt. Diese Teil-Texturen entsprechen jeweils einer festgelegten Größe von maximal zwei Megabyte, um in kurzer Zeit eine visuelle Rückmeldung zu erhalten. Im weiteren Verlauf wird, in Abhängigkeit von der Größe des Volumens, die Anzahl der notwendigen Teil-Texturen und deren Größe berechnet. Zusätzlich wird festgehalten, wieviele Schnittbilder jede Teil-Textur insgesamt beinhaltet und wie diese angeordnet sind. Mit Hilfe dieser Angaben kann von der 3D-Position im Volumen der entsprechende Wert in der 2D-RGBA-Textur ausgelesen werden. Ein exemplarisches Ergebnis dieser Umformung ist in Abbildung 22 abgebildet.

Wie oben beschrieben wird aufgrund der Komplexität dieser Ansatz nicht weiter berücksichtigt. Stattdessen folgt die Erläuterung der Bereit-

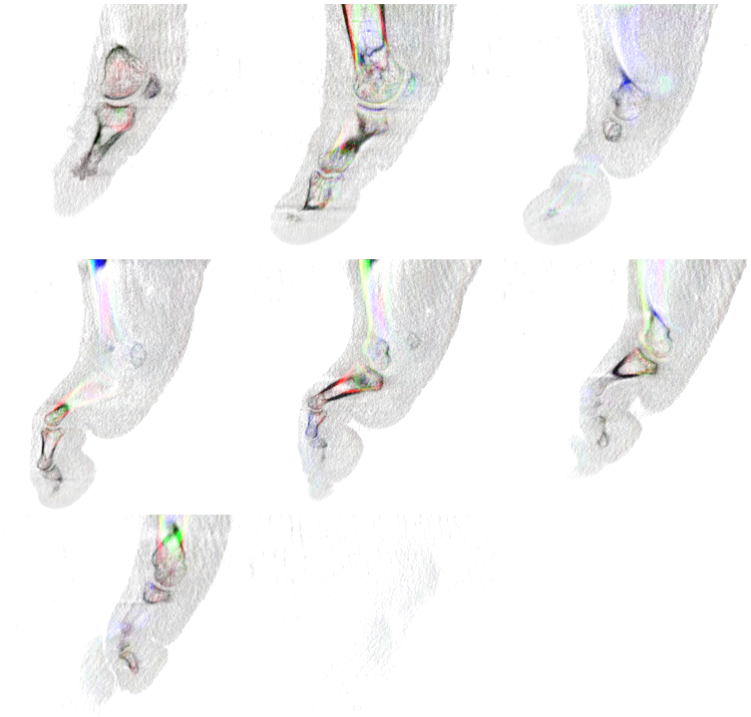

Abbildung 22: Teil-Textur nach der RGBA-Flat3D-Umstrukturierung.

stellung der Volumendaten ohne vorherige Umsortierung und Komprimierung.

Upload eines Datensatzes Sofern der Anwender die notwendigen Rechte besitzt, kann dieser über das Benutzer-Interface mittels einer Upload-Funktion Datensätze in den Blobstore laden. Die Datensätze müssen im RAW-Format vorliegen. Zusätzlich werden über Textfelder die notwendigen Parameter für die Dateninterpretation festgelegt. Diese umfassen die Angabe der Dimension und der Voxelabstände in alle Raumrichtungen sowie der Bit-Tiefe pro Voxel. Außerdem muss der Nutzer einen Volumennamen festlegen, der für die Identifizierung verwendet wird. Die Größe der Datensätze darf für den Upload die 2 Gigabyte Grenze nicht überschreiten. Diese Einschränkung besteht aufgrund von Festlegungen der App Engine. Sobald der Datensatz vollständig in den Blobstore geladen wurde, wird die Ausführung an das *RawDatasetUpload*-Servlet weitergeleitet. Dieser Service hat mit Hilfe von Objectify Zugriff auf den Datenspeicher, um ein Volumendaten-Objekt mit den notwendigen Parametern und der Referenz auf den Daten-Blob zu sichern.

4.2.4 Inkrementelles Laden der Texturdaten

Die Namen der zur Verfügung stehenden Volumendatensätze werden beim Aufruf der Anwendung geladen und als Kontrollelemente für den Benutzer dargestellt. Sobald eine Auswahl getroffen wurde, wird der Visualisierungsprozess gestartet. Zunächst wird hierzu die Entität des selektierten Datensatzes vom Server angefordert. Das entsprechende Datenobjekt wird dazu vom DatasetProvider-Service aus dem Datenspeicher ausgelesen. Dieses Datenobjekt enthält neben dem Blob-Key alle notwendigen Beschreibungen des Datensatzes, jedoch noch nicht die Binärdaten. Der Client erhält dieses Datenobjekt als Antwort auf seine Anfrage. Mit Hilfe dieser Angaben wird die notwendige Größe der zweidimensionalen Flat3D-Textur berechnet, so dass diese den gesamten Datensatz beinhalten kann. Anschließend wird der WebGL-Kontext initialisiert und eine leere Textur mit der zuvor berechneten Größe erzeugt. Die Parameter des Datensatzes und die Anzahl der Schnittbilder in x und y Richtung werden über Uniform-Variablen im aktiven Shaderprogramm gesetzt. Es folgt das Rendern der Pseudo-Geometrie und der Einsatz des Raycasting-Programms. Zeitgleich zum Rendervorgang, werden über asynchrone Remote-Procedure-Calls die Bytedaten des Datensatzes angefordert. Dazu wird der Service-Methode ein Index mitgegeben. Daran erkennt die Methode, welcher Byte-Abschnitt des Datensatzes angefordert werden soll, der einem Schnittbild in z-Dimension entspricht. Die Antwort des Services ist demnach ein Byte-Array, das als WebGLArray interpretiert und als Textur geladen werden kann. Es wird eine Teil-Textur erstellt. Die Größe dieser Textur entspricht

