

Implementierung eines Clustering-basierten Verfahrens zur Segmentierung von Volumenmodellen

Lukas Diewald

An der Fakultät für Informatik
Institut für Anthropomatik und Robotik (IAR) -
Intelligente Prozessautomation and Robotik (IPR)

Erstgutachter: Prof. Dr.-Ing. habil. Björn Hein
Zweitgutachter: Prof. Dr.-Ing. Torsten Kröger
Betreuernder Mitarbeiter: Christian Kunz M.Sc.

04. Juni 2018 – 28. September 2018

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, 10. Oktober 2018

.....
(Lukas Diewald)

Zusammenfassung

Die Visualisierung von Daten spielt auch bei medizinischen Anwendungen eine wichtige Rolle. Im Rahmen des HoloMed Projektes wird ein Arzt bei einer Ventrikelpunktion, einem sehr präzise auszuführenden neurochirurgischen Eingriff, unterstützt, indem ihm das Ventrikelsystem mithilfe einer AR-Brille angezeigt wird. Dabei entsteht die Aufgabe das System anhand von CT-Daten zu segmentieren.

Im Rahmen dieser Arbeit wurde eine Transferfunktion implementiert, die das Ventrikelsystem segmentiert und hervorhebt. Dabei wurde das Clustering-basierte Verfahren von Nguyen [1] verwendet. Dies wendet ein zweistufiges Clustering an. Dabei werden zunächst die LH-Werte des Volumens berechnet und ein Histogramm erstellt. Die LH-Werte beschreiben die Grenzen von verschiedenen Materialien im Volumen. Auf den Histogramm findet anschließend der erste Clusteringschritt statt. Danach werden die entstandenen Cluster nochmals anhand ihrer räumlichen Informationen im Volumen geclustert. Die beiden Clusteringschritte benutzen dabei jeweils *Meanshiftclustering*. Aus den entstehenden finalen Clustern kann das Ventrikelsystem aus mehreren berechneten Clustern zusammengesetzt werden.

Die Implementierung im Rahmen dieser Arbeit war erfolgreich. Es ist nun möglich die Ventrikelsysteme von gesunden Menschen zu segmentieren.

Stickwörter: *HoloMed, Clusteringbasiertes Verfahren, Implementierung*

Inhaltsverzeichnis

1. Einleitung	1
1.1. Motivation	1
1.2. Problemstellung	1
1.3. Zielsetzung	2
1.4. Aufbau der Arbeit	2
2. Stand der Wissenschaft und Technik	3
2.1. Eindimensionale Verfahren	3
2.2. Zweidimensionale Verfahren	4
2.3. Raum-basierte Verfahren	4
2.4. Machine Learning Verfahren	6
2.5. Benutzer-zentrische Verfahren	7
2.6. Clustering-basierte Verfahren	8
2.7. Wahl des Verfahrens	9
3. Methoden	11
3.1. Gradient	11
3.2. LH-Werte	13
3.3. LH-Clustering	14
3.4. Räumliches-Clustering	14
3.5. Hierarchisches-Clustering	15
4. Design	16
5. Implementierung	23
5.1. Gradient	23
5.2. LH-Werte	23
5.3. LH-Clustering	25
5.4. Räumliches-Clustering	27
5.5. Unity	28
6. Ergebnisse	30
6.1. Visualisierung	30
6.1.1. Ventrikelsystem	30
6.1.2. Visualisierungen	30
6.1.3. Datensätze	32
6.1.4. Auswertung	33
6.2. Nutzerstudie	35
6.3. Berechnungszeit	37
7. Fazit	39
8. Ausblick	40
Literaturverzeichnis	41
Anhang	43

A. Technische Dokumentation über die Benutzung der Segmentierungs-tools der MITK-Workbench	43
---	-----------

Abbildungsverzeichnis

1.	Fotografie einer Ventrikelpunktion Entnommen aus dem Forschungsprojekt HoloMed [2]	1
2.	Darstellung der lokalen 4x4x4 Nachbarschaft Entnommen aus [3]	11
3.	Beispiel Gradienten	12
4.	Beispiel LH-Werte	14
5.	UML-Diagramm des Volumens	17
6.	UML-Diagramm über die Module	18
7.	Aktivitätsdiagramm über den Prozessablauf	20
8.	2D Beispiel für die Verkleinerung des Volumens	21
9.	LH-Histogramm	27
10.	Parameterfelder in Unity	29
11.	Zeichnung des Ventrikelsystems Frei nach [4]	30
12.	Visualisierung des ersten normalen Ventrikelsystems von der Seite	31
13.	Visualisierung des ersten normalen Ventrikelsystems von unten	31
14.	Visualisierung des zweiten normalen Ventrikelsystems von der Seite	31
15.	Visualisierung des zweiten normalen Ventrikelsystems von unten	31
16.	Visualisierung des Ventrikelsystems mit Atrophie von der Seite	32
17.	Visualisierung des Ventrikelsystems mit Atrophie von unten	32
18.	Visualisierung des ersten normalen Ventrikelsystems von der Seite mithilfe von Unity	34
19.	Visualisierung des ersten normalen Ventrikelsystems von unten mithilfe von Unity	34
20.	Visualisierung des ersten normalen Ventrikelsystems von oben mithilfe von MITK	35
21.	Visualisierung des ersten normalen Ventrikelsystems von vorne mithilfe von MITK	35
A.1.	Ordner finden	43
A.2.	Überblick über die Workbench	44
A.3.	Transferfunktionstool der MITK-Workbench	44
A.4.	Statistiktool der MITK-Workbench	45
A.5.	Segmentierungstools der MITK-Workbench	45
A.6.	3D-Segmentierungstools der MITK-Workbench	45
A.7.	Region Growing Tool der MITK-Workbench	46
A.8.	Anpassen der Parameter des Region Growing Tools der MITK-Workbench	46
A.9.	Filtertool der MITK-Workbench	47
A.10.	Darstellungstool der MITK-Workbench	47

