Hochschule Düsseldorf
Fachbereich Medien
B.Sc. Medieninformatik PO2010

Arbeit wurde gefertigt in:

antwerpes AG
Vogelsanger Straße 66
50823 Köln

Immersive medizinische Visualisierung - Auswertung und Darstellung von DICOM-Daten

von
Mustafa Talha Özdogan
April 2020

Prüfer
Erstprüfer: Prof. Dr. rer. nat. Christian Geiger
Zweitprüfer: M. Sc. Philipp Ladwig

Fachliche Betreuung
B. Sc. Alexander Kals

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und dabei keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht. Die Arbeit wurde bisher weder gesamt noch in Teilen einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Sperrvermerk

Die vorgelegte Bachelorarbeit mit dem Titel „*Immersive medizinische Visualisierung - Auswertung und Darstellung von DICOM-Daten*“ beinhaltet vertrauliche Informationen und Daten des Unternehmens **antwerpes AG**.

Die Bachelorarbeit darf nur vom Erst- und Zweitprüfer sowie berechtigten Mitgliedern des Prüfungsausschusses eingesehen werden. Eine Vervielfältigung und Veröffentlichung der Bachelorarbeit ist auch auszugsweise nicht erlaubt.

Dritten darf diese Arbeit nur mit der ausdrücklichen Genehmigung des Verfassers und Unternehmens zugänglich gemacht werden.

Mustafa Talha Özdogan
Wuppertal, den 6. April 2020

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Bachelorarbeit unterstützt und motiviert haben.

Zuerst gebührt mein Dank Prof. Dr. rer. nat. Christian Geiger für die Möglichkeit, diese Arbeit anfertigen zu können.

Zudem gilt mein Dank Philipp Ladwig als Zweitprüfer, der mir stets Denkanstöße für meine Arbeit gab.

Ein besonderer Dank gilt Alexander Kals, der meine Bachelorarbeit betreut und begutachtet hat.

Ich bedanke mich bei antwerpes AG für die finanzielle Unterstützung meiner Forschung.

Weiterhin möchte ich meiner Familie und meinen Bekannten danken, die mich immer unterstützten. Ganz besonders danke ich hierbei meiner Frau Elif Özdogan dafür, dass sie mir auch durch die schwersten Zeiten dieser Arbeit half.

Abstract

The DICOM format is a powerful protocol in medicine for the management of medical imagery in two dimensions. DICOM series are, in many use cases, represented by volume rendering in three dimensions to facilitate treatments, operations and diagnoses. Volume visualization has been applied in many problem areas and as such has become an important tool for data exploration and knowledge discovery. This bachelor thesis focuses on the extraction of volume data sets from DICOM series and the rendering of volumes in immersive representation in Unity. For this purpose, the GPU processed Volume Raycasting algorithm and the CPU the Marching Cubes algorithm.

Kurzfassung

Das DICOM Format ist in der Medizin ein mächtiges Protokoll, zur Verwaltung von medizinischen Bildern in der zwei dimensionalen Ebene. DICOM Serien werden in vielen Anwendungsfällen durch Volume Rendering drei dimensional dargestellt, um medizinische Behandlungen, Operationen und Diagnosen zu erleichtern. Die Volumen-Visualisierung wurde in vielen Problembereichen angewandt und ist als solches zu einem wichtigen Werkzeug für die Erforschung von Daten und die Entdeckung von Wissen geworden. Die Bachelorarbeit konzentriert sich auf die Extraktion der Volumendatensatzes aus DICOM Serien und das Rendern der Volumina in der immersiven Darstellung in Unity. Für das Volume Rendering werden der Volume Raycasting Algorithmus auf der GPU und der Marching Cubes Algorithmus auf der CPU verwendet.

Inhaltsverzeichnis

Abkürzungsverzeichnis

DICOM	-	Digital Imaging and Communications in Medicine
IE	-	Information Entity
SOP	-	Service-Object Pair
IOD	-	Information Object Definition
VTK	-	Visualization Toolkit
ITK	-	Insight Toolkit
(xxxx,yyyy)	-	DICOM Tag
HU	-	Hounsfield-Skala
CT	-	Computertomographie
MRT	-	Magnetresonanztomographie
MC	-	Marching Cubes
DVR	-	Direct Volume Rendering
MIP	-	Maximum Intensity Projection
VR	-	Virtual Reality
HMD	-	Head-Mounted-Display

Abbildungsverzeichnis

Tabellenverzeichnis

Codeverzeichnis

Kapitel 1: Einleitung

1.1 Motivation

Es gibt viele Möglichkeiten medizinische Daten in 3D umzuwandeln. Durch die Extrusion der medizinischen Daten (DICOM, RAW, INI) erhält man ein klinisch korrektes 3D Model von dem Patienten. Diagnosen, Behandlungen und Operationen werden somit für Ärzte erleichtert. Fremdkörper können in der 3D Visualisierung erkannt und lokalisiert werden. Mit dem Rendering oder Drucken der 3D Modelle vom Patienten können die Operationen mit realen Dimensionen simulieren werden. Es ist auch ein Echtzeit-Rendering in Operationen möglich, die dem Arzt helfen. [1]

Die 3D Visualisierung von DICOM Daten in der Medizin ist verbreitet. Für die 3D Darstellungen werden Techniken wie Volumengrafiken (Volume Rendering) oder eine Extraktion eines Polygon Meshes aus einer Isofläche (Isosurface Rendering) verwendet. Mit weiteren Algorithmen, wie die Segmentierung von gewünschten Bereichen am Körper, kann man detaillierte Diagnosen festlegen.

Für die Verwirklichung dieser Szenarien existieren viele Bibliotheken, Frameworks, frei und kommerzielle Software für das Volumen Rendering im medizinischen Bereich, die je nach Anwendungsfall verwendet werden. [2] [3] [4]

Die Darstellung der medizinischen Volumina in der virtuellen Realität würde eine verbesserte Wahrnehmung für den Betrachter zur Verfügung stellen.

1.2 Ziele

Diese Bachelorarbeit soll beantworten, welche Metadaten aus einer DICOM Serie benötigt werden, um Volume Rendering Verfahren, wie den Marching Cubes, anzuwenden. Das Ziel ist es, eine Erkenntnis über den Aufbau, das Auslesen und die Auswertung von DICOM-Daten zu erhalten. Dabei wird der Fokus nur auf das medizintechnische Standard Format DICOM gesetzt, weil es mit den Rohdaten zusammen auch Patienteninformationen anbietet.

Das Ziel dieser Arbeit ist es mit einer Bibliothek eine Reihe von DICOM Dateien auszulesen und eine eigene Implementierung des Volume Renderings auszuführen.

Diese Arbeit soll es ermöglichen, medizinische Bilder korrekt immersiv darzustellen, dynamische Änderungen durchzuführen und einen einfachen Import der Daten anbieten. Das Volume Rendering in der immersiven Darstellung wird eine Schwierigkeit darstellen. Performanceverluste müssen vermieden werden, um den Betrachter nicht zu stören.

Diese Arbeit zielt darauf hin, Computertomographie (CT) und Magnetresonanztomographie (MRT) DICOM Daten umzuwandeln. Für andere bildgebende Verfahren in der Medizin, wie Ultraschall, sind alternative Vorgehensweisen notwendig, da sich dort der DICOM Standard unterscheiden kann.

1.3 Projektstruktur

Zunächst sollen die DICOM Dateien ausgelesen und korrekt nach dem DICOM Standard verarbeitet werden. Dazu benötigt man einen 3D Datensatz aus einer DICOM Serie (eine Reihe von DICOM Dateien). Für das 3D Rendering werden der Marching Cubes Algorithmus und Volume Raycast Algorithmus verwendet.

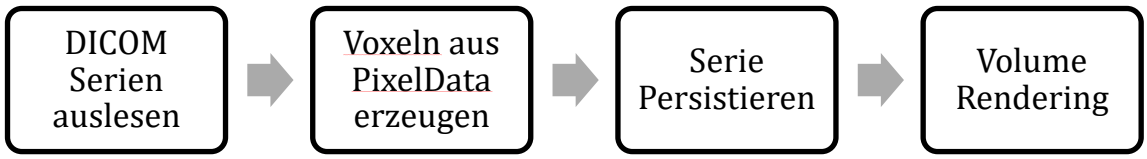


Abbildung a: Projekt Struktur Verlauf

1.4 DICOM Format

Digital Imaging and Communications in Medicine (DICOM) ist der Internationale Standard (ISO 12052) für die Kommunikation und Verwaltung von medizinischen Bildgebung Informationen in Kliniken und wird seit 1980 entwickelt. DICOM bildet den Bilderfassungsprozess und die mit der bildgebenden Diagnostik verbundenen Informationsobjekte. Sie bietet eine Spezifikation dafür, wie Bilddaten, Metadaten und zugehörige Informationsobjekte in einem Binärformat dargestellt und über Computernetzwerke übertragen werden. [5]

1.4.1 Vorteile in der Medizin

Das Hauptziel des DICOM-Standards ist es, eine Interoperabilität sicherzustellen zwischen medizinischen Geräten und Informationssystemen, die mit digitalen medizinischen Bildern und Informationen arbeiten. Fast alle medizinischen Geräte wie CT, MRT oder Ultraschall implementieren den DICOM Standard. Sie erzeugen DICOM-Daten und stellen die Kompatibilität von Systemen sicher. Die Systeme sind zuständig für das Produzieren, Speichern, Anzeigen, Senden, Abfragen, Verarbeiten, Abrufen und Drucken von medizinischen Bildern und Dokumenten.

Die Interoperabilität wird durch das *Patient Root Query/Retrieve Information Model* unterstützt. DICOM Daten werden bei Erzeugung dem Patienten eindeutig zugeordnet. Das Entitäts-Beziehungsdiagramm veranschaulicht das DICOM-Realitätsmodell, bei dem ein Patient eine oder mehrere Studien hat und jede Studie eine oder mehrere Serien enthält. Die Serie dient als ein Container, der als gar keine oder mehrere Objekte wie Wellenformen, Bilder, Rohdaten und Präsentationszustände aggregiert. [6]

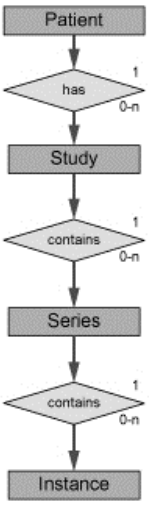


Abbildung b: Das Patient Root Query/Retrieve Information Model E/R Diagramm [6]

1.4.2 Grundlagen des DICOM Formats

DICOM ist ein komplettes Protokoll, das ständig weiterentwickelt wird und sehr gut dokumentiert ist. [7] Das DICOM Format enthält u.a. Bildinformationen, Patienteninformationen und Geräteinformationen. DICOM besteht aus zwei Hauptkomponenten: Dem DICOM File Format und DICOM Network Protocol. Die DICOM File Format Komponente ist der relevante Teil für diese Bachelorarbeit.

Alle medizinischen Bilder werden als DICOM Dateien abgespeichert. DICOM Viewer werden genutzt, um diese Daten auszuwerten und im gewünschten Format anzuzeigen. Die Dokumentation des Standards ist in mehreren Kapiteln unterteilt und besteht aus über 3000 Seiten.

1.4.3 DICOM Objects und SOP Class Definition

Eine *Service-Object Pair* (SOP) -Klasse wird durch die Vereinigung einer *Information Object Definition* (IOD) und eines *DICOM Service Elements* (DIMSE) definiert. Die SOP-Klassendefinition enthält die Regeln und die Semantik, die die Verwendung der Dienste in der DIMSE-Dienstgruppe oder die Attribute des IOD einschränken können. [8] Populäre IODs sind die *CT Image IOD* und *MR Image IOD*.

Jede IOD ist in Pflicht Information Entities (IE) aufgeteilt die im längstem DICOM Standard 3 definiert ist. Jedes IE wird in Modulen unterteilt, welche mandatorisch, optional oder bedingt sind. Die Module sind im Standard 3 Anhang C definiert. [9]

Jedes DICOM Object muss eine definierte SOP Class Model aufweisen. Die Verschachtlung weist eine Baum Datenstruktur auf. Die *Patient* IE ist dabei die Wurzel. Daher kann ohne Patienteninformationen keine *Study*, *Serie*, *Equipment* und *Image* IE existieren. [10]

IE	Module	Reference	Usage
Patient	Patient	C.7.1.1	M
	Clinical Trial Subject	C.7.1.3	U
Study	General Study	C.7.2.1	M
	Patient Study	C.7.2.2	U
	Clinical Trial Study	C.7.2.3	U
Series	General Series	C.7.3.1	M
	Clinical Trial Series	C.7.3.2	U
Frame of Reference	Frame of Reference	C.7.4.1	M
Equipment	General Equipment	C.7.5.1	M
Image	General Image	C.7.6.1	M
	Image Plane	C.7.6.2	M
	Image Pixel	C.7.6.3	M
	Contrast/bolus	C.7.6.4	C - Required if contrast media was used in this image
	Device	C.7.6.12	U
	Specimen	C.7.6.22	U
	CT Image	C.8.2.1	M
	Overlay Plane	C.9.2	U
	VOI LUT	C.11.2	U
	SOP Common	C.12.1	M

Beispiel für ein CT Image IOD. Die erste Spalte enthält den Namen jeder Information Entity, die zweite Spalte den Namen eines DICOM-Moduls und die dritte Spalte den Abschnitt von Teil 3 des DICOM-Standards, in dem das Modul definiert ist. Die letzte Spalte gibt an, ob ein Modul mandatorisch (M), optional (U) oder bedingt (C) ist. [11]

Abbildung c: CT Image IOD [11]

Alle Module bestehen aus DICOM Elementen [12]. DICOM Elemente werden durch Attribute definiert: eindeutiger Tag (Tag), ein Datentyp (VR), einer Länge (Value Length), einem Wert (Value), einer Kodierung (VM) und einem Namen (Tag Name). Im unteren Beispiel erkennt man ein DICOM Bild mit 512 Zeilen.

DICOM Element Beispiel für Row: (0028,0010) US 512 # 2, Rows

Mehrere DICOM Elemente bilden einen DICOM Data Set Stream. DICOM Elemente sind der eigentliche Inhalt, aus den DICOM Daten.

Kapitel 2: Stand der Technik

2.1 DICOM Viewer

DICOM Dateien (.dcm) können nur mit spezieller Software gelesen und manipuliert werden. Häufig werden diese Software schlicht “DICOM Viewer” genannt. DICOM Viewer sollten professionell nutzbar und klinisch korrekt sein. Ganze Studies von Patienten und einzelne Serien werden gefiltert und angezeigt. Eine sehr Hilfreiche Funktion ist die tabellarische Anzeige von DICOM Elementen mit Suchfunktion.

Einige DICOM Viewer bieten das 3D Volume Rendering an, um Serien in Echtzeit anzuzeigen. Einer der bekanntesten DICOM Viewer sind RadiAnt und 3DSlicer. Für die Analysen der DICOM Daten und als Hilfestellung wurden in dieser Arbeit der RadiAntViewer und MicroDicomViewer genutzt. [13] [14]

2.1.1 Multiplanare Rekonstruktion von DICOM Images

Die multiplanare Rekonstruktion (MPR) ist die zweidimensionale Bildrekonstruktion aus denen transversale Ebenen/Schnitte berechnet werden. [15] Bei einem CT oder MR werden Serien in der Regel nur von einer Richtung gescannt. Um die anderen Querschnitte zu erhalten wird der Bildrekonstruktionsalgorithmus multiplanare Rekonstruktion angewendet. Die Querschnitte werden standardmäßig in die Coronalle (y-Achse), Axiale (z-Achse) und Sagittale (x-Achse) aufgeteilt. Die Funktion ist sehr hilfreich, um ein Verständnis für die Dimensionen im dreidimensionalen Raum in DICOM zu erhalten.

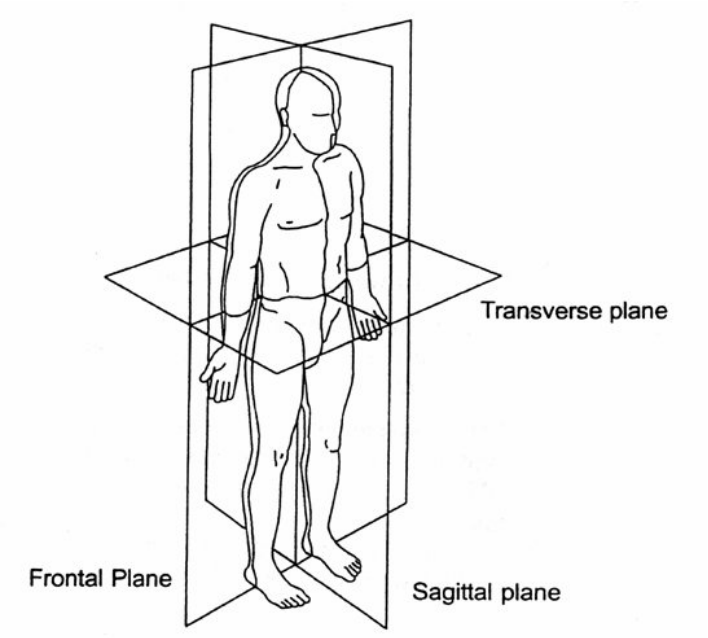


Abbildung d: Die Bezugsebenen des menschlichen Körpers in der anatomischen Standardposition (Vaughan et al., 1987)

2.2 Open Source Software Lösungen von Kitware

2.2.1 Visualization Toolkit

Das Visualization Toolkit (VTK) ist eine Open-Source-Software zur Bearbeitung und Anzeige wissenschaftlicher Daten. Es verfügt über modernste Tools für das 3D-Rendering, eine Reihe von Widgets für die 3D-Interaktion und umfangreiche 2D-Plotfunktionen. VTK ist in der Forschung und Entwicklung weit verbreitet und wird für Informationsvisualisierung, 3D-Computergrafiken, Modellierung, Bildverarbeitung, Volumenrendering und wissenschaftliche Visualisierung genutzt. Das Toolkit ist eine Plattform-agnostische Software und unterstützt Windows, Linux, Mac, Web und Mobiltelefone. Aktuell wird VTK von Kitware weiterentwickelt. Es unterstützt eine Vielzahl von Visualisierungsalgorithmen und fortschrittlichen Modellierungstechniken und nutzt die Parallelverarbeitung mit Thread- und verteiltem Speicher für Geschwindigkeit bzw. Skalierbarkeit. [2]

2.2.2 Insight Toolkit

Insight Toolkit (ITK) ist eine plattformübergreifende Open-Source-Bibliothek, die den Entwicklern ein umfangreiches Paket von Software-Tools für die Bildanalyse zur Verfügung stellt. Entwickelt durch ausgesprochenen Programmiermethoden, baut ITK auf einer bewährten, räumlich orientierten Architektur zur Verarbeitung, Segmentierung und Registrierung wissenschaftlicher Bilder in zwei, drei oder mehr Dimensionen auf. [3] ITK bietet Algorithmen und Funktionen für die Visualisierung von medizinischen Bildern an.