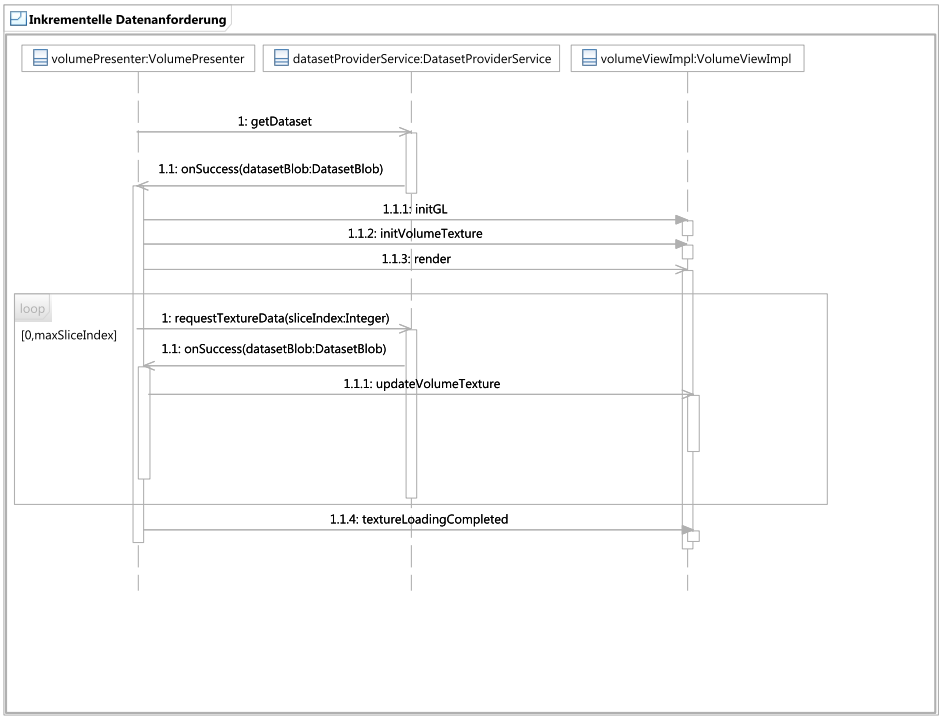

Abbildung 23: Sequenzdiagramm zur Beschreibung des inkrementellen Ablaufs der Datenanforderung bei gleichzeitiger Visualisierung.

der Dimension des Volumens in x- und y-Richtung. Der aktuelle Index bestimmt die Position der Teil-Textur in der zuvor erstellten leeren Textur. Der parallel laufende Renderprozess kann die eingetroffenen Daten so direkt visualisieren.

Das Übertragen der gesamten, oft sehr großen Datenmengen eines Volumendatensatzes ist sehr zeitaufwändig. Die Daten werden daher sequentiell übertragen und mit einer direkten interaktiven Darstellung verbunden. Um zusätzlich einen räumlichen Eindruck der Darstellung zu erhalten, können mehrere Schnittbilder so sortiert übertragen werden, dass eine Visualisierung in gleichmäßigen Abständen erfolgen kann.
Abbildung 24 zeigt die Visualiserung dieses Prozesses. Sobald das Rendern beginnt, werden die Texturen sequentiell vom Server angefordert. Mit Hilfe der zuvor geladenen Texturinformationen kann eine entsprechend große Textur erzeugt werden, die sukzessiv mit den Teiltexturen gefüllt wird. Des Weiteren hat das Fragmentshader-Programm, welches den Raycasting-Algorithmus durchführt, Zugriff auf diese Textur-Parameter, um bei der Abtastung den Wert der 3D-Position aus der 2D-Textur auszulesen.

Abbildung 24: Inkrementelles Laden und direktes Visualisieren eines Volumendatensatzes

Histogramm Sofern dem Datensatz auf Serverseite noch kein Histogramm vorliegt, wird dieses bei der Abfrage der Teildaten erstellt. Sobald der Datensatz vollständig zum Client übertragen wurde, speichert der Service das Histogramm als Datenobjekt ab und trägt eine Verknüpfung in der Datensatz-Entität ein. Bei der erneuten Abfrage des Datensatzes wird das Histogramm nicht erneut berechnet, sondern kann direkt aus dem Datenspeicher geladen werden.