Tabellenverzeichnis

1.	Verwendeten Datensätze	32
2.	Ergebnisse des Interviews mit einem Arzt	34
3.	Durchschnittlichen Ergebnisse des NASA-TLX Bogens	36
4.	Überblick über die Berechnungszeiten der verschiedenen Volumengrößen	37
5.	Überblick über die Berechnungsraten der verschiedenen Volumengrößen .	37

1. Einleitung

1.1. Motivation

Computergestützte Verfahren werden in der Medizin immer wichtiger. Sie erleichtern dem Arzt seine Arbeit und senken die Wahrscheinlichkeit, dass bei einer Operation ein Fehler gemacht wird.

Eine Routineprozedur der Neurochirurgie ist die Punktions des Ventrikelsystems zur Drainage von Liquor. Diese wird häufig nötig, wenn ein Patient beispielsweise unter einer Gehirnblutung, einem Schädelhirntrauma oder einem Schlaganfall leidet.

Um die Punktions durchzuführen, muss der Chirurg eine Bohrlochreparation am sogenannten Kocherpunkt durchführen. Der Arzt muss dabei anhand äußerer anatomischer Landmarker diesen Punkt auf wenige Zentimeter genau finden und die Stichrichtung der Punktions ausmachen.

1.2. Problemstellung

Das aktuelle Vorgehen bei einer Ventrikelpunktion ist sehr fehleranfällig. So kommt es nur in zwei Dritteln aller Operationen zu einem optimalen Ergebnis, wofür oftmals mehrfach punktiert werden muss. In Abbildung 1 ist ein Bild eines solchen Eingriffs zu sehen. Es ist zu erkennen, wie der Chirurg mit dem linken Zeigefinger den inneren Augenwinkel zur Orientierung abtastet, um die Punktions im richtigen Winkel auszuführen.



Abbildung 1: Fotografie einer Ventrikelpunktion
Entnommen aus dem Forschungsprojekt HoloMed [2]

1.3. Zielsetzung

Im Rahmen des HoloMed Projektes wird daran gearbeitet, den Chirurg bei diesem Eingriff zu unterstützen. Der Plan ist, dass der Arzt mithilfe einer AR-Brille angezeigt bekommt, wo sich das Ventrikelsystem befindet und somit mit einer niedrigeren Fehlerwahrscheinlichkeit die Operation durchführen kann. Daraus ergibt sich die Aufgabe das Ventrikelsystem anhand von CT-Daten des Gehirns zu segmentieren und hervorzuheben.

Zum Lösen dieser Problemstellung eignen sich Übergangsfunktionen, auch Transferfunktionen genannt. Allgemein gesprochen ordnet eine Transferfunktion volumetrischen Daten optische Eigenschaften zu. Diese Aufgabe teilt sich in zwei Bereiche auf. Zum einen muss die Funktion entscheiden, welche Daten gezeigt werden und zum anderen, wie diese hervorgehoben werden, beispielsweise welche Farb- und Okklusionswerte diese erhalten sollen. Dabei ist der erste Teil wichtiger und schwieriger umzusetzen.

Das Ziel dieser Bachelorarbeit ist es, eine geeigneten Transferfunktion zu implementieren, um das Ventrikelsystem in CT-Daten segmentieren und hervorheben zu können.

1.4. Aufbau der Arbeit

Die Arbeit teilt sich folgendermaßen auf. In Kapitel 2, Stand der Wissenschaft und Technik, wird ein Überblick über existierende Transferfunktionen und deren Funktionsweisen gegeben. Kapitel 3, Methoden, beschäftigt sich mit der in dieser Arbeit verwendeten Transferfunktion. In Kapitel 4, Design, wird das Softwaredesign der Implementierung erklärt. In Kapitel 5, Implementierung, die Umsetzung dessen besprochen. In Kapitel 6, Ergebnisse, werden die Resultate dieser Arbeit gezeigt und evaluiert. In Kapitel 7, Fazit, werden die Ergebnisse diskutiert. In Kapitel 8, Ausblick, werden mögliche Verbesserungen aufgezeigt.

2. Stand der Wissenschaft und Technik

Das Gebiet der Transferfunktionen ist bereits weit erforscht und es existieren viele unterschiedliche Methoden und Herangehensweisen, um medizinische Daten abhängig von verschiedenen Problemstellungen passend darzustellen.

Dabei ist zwischen den Levels an Automation eines Systems zu unterscheiden. Es gibt vollautomatische Verfahren, bei denen keine Interaktion mit dem Benutzer von Nöten ist, semiautomatische Verfahren, bei denen der Benutzer noch an gewissen Stellschrauben drehen kann, um das Ergebnis zu beeinflussen und manuelle Verfahren, bei denen der Anwender vieles selbst einstellen muss.