2.2.3 VTK und ITK

VTK und ITK bilden zusammen eine Kombination in Bezug auf allgemeines Visualisieren von drei Dimensionalen Anwendungen und medizinische Darstellung in 2D und 3D.

Für die schnelle Umwandlung von DICOM Dateien zu Volumen, sind diese open source Softwares VTK und ITK gut geeignet. Es gibt viele Visualisierung Tools für medizinische Bilder, jedoch hat sich VTK und ITK zum inoffiziellen Standard entwickelt und wird von vielen Bekannten Anwendungen wie 3D-Slicer, OsiriX und ParaView genutzt. VTK und ITK wird in der Regel in C++ und Python genutzt. Kann aber auch in Java und C# genutzt werden. [16] Die Einbindung in Unity ist via Plugin oder die ActiViz.NET/C# Erweiterung möglich.

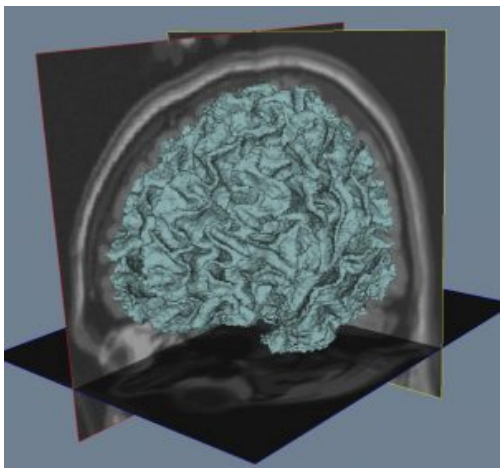


Abbildung e: ITK Volume Rendering: Multiplanare Rekonstruktion mit Volume Rendering [3]

Kapitel 3: Rendering mit DICOM Pixel Data

3.1 Einleitung in das allgemeine Volumen Rendering

In der wissenschaftlichen Visualisierung und Computergrafik ist das Volumen Rendering eine Reihe von Techniken, die zur Darstellung einer 2D-Projektion eines diskret abgetasteten 3D-Datensatzes, typischerweise eines 3D-Skalarfeldes (eine Menge von Voxeln), verwendet werden. Aus vielen Anwendungen werden dreidimensionale Arrays aus digitalen Daten erzeugt. Das DICOM Protokoll ist eines davon. In diesem Fall ist es notwendig die DICOM Serie in ein 3D Datensatz umzuwandeln und ein Skalarfeld zu berechnen. Die Volumenvisualisierung ist eine Methode zur Extraktion aussagekräftiger Informationen aus volumetrischen Datensätzen durch den Einsatz interaktiver Grafiken und Bildgebung und befasst sich mit der Darstellung, Manipulation und Wiedergabe von volumetrischen Datensätzen. [17]

Ein Skalarfeld φ ist eine Funktion, die uns für jeden Punkt im Raum einen einzelnen Wert p liefert und bildet jeden Punkt p einer Mannigfaltigkeit M auf einen Skalar $\varphi(p)$ ab. [18]

$$\varphi: M \rightarrow \mathbb{R}$$

In einem diskreten Skalarfeld existieren Isoflächen, die für den Marching Cubes benötigt werden. Eine Isoflächen (eng. Isosurface) bildet eine Menge von Punkten (x,y,z) ab, die sich in einen Wertebereich abhängig vom Isowert (Schwellenwert) befinden. Isosurfaces (3D) sind analog zu Isolinien (2D). Ein gutes Beispiel dafür liefert die Messungen der Temperatur in einem Raum an bestimmten Punkten. Die Menge wird definiert: [19]

$$\{< x, y, z > \in \mathbb{R}^3 : f(x, y, z) = h \in \mathbb{R}\}$$

Die Idee von der Erzeugung eines Skalarfeldes ist es, einer Menge P von Punkten p im Skalarfeld einen Farbwert oder Material zuzuweisen und somit ein dreidimensionales Bild zu erzeugen. Die Voraussetzung dafür ist, dass jeder Punkt p einen Wert für seine eigene Dichte besitzt. Somit sind die Punkte in der Menge unterscheidbar.

In nachfolgenden Abschnitten werden die Volume Rendering Verfahren “Volume Ray Casting” und der “Marching Cubes” beschrieben. Die Berechnung des Skalarfeldes für eine DICOM Serie wird in Abschnitt 3.2 beschrieben.

Typischerweise werden 3D Datensätze als quadratische Zellen oder einfache Punkte/Werte in Arrays gespeichert. Diese Punkte oder Zellen bezeichnet man als Voxel. Eine Voxel repräsentiert einen Wert in einem Gittermodell im dreidimensionalen Raum (Skalarfeld). Die Werte einer Voxel sind variabel und vom Anwendungsfalls abhängig. [17]

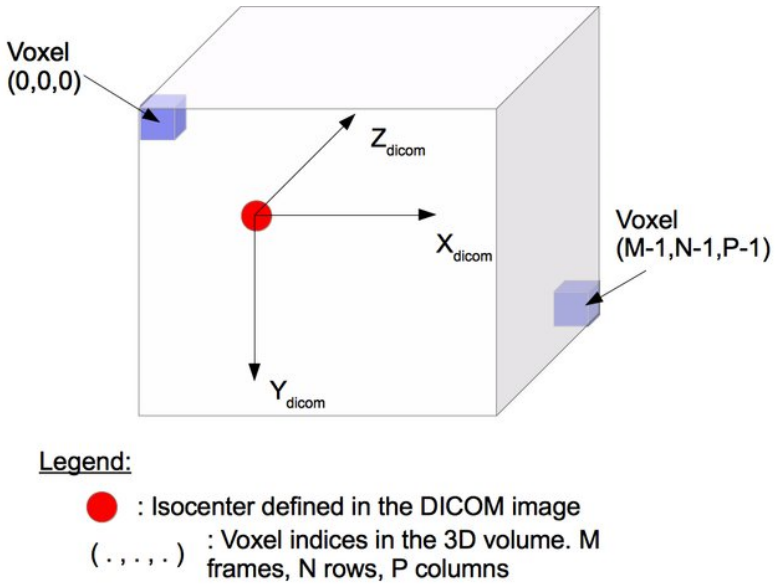


Abbildung f: Repräsentation eines Voxel Felds [50]

3.2 Voxel Berechnung einer DICOM Serie

3.2.1 DICOM Serien

Eine DICOM Serie besteht aus mehreren aufeinander folgenden einzelnen DICOM Slices (.dcm). Für die Erzeugung eines 3D Datensatzes sind einige Voraussetzungen notwendig. Die DICOM Serie muss einheitlich und regelmäßig aufgenommen sein. Das bedeutet, die Folge der Dateien muss in einem regelmäßigen Abstand und im selben Winkel gescannt worden sein. Ansonsten kann das Volumen Rendering eine schlechte 3D Projektion erstellen, Artefakte bilden oder gar kein 3D Rendering durchführen. Verzerrungen in der Darstellung und bei den Messungen sind dann zu erwarten. Medizintechnisch werden DICOM Serien wie beschrieben gescannt:

Die Strahlen, die ein CT oder MR erzeugt, dringen durch einen Körper. Die Strahlen werden vom Körper und dem Detektor, der sich unter dem Medizingerät befindet, aufgefangen. Je nach der Dichte und Material des betroffenen Mediums im Körper (Knochen, Flüssigkeit, Weichteile), dringen stärkere oder schwächere Strahlen bis zum Detektor durch. Der Detektor wird wie ein Film ausgelesen und erzeugt die Aufnahme mit den abgefangenen Strahlungen. Dabei wird eine Aufnahme als "Slice" bezeichnet. Slices werden in einem bestimmten Abstand von A nach B gescannt. Außerdem besitzen Slices eine Dicke. Das Verhältnis zwischen dem Abstand und Dicke beschreibt, wie regelmäßig der Körper gescannt wurde. Dabei gilt das Prinzip "As Low As Reasonably Achievable" (ALARA), welches vorschreibt, nur so viel wie nötig eines Körpers zu scannen (Strahlenschutz). Daher kann eine DICOM Serie auch nur aus einem einzigen Slice bestehen, falls das dem behandelten Arzt ausreicht. [20]

Je geringer der Slice Abstand und die Slice Dicke ist, desto genauer und hochauflösender wird das Volumen Rendering, da mehr Werte im 3D Datensatz enthalten sind. Das würde jedoch zu großen DICOM Studies führen (Terabyte Bereich) und das Auslesen verlangsamen.

3.2.2 Voxel aus Image Pixel und Plane Module für CT/MR Image IOD

Für das Erzeugen einer Voxel sind die Image Position (0020,0032) und Image Orientation (0020,0037) aus dem Image Plane Module nötig. Das Image Plane Module ist für das CT/MR IOD ein Pflichtmodul. Das Image Plane Module liefert die 3D Daten. Die Stellung des Patienten legt ebenfalls die Transversale-Ebene (Coronall, Axial, Sagittal) fest, die aus der Image Orientation ermittelt werden kann. Der Mittelpunkt des Körpers bzw. des Strahlers legt den Koordinatenursprung des dreidimensionalen Raumes fest.

Ein DICOM Image muss nicht zwingend quadratische Pixel enthalten, wie übliche Bild Kodierungen. Daher besitzt das Image Plane Module das Element Pixel Spacing (0028,0030). Alle auf den Pixelabstand bezogenen Attribute werden als der physikalische Abstand zwischen den Zentren jedes zweidimensionalen Pixels kodiert, der durch zwei numerische Werte angegeben wird. Der erste Wert ist der Zeilenabstand in mm, d.h. der Abstand zwischen den Mittelpunkten benachbarter Zeilen, oder der vertikale Abstand. Der zweite Wert ist der Spaltenabstand in mm, d.h. der Abstand zwischen den Mittelpunkten benachbarter Spalten, oder der horizontale Abstand. [21]

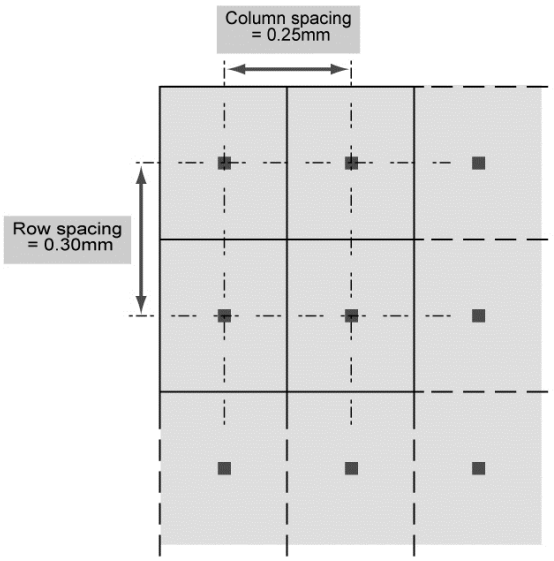


Abbildung g: PixelSpacing Repräsentation [55]

Als Beispiel für das Pixel Spacing kann man sich folgende Fälle ansehen. Wenn beide Fälle genau dieselben Aufnahmen sind:

Fall 1: DICOM Image	Rows (x): 100,	Columns (y): 100,	Pixel Spacing (1 | 1)
Fall 2: DICOM Image	Rows (x): 200,	Columns (y): 100,	Pixel Spacing (0.5 | 1)

Beide Fälle werden genau dasselbe Bild anzeigen. Fall 2 lässt vermuten, es sei verzerrt, weil es doppelt so viel Rows hat als Fall 1. Jedoch hat die Pixel Spacing in X-Richtung die halbe Größe. Daher geben die Rows und Columns nur die Anzahl der Zeilen und Spalten an. Die Tatsächliche Pixel Anzahl ist das Produkt des Pixels Spacing und Rows/Columns.

Die Image Position Patient (0020,0032) liefert einen dreidimensionalen Vektor $\vec{v}$ und beschreibt die X-, Y- und Z-Koordinaten der linken oberen Ecke (Mitte des ersten übertragenen Voxels) des Bildes in mm. Die Image Orientation (0020,0037) gibt die Richtungskosinus der ersten Zeile und der ersten Spalte in Bezug auf den Patienten an. Diese Attribute sind als Paar anzugeben: Zeilenwert

für die X-, Y- und Z-Achse, gefolgt vom Spaltenwert für die X-, Y- und Z-Achse. Eine Voxel Position (P_x, P_y, P_z) bildet eine Pixelposition (i, j) auf das Gerät wie folgt ab: [22]

$$\begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} X_x\ \Delta_i & Y_x\ \Delta_j & 0 & S_x \\ X_y\ \Delta_i & Y_y\ \Delta_j & 0 & S_y \\ X_z\ \Delta_i & Y_z\ \Delta_j & 0 & S_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ 0 \\ 1 \end{bmatrix} = M \begin{bmatrix} i \\ j \\ 0 \\ 1 \end{bmatrix}$$

P_{xyz} : Die Koordinaten des Voxels (i, j) in der Bildebene des aktuellen Bildes in mm.
S_{xyz} : Der Vektor von Image Position (0020,0032). Es ist die Position des Ursprungs.
X_{xyz} : Die Werte aus der Zeile (X) Richtung des Image Orientation (Patient) (0020,0037)
Y_{xyz} : Die Werte aus der Spalte (Y) Richtung des Image Orientation (Patient) (0020,0037)
i : Spaltenindex zur Bildebene.
Δ_i : Spalten Pixel Spacing
j : Reihenindex zur Bildebene.
Δ_j : Reihen Pixel Spacing

Die Matrix lässt sich vereinfacht darstellen durch eine Funktion für Reihe i und Spalte j des aktuellen Bildes n.

$$f_n(i,j) = \overrightarrow{S_n} + \left(\overrightarrow{X_n} * \Delta_i * i\right) + \left(\overrightarrow{Y_n} * \Delta_j * j\right) = \vec{P}_{ij}$$

Daraus folgt die Voxel für die Serie mit der Collection V:

$$\sum_{\substack{0 \le i < rows \\ 0 \le j < columns}}^{n} f_n(i,j)$$

Für die Implementierung der Voxel muss die Image Patient Orientation nicht zwingend beachtet werden, solange man das Volumen initial nicht nach einer bestimmten Orientierung rotieren möchte oder die Lage des transversalen Schnittes nicht benötig. Die Collection V kann durch eine Schleife abgebildet werden und die Dimension vereinfachter berechnet werden.

Durch die Berechnung von dem Abstand zwischen zwei Slices $Distance(\overrightarrow{v_1}, \overrightarrow{v_2}) = d$ und der Rows (0028,0010) und Columns (0028,0011) Angaben in dem Image Pixel Module lassen sich die Dimensionen des Volumens ermitteln und berechnen.

Das Produkt des Abstands d und der Anzahl der Slices n ergibt die Tiefe des Volumens z. Rows und Columns stellen die x, y Dimensionen dar. Die Pixel Spacing zu beachten ist hierbei wichtig.

$$Dimension\ x = Rows * PixelSpacing.x$$
$$Dimension\ y = Columns * PixelSpacing.y$$
$$Dimension\ z = d * n$$

Die Größe des Skalarfeldes beträgt somit $v(Dimension\ x, Dimension\ y, Dimension\ z)$.

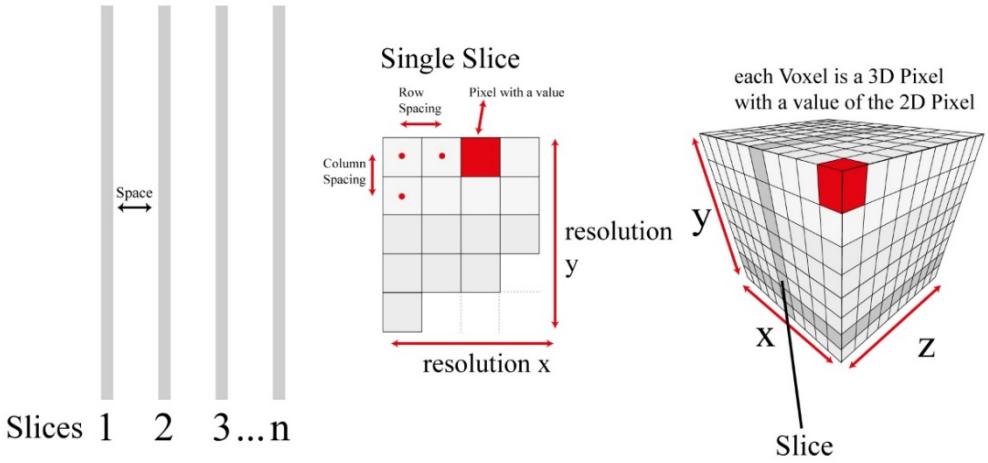


Abbildung h: Darstellung der Dimensionen einer DICOM Serie

3.2.3 Extrahieren der Pixel Values

3.2.3.1 Raw Pixel Data

Jeder Pixel enthält einen Pixel Value als Rohdaten. Die ursprünglichen Pixelwerte könnten beispielsweise einen gerätespezifischen Wert speichern, der nur dann eine Bedeutung hat, wenn er von dem Gerät, das die Werte erzeugt hat, verwendet wird. Für die Umwandlung in allgemeine oder vordefinierte Werte werden Lookup Tables (LUT) oder die Rescale Intercept/Slope Werte verwendet. Die Ursprünglichen Rohdaten werden auch Modality LUT genannt, die in der jeweiligen Standard Modalitäten definiert sind. Modalitäten beschreiben in den DICOM Daten, um welche Aufnahmetechnik es sich handelt (z.B. CT, MRT, US).

Die Rohdaten können durch die Rescale Intercept (0028, 1052) und Rescale Slope (0028, 1053) in bestimmte Werte, die für die Anwendung sinnvoll sind, umgewandelt werden. Die DICOM Elemente Rescale Intercept und Rescale Slope geben die lineare Transformation von Pixeln in ihrer, auf der Festplatte gespeicherten Darstellung, zu ihrer Darstellung im Speicher an.

Nach einer Transformation (LUT oder Rescale Intercept/Slope) geben die Werte Window Center (0028,1050) und Window Width (0028,1051) an, welche Graustufen bzw. Pixelwerte sichtbar sein sollen. Alle Pixel, die außerhalb des festgelegten Wertebereiches sind, werden schwarz oder weiß angezeigt (transparent in Volume Rendering). Der Wertebereich wird durch die Window Settings festgelegt.