4.2.5 Integration einer interaktiv konfigurierbaren Transferfunktion

Sobald das Rendering des Datensatzes aktiv ist, kann eine 1D-Transferfunktion in das Raycasting-Verfahren integriert werden. Für die interaktive Konfiguration der Transferfunktion wird ein Editor eingesetzt. Über diesen Editor ist es möglich, über Mausinteraktionen Knotenpunkte zu setzen, denen eine Farbe zugewiesen wird. Die Position des Knotens auf der x-Achse bestimmt den skalaren Wert. Die Opazität wird über die y-Position festgelegt. Zur Orientierung der skalaren Verteilung wird der Editor mit Hilfe des Histogramms visualisiert.

Nachdem eine Benutzerinteraktion erfolgt ist, wird die Transferfunktion neu berechnet. Das heißt, dass zwischen den Knotenpunkten die Farbe und die Opazität interpoliert werden. Das Ergebnis wird als ein-dimensionale RGBA-Textur in den WebGL-Kontext geladen. Der Raycasting-Shader führt dann bei der Evaluation des Volumenrendering-Integrals für die Abtastungsstellen einen Lookup auf dieser Textur aus. Der aktuelle Sampling-Wert entspricht dabei der Texturposition. Das Ergebnis des Lookups, die Farbe und die Opazität fließen anschließend in die weitere Berechnung ein. Der Transferfunktions-Editor wurde mit Hilfe der externen GWT-Bibliothek *gwt-links* [7] umgesetzt. Dieses quelloffene GWT-Modul ermöglicht das Verknüpfen und Drag-and-Drop von Widgets. Aus diesem Grund eignet es sich für die Erstellung eines interaktiven Editors. Für die Farbdefinition der einzelnen Knoten wurde ein Colorpicker-Widget [8] eingesetzt.

Authorisierung zum Upload von Datensätzen In das System wurde die Benutzerverwaltung von Google integriert. Anwender können sich mit ihrem Google-Account anmelden. Mit Hilfe eines Services kann so festgestellt werden, welcher Benutzer zur Zeit mit der Anwendung kommuniziert und welche Rechte dieser besitzt. Über ein Web-Interface der Applikation können Benutzer registriert werden, um diesen erweiterte Funktionalität zur Verfügung zu stellen. So ist es zur Zeit beispielsweise nur administrativen Anwendern erlaubt neue Datensätze in den Blobstore zu laden. Des Weiteren wurde die Benutzerverwaltung im Hinblick auf die Verwen-

[7] http://code.google.com/p/gwt-links/
[8] http://code.google.com/p/auroris/

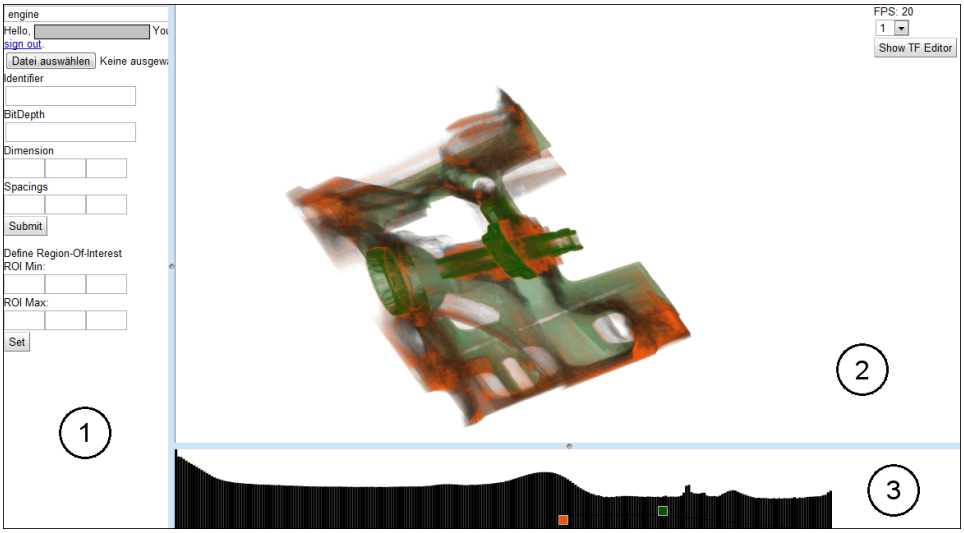

Abbildung 25: Screenshot der Benutzeroberfläche. 1: Auswahl und Upload von Datensätzen. Definition der ROI. 2: WebGL-Canvas Element zur direkten Volumenvisualisierung. 3: Histogramm und Editor zur Konfiguration einer eindimensionalen Transferfunktion.

dung der Channel-API zur möglichen Umsetzung einer kollaborativen Applikation realisiert.

4.2.6 Aufbau der Applikation und Beschreibung der Benutzeroberfläche

In diesem Abschnitt soll der Aufbau der Applikation unter Berücksichtigung des Model-View-Presenter Entwurfsmusters beschrieben werden. In diesem Kontext erfolgt auch eine Beschreibung der Benutzeroberfläche. In Bezug auf das Model-View-Presenter Entwurfsmuster wurde für jeden Teilbereich der Anwendung eine eigene Ansicht mit zugrundeliegendem Modell und Präsentator entwickelt. So konnte eine getrennte Entwicklung der einzelnen Bereiche realisiert werden. In Abbildung 25 ist das Benutzer-Interface abgebildet.