Transferfunktionen unterscheiden sich weiterhin in ihrer Dimensionalität. Es existieren ein- zwei- und allgemein -mehrdimensionale Transferfunktionen. Des Weiteren sind die Grundlagen, auf denen die Berechnungen ruhen teils völlig verschieden. Manche Verfahren basieren auf den Intensitätswerten oder deren Änderung im gegebenen Volumen. Andere basieren auf der Größe der Features, die für den Nutzer von Interesse sind. Wieder andere sind Benutzer-zentrisch orientiert oder wenden Machine Learning an, um zu einem gewünschten Ergebnis zu gelangen.

In diesem Abschnitt wird ein Überblick über die unterschiedlichen Vorgehensweisen von Transferfunktionen gegeben. Dazu werden diese im Folgenden in die Unterkapitel Eindimensionale, Zweidimensionale, Raum-basierte, Machine Learning, Benutzer-zentrische und Clustering-basierte Verfahren aufgeteilt. Eine Einteilung in verschiedene Kategorien fällt dabei schwer, da viele der Verfahren mehrere der genannten Herangehensweisen verbinden. Aus diesem Grund können manche Verfahren nicht eindeutig in die hier genannte Unterteilung eingeteilt werden, da eine Mehrfachnennung möglich wäre.

2.1. Eindimensionale Verfahren

Die eindimensionalen Transferfunktionen sind die einfachste Form. Bei dieser werden den Voxeln Farbe und Okklusion nur abhängig von deren Intensitätswerten zugewiesen. Beispielsweise ist die einfachste Herangehensweise, dass ein Benutzer einen oder mehrere Werte angibt. Diese werden anschließend von der Transferfunktion eingefärbt.

Es existieren auch etwas aufwändigere eindimensionale Transferfunktionen, wie zum Beispiel das Vorgehen, das im Paper von Drebin [5] vorgestellt wird. In diesem werden die Voxel anhand von auf den Intensitätswerten basierenden Wahrscheinlichkeiten klassifiziert. Abhängig von dieser Klassifizierung werden den Voxeln Farbwerte zugewiesen.

Allgemein werden eindimensionale Transferfunktionen nicht mehr benutzt, da sie sich nur für simple Aufgaben eignen. Für das Lösen der komplexen Aufgabe des Ventrikelsystems zu segmentieren, sind sie jedoch unbrauchbar. Medizinischen Daten sind Messwerte und deshalb mit Rauschen behaftet. Dies erschwert die Identifizierung eines genauen Wertebereichs der Strukturen von Interesse. Weiterhin sind die Intensitätswerte verschiedener Bereiche nah beieinander oder gar gleich. Deshalb ist es mit eindimensionalen Transferfunktionen nicht möglich, verschiedene Materialien von einander zu unterscheiden.

2.2. Zweidimensionale Verfahren

Bei zweidimensionalen Transferfunktionen werden häufig als zweite Eingabegröße neben den Intensitätswerten die Länge der Gradienten hinzugezogen. Der Gradient eines Voxels, ist ein Vektor der in die Richtung der größten Änderung der Intensitätswerte zeigt.

Ein frühes Beispiel für die Verwendung der Gradientenlänge ist aus dem Jahr 1988 aus dem Paper von Levoy [6]. In diesem werden mithilfe einer simplen Funktion Voxeln Opazitätswerte abhängig von deren Intensitätswerten und Gradientenlängen zugewiesen.

Kniss stellt in seiner Arbeit [7] ebenfalls Transferfunktionen vor, die auf Intensitätswerten und Gradientenlängen basieren. Der Benutzer kann über eine graphische Oberfläche sogenannte *widgets* auf dem zweidimensionalen Histogramm erstellen, welche die Visualisierung der Daten bestimmen.

Das Paper von Shouren Lan [8] befasst sich hingegen mit der Verbesserung solcher zweidimensionalen Transferfunktionen, die auf Skalarwerten und Gradienten(kurz: SG-TF) basieren. Genauer geht es darum, die bei solchen Transferfunktionen immer wieder vorkommende Überlappung von unterschiedlichen Bereichen, zu verbessern. Dabei wird im Paper zwischen 3 verschiedenen Klassen von Strukturen unterschieden:

- (i) Strukturen, die keine andere Struktur berühren
- (ii) Strukturen, die keine andere Struktur berühren, jedoch nah an einer anderen liegen
- (iii) Strukturen, die andere Strukturen berühren

Wenn der Benutzer eine Region ausgewählt hat, werden zunächst alle Strukturen in dem Bereich klassifiziert und kleine Fragmente entfernt. Durch verschiedene Algorithmen werden Strukturen der Klassen (ii) durch Erosion, Dilatation, Aufteilen und neu Zusammenfügen voneinander getrennt. Strukturen der Klasse (iii) werden durch eine weitere niedrig dimensionale Transferfunktion getrennt. Anschließend werden Löcher, die beim Aufteilen der Strukturen entstehen mithilfe von einem Dilatationsoperator geschlossen. Als letztes wird den unterschiedlichen Strukturen verschiedene Farben und Okklusionen zugewiesen.

Das Vorgehen von Lan führt zwar zu guten Ergebnissen und wurde sogar schon am Ventrikelsystem getestet, jedoch wäre die Implementierung des Verfahrens mit der Unterscheidung der Klassen und den verschiedenen Algorithmen zum Teilen der Strukturen, Schließen von Löchern etc. sehr zeitaufwändig und damit im Rahmen dieser Bachelorarbeit nicht umsetzbar.