3.2.3.2 Values of Interest Lookup Table Funktionen

Die DICOM-Pipeline für das Rendern von Bildern enthält eine Reihe von Stufen, in denen ein Lookup Table (LUT)-Typ angewendet werden kann. Lookup Table Funktionen interpolieren die Dichte der sichtbaren Graustufen. Die Anwendung von LUT ist nur für DICOM Daten mit der Photometrische Auswertung (0028,0004) von MONOCHROME1/2 sinnvoll. Eine Values of Interest Lookup Table (VOI LUT) Funktion wird angewendet, wenn die Transformation von den ursprünglichen Pixelwerten in relevante Werte nicht linear ist. [23]

Bei nicht linearen Interpolationen nutzt das VOI LUT die Aktivierungsfunktion SIGMOID. Die SIGMOID Funktion wird in Deep Learning häufig genutzt, um das Output Neuronen zu bewerten. Die SIGMOID Interpolation verfügt einen weichen Gradienten bei der Approximation auf den Minima und Maxima Wert. Gleichzeitig bietet es einen linearen Teil an, der in dem Definitionsbereich streng monoton steigend ist. Somit erzielt man in der Normalisierung der Minima und Maxima Werte auf 0 und 1 (Alpha-Werte) eine höhere Verteilung in den mittleren Werten. [24] [25]

3.2.3.3 Hounsfield-Skala

Die Hounsfield-Skala (HU) beschreibt in der CT die Abschwächung der Röntgenstrahlen in Gewebe, die der Detektor abfängt und als Graustufenbilder darstellt. Die HU Werte können Gewebearten zugeordnet werden und somit pathologische Abweichungen erfasst werden. Daher kann man auch sagen, dass der HU Wert die Röntgendichte (Röntgenopazität) beschreibt. [26]

Bei der Darstellung auf der Festplatte und im Speicher kann ein anderer Wertebereich möglich sein. Ein Beispiel: CT-Bilder, deren Pixelwerte in Hounsfield-Einheiten gemessen werden, die negative Werte haben können, werden üblicherweise mit einer vorzeichenlosen Ganzzahl gespeichert. Folglich ist es üblich, dass CT-DICOM-Dateien für die Umwandlung einen negativen Rescale Intercept haben. Die lineare Skalierung wird auch in Fällen angewandt, in denen ein Pixel einen großen Wertebereich haben kann, wobei die Werte mit möglichst wenigen Bits gespeichert und Quantisierungsfehler vermieden werden. [27]

Die Formel für die Berechnung der HU Values lautet:

$$hu = rawPixelValue * slope + intercept$$

Die Rescale Intercept/Slope Werte werden vom Hersteller der Hardware bestimmt und sind mandatorische Tags. Mit folgender Formel lassen sich die Window Settings anwenden:

$$minHU = windowCenter - (windowWidth / 2)$$
$$maxHU = windowCenter + (windowWidth / 2)$$

Durch den HU Value lassen sich Substanzen spezifizieren. Die Zuordnung von Substanz und Wert können sehr stark nach Gerätehersteller und Geräteeinstellung durch den Techniker schwanken. Deshalb ist es gut ein Histogramm zu nutzen, um schnelle Analysen durchzuführen. Trotzdem gibt es HU Values die sich bewährt haben und indirekt einen Standard gesetzt haben. Die berechneten HU Values werden der Voxeln zugewiesen. Mit diesen Voxeln ist nun ein Volumen Rendering möglich.

Material	Hounsfield Unit
Air	-1000
Lung	-500 to -200
Fat	-200 to -50
Water	0
Blood	25
Muscle	25 to 40
Bone	200 to 1000

Tabelle 1: Beispiel für HU Values [28]

3.3 Grundlagen des Volume Raycasting

Robert A. Drebin, Loren Carpenter und Pat Hanrahan stellten diese Technik erstmals im Jahr 1988 vor. [29] Nachfolgend wird das Direct Volume Rendering, Maximum Intensity Projection und Isosurface Rendering detaillierter beschrieben. Für die Implementierung wird das offene GitHub Projekt von Matias Lavik verwendet und angepasst. [30]

3.3.1 Algorithmus

Das Volume Raycasting, auch oft Volume Ray Marching genannt, ist ein Volume Rendering Verfahren. Es basiert auf dem Raycasting Render Verfahren. Das Ray Marching kann je nach Anwendungsfall einen modifizierten Algorithmus implementieren (z.B. Sphere Tracing). Nachfolgend wird nur das Volume Ray Casting betrachtet.

Raycasting/-tracing: Das Ray Casting feuert vom Auge (z.B. Kamera) durch jeden Pixel im 2D Grid einen Strahl (Ray) ab und zeichnet alle getroffenen (hit) Triangles auf den Bildschirm. Bei einer erweiterten Version des Raycasting (dem Raytracing) ist es möglich realitätsnahe fotorealistische Lichtberechnungen durchzuführen. Dabei werden vom aktuellen Schnittpunkt aus Refraktion-, Reflektion- und Lichtvektoren abgefeuert und verfolgt (Tracing), um die Licht- und Shader (Phong, Lambert) Berechnung durchzuführen. Dadurch erhält man eine physikalisch korrekte und realitätsnahe 3D Szene.

Der wesentliche Unterschied zu dem Raytracing ist im Aufbau der Szene. Beim Raytracing ist es üblich in der Szene primitive Geometrien (Kugeln, Würfeln, Dreiecke) zu definieren. Der Volume Raycasting zielt darauf ab nicht einheitlichen Volumen (z.B. Wolken) oder implizite Funktionen und Fraktale zu rendern. In diesem Fall ist die DICOM Serie eine variables Skalarfeld, welches mit der Volume Raycasting abgetastet werden muss.

Beim Volume Raycasting wird wie beim konventionellen Raycasting einen Strahl abgefeuert. Als Ursprung ist die Kamera definiert. Die Länge L des Strahles wird in Samples n aufgeteilt. Jeder Sample Punkt definiert eine Farbe. Die Berechnung der Farbe hängt vom Modus ab. Folgende Modi werden vorgestellt: Direct Volume Rendering, Maximum Intensity Projection, Average Intensity Projection und Isosurface Rendering.

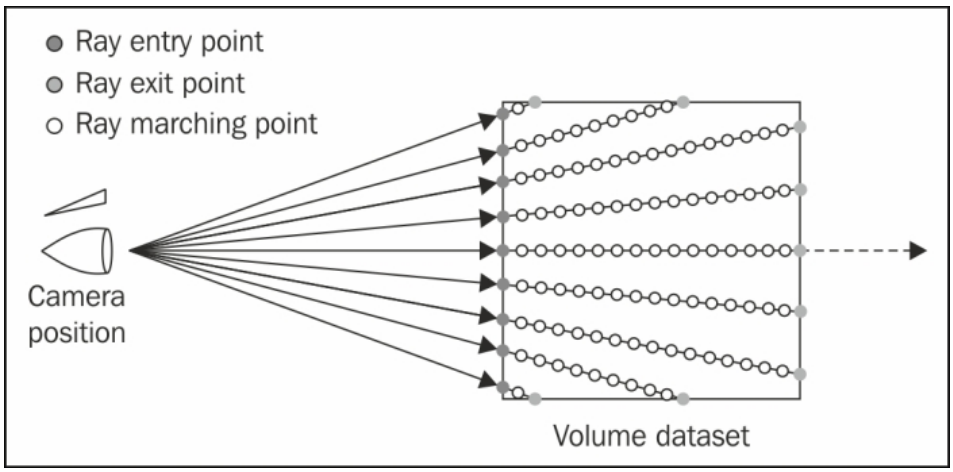


Abbildung i: Volume Raycasting Repräsentation [51]

Bei allen Rendering Methoden werden allen abgetasteten Samples eine Compositing ausgeführt und der Farbwert auf dem Bildschirm projiziert.

Das Volume Raycasting kann in drei Phasen aufgeteilt werden: [31]

1. Abtastung des Skalarfeldes
    a. Reguläre Abtastung: Nutze alle abgetasteten Werte aus den Voxeln. Es kann vorkommen, dass eine Voxel mehrmals abgetastet wird und zur Redundanz führt.
    b. Nearest-neighbor Abtastung: Nutze nur die Voxel, die sich am nächsten zu dem aktuellen Sample befindet → niedrige Auflösung
    c. Voxel Schnittstellen: Nutze die Voxel Kanten, die durch den Strahl geschnitten werden.
2. Erzeugen eines Intensitätsprofil für je Strahl
    a. Erstelle ein Intensitätsprofil als 2D Graphen, der beschreibt, bei welcher Tiefe welcher Voxelwert vorkommt.
3. Durch Anwendung einer Ray Funktion Voxel anzeigen
    a. Wende eine Ray Funktion an, z.B. Isosurface Rendering, Direct Volume Rendering, Maximum Intensity Projection oder Average Intensity Projection

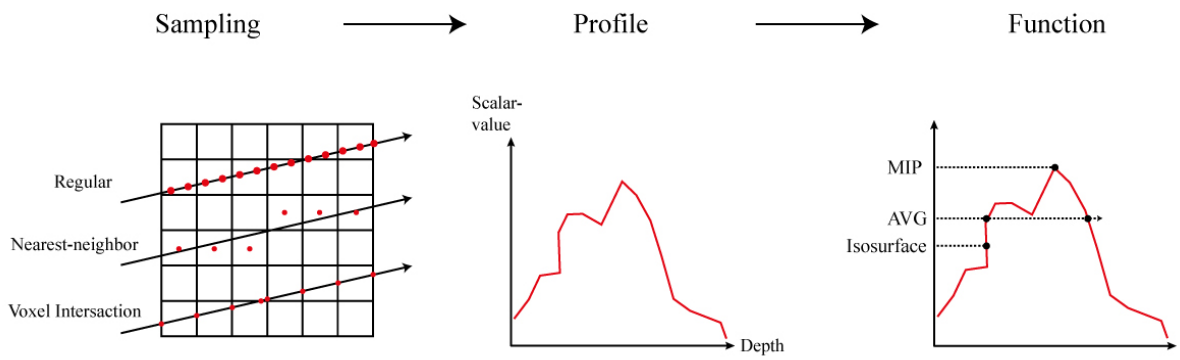


Abbildung j: Darstellung der Schritte

3.3.1.1 Isosurface Rendering

Der Isosurface Rendering Algorithmus ist wie der Marching Cubes abhängig von einem Isowert h. Der erste getroffene Voxel mit dem Skalar $f(x) > h$, wird gezeichnet und kein weiterer Strahl wird abgefeuert, da ein voller Farbwert ohne Transparenz gesetzt wird. Ein einfacher Farbwert ist ohne Licht- und Schattenberechnung nicht hilfreich und unästhetisch. Daher ist es anschließend notwendig eine Lichtberechnung (Shading) durchzuführen.

Es sind beliebige Beleuchtungsmodelle möglich (Lambert, Phong, Blinn). Für den Proof of Conecpt wird die diffuse Lambert Beleuchtung ausreichen. Die Lichtintensität i_d wird durch den Winkel θ zwischen der Richtung $\vec{l}$ zur Lichtquelle und der Oberflächennormale $\vec{n}$ und einer diffusen Konstante K_d bestimmt. Da wir durch dieses Rendering Verfahren keine Polygone erhalten, muss die Normale ermittelt werden. Die Steigungsänderung beschreibt die Normale. Die Richtung der Steigungsänderung (Gradient) in Abhängigkeit auf die Nachbarzellen bildet die Normale $\vec{n}$. Zur einfachen veranschaulichung wäre das in einem zweidimensionalen Raum die Geradensteigung durch zwei Punkte P_1 und P_2 mit $m = \frac{\Delta y}{\Delta x}$. Analog zum dreidimensionalen Raum ist die Normale $\vec{n} = (\Delta x, \Delta y, \Delta z)$, die bei dem aktuellen Voxel mit den benachbarten Voxeln berechnet wird. Durch die Berechnung der Normalen aller Voxeln entsteht ein Vektorfeld an Normalen. Mit der Normale und dem Einfallswinkel des Lichtvektor kann die Lichtintensität berechnet werden. Wir nehmen an, dass $v = data(x, y, z)$ den Skalarwert in der Voxel liefert. [32]

$$\Delta x = data(x-1, y, z) - data(x+1, y, z)$$
$$\Delta y = data(x, y-1, z) - data(x, y+1, z)$$
$$\Delta y = data(x, y, z-1) - data(x, y, z+1)$$

$$\vec{n} = \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix}$$

$$i_d = K_d * (\vec{n} \cdot \vec{l})$$

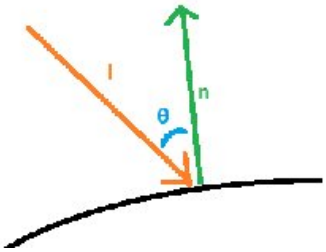


Abbildung k: Einfache Lichtberechnung mit Vektoren

3.3.1.2 Maximum Intensity Projection

Die Maximum Intensity Projection (MIP) ist in der Implementierung eine einfache Projektion. Die MIP projiziert entlang dem aktuellen Strahl nur die höchsten Voxelwerte. Diese sind meistens solide Materialien wie die Knochen und eine dünne Schicht der Haut. Diese Projektion wird verwendet, um Angiogramme (Blutgefäße mit Kontrastmittel) oder Gefäßstrukturen aus einem CT- oder MRT-Scan zu extrahieren. [33]

3.3.1.3 Average Intensity Projection

Der Average Intensity Projection zeigt die den durchschnittlichen Dichtewert entlang des aktuellen Strahls an. Mit dem Average Intensity Projection erzielt man eine X-Ray ähnliche Projektion. [33]

3.3.1.4 Direct Volume Rendering

Das Direct Volume Rendering (DVR) stellt wohl den fortgeschrittensten Modus dar. Bei dem Direct Volume Rendering werden alle Samples auf eine Farbe und Deckkraft (RGBA) abgebildet. Die RGBA Werte erhalten die Samples aus einer Transferfunktion. Die Grundidee ist es bei jedem Sample den Farbwert mit dem vorherigen Sample zu kombinieren. Die Kombination aus den zwei RGBA-Werten kann eine beliebige Interpolation zwischen zwei Funktionswerten sein. In der einfachsten Form beschreibt eine eindimensionale Transferfunktion einen Graphen eines unabhängigen Skalaren (HU-Value) im Vergleich zum abhängigen skalares (RGBA-Wert). Die Konzeption lautet: Zeige einen RGBA-Wert bei Skalar x.

Es kann vorkommen, dass mehrere Körperteile ähnliche Dichten aufweisen und nicht direkt mit der eindimensionalen Transferfunktion ausgewählt werden können, weil die Unterschiede zu gering sind. Deshalb ist es nötig eine weitere Dimension hinzuzufügen. $f(x)$ beschreibt in zweidimensionalen Transferfunktionen die Skalardaten des Skalarfeldes. Mit der ersten Ableitung $f'(x)$ lassen sich, wie beim Isosurface Rendering, als Vektor $\overrightarrow{v_i}$, die Richtung der größten Steigungsänderung beschreiben. Die normalisierte Steigungsänderung wird oft als Normale für das Shading genutzt. Die Länge der Steigungsänderung bei $|\overrightarrow{v_i}|$ für $f'(x_i)$ beschreibt die lokale Änderung im Skalarfeld $f'(x_i) = |f(x)|$. [34]

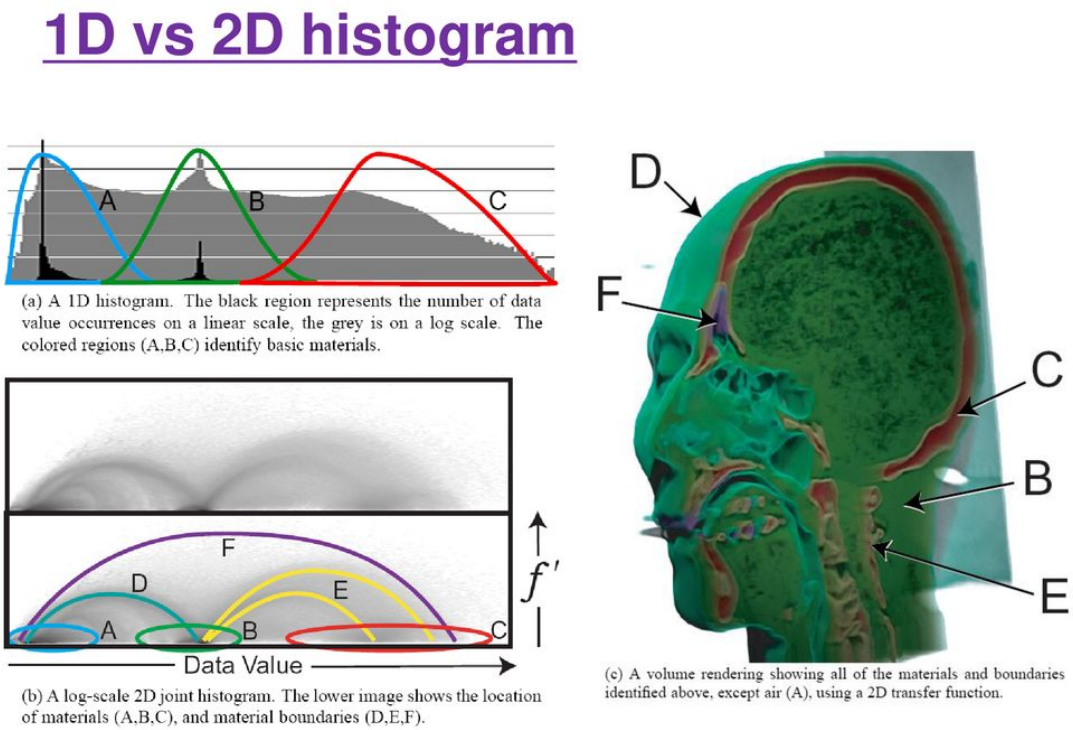


Abbildung l: Transfer Funktionen Beispiel [56]

Mit der Kombination der Dichte (HU), die Richtung und Länge der Steigungsänderung lassen sich leichter Regionen definieren. Für die Verdeutlichung wird folgendes Beispiel betrachtet: Die Haut hat eine relativ geringe Dichte. Viele andere Regionen können dieselbe Dichte aufweisen. Da jedoch die Haut die äußerste Schicht des Körpers bildet, hat es gleichzeitig eine relativ hohe Steigungsänderung (durch die Anzahl der Voxel) im Vergleich auf seine Dichte. [35]

3.3.1.5 Volume Rendering mit Window Settings

Die Voxeln mit einem Schwellenwert auf einen Wertebereich zu limitieren, erzielt nicht immer ein gutes Ergebnis. Für eine verbesserte Ausgrenzung der Dichtwerte können die Window Settings des DICOM Standard verwendet werden.