Objekt-orientierte Struktur nach dem MVP Beim Start der Applikation wird zunächst über die *onModuleLoad()*-Methode der Application-Controller initialisiert. Zudem wird mit Hilfe des UIBinders ein Layout (Template) erzeugt. Der AppController verwaltet dieses Template und erhält die Referenzen zu den beteiligten Services. Darunter fallen der DatasetProvider- und der Authorization-Service, sowie die beiden Upload-Servlets für das Bereistellen von RAW-Datensätzen beziehungsweise komprimierter Daten in einem eigenen Zip-Format. Zur Veranschaulichung dieses Aufbaus dient

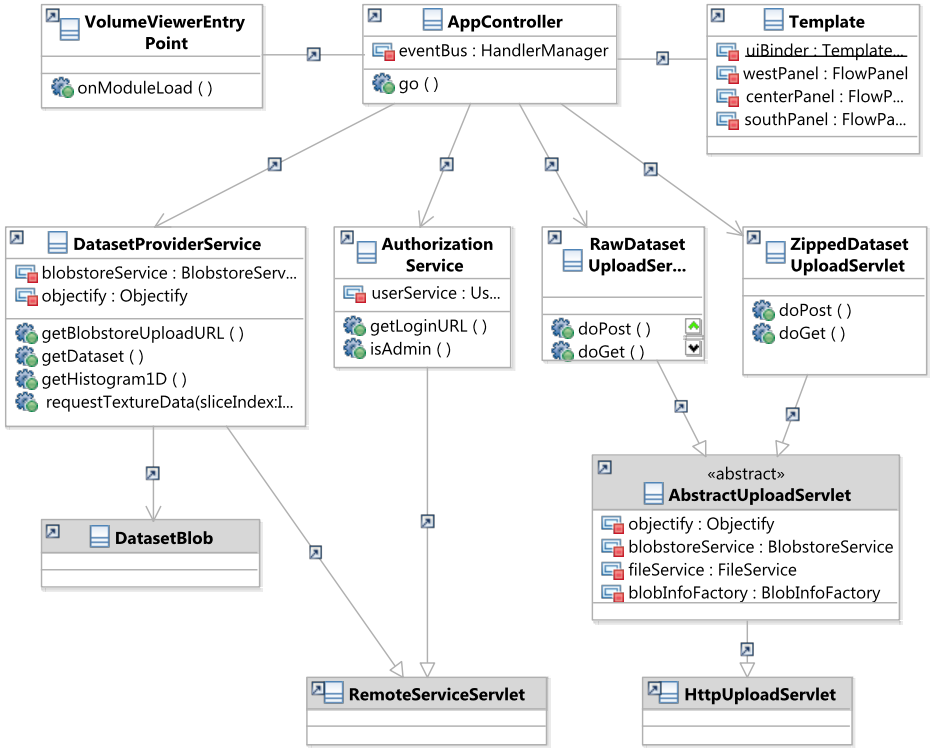

Abbildung 26: Klassendiagramm für den Aufbau der Applikation

das Klassendiagramm in Abbildung 26.

Für die Visualisierungs-Ansicht soll an dieser Stelle exemplarisch der Einsatz des MVP aufgezeigt werden. Der Application-Controller aktiviert den VolumePresenter über die allgemeine Presenter-Schnittstelle. Der VolumePresenter steuert die Kommunikation mit dem DatasetProvider-Service und hat über die Schnittstelle VolumeView Zugriff auf die Methoden zur Manipulation der Ansicht. Diese umfassen unter anderem Befehle für das Rendern, die Erstellung und Aktualisierung der Volumen-Textur sowie das Erstellen der LookUp-Tabelle für die Transferfunktion. Die Implementierung der Ansicht (VolumeViewImpl) besitzt einen WebGL-Kontext, mit dessen Hilfe hardware-beschleunigte 3D-Grafik angezeigt werden kann. Der Volume-Presenter wird durch ein Interface über die Aktualisierung von Parametern informiert und leitet die entsprechende Weiterverarbeitung ein. In Abbildung 27 ist die MVP-Umsetzung für Volumen-Ansicht dargestellt.

Die anderen Elemente der Applikation sind nach dem selben Schema erstellt und sollen daher nicht näher erläutert werden.