2.3. Raum-basierte Verfahren

In der Arbeit von Wesarg und Kirschner [9, 10] wird das *Structure-Size-Enhanced Histogram* vorgestellt. Zur Berechnung des Histogramms muss für jeden Voxel die Strukturgröße abgeschätzt werden. Dies geschieht, indem die Anzahl an Schritten, die in die jeweiligen Richtungen der 26 Nachbarvoxel vollbracht werden kann, aufaddiert wird. Ein Schritt kann ausgeführt werden, wenn der Intensitätswert des erreichten Voxels nicht mehr als ein gegebener Parameter von dem Intensitätswert des Ausgangspunkt abweicht.

Der erste Schritt hat eine Länge von einem Voxel, der zweite von zwei Voxeln, der dritte von vier Voxeln... und so weiter. Die Schrittweite verdoppelt sich mit jedem weiteren

Schritt bis hin zur Hälfte der Größe des gesamten Volumens. Hierbei kann ein Schritt trotzdem nur ausgeführt werden, wenn alle Voxel auf dem Weg das Kriterium erfüllen. Die Akkumulation aller Werte ergibt die geschätzte Größe der Struktur des Voxels. Dabei wird, um das Ergebnis weiterhin zu verbessern, jeweils nur der kleinere Wert von zwei entgegengesetzten Richtungen in der Berechnung verwendet. Aus der geschätzten Größe und dem Intensitätswert wird ein zweidimensionales Histogramm erstellt. Abhängig von der euklidischen Distanz der Kästchen des Histogramms zu einem vom Benutzer gegebenen Punkt, werden die Farbwerte der einzelnen Voxel aus einer Farbtabelle ausgelesen. Das Verfahren eignet sich gut, um relativ große Strukturen zu erkennen, jedoch ist es unklar, ob das relativ schmale Ventrikelsystem erkannt werden würde.

Eine andere Idee, Strukturen basierend auf räumlichen Informationen zu segmentieren, ist die Verwendung von Region Growing. Beispielsweise wird im Paper von Huang [11] ein solches Region-growingverfahren vorgestellt. Der Benutzer kann einen Punkt von Interesse im Volumen wählen, den sogenannten *seed*. Es werden alle 26 Nachbarn des *seeds* besucht und anhand einer Kostenfunktion, die den entsprechenden Wert des besuchten Voxels und des *seed*-Voxels vergleicht, entschieden, ob sie zu der Region dazugehören oder nicht. Sind sie Teil der Struktur, werden auch ihre Nachbarn besucht und alle passenden Voxel zu der Region hinzugefügt. Diese Vorgang wiederholt sich so lange bis alle Voxel gefunden wurden, oder ein anderes, internes Abbruchkriterium erfüllt wurde.

Es stehen dem Anwender drei verschiedene Kostenfunktionen zu Verfügung. Die erste Funktion bezieht sich auf die Intensitätswerte der Voxel, die zweite auf die Länge der jeweiligen Gradienten und bei der dritten werden die Gewichte der Voxel verglichen, die vorher vom Benutzer definiert werden müssen. In einem Nachbearbeitungsschritt ist es anschließend noch möglich, Elemente, die nicht zur Struktur von Interesse gehören, zu entfernen. Beispielsweise wenn eine ganze weitere Struktur auch visualisiert wird, da sie über eine kleine Brücke von ein bis zwei Voxeln mit der eigentlich gesuchten Struktur verbunden ist. Da das Region-growing sehr zeitaufwändig ist, kann der Anwender auswählen, dass er zunächst nur eine gewisse Teil der Region sich errechnen lässt.

Im Vergleich zu Huan [11] benutzt Chen in seiner Arbeit [12] nicht nur ein *seed*basiertes Vorgehen, sondern fügt noch ein sketchbasiertes Verfahren davor ein. Zunächst wählt der Anwender eine Reihe an Intensitätswerten im Histogramm, die für ihn interessant sind. Danach kann er direkt im Volumen eine Region von Interesse einzeichnen und markieren. Das Programm schneidet im Anschluss alle Teile des Volumens, die außerhalb der gewählten Region liegen, weg. Jetzt kann der Nutzer wie im vorher vorgestellten Verfahren seinen *seed* setzen. Dies erleichtert dem Benutzer die Anwendung, da er schneller zu seinem Punkt von Interesse gelangt, ohne vorher durch diverse Querschnittsbilder iterieren zu müssen. Des Weiteren ist es zeitsparend für den User, falls dieser sich nicht genau mit dem Datensatz und der zu Visualisierenden Region auskennt.

Correa und Ma zeigen in ihrer Arbeit [13] hingegen einen Ansatz, der auf der relativen Größe der zu visualisierenden Features basiert. Dazu benutzen sie den sogenannten *scale-space*, welcher für das Volumen berechnet wird, um anschließend eine auf Größe basierende Transferfunktion anzuwenden. Diese ordnet Farbe und Okklusion den entsprechenden Größen der Features des Volumens zu.

In einem weiteren Paper [14] beschreiben die beiden Forscher ein Verfahren, das auf der Okklusion der Voxel basiert. Dafür betrachten sie die Umgebung einzelner Voxel und berechnen abhängig davon, die Okklusion. Die Ergebnisse werden in einem zweidimensionalen Histogramm in Kombination mit den Intensitätswerten der Voxel gespeichert.

Durch die Umgebungsokklusion der Voxel ist es leicht mit einer auf dem Histogramm basierenden Transferfunktion unterschiedliche Materialien mit gleichen Intensitätswerten zu unterscheiden.