Wenn die VOI LUT-Funktion (0028,1056) nicht vorhanden ist oder den Wert *LINEAR* hat, spezifizieren der Window Center (0028,1050) und die Window Width (0028,1051) eine lineare Transformation von gespeicherten Pixelwerten (nach Anwendung einer beliebigen Modality LUT oder Rescale Slope und Intercept, die im IOD spezifiziert wurden) auf die anzuzeigenden Werte. Window Center enthält den Eingabewert, der die Mitte des Fensters darstellt. Window Width enthält die Breite des Fensters. [36]

Je größer der Window Center, desto mehr Voxel mit größeren Dichtwerten werden angezeigt.
Je größer der Window Width, desto größer der Wertebereich der angezeigten Dichtewerte. Deshalb gilt für die Window Width $WindowWidth \geq 1$. Die lineare Transformation wird im DICOM Standard mit einem pseudo Code beschrieben: [36]

Input: x (z.B. HU Value)
Window Center: c
Window Width: w
Output: y

```
if (x <= c - 0.5 - (w-1) /2), then y = ymin
else if (x > c - 0.5 + (w-1) /2), then y = ymax
else y = ((x - (c - 0.5)) / (w-1) + 0.5) * (ymax- ymin) + ymin
```

Code 1: Lineare Transformation pseudo Code

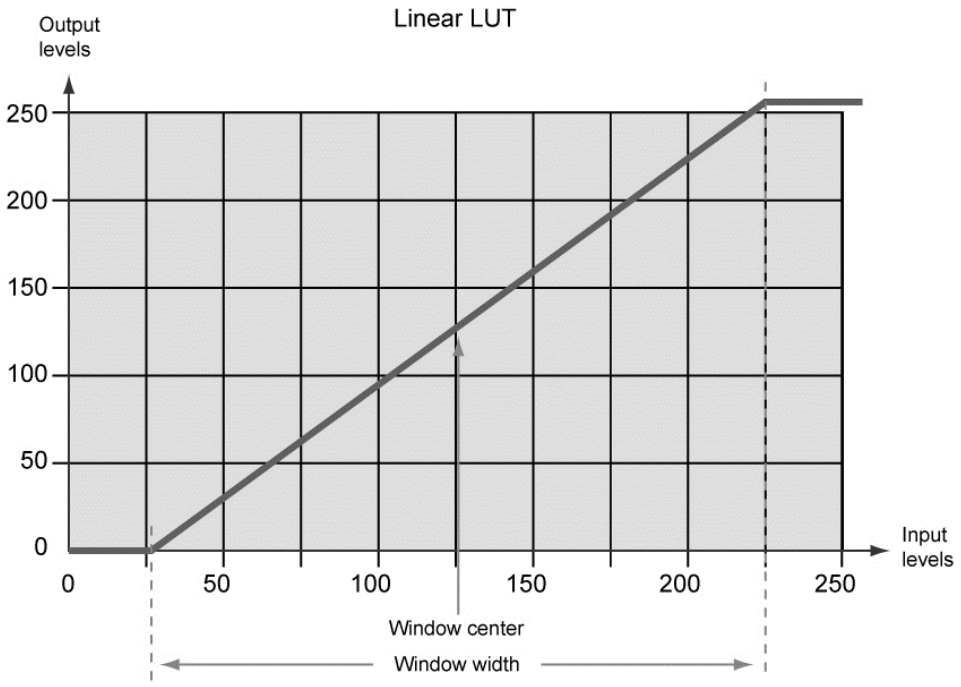


Abbildung m: Window Settings als Lineare Funktion

3.3.2 Komplexität

Es ist schwierig eine allgemeine Komplexität zu definieren, da die Komplexität des Volume Raycasting stark abhängig vom Rendering Modus ist. Bei allen Modi ist die Anzahl der Voxel und die tiefe des Sampling signifikant. Beim Isosurface Rendering ist das Sampling weniger Aufwändig, da es beim ersten Hit einer Voxel das Sampling für den aktuellen Strahl unterbricht. Man könnte einen absoluten Worst Case für die vorgestellten Verfahren definieren mit $O(x * y * sampleRate)$ wobei x und y die zweidimensionale Auflösung eines Slice ist. Die $sampleRate$ gibt an, mit welcher Frequenz die Tiefe z abgetastet werden soll. Allgemein ist das Volume Raycasting auf der GPU in Echtzeit rechenaufwendig, welches in der immersiven Darstellung zu Problemen führen kann.

3.3.3 Pseudo Code

```
START
Define the length of a sample step:
float stepsize = max / numberOfSteps;
Define Ray as 3D Vector:
              Vector rayStart = cameraPosition;
              Vector rayDirection;
       Loop until steps are done
       FOR currentStep = 0 to numberOfSteps
              Calculate the step with step size
                     float step = currentStep * stepsize
              Calculate the current pos along the ray
                     Vector currentPos = rayStart + rayDirection*step;
              Get the scalar value
                     float density = getDensity(currenPos);

              Modify the color based on the density.
              This step is variable according to rendering mode
              color = someColorFunction(density);
       ENDFOR
return color;
END
```

Code 2: Pseudo Entwicklung Ray Marching

[35]

3.4 Grundlagen des Marching Cubes

Der Marching Cubes Algorithmus wurde 1987 von William E. Lorensen und Harvey E. Cline veröffentlicht. [37] Marching Cubes ist ein Algorithmus in der Computergrafik um Polygon Meshes von Isoflächen (Isosurfaces) zu extrahieren bzw. darzustellen. Die Isoflächen befinden sich in einem dreidimensionalen diskreten Skalarfeld. Die verwendete Implementierung orientiert sich an die Dokumentation von Paul Bourkes "Polygonising a scalar field". [38] [39]

3.4.1 Algorithmus

Es ist von Vorteil zunächst einen Einblick in den Marching Squares Algorithmus zu erhalten, um den Marching Cubes genauer zu verstehen.

Der Marching Squares Algorithmus ist analog zu den Marching Cube. Wobei der Algorithmus mit Isolinien im zweidimensionalen Skalarfeld arbeitet. Der Marching Squares wird typischerweise für Höhenlinien auf topografischen Karten oder die Erzeugung von Isobaren für Wetterkarten verwendet.

Der Algorithmus zeichnet Linien zwischen interpolierten Werten entlang der Kanten eines Quadrats. Dabei wird ein Referenzwert (Isowert) berücksichtigt. Das Skalarfeld wird in eine gewünschte Auflösung in ein Gitterraster mit Quadraten mit 4 Knoten eingeteilt. Die Kanten eines Quadrats erhalten je ein Knoten, dessen Position entlang der Kante variieren kann.

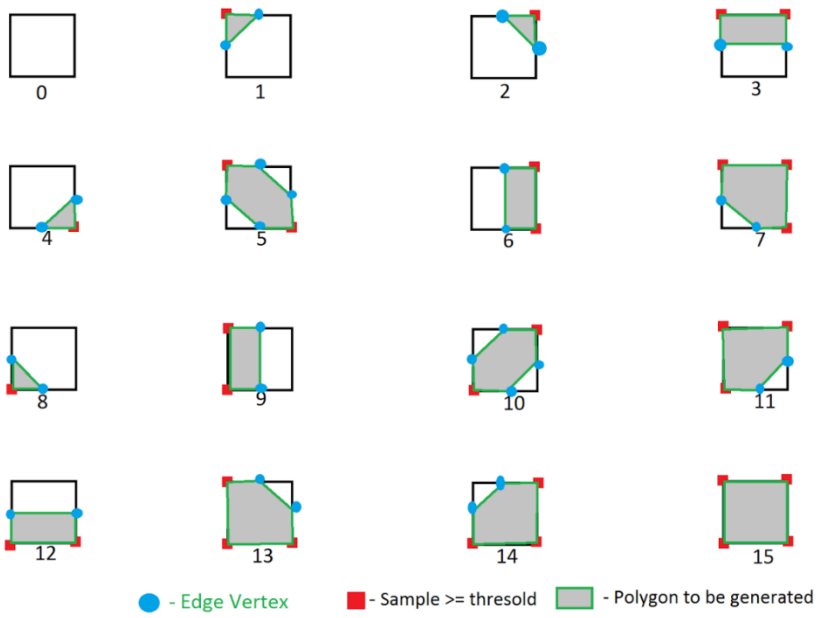


Abbildung n: Marching Squares Fälle

Der Isowert ist frei wählbar. Ist der Wert eines Knoten $f(x) \geqq Isovalue$, dann erhält der Knoten den Binärwert "1". Ansonsten ist der Binärwert bei "0". Knoten mit dem Binärwert 1, sind aktivierte Knoten. Man unterscheidet unter 16 möglichen Fällen, wie ein Square die Kanten interpolieren kann. Die Dezimalzahl bildet den Index. Der Index referenziert in einer Lookup Table eine Interpolation oder die Knoten eines Polygons, welche dann gezeichnet werden kann.

Der Marching Cubes (MC) Algorithmus verhält sich im dreidimensionalen Raum Analog zu dem Marching Squares. Der MC beschreibt die Erstellung einer polygonalen Oberflächendarstellung einer Isofläche eines 3D-Skalarfeldes. Dabei bietet der MC gute Performance, da es hauptsächlich auf Lookup Tabellen basiert. Der MC ist in der Medizin, Mathematik und Spielebranche häufig vertreten.

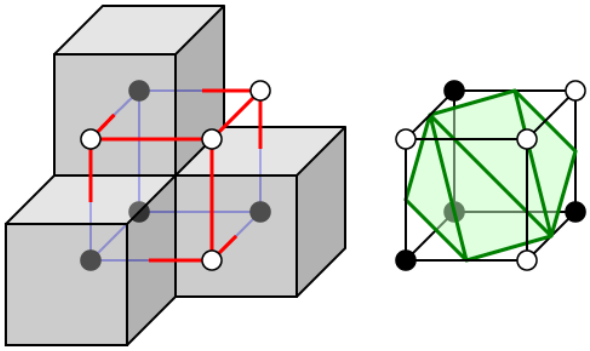


Abbildung o: Die Erzeugung eines Würfels und dessen Polygon [59]

Der Marching Cubes arbeitet nach dem "divide and conquer" Prinzip. Das Skalarfeld wird in logische Würfel aufgeteilt. Jeder Würfel besteht aus 8 Pixel; jeweils vier aus benachbarten Slices. Der MC beginnt beim ersten Voxel und ermittelt die Schnittpunkte mit der Isofläche. Anschließend "marschiert" der Würfel zum nächsten Voxel. Das wesentliche Problem des Marching Cubes ist es, eine Approximation an die Isofläche in einem Skalarfeld zu ermitteln. Die Schnittpunkte in einem Würfel werden durch die 8 Knoten des Würfels ermittelt. Ein Knoten kann zwei Zustände annehmen. [19]

$$1.Zustand\text{: } f(x) \geqq h$$
$$2.Zustand\text{: } f(x) < h$$

Da wir 8 Knoten des Würfels haben mit je 2 Zuständen, existieren $2^8 = 256$ Möglichkeiten an Schnittstellen zwischen Isosurface und Knoten, welches den schwierigen Teil des Algorithmus bildet. Die Schnittstelle definiert, ob ein Knoten "innerhalb" oder "außerhalb" der Isofläche ist. Die Position, an der sie die Kanten schneiden, wird linear interpoliert, das Verhältnis der Länge zwischen den beiden Knoten ist das gleiche wie das Verhältnis des Isowertes zu den Werten an den Knoten des aktuellen Würfels. [37]

Die geschnittenen Knoten geben einen Zustands Index $cubeIndex$ zurück. Mit dem $cubeIndex$ lässt sich aus der Lookup Tabelle der $edgeIndex$ ermitteln und somit der Zustand des aktuellen Würfels, der in der Isofläche dargestellt werden soll. Der $cubeIndex$ ist eine 8-Bit integer, wobei je Bit einen Knoten repräsentiert. Durch den $cubeIndex$ erhält man einen 12-Bit $edgeIndex$, wobei je Bit eine Kante repräsentiert. [38]

Für beide Index gilt bei $cubeIndex = 0 \wedge edgeIndex = 0$, dass die Isofläche gar nicht geschnitten wird. Die Schnittpunkte werden durch lineare Interpolation berechnet. Wenn P_1 und P_2 die Knoten einer geschnittenen Kante und V_1 und V_2 der Skalarwert $f(x)$ an den Knoten sind, ist der Schnittpunkt P gegeben durch [38]:

$$P = P_1 + (h - V_1) * (P_2 - P_1) \,/\, (V_2 - V_1)$$

Anschließend folgt eine Triangle Lookup Table, die denselben Index wie der $cubeIndex$ referenziert, um die nötigen Triangles auszulesen. Es werden maximal fünf Triangles benötigt. Je höher die Auflösung der Voxeln sind, desto genauer wird das generierte Mesh werden.

Zusammenfassung des Marching Cubes Algorithmus: [37]

1. Zwei aufeinander Folgende Slices n_i, n_{i+1} aus dem Skalarfeld wählen.
2. Voxel/Würfel aus 8 benachbarten Punkten bilden. Je vier Punkte aus n_i, n_{i+1}. Die Punkte bilden die Knoten $v_1 \cdots v_8$. Resultierende Kanten $e_1 \cdots e_{12}$ indexieren.
3. $cubeIndex$ ermitteln. Wenn $v_i > h$ wird der aktuelle Bit auf 1 gesetzt. Der 8-Bit integer liefert den gewünschten $cubeIndex$ aus $cubeIndex \in \{0, 1, \ldots, 255\}$ z.B: $1100\,0000 \triangleq 192$
4. Kanten erzeugen. Tirangle aus Lookup Table erhalten mit $cubeIndex$
5. Position der erzeugten Knoten auf den Kanten durch lineare Interpolation bestimmen.
6. Für jede Knoten die Normale berechnen und interpolieren der Einheitsnormale des Triangles
7. Triangles cachen und zur nächsten Zelle “marschieren”
8. Triangles zeichnen in Echtzeit oder nach der letzten Zelle zeichnen.

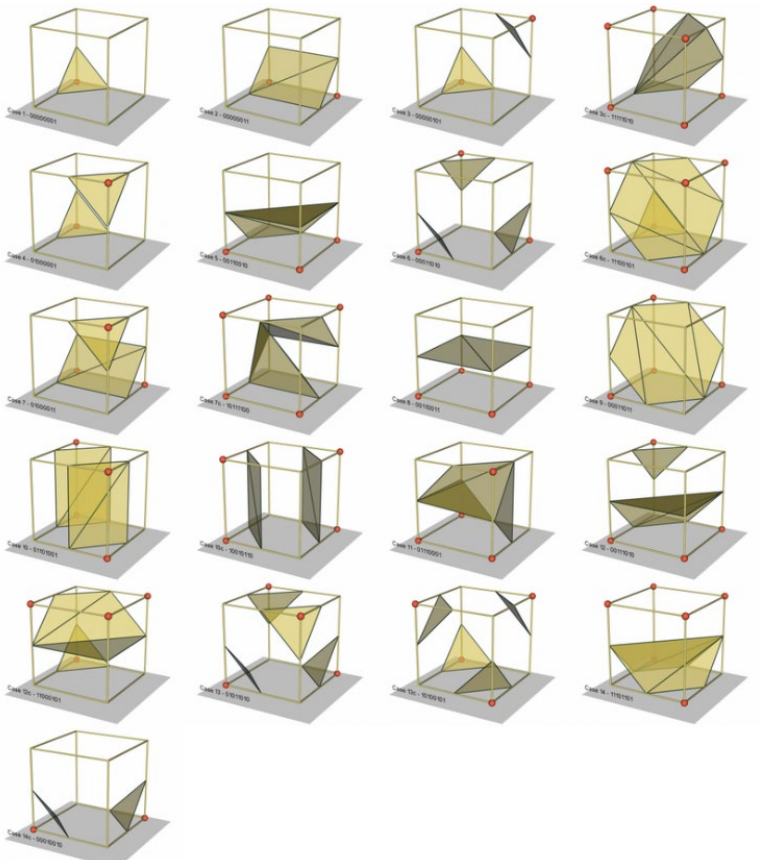


Abbildung p: Darstellung der möglichen Fälle. [53]

3.4.2 Komplexität

Der Standard MC hat im Worst Case eine Komplexität von $O(n)$, wobei n die Anzahl der Zellen ist. [40] Ohne Optimierungen entsteht immer der Worst Case, da immer alle Zellen iteriert werden. Die Summe der Zellen kann in Millionen Bereich liegen, welches eine große Summe ist. Für die meisten CPUs heutzutage wird das kein Problem sein, trotzdem sind einige Optimierungen möglich, falls der MC in Echtzeit auf der GPU laufen soll:

- Es ist möglich vor dem MC nur die aktiven Zellen zu ermitteln und in MC nur diese Zellen zu betrachten.
- Einige Zellen teilen sich dieselbe aktiven Kanten, welche dann zweimal interpoliert wird. Das könnte man vermeiden, wenn man sich aktive Kanten cached und nicht erneut abfragt.
- Anwendung einer k-d- Tree [41]

3.4.3 Pseudo Code

Eingabeparameter:

- Volumen Data oder Voxelgrafik
- Ein Isowert /Dichtewert

Ausgabeparameter:

- Liste aller anzuzeigenden Knoten und List aller zugehörigen normals

```
1: for each voxel (cube) v of V do
   2: Index vom Cube berechnen: Vergleiche alle 8 Knotenwerte mit   Isowert h.
   3: Berechnetes Index in der lookup-table prüfen
   4: Schnittstellen bestimmen von den vertices der edges mit den    surface-
edges durch lineare Interpolation
   5: Einheitsnormalen berechnen für jeden vertex im cube
       (Finite difference). Normals zu je triangle interpolieren.
   6: Return triangle vertices und vertex normals.
7: end for
```

Code 3: Pseudo Entwicklung Marching Cubes [19]

Kapitel 4: Praktische Umsetzung in Unity

4.1 Verfügbare Technologien für die immersive Visualisierung von DICOM

4.1.1 Unity Echtzeit-Entwicklungsplattform und Virtuelle Realität

Für den Praxisteil wird die Unity Engine genutzt. Unity ist eine Echtzeit-Entwicklungsplattform, die mit C++ entwickelt und eine C# API nutzt. Dank Unity wird der Fokus auf die Algorithmen leichtfallen, da die Engine bereits vieles bezüglich 3D-Visualisierung, Transformationen, ideale Entwicklung für VR und fortgeschrittene 3D Interaktionen übernimmt. Für die Einstellungen und Einrichtung der Virtuellen Umgebung gibt es Plugins (Steam VR) und Beispiel Implementierungen, die die Arbeit vereinfachen wird.