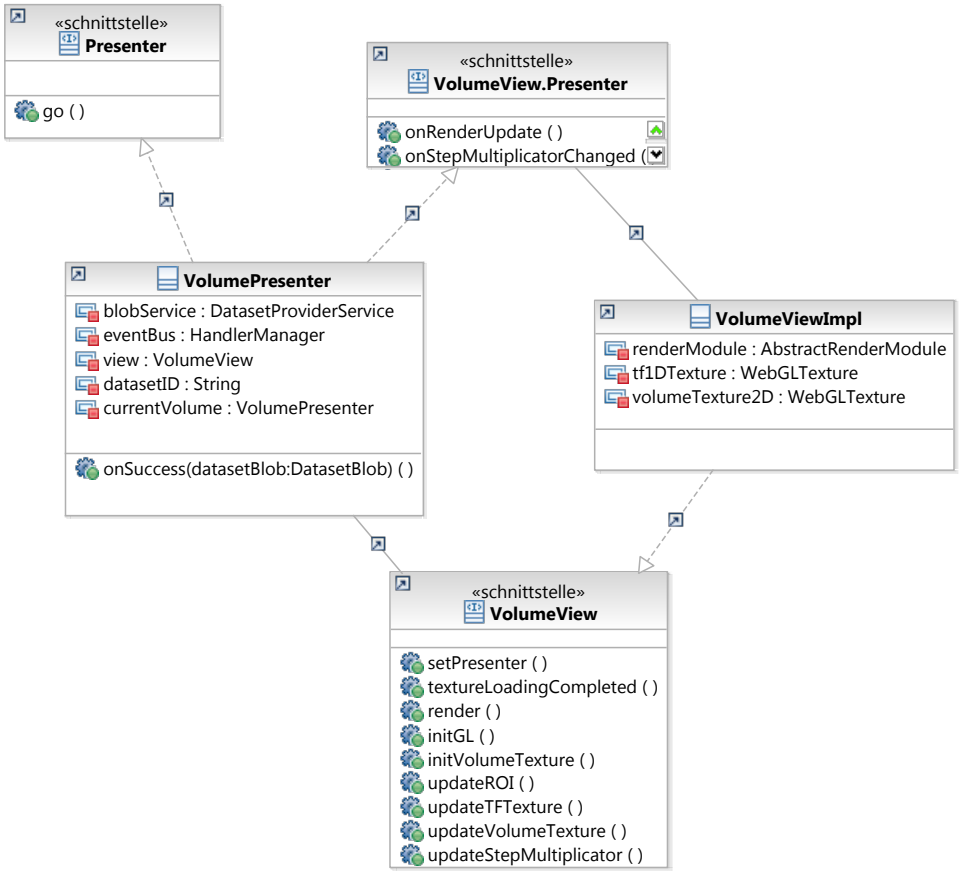

Abbildung 27: Klassendiagramm der MVP-Ausprägung für die Volumen-Ansicht

5 Ergebnisse

Das vorgestellte System wurde bezüglich der Effizienz des implementierten Raycasting-Algorithmus im WebGL-Kontext untersucht. Außerdem wurde die visuelle Qualität der Darstellung dazu in Bezug gesetzt. Des Weiteren soll der Speicherverbrauch der Applikation für unterschiedliche Browser gegenübergestellt werden.

5.1 Konfiguration des Testsystems

Hardware Die Applikation wurd auf einem System mit Pentium Dual-Core Prozessor T4200 mit 2x2.00 GHz und 3GB Arbeitsspeicher getestet. Bei der Grafik-Hardware handelte es sich um eine ATI Mobility Radeon HD 4500 Series Grafikkarte mit 512 MB RAM dezidiertem GPU-Speicher und aktueller Treiber-Version.

Software Die Untersuchungen wurden auf einem Testrechner mit Windows 7 Home Premium 64-Bit Betriebssystem ausgeführt. Die Tests wurden auf folgenden Webbrowsern mit WebGL-Unterstützung durchgeführt:

- Google Chrome (16.0.912.63)
- Mozilla Firefox (8.0.1)
- Opera Next (12.00 alpha)

Die Applikation ist aufgrund der Komplexität des Raycasting-Shaders nur mit nativer OpenGL Unterstützung lauffähig. Daher wurden entsprechende Konfigurationen wie in 4.1.8 beschrieben vorgenommen. Dazu muss der Google Chrome mit dem Paramterzusatz '–use-gl=desktop' gestartet werden. Der Mozilla Firefox erlaubt diese Konfiguration über das 'about:config'-Menü. Hier muss 'webgl.prefer-native-gl' auf 'true' gesetzt werden. Der Opera Next benötigt diese Einstellung nicht, da er standardmäßig den OpenGL-Treiber unterstützt.

5.2 Auswahl der medizinischen Datensätze für die Messung

Für die Auswertung wurden die Datensätze mit unterschiedlicher Größe und Bit-Tiefe ausgewählt um die Ausführungsgeschwindigkeit der Umsetzung hinreichend zu überprüfen. Neben einem 8Bit-Volumendatensatz von geringer Größe (Engine) wurde das System mit einem verhältnismäßg größeren Datensatz (Stent8) getestet. Letzterer konnte auch als 16-Bit Variante untersucht werden. Tabelle 2 veranschaulicht die unterschiedlichen Konfigurationen.

Datensatz	Dimension (x, y, z)	Bit-Tiefe	Größe (KB)
Engine	(256, 256, 128)	8	8.192
Stent8	(512, 512, 174)	8	44.544
Stent16	(512, 512, 174)	16	89.088