Eine weitere Arbeit [15] von Correa und Ma beschäftigt sich mit Transferfunktionen abhängig von der Sichtbarkeit einzelner Voxel. Die Sichtbarkeit jedes Voxels wird abhängig vom Sichtpunkt auf das Volumen berechnet, indem die Opazität vom Standpunkt der Kamera bis hin zum Voxel akkumuliert wird. Anschließend wird ein Histogramm über die Sichtbarkeitswerte erstellt. Auf diesem kann der Benutzer eine Transferfunktion zur Bestimmung der optischen Visualisierung erstellen, bei der er ein direktes Feedback über die Darstellung erhält. Um das gewünschte Ergebnis zu erhalten, wäre jedoch ein sehr genaues Einstellen dieser Funktion von Nöten, was der User nur schwer umsetzen kann. Aus diesem Grund wird eine Energiefunktion erstellt. Damit wird das Problem zu einem Optimierungsproblem, dem Minimieren der Energiefunktion. Dies kann mithilfe von progressiver Suche gelöst werden. Der Anwender muss dafür lediglich ein Opazitätsfunktion angeben, die seine gewünschten Visualisierungsziele beschreibt. Hierbei kann er auch aus vorgefertigten Funktionen wählen. Diese Histogramme wurde von Correa und Ma in einer erweiterten Arbeit [16] nochmals verbessert. Sie führten multidimensionale Sichtbarkeits-histogramme ein, die beispielsweise auch die Gradientenlänge in Betracht ziehen. Des Weiteren stellen sie zwei Methoden zur Berechnung von Sichtpunkt unabhängigen Sichtbarkeitshistogrammen vor. Zum einen ein omni-direktionales Sichtbarkeitshistogramm, bei dem die Sichtbarkeit von allen möglich Sichtpunkten berechnet wird. Und ein radiales Sichtbarkeitshistogramm, bei dem radiale Strahlen verwendet werden. Dazu wird das kartesische in ein sphärisches Koordinatensystem umgerechnet.

2.4. Machine Learning Verfahren

Die Arbeit von Tzeng [17] benutzt Machine Learning, um für den Nutzer interessante Strukturen darzustellen. Hierbei wird ein *Neuronales Netz* und eine *Support Vector Machine* benutzt. Als Input für das Verfahren kann der Benutzer im Volumen mit zwei verschiedenen Farben Regionen anmalen und damit markieren. Mit der einen Farbe markiert der Nutzer die Stellen von Interesse, die er hervorgehoben haben möchte, mit der anderen Farbe Regionen, die ihn explizit nicht interessieren.

Das Programm nimmt im Anschluss die Intensitätswerte, Länge der Gradienten und Intensitätswerte der Nachbarn aller markierter Voxel als Input um eine sinnvolle Segmentation zu finden. Das Ergebnis wird dem Anwender in Form einer farbigen Darstellung gezeigt, bei der er abhängig von der Farbe der Regionen erkennen kann wie ähnlich sie den mit der ersten Farbe angemalten Voxeln sind. Gefällt dem Benutzer das Ergebnis noch nicht, so kann er durch weiteres Einfärben von Regionen das Ergebnis verbessern bis das gewünschte Resultat erreicht wird.

Soundararajan stellt ebenfalls ein Verfahren vor [18], bei dem der Anwender direkt im Volumen markieren kann, welche Gebiete für ihn von Interesse sind.

Die Arbeit beschäftigt sich dabei mit dem Vergleich von fünf verschiedenen überwachten Klassifikationstechniken, die anhand der Markierungen die Volumendaten klassifizieren müssen. Es wurden *Gaussian Naive Bayes*, *k Nearest Neighbor*, *Support Vector Machines*, *Random Forests* und *Neural Networks* verglichen, mit dem Ergebnis, dass sich *Random Forests* am besten für die Aufgabe eignet. Die Forscher fanden heraus, dass es schnell, robust, einfach zu benutzen sei und zu guten Ergebnissen gelangt.

2.5. Benutzer-zentrische Verfahren

Eine weitere Art Transferfunktionen anzuwenden sind Benutzer-zentrische Verfahren. Hierbei werden Visualisierungen hauptsächlich abhängig von Eingaben des Benutzers erstellt. Der Anwender hat also die Möglichkeit mit dem Programm zu interagieren. Dies ist für einen unerfahrenen User intuitiver und er kann durch Ausprobieren ein gewünschtes Ergebnis erzielen. Sie vorgestellten Region Growing und Machine Learning Verfahren könnten auch an dieser Stelle erwähnt werden.

Fang stellt in seinem Paper [19] ein Verfahren vor, bei dem die Transferfunktion eine Abfolge von verschiedenen 3D Bildverarbeitungsverfahren ist, deren Parameter vom Benutzer angepasst werden können.

Weiterhin gibt es auch Methoden, bei denen ein Hilfswerkzeug zum Einsatz kommt. Zum Beispiel kann der Benutzer im Paper von Reitinger [20] sich gezielt Bereiche des Volumens hervorheben lassen, indem er sie mit einem Stift auswählt. Hierbei wird die Intensität des ausgewählten Punktes als auch der räumliche Abstand zum Stift in Betracht gezogen.