4.1.2 Visualization Toolkit in Unity

4.1.2.1 Visualization Toolkit und ActiViz.NET/C# in Unity

Das Visualization Toolkit wird für C# Entwicklungsumgebungen als ältere Version von VTK ActiViz.NET/C# Edition 5.8.0 (Stand März 2020) kostenlos unterstützt. Die Einbindung in Visual Studio erfolgt über das NuGet Manager. Der Unity Compiler entfernt manuell eingefügte Pakete aus dem NuGet Manager entfernt. Diese müssen als Plugin in Unity eingefügt werden, welches mit einem NuGet Manager Plugin in Unity möglich ist. Auf diesem Weg ist das Visualization Toolkit C# in Unity nutzbar. Die VTK nutzt eine eigene Renderpipeline mit OpenGL, die in Unity nicht direkt unterstützt wird. Es ist möglich die Renderpipeline per Plugin anzusprechen und die gerenderte VTK Scene in Unity anzuzeigen. [42] [43] Mit der VTK Einbindung hätte man Zugriff auf die Funktionen, um DICOM Serien einzulesen und in Meshes umzuwandeln. Das ist bereits mit wenigen Zeilen Code möglich. Dieser Weg sorgt für Einschränkungen, da man die volle Bandbreite der Unity Umgebung nicht direkt nutzen kann. Deshalb wäre ein einfacher Weg, die VTK Library ohne Anbindung an die Renderpipeline zu nutzen und die gerenderten Objekte in der VTK Scene in gängige 3D-Formate wie STL oder OBJ zu konvertieren und in Unity zu exportieren. Das hat zur Folge, keine Echtzeit Algorithmen anwenden zu können und durch die lange Pipeline würde die Performance sinken.

4.1.2.2 Native Unity Plug-ins in C++ und Visualization Toolkit

Die Einbringung von VTK ActiViz.NET/C# in Visual Studio und Unity ist einfach, aber bringt einige Nachteile mit sich. ActiViz.NET/C# läuft auf einer veralteten Version 5.8.0. Die aktuelle Version liegt bei 8.2 (Stand April 2020). Die C# Version ist nicht so gut dokumentiert, wie die Python oder C++ Dokumentationen. Es ist schwierig eine Analogie zwischen C# und C++ herzustellen. Die C#-Version dient als Wrapper und greift wiederum auf die eigene C++ Bibliothek zu. Deshalb war es einen Versuch wert einen eigenes natives C++ Plugin für Unity zu entwickeln und VTK zu nutzen. Dadurch lief der Zugriff auf VTK deutlich schneller und die DICOM Serien wurden schneller berechnet. Jedoch ist das Problem mit dem Export von den 3D-Formaten immer noch vorhanden. VTK ist eine abstrakte Bibliothek. VTK erfordert eine lange Einarbeitungszeit und somit wurde der Zeitrahmen für diese Arbeit begrenzt. Aus diesem Grund war es schneller die DICOM Serie als Rohdaten auszulesen, die Voxeln selber zu berechnen und fertige

Implementierungen des Marching Cubes [39] und Volume Raycastings [30] zu nutzen und anzupassen.

Während diese Bachelorarbeit geschrieben wurde, veröffentlichte Kitware einen "VTKUnity-MedicalViewer" für Unity im Asset Store (Version 1.0). Nach einer Evaluierung weist dieses Plugin dieselben Nachteile auf.

4.1.3 Verwaltung DICOM Daten mit Fellow Oak DICOM in Unity

Fellow Oak DICOM (fo-dicom) ist eine open-source Bibliothek zur Verwaltung von DICOM Daten. Fo-dicom unterstützt C# für .NET Framework, .NET Core, Universal Windows, Android, iOS, Mono und Unity. [4] Mit fo-dicom lassen sich DICOM Daten auslesen, manipulieren, erzeugen, die Meta Daten anzeigen und in Bilder kodieren. Die Einbindung in Unity läuft über das Plugin, welches über den Asset Store erhältlich ist. [44] Das Plugin funktioniert ausgezeichnet mit Unity und reicht vollkommen für den Anwendungsfall dieser Arbeit aus. Mit fo-dicom sollen die DICOM Serien ausgelesen und die nötigen DICOM Elemente abgerufen werden.

4.2 Auslesen einer DICOM Serie

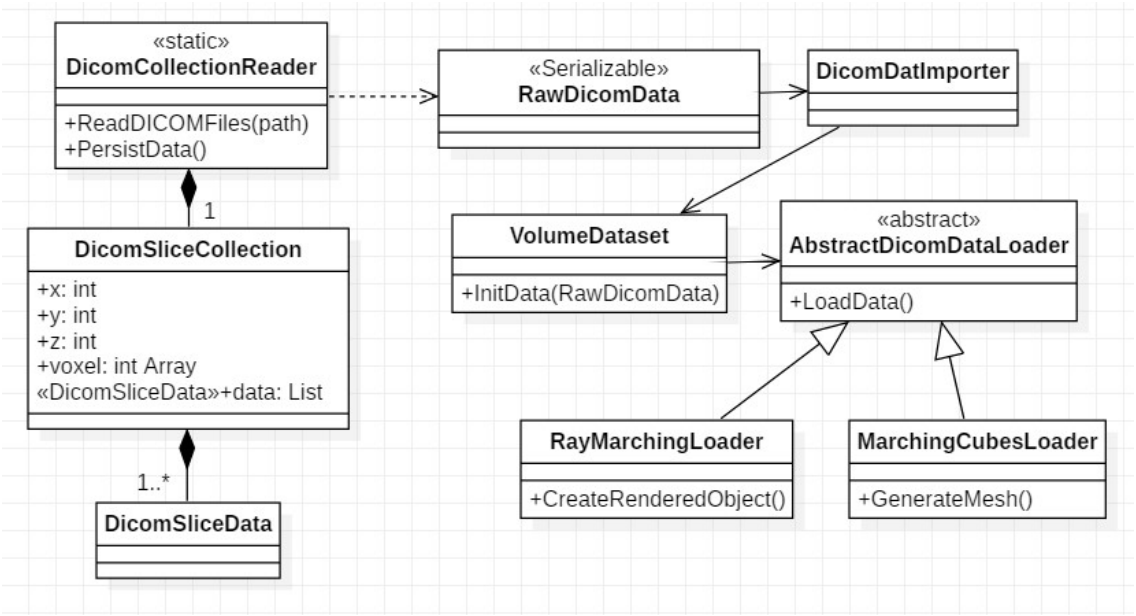


Abbildung q: Übersicht der Klassen

Mit dem .NET Framework System.IO werden die Dateien ermittelt und von der fo-dicom ausgelesen. Die ausgelesenen DICOM Dateien werden in einer Collection gecached. Die Collection enthält nur die nötigsten Informationen über die DICOM Serie.

```
foreach (var file in files)
{
   var dicomFile = Dicom.DicomFile.Open(file.FullName);
   DicomSliceData data = new DicomSliceData(dicomFile);
   dicomCollection.Add(data);
}
```

Code 4: Auslesen der DICOM Daten

Die DICOM Dateien können durch die Dateinamen oder das System falsch sortiert sein. Deshalb kann man die DICOM Dateien nicht nacheinander auslesen. Eine Sortierung ist hier erforderlich für eine korrekte Anzeige der Serie in 2D und 3D. Alle Slices besitzen eine Instance Number (0020,0013), die die Reihenfolge der Aufnahmen angibt.

```
dicomDatas = dicomDatas.OrderBy(dat => dat.InstanceNumber).ToList();
```

Code 5: Sortieren der DICOM Daten nach Instanznummer

DICOM Serien werden durch die SeriesInstanceUID (0020,000E) oder der SeriesNumber (0020,0011) unterschieden. Wenn DICOM Dateien in einem Ordner sind, kann die Unterscheidung so gewährleistet werden. Fälschlicherweise können in einer DICOM Serie mehrere Serien existieren. Durch falsche Einstellungen des Gerätes können Slices dieselbe Serien Zuordnung erhalten. Aus der sortierten DICOM Slices müssen die einzelnen Serien gefiltert werden, um die Berechnung mehrerer Volumen zu vermeiden. Die konkrete Trennung kann durch die Acquisition Number (0020,0012) erfolgen. Alle Slices, die eine gleiche Akquisitionsnummer besitzen, gehören zu derselben Serie.

```
dicomDatas = (from dat in dicomDatas
              where dat.AcquisitionNumber == firstAcquisitionNumber
              select dat).ToList();
```

Code 6: Ermitteln der Daten nach Akquisitionsnummer

4.3 Erzeugen der Voxeln mit dem Hounsfield-Skala

4.3.1 PixelData eines DICOM Slices

Die PixelData (7FE0,0010) ist der rohe Datenstrom des abgetasteten Pixels, woraus ein einzelnes DICOM Slice besteht. Jede `DicomSliceData` enthält seinen eigenen Byte Stream von PixelData die das `IPixelData` von fo-dicom implementiert.

```
// Create PixelData object to represent pixel data in dataset
DicomFile uncompressedClone =
dicomFile.Clone(DicomTransferSyntax.ExplicitVRLittleEndian);

DicomPixelData pixel_data =
DicomPixelData.Create(uncompressedClone.Dataset);

// Pixel data interface implemented by various pixel format classes
pixelData = PixelDataFactory.Create(pixel_data, 0);
```

Code 7: Erzeugen der PixelData

Auf die PixelData kann eine LUT Funktion angewendet werden oder manuell die Hounsfield-Skala berechnet werden.

4.3.2 Berechnung des Hounsfield-Skala

Die Formel für ein HU-Value lautet $hu = rawPixelValue * slope + intercept$. Die $rawPixelValue$ erhaltet man aus der PixelData. Die Rescale Slope (0028,1053) und Rescale Intercept (0028,1052) müssen je DICOM Slice ausgelesen werden. Somit kann die aktuelle HU-Value an einer bestimmten Position x und y berechnet werden.

```
rescaleSlope = dicomDataSet.Get<int>(DicomTag.RescaleSlope);
rescaleIntercept = dicomDataSet.Get<int>(DicomTag.RescaleIntercept);
...
int orgPixelValue = (int) pixelData.GetPixel(x, y);
int huValue = orgPixelValue * rescaleSlope + rescaleIntercept;
return huValue;
```

Code 8: Berechnung der Hounsfield Skala

4.3.3 Zusammenführung in ein Skalarfeld

Die Zusammenführung in ein dreidimensionales Skalarfeld ist abhängig von der Auflösung und Anzahl der DICOM Slices. Die Rows (0028,0010) und Columns (0028,0011) und die Anzahl der Slices müssen in einer dreifachen For-loop iteriert werden. Jede einzelnen HU-Value werden in einem Array Stream von Integers abgespeichert. Die Größe des Arrays ist das Produkt der Rows, Columns und Anzahl der Slices. Die For-loop ist der Kern der Anwendung, welche auch bei der Arbeit den Wendepunkt zum Erfolg gebildet hat. Denn mit diesem Stream lassen sich alle Volume Renderverfahren anwenden. Der Stream enthält die dreidimensionale Eigenschaft mit ihren Voxeln und alle Voxelwerte.

```
int xSize = dicomDataSet.Get<int>(DicomTag.Rows);
int ySize = dicomDataSet.Get<int>(DicomTag.Columns);
int zSize = collection.Count;
int[] voxels = new int[xSize * ySize * zSize];

for (int z = 0; z < zSize; z++)
{
    for (int x = 0; x < xSize; x++)
    {
        for (int y = 0; y < ySize; y++)
        {
            int huValue = dicomDatas[z].GetHUValue(x, y);
            voxels[x + y * xSize + z * (xSize * ySize)] = huValue;
        }
    }
}
```

Code 9: Erzeugen der Voxel mit HU

4.3.4 Ermittlung der Skalierung des Skalarfeldes einer DICOM Serie

Die Rows, Columns und Slice Anzahl repräsentieren die Dimensionen des Datenstreams, jedoch nicht die reale Skalierung des Modells bzw. des gescannten Mediums.

Die Rows und Columns müssen mit der PixelSpacing (0028,0030) multipliziert werden, um die realen Maße in mm zu erhalten. Die Anzahl der Slices ist die Tiefe der Voxel. Die Anzahl muss mit dem Abstand zwischen den Slices multipliziert werden. Den Abstand zwischen den Slices kann man durch die Image Position Patient (0020,0032) ermitteln. Der absolute Abstand ist der Abstand zwischen den beiden Vektoren des Image Position von benachbarten Slices. [28]

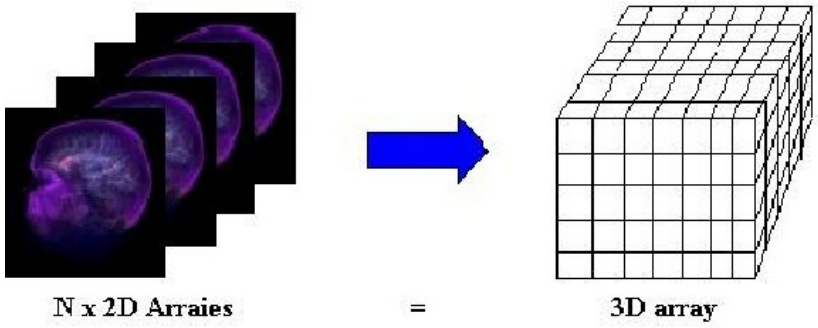


Abbildung r: Slices Sampling in 3D. [54]

```
// Absoulute distance between two slices
SpaceBetweenSlices = Vector3.Distance(data[2].ImagePosition,
data[1].ImagePosition);
```

Code 10: Ermitteln der Abstände zwischen Slices

Somit ist die Skalierung der Serie:

```
float xDimension = Rows     * PixelSpacing.x;
float yDimension = Columns  * PixelSpacing.y;
float zDimension = Count    * Space;

Scale = new Vector3(xDimension, yDimension, zDimension);
```

Code 11: Berechnung der Skalierung

4.4 Anwenden des Volume Renderings auf das Skalarfeld

4.4.1 Volume Raycasting als Unity Shader mit 3D Texturen

Für die Anwendung des Volume Raycastings wird das Unity Shader System genutzt, die in Echtzeit das Volumen mit der GPU rendert. Die berechneten Voxeln werden auf einen 1:1:1 Box (Quader) mit einer 3D Textur gemapped. Die 3D Textur wird als die Daten Textur genutzt. Der Wertebereich des Volumendatensatzes wird zwischen 0-1 normalisiert. Die eingetroffenen Strahlen erhalten bei einem Sample die Voxelwerte aus der Texturkoordinate. Die Unity Shader Syntax und Semantik wird nicht weiterhin vertieft. Eine grobe Erklärung der Implementierung: [45]

1. Zeichne die Back-Faces der Box, um die gesamte Box sehen zu können
2. Für jeden Knoten:
    a. Setzte die Startposition des Strahls auf die lokale Position des Knotens.
    b. Setzte die Richtung des Strahls zur Richtung zum Auge
    c. Teile den Strahl in n samples auf
    d. Für jeden sample (Von Startposition entlang des Strahls)
        i. Voxelwert: aktuelle Position als Texturkoordinate verwenden und den Wert aus der 3D-Textur holen
        ii. Voxelwert je nach Modus auswerten und als Ausgabefarbe als Pixel ausgeben

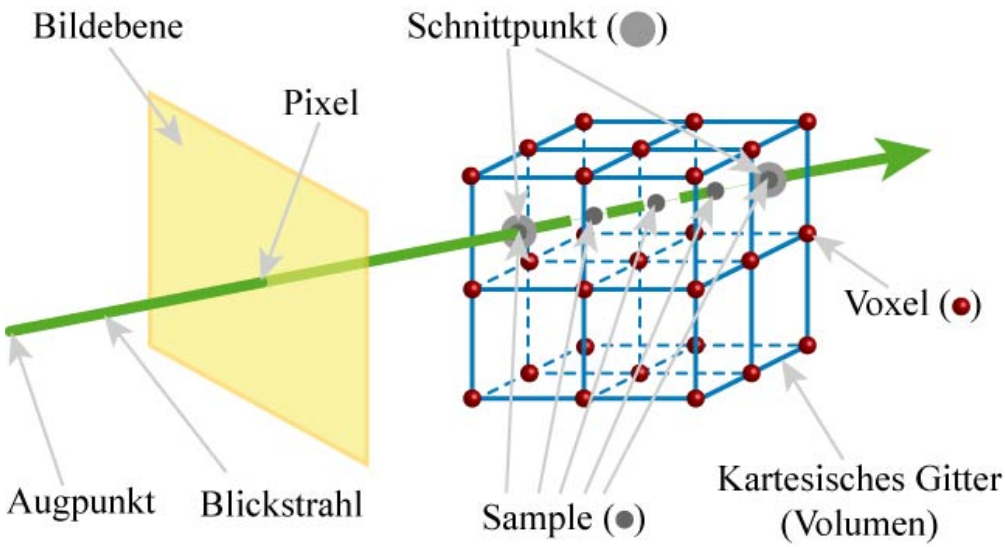


Abbildung s: Volume Raycasting Visualisierung [57]

4.4.1.1 Direct Volume Rendering

Der Direct Volume Rendering erhält die ausgegebene Pixelfarbe aus Transferfunktionen.

```
#if TF2D_ON
   float3 gradient = getGradient(currPos);
   float mag = length(gradient) / 1.75f;
   float4 src = getTF2DColour(density, mag);
#else
   float4 src = getTF1DColour(density);
#endif
```

Code 12: Transferfunktion für DVR

Die Farbe aus der eindimensionalen Transferfunktion wird lediglich aus einer bestimmten Position x aus einem einfachen Farbverlauf mit Alpha Werten entnommen. Für die zweidimensionale Transferfunktion wird zunächst der Gradient (Steigungsänderung) aus der Gradienten Textur geholt. Der Gradienten Textur wird ebenfalls als 3D Textur abgespeichert, um Berechnungen in Echtzeit zu vermeiden. Mit der Länge des Gradienten Vektors wird aus der zweidimensionale Transferfunktion der Wert entnommen. Der Farbwert wird mit dem alten Farbwert interpoliert, bis die Farbe eine hundertprozentige Transparenz besitzt.

```
col.rgb = src.a * src.rgb + (1.0f - src.a) * col.rgb;
col.a = src.a + (1.0f - src.a) * col.a;
```

Code 13: Farbe Interpolation DVR

4.4.1.1.1 Direct Volume Rendering mit Window Settings

Der Direct Volume Rendering wurde mit den Window Settings (siehe 3.3.1.4) erweitert. Die Windows Settings können beliebig in jedem Modus verwendet werden. Für den Proof of Concept wurde die lineare Transformation auf das Direct Volume Rendering angewendet.

```
float absVal = ApplyWindow(NormToAbs(density));
float normVal = AbsToNorm(absVal);
…
src.a = normVal;
```

Code 14: Anwendung der Window Settings

Die Dichte wird vor der Anwendung des Window Settings in den Absoluten Wert umgewandelt, um die lineare Transformation wie in den Standard (3.3.1.5) auszuführen und um einen Abrundungsfehler zu vermeiden. Der absolute Rückgabewert wird erneut normalisiert. Der normalisierte Rückgabewert stellt die Transparenz der Farbe aus der Transferfunktion dar.