Tabelle 2: Auswahl der Datensätze für die Auswertung

5.3 Auswertung der Messungen

Die Ausführungsgeschwindigkeit wurde hinsichtlich der Bildwiederholfrequenz bei Interaktion ausgewertet. Im Rahmen der Untersuchung wurde leider festgestellt, dass die exakte Auswertung nur in Bezug auf den Google Chrome Webbrowser möglich ist. Die Messungen wurden mit Hilfe der Benchmark Software FRAPS [9] durchgeführt. Das Tool ist jedoch im Zusammenspiel mit dem Mozilla Firefox nicht funktionsfähig. Die Anzeige im Browser selbst konnte ebenfalls nicht verlässlich umgesetzt werden. Der Opera Webbrowser mit WebGL-Unterstützung liegt aktuell lediglich in einer Alpha-Version vor. Aufgrund dessen kommt es zu einer fehlerhaften Darstellung und eine Auswertung ist daher nicht aussagekräftig. Aus diesem Grund kann die Bewertung der Applikation bezüglich der Framerate nur im Hinblick auf den Google Chrome stattfinden. In Tabelle 3 sind die entsprechenden Ergebnisse dargestellt.
Gemessen wurde für die drei Testdatensätze jeweils die minimale, maximale und durchschnittliche Bildwiederholrate in einem Zeitintervall von 60 Sekunden. Während der Messung wurde die Visualisierung durchgehend rotiert um ein aussagekräftiges Ergebnis bezüglich der Interaktion zu erhalten. Die Messungen wurden unabhängig voneinander und jeweils mit einer unterschiedlichen Schrittweite durchgeführt.
Die Ergebnisse zeigen einen deutlichen Performance-Unterschied zwischen 8-Bit- und 16-Bit-Datensätzen. Dies kann vermutlich damit begründet werden, dass die Auflösung der Werte feiner ist und dadurch das Abbruchkriterium später erreicht wird. Außerdem muss eine zusätzliche Umrechnung bei der Abtastung von zwei 8-Bit-Komponenten zu einem 16-Bit-Wert zwischengeschaltet werden. Der 16-Bit Datensatz ermöglicht keine zufriedenstellende Interaktivität aufgrund der geringe Bildwiederholfrequenz bei kleiner Abtastungsrate. Erst bei einer Multiplikation der Schrittweite auf das Dreifache ist das Ergebnis mit einer durchschnittliche Rate von ungefähr 12 FPS für die medizinische Visualisierung akzeptabel. Bei einer fünffachen Schrittweite ist eine relativ flüssige Darstellung bei durchschnittlich 20 FPS möglich. Die Erhöhung der Schrittweite hat allerdings einen enormen Qualitätsverlust der Visualisierung zur Folge.
Die beiden 8-Bit Datensätze sind hingegen auch bei hoher visueller Qua-

[9] www.fraps.com

Schrittweite	**Engine**			**Stent8**			**Stent16**		
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
x1	27	31	29.8	13	31	20.5	3	7	4.8
x2	26	61	50.8	21	31	29.1	6	11	8.9
x3	57	61	59.9	29	49	30.9	9	15	12.2
x5	59	61	60.0	43	61	53.5	16	16	20.2

Tabelle 3: Messergebnisse in Bilder pro Sekunde innerhalb eines Zeitintervalls von 60 Sekunden für den Google Chrome Browser

lität interaktiv explorierbar. Die visuellen Unterschiede bei Änderung der Schrittweite sind in Abbildung 28 zu erkennen.
Auch wenn keine Messdaten für den Mozilla Firefox erstellt werden konnten, soll an dieser Stelle erwähnt werden, dass der subjektive Eindruck hinsichtlich der Ausführungsgeschwindkeit mit der des Google Chrome vergleichbar ist.

6 Diskussion

Abschließend werden die Ergebnisse bewertet und ein persönliches Fazit gezogen. Zuletzt werden weitere Entwicklungsmöglichkeiten und relevante Untersuchungsaspkete in Ausblick gestellt.

6.1 Fazit

Zunächst einmal kann festgehalten werden, dass die webbasierte und GPU-unterstützte medizinische Visualisierung durch die neuen Web-Technologien HTML5 und WebGL realisierbar ist. Die Umsetzung liefert ein Framework für den Ansatz der client-seitigen direkten Volumenvisualisierung im Webbrowser. Mit Hilfe der Google App Engine konnte ein Beispiel für eine Client-Server Webapplikation realisiert werden, die externe medizinische Bilddaten über die HTTP-Verbindung bereitstellt und diese clientseitig direkt mit Hilfe von Volumen-Raycasting darstellt. In Anbetracht der Größe volumetrischer Daten ist die Übertragung ein zeitlich kostenspieliger Prozess. Für die unmittelbare Visualisierung wurde demnach ein inkrementelles Konzept entwickelt um aktuell eingetroffene Daten direkt darzustellen. Hierzu wurden ein verteilter und sequentieller Aufbau der Visualisierung umgesetzt. Die Daten können mit Hilfe eines Transferfunktion-Editors interaktiv exploriert werden.
Die Auswertung wurde auf einem System mit durchschnittlicher Leistung durchgeführt. Da in Bezug auf die Systemkonfiguration eine, im Rahmen der medizinischen Visualisierung, interaktiv bedienbare Anwendung rea-