Wu und Qu stellen in ihrer Arbeit [21] ein intuitives Verfahren zum Verändern von Features von schon existierenden Transferfunktionen vor. Der Benutzer lädt zwei verschiedene *direct volume rendered images(DVRIs)* in das Programm. Hierbei wird ihm die jeweilige Visualisierung und Transferfunktion angezeigt. Der User kann entscheiden, ob er gewisse Features der beiden DVRIs zu einem verschmelzen, aus beiden mischen oder einzelne löschen möchte. Die Zusammenführung geschieht im Anschluss mithilfe einer Energiefunktion, die die Ähnlichkeit zweier Bilder beschreibt. Wie bei Correa und Ma [15] wird das Zusammenführen somit zu einem Optimierungsproblem und zwar dem Minimieren der Energiefunktion. Dies wird im Paper mithilfe eines stochastischen Suchalgorithmus gelöst. Der Anwender erhält am Ende eine Visualisierung mit allen gewünschten Features.

Die Arbeit von Buckner [22] hingegen basiert auf zwei Volumen, eines, in dem die Intensitätswerte gespeichert sind und eines, das vom Benutzer angegebene Regionen speichert. Das zweite Volumen entsteht, indem der Benutzer die Voxel der Regionen von Interesse markiert. Diese Voxel erhalten den Wert eins, wohingegen allen anderen der Wert Null zugewiesen wird. Anschließend werden drei verschiedene Objekte für die Interaktion mit dem Volumen vorgestellt:

- Die *selection*, die vom Benutzer ausgewählten Regionen, auf denen eine Transformation, wie Drehen oder Vergrößern, angewandt werden kann
- Der *ghost*, die zum ursprünglichen Ort der *selection* korrespondierenden Punkte im Datenvolumen
- Der *background*, der das restliche Datenvolumen repräsentiert

Für diese 3 Objekte, wird jeweils mithilfe der Intensitätsfunktion des Volumens, eine Funktion erstellt, die für einen gegebenen Punkt die Zugehörigkeit zu den jeweiligen Objekten zurückgibt.

In einem weiteren Schritt werden den Voxeln mithilfe einer Transferfunktion, die auf dem jeweiligen Grad an Dazugehörigkeit zu den verschiedenen Objekten beruht, Farb- und Opazitätswerte zugewiesen. Weiterhin wurden zwei weitere Transferfunktionen erstellt, die auf Voxel angewandt werden, die eine Überlappung von zwei Objekten aufweisen.

2.6. Clustering-basierte Verfahren

Sereda baut seine Arbeit [23] auf den von Serlie [24] vorgestellten LH-Histogrammen auf und zeigt, wie es mithilfe von ihnen möglich ist, Objekte zu klassifizieren. Die Berechnung eines LH-Histogramms ist eine Methode zur Erkennung von Kanten, unter der Verwendung von Low- und High-Werten. Dabei werden die Voxel in zwei verschiedenen Kategorien eingeteilt. Es gibt Voxel, die innerhalb eines Materials liegen und welche, die an der Grenze zweier Materialien liegen. Ist ein Voxel innerhalb, so sind seine LH-Werte gleich. Grenzvoxel hingegen haben unterschiedliche Low- und High-Werte, wobei diese die Intensitätswerte der beiden Materialien, zwischen denen die Grenze verläuft, beschreiben.

Bei der Berechnung des Histogramms wird als erstes getestet, ob der betrachtete Voxel an einer Grenze liegt. Ein Punkt liegt innerhalb eines Materials, wenn die Länge des Gradienten kleiner als ein gewisses epsilon (bei MRT-Daten) oder gleich Null (bei CT-Daten ist). In diesem Fall wären die Low- und High-Werte der Intensitätswert des Voxels. Ist dies jedoch nicht der Fall, wird in Richtung (für die High-Werte) und entgegengesetzter Richtung (für die Low-Werte) des Gradienten schrittweise integriert. Die Integration stoppt, sobald ein Material gefunden wurde. Dies wird für jeden Punkt im Volumen berechnet und danach aus allen LH-Werten ein Histogramm erstellt. Zur Visualisierung benutzt Sereda eine dreidimensionale Transferfunktion. Diese nimmt die beiden LH-Werte als auch die Gradientenlänge, da vor allem Voxel nah an der Grenze interessant sind und diese dadurch hervorgehoben werden, als Parameter entgegen. Anschließend verwenden die Forscher Regiongrowing, um Strukturen noch deutlicher erkennen zu können. Dabei basiert die Kostenfunktion auf dem LH-Histogramm. Dies beschreiben sie als deutlich besser als Kostenfunktionen, die auf dem Intensitätswert und der Gradientenlänge basieren, da Kanten trotz Überlappungen besser erkannt werden können.

In einer späteren Arbeit [25] stellt Sereda ein hierarchisches Clusteringverfahren vor. Hierbei werden in einer Menge von Clustern immer die zwei verschmolzen, die sich bei dem ausgewählten Vergleichsverfahren am ähnlichsten sind. Es wird eine Kombination aus zwei solcher Vergleichsverfahren vorgestellt. Zum einen wird die räumliche Nähe in Betracht gezogen, bei der gezählt wird, wie viele direkte Nachbarn zwei Cluster besitzen. Zum anderen wird die Nähe im LH-Raum untersucht. Als Startcluster dienen hierbei die Kästchen des LH-Histogramms. Die einzelnen Cluster bekommen für die Visualisierung am Ende eine zufälligen Farbwert zugewiesen.

Das Paper von Binh P. Nguyen [1] stellt ebenfalls ein clusteringbasiertes Verfahren vor. Zunächst wird in einem Vorverarbeitungsschritt die Gradienten des Volumens berechnet. Hierzu wurde Hong's Methode [3] verwendet. Im Anschluss daran wird anhand der Gradienten die LH-Werte mithilfe von Heuns Methode, ein modifizierte Euler Methode, ermittelt. Hierbei wird eine Gewichtung abhängig von der zurückgelegten Strecke bei der Interpolation für den Low- beziehungsweise High-Wert errechnet. Aus den LH-Werten und deren Gewichten wird anschließend ein LH-Histogramm erzeugt.