4.4.1.2 Isosurface Rendering

Das Isosurface Rendering zeichnet den nächsten ersten Punkt zur Kamera, mit einem Skalarwert innerhalb der benutzerdefinierten Schwellenwerte. [45] [35] Den Farbwert kann man wieder aus der eindimensionalen Transferfunktion erhalten, ist jedoch keine Pflicht. Das Licht wird diffus berechnet. Nach der Farbberechnung wird der nächste Strahl abgefeuert, da hier das Sampling beim ersten Hit unterbrochen wird.

```
if (density > _MinVal && density < _MaxVal)
{
    float3 normal = normalize(getGradient(currPos));
    float lightReflection = dot(normal, lightDir);
    lightReflection =  max(lerp(0.0f, 1.5f, lightReflection), 0.5f);
    col = lightReflection * getTF1DColour(density);
    col.a = 1.0f;
    break;
}
```

Code 15: Isosurface Lichtberechnung

4.4.1.3 Maximum Intensity Projection

Der Maximum Intensity Projection zeigt die höchsten Voxelwerte entlang des aktuellen Strahls an.

```
for(..) {
    if (density > _MinVal && density < _MaxVal)
        maxDensity = max(density, maxDensity);
}
col = float4(1.0f, 1.0f, 1.0f, maxDensity);
```

Code 16: MIP Shader Code

4.4.1.4 Average Intensity Projection

Für das Bilden der durchschnittlichen Dichte müssen alle Dichtwerte entlang des Strahles gesammelt werden und durch die Anzahl der Samples geteilt werden.

```
countDensity = countDensity + density;
iDepth++;
…
float4 src = float4(1.0f, 1.0f, 1.0f, (countDensity / iDepth));
```

Code 17: Average Shader Code

4.4.1.5 Slice Renderer

Eine weitere viel genutzte Technik für das Volumen-Rendering ist das Slice-Rendering. Dabei handelt es sich um eine 2D-Visualisierung des Volumendatensatzes, bei der ein Slice des Volumens gerendert wird. Mit dem Slice Renderer wird gleichzeitig die Multiplanare Rekonstruktion in Echtzeit ermöglicht. Dazu kann man ebenfalls die 3D Daten Textur verwenden und mit einer bewegbaren Plane in Echtzeit die aktuellen Datensätze abfragen und anzeigen. Hier ist die Farbwiedergabe ebenfalls beliebig.

4.4.1.6 Querschnitt Renderer

Den Querschnitt des Volumens zu bilden kann sehr hilfreich sein, um bestimmte Bereiche des Volumens hervor zu heben oder bestimmte Organe zu schneiden. Durch den intensiven Voraufwand wird diese Implementierung relativ einfach. Lediglich müssen die Schnittstellen mit dem Volumen definiert werden (z.B. mit einem Quader). All diese Schnittpunkte werden auf volle Transparenz gesetzt bzw. kein Volume Rendering ausgeführt. Somit kann man beliebige Querschnitte bilden und das Volumen aufschneiden.

4.4.2 Marching Cubes mit Mesh Creation

Die Implementierung des Marching Cubes ist entnommen von Scrawk. [39] Die berechneten Voxeln werden mit den Dimensionen des Voxeln dem Marching Cubes Algorithmus übergeben. Abhängig von einem Isowert werden die Knoten durch Lookup Tabellen in einer Liste gesetzt. Aus den Knoten werden einmalig Meshes generiert. Das fertige Polygon Modell hat genau dieselbe Skalierung wie die Dimensionen der Voxeln. Das Modell wird auf die Skalierung der zuvor berechneten Skalierung der Serie gesetzt.

Diese Art der Implementierung auf der CPU ist für die immersive Darstellung sehr gut geeignet. Das Mesh wird einmal berechnet und dann nur noch als Polygon Modell angezeigt.

4.5 Persistieren von DICOM Serien mit Voxeln durch Serialization

4.5.1 Serialization in Unity C#

Serialization in C# ist unter anderem ein automatischer Speicherprozess als Byte-Stream von Datenstrukturen und Objektzuständen in eine Datei, in den Speicher oder in eine Datenbank. Das Auslesen, also Deserialization, kann diese binär auslesen und zum Laden rekonstruieren. Einige Untiy Funktionen laufen mit der Serialization, wie z.B. das Speichern und Laden der Scene, das Inspektor Fenster, das Initialisieren und die Prefabs.

4.5.2 Serialization der DICOM Serien

Mit der Serialization werden ausgelesene DICOM Daten serialisiert und als einzelne binäre DAT Dateien abgespeichert. Es werden nur die nötigen Attribute einer Serie gespeichert wie die Voxeln, Dimensionen und Abstände. Vor dem Volume Raycasting werden die Texturen aus den gespeicherten Informationen einmalig berechnet und wie gewohnt der Algorithmus angewendet. Für den Marching Cubes reichen die Voxeln und Dimensionen aus. Mit der Serialization wird der Schritt einzelne DICOM Daten auszulesen komplett übersprungen und somit in der Anwendung schneller geladen. Außerdem können komplette DICOM Serien immense Dateigrößen annehmen. Mit dieser Technik wird gleichzeitig eine Komprimierung durchgeführt.

```
BinaryFormatter bf = new BinaryFormatter();
FileStream file = File.Create(_datFilePath + "/" + _datName + ".dat");

RawDicomData rawDicomData = new RawDicomData(...);
bf.Serialize(file, rawDicomData);
file.Close();
```

Code 18: Serialization der DICOM Serie

Das Lesen von DICOM Serien und die Verwaltung der DAT Dateien läuft über das Editor Window Scripting von Unity. Es wird ein DICOM Folder ausgewählt und die DAT erzeugt. Die DAT Datei kann anschließend im Editor Mode oder Play Mode beliebig geladen und genutzt werden.

4.6 Die Immersive Darstellung der Voxeln mit Volumen Rendering

4.6.1 Wahl bei dem Head-Mounted Display System

Das genutzte Head-Mounted Display (HMD) System muss leistungsstark und eine hohe Auflösung bieten, um die detailreiche DICOM Serie in Echtzeit in einer guten bis hohen Framerate anzuzeigen. Selbstverständlich spielt hierbei auch der Rechner, auf dem die Anwendung läuft, eine sehr große Rolle. Je höher und leistungsstärker die HMD ist, desto mehr muss der Computer rechnen und leisten. Die Anwendung wurde hauptsächlich mit Oculus Rift S und Rift genutzt. Die Controller werden durch Infrarot Sensoren getracked, die direkt an das Headset angebracht sind. Die Rift S ist ein gutes Mittelmaß, um die Anwendung zu schnell und einfach zu testen. Die Rift S ist in der Handhabung vorteilhaft, da nur das Headset verkabelt werden muss.

4.6.2 Einbindung eines HMD-Systems in Unity

Die Einrichtung in Unity erfolgt durch das SteamVR Plugin. Das Plugin sorgt für eine HMD-System unabhängige Entwicklung und beschränkt nicht die Anwendung auf ein System. Durch eine einfache Installation des Plugins, ist die VR Umgebung bereits eingerichtet und kann sofort gestartet werden. Interaktionen mit dem Controller wurden selbst implementiert.

4.6.3 Leistungsoptimierung für die immersive Darstellung

DICOM Serien bestehen meistens aus mehreren Hundert Slices. Dadurch kann die dreidimensionale Auflösung in die Höhe steigen. Für die Optimierung der DICOM Serien ist es möglich die Auflösung durch die Reduzierung der Slices und mit Downsampling [46] einzelner Slices zu verringern.

4.6.3.1 Slice Reduzierung durch arithmetisches Mittel benachbarter Slices

Das selbst entwickelte Verfahren bildet das arithmetische Mittel der Pixelwerte zwischen zwei oder mehrerer Slices. Das Verfahren ist sehr einfach, aber kann die Slice Anzahl um 50% reduzieren (Tabelle 6). Die Verschmelzung der Slices sollte nur bei einem geringen Abstand zwischen den Slices erfolgen. Bei großen Abständen würde das Verfahren das Ergebnis verfälschen und das gerenderte Modell würde nicht mehr der Realität entsprechen. Dennoch öffnet diese Implementierung weitere Wege, um die Interpolation mit anderen Verfahren zu erweitern oder zu ersetzen, die ein gutes Ergebnis erzielen würde.

```
huValue = (huValue + nextDicomFile.GetHUValue(x, y)) / 2;
```

Code 19: Arithmetisches Mittel für HU Value

4.6.3.2 Downsampling einzelner Slices mit bilinearer Interpolation

Eine weitere Idee ist es die Auflösung einzelner Slices mit einer Interpolationsfunktion zu reduzieren. Die Fellow Oak Bibliothek bietet bereits eine Erweiterung an, Datenströme in eine gewünschte Auflösung mit der bilinearen Interpolation zu interpolieren. Auflösungen von 512x512 sind für DICOM Slices üblich.

```
compressedPixelBuffer = BilinearInterpolation.RescaleGrayscale(pixelBuffer,
rows, columns, newX, newY);
```

Code 20: Bilineare Interpolation des Pixel Buffers

4.6.3.3 Samplerate des Volume Raycastings verringern

Um beim Volume Raycasting in Echtzeit weniger Rechenaufwand zu erzielen, kann man die Samplerate des Algorithmus verringern. Je weniger Voxeln abgetastet werden, desto schneller wird gerendert. Die Samplerate beeinflusst nicht nur die Framerate, sondern auch das gerenderte Volumen. Unterschiedliche Sampleraten können auch eingesetzt werden, um die gewünschte Optik zu erzielen.

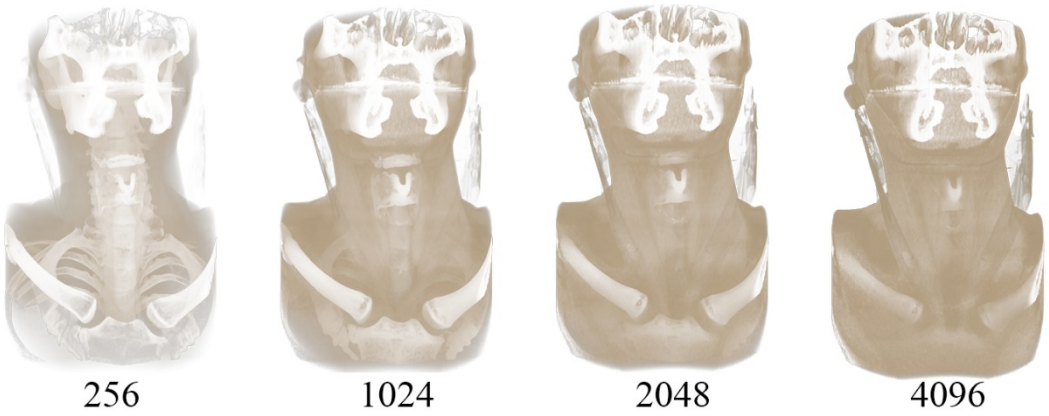


Abbildung t: Sample Rate Vergleiche bei selbem Volumen

4.6.3.4 Beispiel Rechnung

DICOM Serie [47]
Slices: 361, Rows: 512, Columns: 512, Abstand der Slices: 0.5mm
Anzahl der Voxel

Ohne Optimierung	⇒	$512 * 512 * 361 = 94.633.984$
Slice Reduzierung	⇒	$512 * 512 * 180 = 47.185.920$
Slice Reduzierung und bilineare Interpolation	⇒	$256 * 256 * 180 = 11.796.480$

Die Voxel Anzahl konnte mit beiden Optimierungen um ca. 88% verringert werden.

4.6.4 Virtuelle Umgebung der Anwendung

Die über Unity entwickelte Anwendung dient als Proof of Concept. Der Fokus soll lediglich auf das Volumen und die Bedienelemente liegen. Der Hintergrund ist in einem solidem schwarz, um einen Kontrast zwischen dem Volumen und Hintergrund zu schaffen. Die Anwendung startet sofort in der Virtuellen Umgebung und zeigt vorhandene DICOM Serien an (DAT). Für das Volume Rendering werden der Volume Raycasting in Echtzeit und der Marching Cubes auf der CPU angeboten.

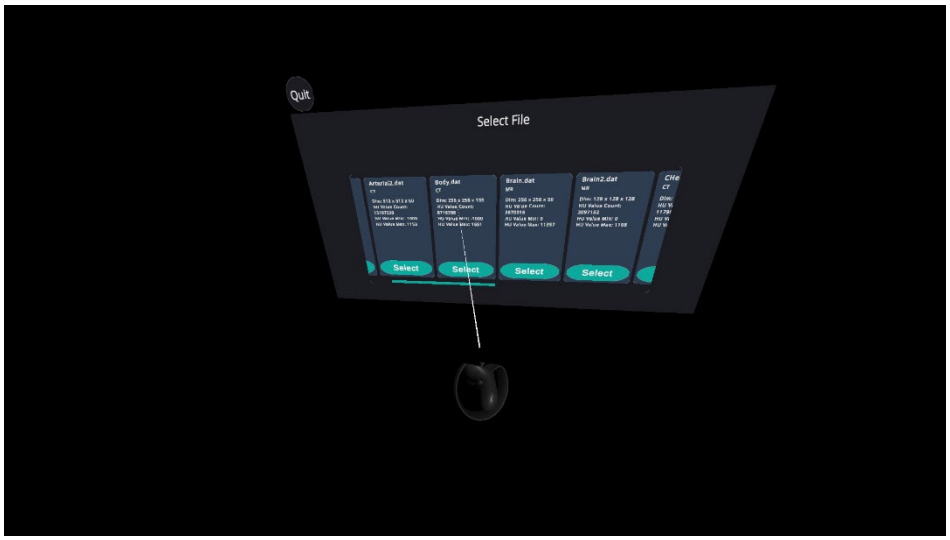


Abbildung u: Menü Auswahl in VR

Die Interaktionen mit dem Volume Raycasting sind beschränkt. Bei näheren Betrachtungen kann die Framrate sinken, sodass der Betrachter gestört wird. Je nach HMD und Rechner kann sich das unterscheiden. Daher kann man bei dem Volume Raycasting das Volumen mit dem Pointer rotieren und mit den Slidern die Schwellenwerte bzw. Window Settings ändern. Die Transferfunktionen sind vorgegeben. Die Anpassung einer Transferfunktion in Echtzeit wäre als Erweiterung der Anwendung möglich. Die Bounds des Volumens werden als Linien dargestellt.

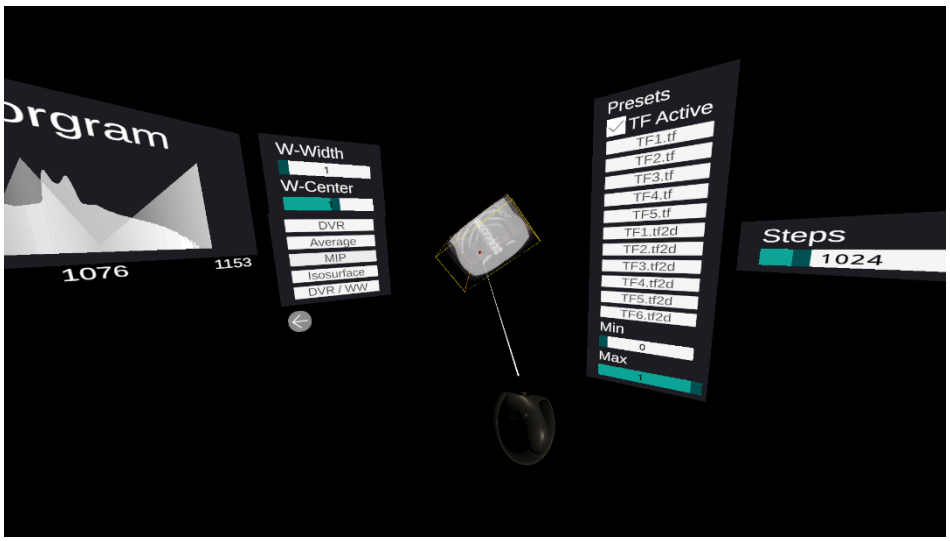


Abbildung v: Volume Raycasting mit Einstellungen

Der Marching Cubes berechnet einmalig das Volumen mit einem Standard Isowert. Der Isowert kann mit einem Slider geändert und das Volume neu berechnet werden. Das Rotieren mit dem Pointer ist ebenfalls möglich. Da der Marching Cubes nicht in Echtzeit berechnet wird, kann man sich dem Volumen nähern und für nähere Betrachtungen greifen.

Abbildung w: Marching Cubes in VR

Die DICOM Serien werden aus dem Streaming Assets Ordner ausgelesen. Unity kann in den Streaming Assets Ordner mit einem relativen Pfad zugreifen. Für die Erstellung der DICOM Serien wird der Editor benötigt. Als Erweiterung könnte es eine getrennte Software geben, die die DICOM Serien ausliest und persistiert. Das Auslesen der DICOM Serien in der VR Anwendung ist auch eine Möglichkeit.