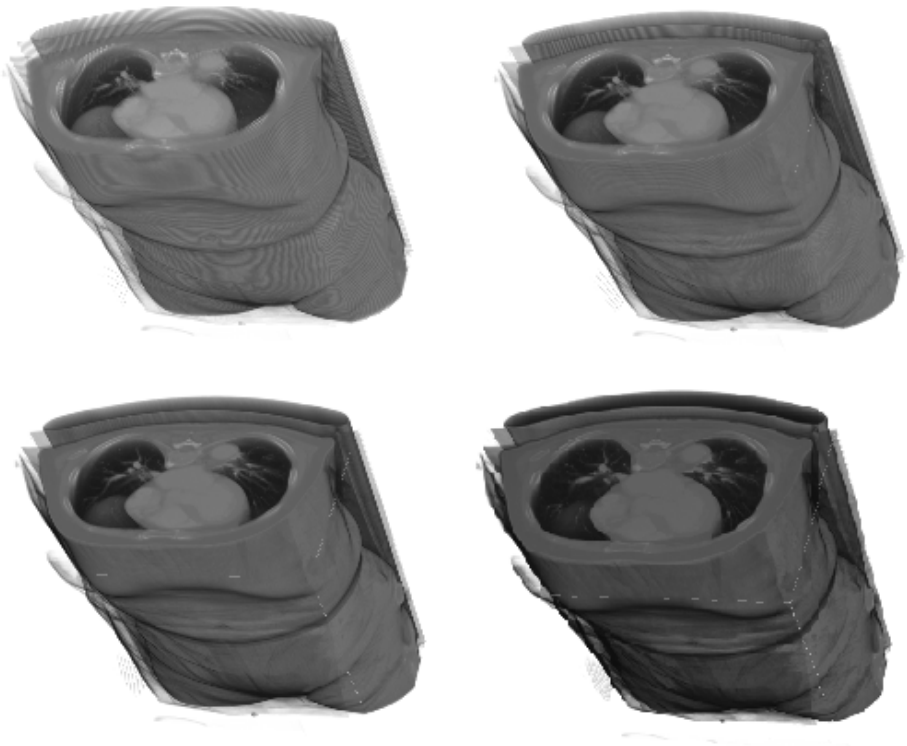

Abbildung 28: Darstellung des Volumen-Raycastings mit unterschiedlicher Schrittweite. Von links oben nach rechts untern: Einfache, zweifache, dreifache und fünffache Schrittweite

lisiert werden konnte, hat sich der Einsatz der Methode als sinnvoll erwiesen.

Die Arbeit liefert somit eine Grundlage für weitere Untersuchungen in Bezug auf WebGL und die dreidimensionale medizinische Visualisierung.

6.2 Ausblick

Zum Abschluss sollen mögliche Weiterentwicklungen und zusätzliche Untersuchungsfelder vorgestellt werden.
Da sich WebGL mit der Version 1.0 noch in einer sehr früher Phase befindet und für geringe grafische Hardware-Voraussetzungen konzipiert ist, könnte in Zukunft die Umsetzung mit einfacheren technischen Mitteln realisierbar sein. An dieser Stelle soll nochmals auf die fehlende Unterstützung von 3D-Texturen verwiesen werden, deren Einsatz in der GPU-unterstützten dreidimensionalen medizinischen Visualisierung ein Basiselement darstellt. Außerdem erschweren die fehlenden Floating-Point-Texturen die Visualisierung von Datensätzen mit hoher Auflösung. Mit Hilfe von zusätzlichen Erweiterungen kann dieses Feature jedoch eingesetzt werden. Eine Untersuchung wäre demnach sinnvoll.
Des Weiteren wäre von Interesse, wie die Verwaltung und Darstellung von Datensätzen realisiert werden kann, die aufgrund ihrer Größe die Limitierungen des Browser-Caches oder des Grafikkartenspeichers überschreiten. Aufgrund fehlender Testdatensätze in dieser Größenordnung konnte dies im Rahmen der Arbeit nicht untersucht werden.
Eine zusätzliche Motivation dieser Arbeit war die Idee einer kollaborativen Visualisierungsanwendung. Die Google App Engine liefert mit der Channel API eine geeigente Technik dies umzusetzen. Nebenläufig zu dieser Arbeit wurde eine Testanwendung entwickelt, deren Integration in das System aber noch aussteht. Es ist somit vorstellbar, dass mehrer Benutzer an einem Visualsierungsprojekt, ohne den Einsatz zusätzlicher Software, gleichzeitig teilnehmen und mit der Darstellung in Echtzeit interagieren können.
Zudem ist eine Verknüpfung mit einem PACS-System vorstellbar. Dazu müssten aber andere Server und Technologien als die Google App Engine eingesetzt werden, da hier die Sicherheit der Daten und die Privatsphäre der Patienten nicht mehr überschaubar wäre. DICOM ist das gebräuchliche Format für medizinische Daten. Die Integration eines serverseitigen DICOM-Parsers wäre für die Bereitstellung dieser Datensätze notwendig.
Hinsichtlich der Effizienz des Raycasting-Algorithmus kann dieser durch Beschleunigungstechniken wie das Empty Space Skipping und adaptives Sampling weiterentwickelt werden. Die Umsetzung dieser Methoden mit WebGL ist daher zu überprüfen.
Des Weiteren erfordert die Erweiterung der Visualisierung durch zusätzliche optische Eigenschaften, wie die lokale und globale Beleuchtung eine