Dem Benutzer steht dann ein zweistufiges und ein dreistufiges Clusteringverfahren zur Auswahl, wobei die ersten beiden Clusteringschritte die Selben sind. Im ersten Clusteringschritt wird im LH-Raum mithilfe von *Meanshiftclustering* geclustert. Es wird für jeden LH-Wert alle Werte gefunden, die in einem Kreis mit einem Radius von 7% - 9% des maximalen LH-Wertes um den ursprünglichen Punkt liegen. Anschließend wird der neue durchschnittliche Mittelpunkt von allen Punkten im Cluster ermittelt. Der Vorgang wiederholt sich so lange, bis der Abstand zweier Mittelpunkt aus aufeinanderfolgenden

Iterationen kleiner als ein Thresholdwert ist. Der komplette Vorgang wird für jeden Punkt im LH-Histogramm wiederholt und am Ende werden alle Cluster, deren Mittelpunkte nah beieinander liegen, zu einem einzigen Cluster verschmolzen.

Der zweite Clusteringschritt wird auf die Cluster des ersten Schrittes angewendet. Hierbei wird auch *Meanshiftclustering* verwendet. Diesmal wird jedoch räumlich, also abhängig von der Position im Volumen, geclustert. Des Weiteren werden die Parameter für den Suchradius und Distanz zweier Mittelpunkte, damit sie verschmolzen werden, angepasst. Das Ergebnis der ersten zwei Schritte sind Cluster, bei denen alle Voxel ähnliche LH-Werte haben und auch räumlich nah beieinander liegen.

Im optionalen dritten und letzten Clusteringschritt wird hierarchisch geclustert. Hierbei wird für jeden Cluster die paarweise Nähe zu jedem anderen Cluster errechnet. Anschließend werden hierarchisch immer die zwei Cluster, die sich am nächsten sind, zu einem verschmolzen, solange bis nur noch ein Cluster existiert. Hierbei speichert das Programm jeweils, welche Cluster wann miteinander verschmolzen wurden. Der Benutzer kann im Anschluss entscheiden wie viel Cluster er haben möchte. Abhängig davon wird das hierarchische Clustern umgekehrt und die Cluster werden wieder getrennt bis die gewünschte Anzahl an Clustern erreicht ist.

2.7. Wahl des Verfahrens

Bei der Wahl des in dieser Bachelorarbeit verwendeten Verfahrens, könnten viele der gerade vorgestellten Herangehensweisen zu einer erfolgreichen Segmentierung des Ventrikelsystems führen. Andere scheiden jedoch prinzipiell aus.

Beispielsweise eignen sich eindimensionale und zweidimensionale Transferfunktionen schlecht für diese Aufgabe. Sie sind von ihrer Herangehensweise zu direkt und leiden oft an Überlappungen. Auch das genaue Abgrenzen von kleineren Strukturen, deren Intensitätswerte womöglich mehrfach im Volumen vorkommen, ist damit schwer bis gar nicht möglich.

Ein Verfahren wie von Lan [8] vorgeschlagen, eignet sich deutlich besser für diese Aufgabe. Jedoch ist, wie schon erwähnt wurde, der Aufwand der Implementierung für den Rahmen dieser Bachelorarbeit zu groß.

Weiterhin wurden verschiedene Transferfunktionen vorgestellt, die auf den räumlichen Informationen der anzuseigenden Regionen basieren. Die Implementation eines solchen Verfahrens würde vermutlich ebenfalls zu einer erfolgreichen Darstellung des Ventrikelsystems führen. Jedoch benötigt der Anwender für all diese Herangehensweisen nicht nur die Information wie das Ventrikelsystem aussieht, sondern auch wo es im Volumen zu finden ist. Folglich funktionieren diese Verfahren, sind aber wenig intuitiv und benötigen ein gewisses Fachwissen seitens des Nutzers. Darum wurde sich gegen einen räumlichbasierten Vorgehen entschieden.

Es wurden Methoden basierend auf Machine Learning vorgestellt. Bei diesen, muss der Benutzer jedoch, ähnlich wie bei den Region Growing Verfahren im Volumen einzeichnen, welche Bereiche für ihn von Interesse sind und welche ihn explizit nicht interessieren. Dies kann vermutlich ebenfalls zu einer erfolgreichen Segmentierung führen. Es wurde sich jedoch (wie beim Region Growing) dagegen entschieden, da erneut sehr viel Interaktion vom Benutzer gefordert wird. Dies ist ebenfalls der Fall bei den Benutzer-zentrischen Transferfunktionen.

2. Stand der Wissenschaft und Technik

Aus diesen Gründen wurde sich in dieser Arbeit auf die Implementierung des clustering-basierten Verfahren von Nguyen [1] festgelegt. Dieses eignet sich mit der Kantenerkennung sehr gut zum Segmentieren von gezielten Strukturen. Der Benutzer muss wenn die richtigen Parameter gefunden wurden, lediglich das Programm ausführen, ohne das Ventrikelsystem bei jedem Datensatz erneut selbst auswählen zu müssen. Ein weiterer Vorteil des clusteringbasierten Verfahrens sind die relativ geringen Berechnungszeiten im Vergleich zu vielen anderen sehr rechenaufwendigen Vorgehensweisen.