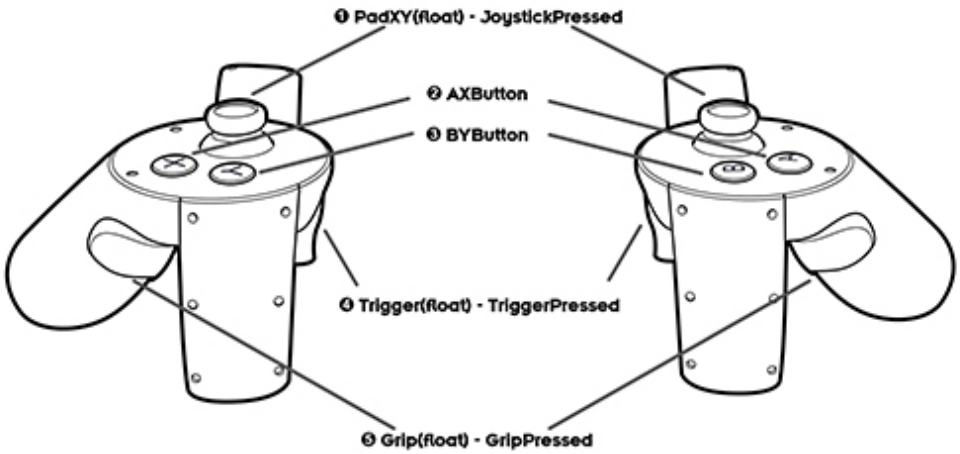


Abbildung x: Oculus Rift Controller [58]

Bedienungsanleitung:

A:	Die View nach der Kamera zentrieren
Rechter Trigger:	Pointer Klick / Rotieren
Rechter und linker Grip:	Uniform Skalierung des Volumens / Greifen bei MC

Kapitel 5: Benchmarking

5.1 Leistungsmessungen bei unterschiedlichen Renderverfahren

Allgemein ist es schwierig den Volume Raycasting flüssig mit hoher Framerate zu nutzen. Die Echtzeit Berechnungen auf der GPU sind zu aufwendig. Nach den Optimierungen war eine deutliche Verbesserung zu sehen. Die Messungen wurden mit derselben DICOM Serie [47] durchgeführt. Die zu rendernden Voxeln waren in einem sehr nahen Abstand zu der Kamera. Die Messungen wurden mit der Oculus Rift S und einer GeForce 970M durchgeführt.

5.1.1 Direct Volume Rendering

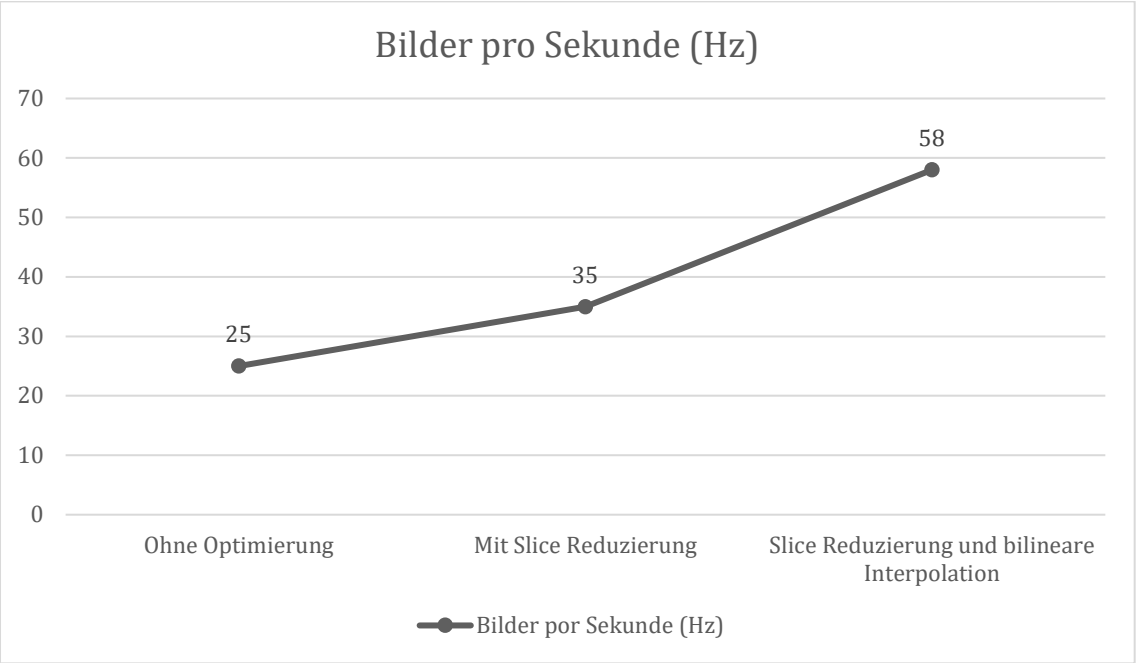


Tabelle 2: Framerate DVR

5.1.2 Isosurface Rendering

Bei dem Isosurface Rendering zeigen die Optimierungen kaum einen Unterschied, da die Framrate sehr stark vom Isowert abhängt und je nach angezeigte Voxeln mehr Lichtberechnungen durchführt werden. Bei dieser Messung wurde nur der Worst Case betrachtet. Die Maximale Framrate betrug 45 Hz.

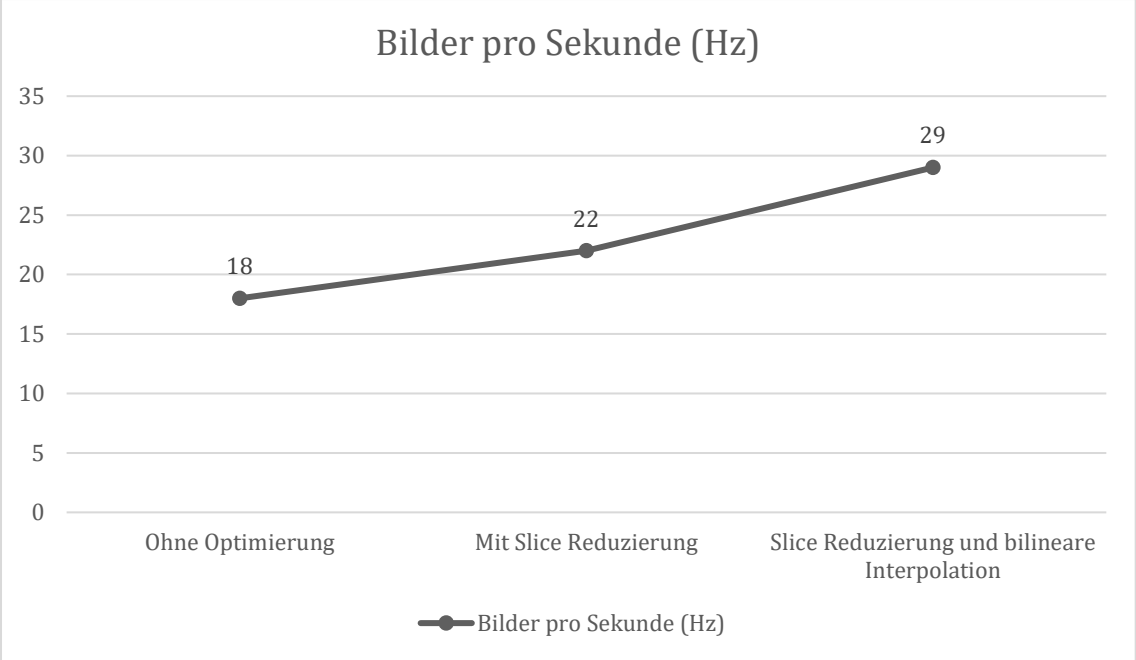


Tabelle 3: Framerate Isosurface Rendering

5.1.3 Maximum Intensity Projection

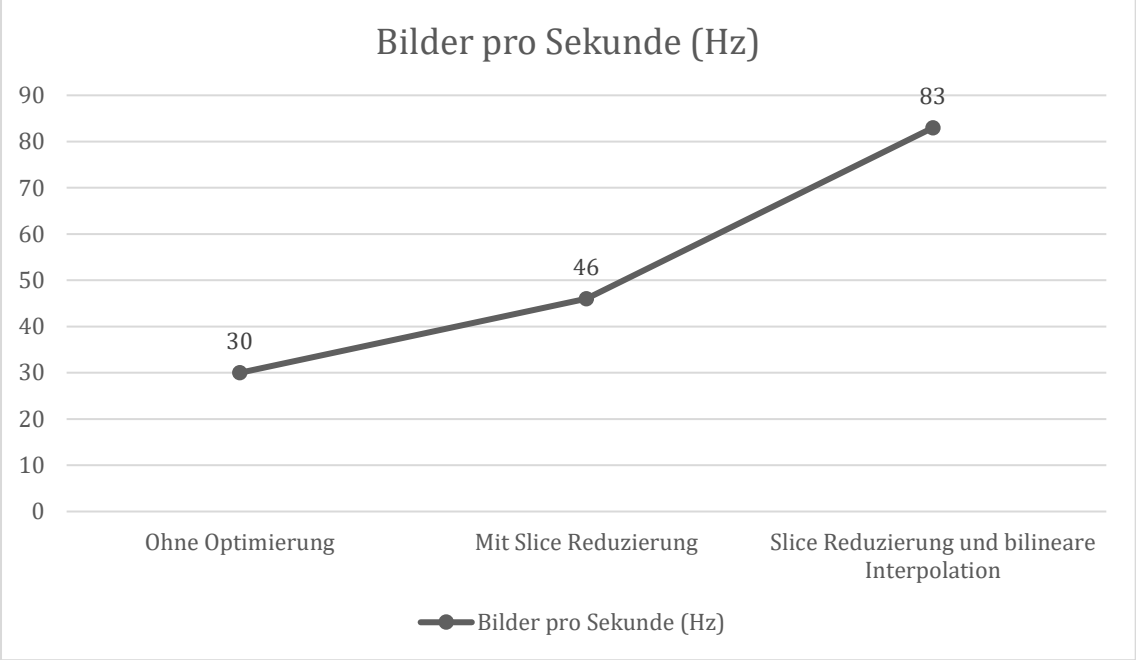


Tabelle 4: Framerate MIP

5.1.4 Direct Volume Rendering mit verschiedener Samplerate

Die Samplerate kann erhebliche Auswirkungen auf die Framerate haben. Die Messungen wurden mit einer Oculus Rift und einer DICOM Serie mit 901 Slices durchgeführt [48]. Die DICOM Serie wurde mit den vorgestellten Komprimier Verfahren gerendert.

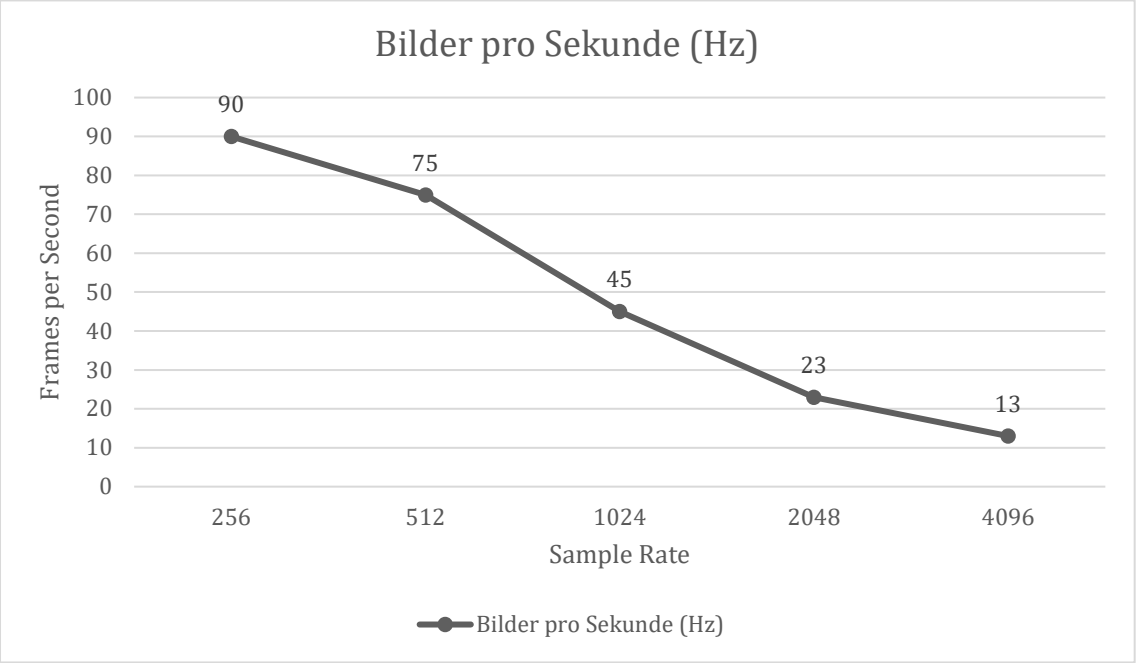


Tabelle 5: DVR mit Samplerate

5.1.5 Datengröße der persistierten DICOM Serien

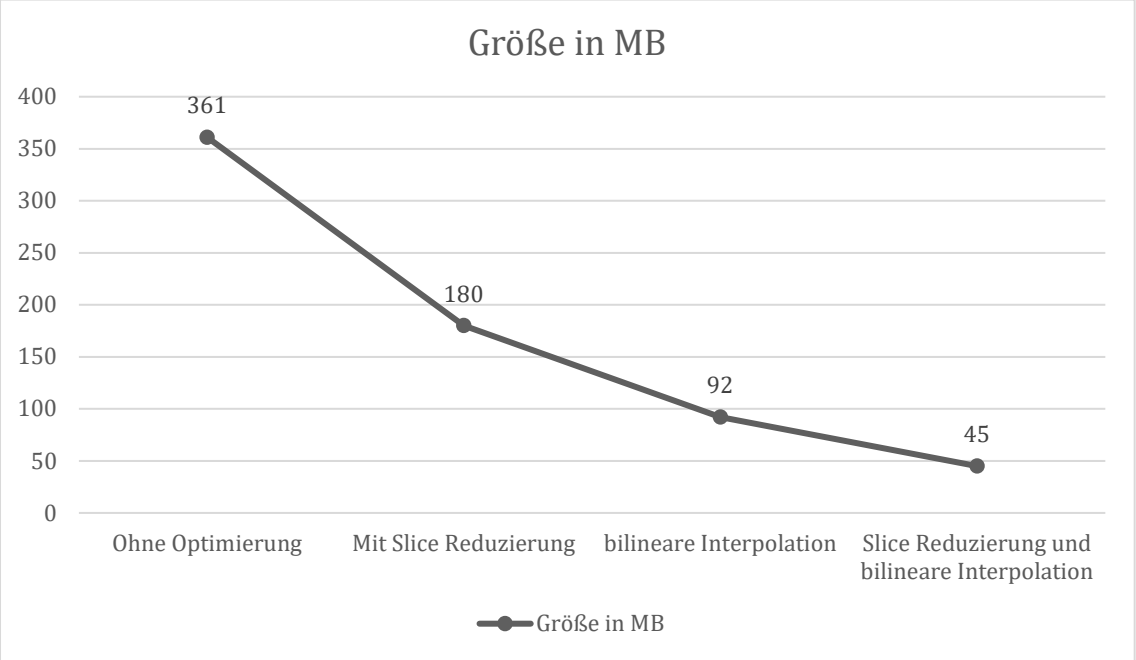


Tabelle 6: Datengröße Messungen

5.1.6 Marching Cubes

Das generiete Mesh durch den Marching Cubes weist eine konstante Framerate mit 80-100 Hz auf. Da der Marching Cubes nicht in Echtzeit gerendert wird, sondern einmal von der CPU berechnet wird, macht es hier Sinn, die Berechnungszeit zu messen. Die Messungen wurden im Editor Modus von Unity und einer Intel i7 7700HQ 2.80 GHz durchgeführt.

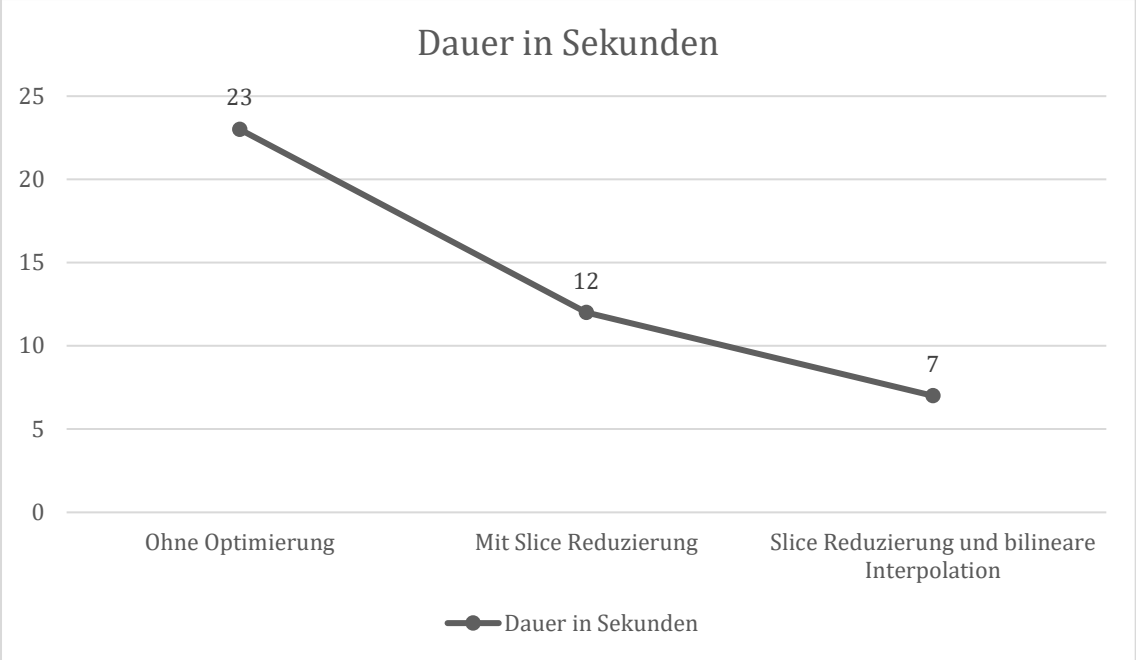


Tabelle 7: Marching Cubes Berechnungsdauer

5.2 Fazit

Beim DICOM Standard handelt es sich um ein komplexes Format. Der DICOM Standard umfasst viele Bereiche von einfachen Informationen bis zu der Medizin, Computergrafik, Bildtechnik und Visualisierungen. Um die DICOM Daten in 3D in Unity zu visualisieren sind einige Umwege nötig, wobei mit anderen Bibliotheken und Frameworks, wie VTK und ITK, es auf Anhieb leichter fallen würde. Für diese Arbeit reichte es aus, nur das DICOM Image Plane Module zu verstehen, um die drei dimensionalen Visualisierungen durchzuführen. Nach der Berechnung der Voxeln sind flexibel diverse Volume Renderverfahren einsetzbar.

In dieser Arbeit wurde festgestellt, dass die immersive Darstellung mit Echtzeit Volume Rendering eine deutlich große Herausforderung darstellen kann. Die vorgestellten Renderverfahren sind für diesen Zweck geeigent. Nach weiteren Optimierungen in der Implementierung und Entwicklung, können bessere Ziele erreicht werden.