Vorverarbeitung der Volumendaten. Ebenso setzt der Einsatz einer mehrdimensionalen Transferfunktion beispielsweise die Berechnung der Gradienten voraus. Aus diesem Grund ist eine serverseitige Manipulation der Bilddaten sowie die Generierung zusätzlicher Informationen denkbar. Außerdem können serverseitig Bibliotheken für die Analyse und Segmentierung zum Einsatz kommen. Ein weiterer Ansatz wäre, diese Berechnungen mit Hilfe von WebGL beim Client durchzuführen. Hierzu müssen Methoden aus der GPGPU-Forschung integriert werden.
Von Interesse ist auch die Darstellung von geometrischen Objekten für die multimodale Visualisierung. Segmentierte Daten oder die Geometrie von extrahierten Nervenfasern aus der DTI-Visualisierung können als komplexe Polygonnetze vom Server angefordert werden und in das direkte Volumenrendering integriert werden.
Zuletzt soll die Möglichkeit einer interaktiven Applikation für anatomische Lernzwecke in Aussicht gestellt werden. Segmentierte Strukturen könnten dazu beispielsweise über Tags mit externen Webinhalten zu weiterführenden Informationen vernetzt werden.
Das Forschungsfeld der webbasierten und GPU-unterstützten medizinischen Visualisierung wurde durch WebGL neu motiviert. Es ist von großem Interesse, Ansätze und etablierte Methoden aus der dreidimensionalen medizinschen Visualisierung im Web-Kontext zu untersuchen.

Literatur

[BA01] Johannes Behr and Marc Alexa. Volume visualization in vrml. In *Proceedings of the sixth international conference on 3D Web technology*, Web3D '01, pages 23–27, New York, NY, USA, 2001. ACM.

[BEJZ09] Johannes Behr, Peter Eschler, Yvonne Jung, and Michael Zöllner. X3dom: a dom-based html5/x3d integration model. In *Proceedings of the 14th International Conference on 3D Web Technology*, Web3D '09, pages 127–135, New York, NY, USA, 2009. ACM.

[BM07] B. Blazona and Z. Mihajlovic. Visualization service based on web services. *Journal of Computing and Information Technology*, 15(4):339–345, 2007.

[BW00] Ken Brodlie and Jason Wood. Volume graphics and the internet, 2000.

[CKY00] M. Chen, A. Kaufman, and R. Yagel. *Volume graphics*. Springer, 2000.

[CRP11] Cantor-Rivera and Peters. Pervasive medical imaging applications - current challenges and possible alternatives -, November 2011. `http://www.imaging.robarts.ca/\%7Edcantor/wp-content/uploads/2010/07/eHealth.pdf`.

[CSK+11] John Congote, Alvaro Segura, Luis Kabongo, Aitor Moreno, Jorge Posada, and Oscar Ruiz. Interactive visualization of volumetric data with webgl in real-time. In *Proceedings of the 16th International Conference on 3D Web Technology*, Web3D '11, pages 137–146, New York, NY, USA, 2011. ACM.

[DN10] Dlc00-NJITWILL. Pacs workflow diagram, June 2010. `http://upload.wikimedia.org/wikipedia/commons/2/28/Workflow_diagram.png`.

[EWE99] K. Engel, R. Westermann, and T. Ertl. Isosurface extraction techniques for web-based volume visualization. In *Visualization'99. Proceedings*, pages 139–519. IEEE, 1999.

[Goo11a] Google. Google app engine developer's guide, November 2011. `http://code.google.com/intl/en/appengine/docs/`.

[Goo11b] Google. Google web toolkit developer's guide, November 2011. http://code.google.com/intl/de-DE/webtoolkit/doc/latest/DevGuide.html.

[Goo11c] Google. Java components of the gwt rpc mechanism, November 2011. http://code.google.com/intl/de-DE/webtoolkit/doc/latest/tutorial/images/AnatomyOfServices.png.

[HKRs+06] Markus Hadwiger, Joe M. Kniss, Christof Rezk-salama, Daniel Weiskopf, and Klaus Engel. *Real-time Volume Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 2006.

[HLSR09] Markus Hadwiger, Patric Ljung, Christof Rezk Salama, and Timo Ropinski. Advanced illumination techniques for gpu-based volume raycasting. In *ACM SIGGRAPH 2009 Courses*, SIGGRAPH '09, pages 2:1–2:166, New York, NY, USA, 2009. ACM.

[KW03] J. Kruger and R. Westermann. Acceleration techniques for gpu-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, pages 38–, Washington, DC, USA, 2003. IEEE Computer Society.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21:163–169, August 1987.

[PB07] Bernhard Preim and Dirk Bartz. *Visualization in Medicine: Theory, Algorithms, and Applications (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[Ram10] Chris Ramsdale. Large scale application development and mvp, 2010. http://code.google.com/intl/de-DE/webtoolkit/articles/mvp-architecture.html.

[Ras09] Matthias Raspe. *GPU-assisted Diagnosis and Visualization of Medical Volume Data*. PhD thesis, Universität Koblenz-Landau, Campus Koblenz, 2009.

[WLC10] H. Wei, E. Liu, and G. Clapworthy. Interactive 3d rendering to assist the processing of distributed medical data. In *Proceedings of the First International Conference on Intelligent Interactive Technologies and Multimedia*, pages 119–126. ACM, 2010.

Abbildungsverzeichnis