Das Volume Raycasting setzt nur Farben aus einer Transferfunktion. Die Erweiterung in eine realitätsnahe Lichtberechnungen mit Textur- und Materialbasierenden Renderings, kann deutliche Verbesserungen für Diagnosen und Behandlungen bieten. Das sogenannte Photo-Realistic Volume Rendering baut darauf auf, mehrere Lichtquellen zu nutzen, bestimmten Segmenten Materialien mit Texturen zu setzten. Das Photo-Realistic Volume Rendering könnte eine Kombination aus dem Raymarching und Raytracing durchführen und somit auch Materialienberechnungen mit Reflektionen und Refraktionen anzeigen. Das Isosfurface Rendering ist bereits eine einfache Lichtberechnung, die man mit mehreren Lichtarten erweitern könnte und Reflektions- und Refraktionsberechnungen hinzufügen. [49]

Der Marching Cubes könnte auf der GPU entwickelt werden, um Echtzeit Rendering anzubieten. Die Optimierung Ansätze für den Marching Cubes sind durchaus machbar und sinnvoll.

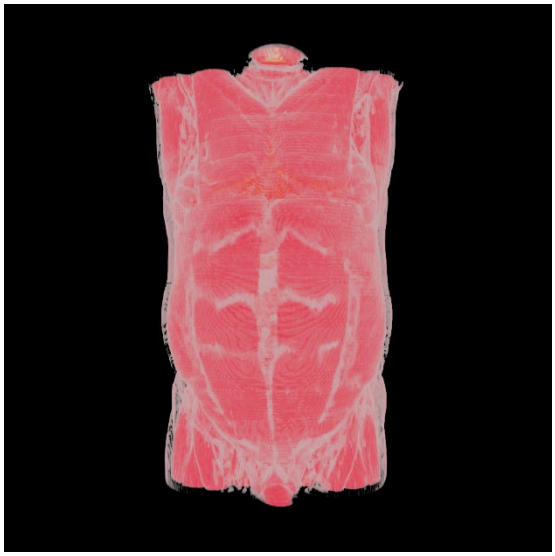
Die Verwaltung der DICOM Daten kann verbessert werden. Die DICOM Serien und persistierten DAT Dateien werden lokal in den Speicher geladen. Das kann zu unnötigen Speicherbelastung führen. Ein asynchrones Laden wäre schneller und benutzerfreundlicher. Fellow Oak, Unity und C# bieten diese Techniken an. Um es für Patienten und Ärzte sinnvoller zu gestalten, könnte man die Patienten Informationen ebenfalls anzeigen.

Einige DICOM Serien sind fehlerhaft: z.B. Fehlende Tags oder beschädigte Dateien. Während DICOM Viewer diese DICOM Serien erkannt und korrekt angezeigt hat, wurden in der eigenen Entwicklung Fehler sichtbar. Weitere Recherchen wären notwendig für eine Fehlerquellen Analyse.

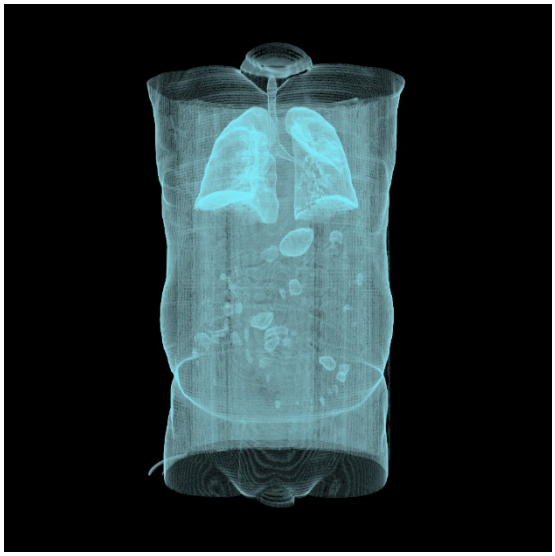
Die Anwendung für die Virtuelle Realität kann in der Gestaltung und Bedingung verbessert werden. Einige UX Elemente sind für Laien nicht direkt verständlich. Außerdem würde es mehr Sinn machen, die Transferfunktionen in der Anwendung manuell zu setzten. Die vorgegebenen Presets sind nicht immer passend zu der aktuellen Serie.

Insgesamt war der Proof of Concept erfolgreich und die Anwendung ist in allen Bereichen erweiterbar.

Anhang



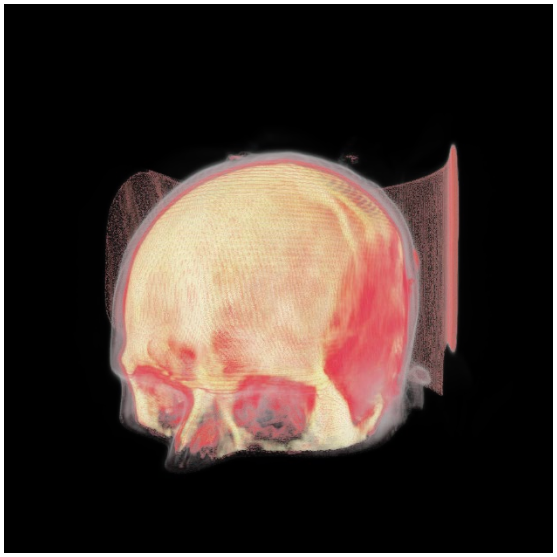
Anhang 4: DVR mit 1D Transferfunktion. Muskeln



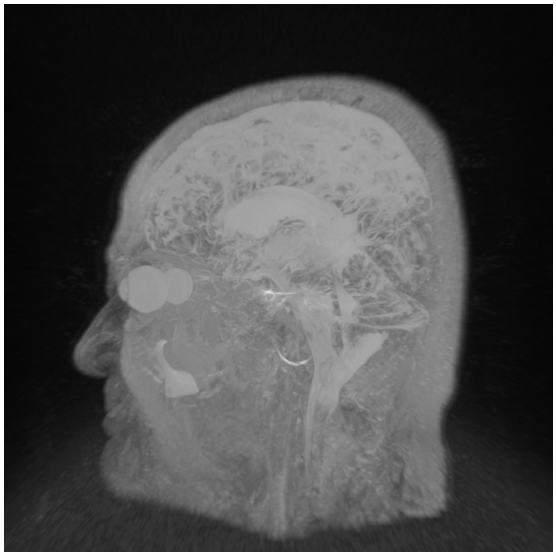
Anhang 3: DVR mit 2D Transferfunktion: Lunge

Anhang 2: DVR Knochen und Luft

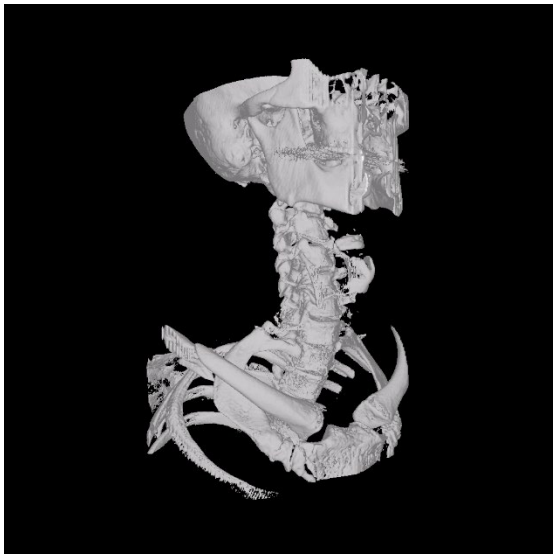
Anhang 1: DVR mit Window Settings



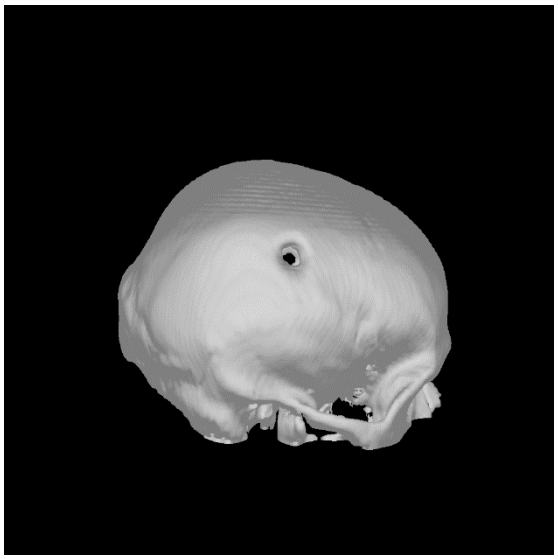
Anhang 8: DVR mit niedriger Samplerate



Anhang 7: MIP eines Kopfes.



Anhang 6: Isosurface vom Knochen



Anhang 5: Isosurface Schädel mit Loch

Verweise

[1] L. Soler, S. Nicolau, P. Pessaux, D. Mutter und J. Marescaux, Real-time 3D image reconstruction guidance in liver resection surgey, http://hbsn.amegroups.com/: Hepatobiliary Surgery and Nutrition, 2014.

[2] Kitware, „VTK," [Online]. Available: https://vtk.org/about/. [Zugriff am März 2020].

[3] Kitware, „ITK," [Online]. Available: https://itk.org/about/#overview. [Zugriff am März 2020].

[4] Fellow Oak DICOM, „github Fellow Oak DICOM," [Online]. Available: https://github.com/fo-dicom/fo-dicom.

[5] National Electrical Manufacturers Association (NEMA), „www.dicomstandard.org," [Online]. Available: https://www.dicomstandard.org/about/. [Zugriff am April 2020].

[6] DICOM Standard, „DICOM PS3.4 2020b - Service Class Specifications," in *C.6 SOP Class Definitions*, pp. 126-127.

[7] „Dicom Standard," [Online]. Available: https://www.dicomstandard.org/current/.

[8] DICOM Library, „DICOM Library," [Online]. Available: https://www.dicomlibrary.com/dicom/sop/. [Zugriff am März 2020].

[9] J. M. M. •. C. P. L. M. P. •. D. S. Charles E. Kahn, „Informatics in Radiology," in *An Information Model of the*, January-February 2011, p. 298.

[10] DICOM Standard, „SOP Class Definitions," [Online]. Available: http://dicom.nema.org/dicom/2013/output/chtml/part04/sect_C.6.html.

[11] J. M. M. •. C. P. L. M. P. •. D. S. Charles E. Kahn, „Informatics in Radiology," in *An Information Model of the*, January-February 2011, p. 299.

[12] DICOM®, „DICOM PS3.6 2020b - Data Dictionary," 2020.

[13] Medixant, „radiantviewer," Medixant, [Online]. Available: https://www.radiantviewer.com/.

[14] MicroDicom, „MicroDicom," [Online]. Available: http://www.microdicom.com/.

[15] Wikipedia, „Multiplanare Reformation," [Online]. Available: https://de.wikipedia.org/wiki/Multiplanare_Reformation. [Zugriff am März 2020].

[16] Kitware, „Kitware," [Online]. Available: https://www.kitware.eu/product/activiz. [Zugriff am März 2020].

[17] Ö. C. ÇELEBİ, „3D multiplanar reconstruction and rendering of medical and non-medical volumetric data.," 18 Juni 2011. [Online]. Available: http://www.byclb.com/TR/Tutorials/volume_rendering/Index.aspx. [Zugriff am März 2020].

[18] „Scalar," 7 Feburar 2011. [Online]. Available: https://www.encyclopediaofmath.org/index.php/Scalar. [Zugriff am März 2020].

[19] H. P. Marcos Vinicius Mussel Cirne, „Marching Cubes Technique for Volumetric Visualization," *Journal of the Brazilian Computer Society,* p. 2, September 2013.

[20] V. Özen und B. Gökirmak, Interviewees, *Recherche und persönliche Befragung Medizintechniker bzgl. DICOM Daten.* [Interview]. 28 Januar 2020.

[21] Innolitics, „Pixel Spacing Attribute," Innolitics, [Online]. Available: https://dicom.innolitics.com/ciods/ct-image/image-plane/00280030.

[22] DICOM Standard, C.7.6.2.1 Image Plane Attribute Descriptions.

[23] DICOM Standard, „VOI LUT Function Attribute," Innolitics, [Online]. Available: https://dicom.innolitics.com/ciods/ct-image/voi-lut/00281056. [Zugriff am März 2020].

[24] DICOM Standard, „Y VOI LUT Functions (Informative)," [Online]. Available: http://dicom.nema.org/DICOM/2013/output/chtml/part17/chapter_Y.html.

[25] M. S. P. C. Reinerth, Interviewee, *Sigmoind Funktionen in der Datenauswertung.* [Interview]. 10 März 2020.

[26] J. Broder und R. Preston, „Imaging the Head and Brain," in *Diagnostic Imaging for the Emergency Physician*, 2011.

[27] M. McCormick, „DICOM Rescale Intercept / Rescale Slope and ITK," 6 Oktober 2014. [Online]. Available: https://blog.kitware.com/dicom-rescale-intercept-rescale-slope-and-itk/. [Zugriff am März 2020].

[28] H. E. T. İ. S. K. G. Ç. A.H. YURTTAKAL, „3D Visualization Thyroid CT Images Using Marching Cubes Algorithms," in *International Conference on Advanced Technologies, Computer Engineering and Science (ICATCES'18)*, Safranbolu, Turkey, May 11-13, 2018.

[29] L. C. P. H. Robert A. Drebin, Volume Rendering, San Rafael, CA, 1988.

[30] M. Lavik, „Volume rendering, implemented in Unity3D," 19 Januar 2020. [Online]. Available: https://github.com/mlavik1/UnityVolumeRendering. [Zugriff am Januar 2020].

[31] A. C. Telea, Data Visualization: Principles and Practice, Second Edition, September: AK Peters, 2014 .

[32] J. S. C. E. L. R. G. S. K. THOMAS WISCHGOLL, Validation of Image-Based Method for Extraction of Coronary Morphometry, 2007.

[33] computer graphics laboratory ETH zürich, Stuttgart Visualization Course: Direct Volume Rendering, 2006.

[34] G. K. a. C. H. JOE KNISS, „Multi-Dimensional Transfer Transfer Functions for Volume Rendering," in *Johnson/Hansen: The Visualization Handbook*, 2004, pp. 184-185.

[35] M. Lavik, „Volume Rendering in Unity," 19 Januar 2020. [Online]. Available: https://matiaslavik.wordpress.com/2020/01/19/volume-rendering-in-unity/.

[36] DICOM Standard, „DICOM PS3.3 2020b - Information Object Definitions," in *Section C.11.2.1.2 Window Center and Window Width*.

[37] W. E. L. u. H. E. Cline, MARCHING CUBES: A HIGH RESOLUTION, Schenectady, New York 12301, 1987.

[38] P. Bourke, „Polygonising a scalar field," Mai 1994. [Online]. Available: http://paulbourke.net/geometry/polygonise/. [Zugriff am März 2020].

[39] J. Scrawk, „Marching cubes in Unity," 24 August 2019. [Online]. Available: https://github.com/Scrawk/Marching-Cubes. [Zugriff am März 2020].

[40] H. Y. Timothy S. Newman, „A survey of the marching cubes algorithm," in *Computers & Graphics 30 (2006)*, Department of Computer Science, University of Alabama in Huntsville, Huntsville, AL 35899, USA, ScienceDirect, 2006, pp. 860-861.

[41] M. Cirne und H. Pedrini, „2.4.1 k-d Tree," in *Marching Cubes Technique for Volumetric Visualization Accelerated with Graphics Processing Units*, 2017, p. 5.

[42] S. D. N. T. K. P. J. A. S. J. M. S. A. G. Gavin Wheeler, „Virtual interaction and visualisation of 3D medical imaging data with VTK and Unity," Healthcare Technology Letters, 10th August 2018.

[43] 3D Heart project, „Gitlab,“ NIHR i4i 3D Heart project, [Online]. Available: https://gitlab.com/3dheart_public/vtktounity.

[44] Cureos, „Assetstore Unity fo-dicom,“ [Online]. Available: https://assetstore.unity.com/packages/tools/integration/fo-dicom-60902.

[45] M. Heyde, „3.3 Volumen-Raycasting,“ in *Masterarbeit: Direktes Volumen-Rendering mehrerer überlappender*, Dresden, TECHNISCHE UNIVERSITÄT DRESDEN, 2015, pp. 10-14.

[46] M. Heyde, „5.4 Optimierungen,“ in *Direktes Volumen-Rendering mehrerer überlappender Schichtenstapel*, Dresden, TECHNISCHE UNIVERSITÄT DRESDEN, 2015, pp. 42-43.

[47] DICOM Library, „DICOM Study CT,“ [Online]. Available: https://www.dicomlibrary.com/?manage=1b9baeb16d2aeba13bed71045df1bc65. [Zugriff am April 2020].

[48] Object Research Systems, „theobjects - Sample Datasets,“ Object Research Systems, [Online]. Available: https://www.theobjects.com/dragonfly/learn-sample-datasets.html. [Zugriff am 05 April 2020].

[49] T. Kroes, F. H. Post und C. P. Botha, Exposure Render An Interactive Photo-Realistic Volume Rendering Framework, 2012.

[50] P. Morel, „MSPT : Motion Simulator for Proton Therapy,“ November 2014. [Online]. Available: https://www.researchgate.net/publication/277596746_MSPT_Motion_Simulator_for_Proton_Therapy.

[51] M. M. Movania, „Implementing volume rendering using single-pass GPU ray casting,“ in *OpenGL Development Cookbook*, 2013.

[52] M. Häggström. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=81072500.

[53] Lorensen, Juli 2007. [Online]. Available: http://marchingcubes.org/index.php/File:MCCases.png.

[54] K. D. Tönnies, „Basics for Volume Rendering in Medicine,“ in *State of The Art for Volume Rendering*, University of Magdeburg, p. 2.

[55] DICOM Standard, „C.7.6.2 Image Plane Module | 10.7.1.3 Pixel Spacing Value Order and Valid Values,“ in *DICOM PS3.3 2020b - Information Object Definitions*.

[56] S. Davidson, „Transfer Function Transfer functions make volume data visible,“ [Online]. Available: https://slideplayer.com/slide/13944836/.

[57] D. Röttger, „Visualisierung und Volumenrendering 2,“ Universit□at Koblenz, 2012.

[58] ventuz, [Online]. Available: https://www.ventuz.com/support/help/V5_04/NodeVrTracked.html.

[59] G. Development, „Voxels with Both Round and Square Features,“ vbstudio, 7 April 2019. [Online]. Available: https://vbstudio.hu/en/blog/20190407-Voxels-with-Both-Round-and-Square-Features. [Zugriff am 29 März 2020].