3. Methoden

Diese Arbeit orientiert sich an dem Clusteringverfahren aus dem Paper von Nguyen [1]. In diesem Kapitel werden die verschiedenen verwendeten Methoden vorgestellt.

3.1. Gradient

Zuerst müssen die Gradienten aller Voxel berechnet werden. Dafür wurde Hong's Methode [3] gewählt. Diese ist ein Verfahren basierend auf Approximation zur Berechnung von Gradienten eines Volumens unter der Betrachtung der lokalen $4 \times 4 \times 4$ Nachbarschaft des Punktes.

Hierbei ist zu beachten, dass es nicht möglich ist, den Gradienten für einen Voxel direkt zu berechnen. Der Gradient drückt die Veränderung der Intensitätswerte im Raum aus, folglich kann er immer nur zwischen mehreren Punkten berechnet werden. Deshalb liegt er im Falle eines dreidimensionalen Volumens im Zentrum eines Würfels, der von 8 benachbarten Voxeln aufgespannt wird.

In Abbildung 2 ist zu erkennen, wie der Gradient im Zentrum der acht Voxel liegt. Des Weiteren ist die $4 \times 4 \times 4$ Nachbarschaft in Form der durchnummerierten Punkte zu sehen.

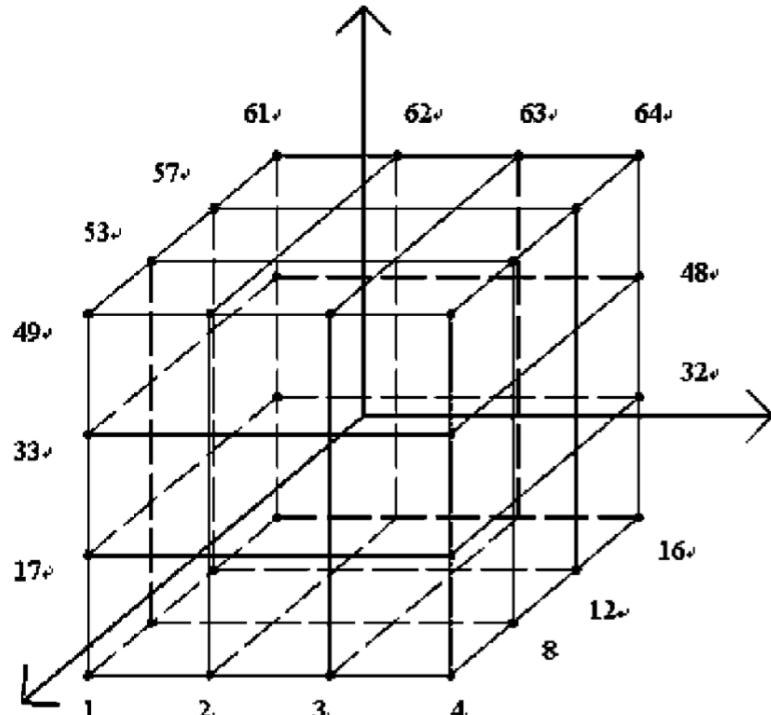


Abbildung 2: Darstellung der lokalen $4 \times 4 \times 4$ Nachbarschaft
Entnommen aus [3]

Ein Gradient ist ein dreidimensionaler Vektor, der in die Richtung der größten Änderung der Intensitätswerte im Volumen zeigt. Ein Beispiel dessen ist in Abbildung 3 zu sehen. Die Gradienten zeigen von rechts nach links, in die Richtung, in der die Intensitätswerte anwachsen.

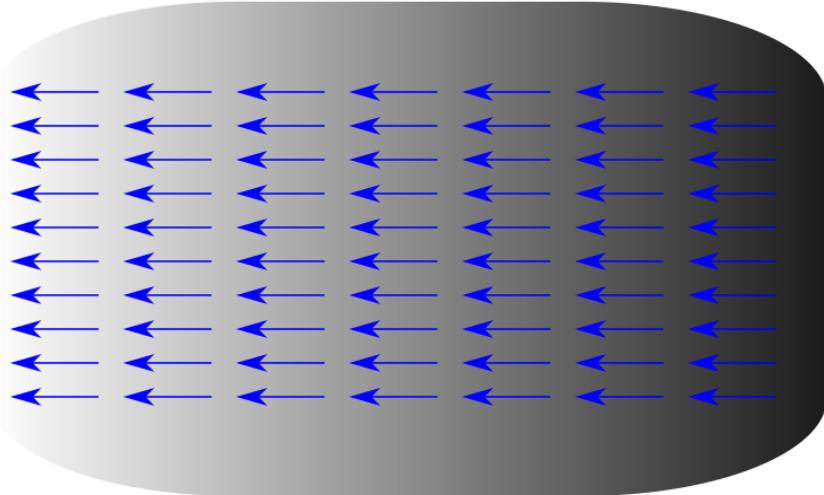


Abbildung 3: Beispiel Gradienten

Gäbe es eine Funktion, die die Intensitätswerte des Volumens in der lokalen Nachbarschaft beschreibt, wäre der Gradient die Ableitung dieser Funktion. Folglich muss für die Berechnung eines Gradienten zunächst eine Intensitätsfunktion für die lokale 4x4x4 Nachbarschaft aufgestellt werden. Im Paper wird dies mit der Formel:

$$f(x, y, z) = Ax^2 + By^2 + Cz^2 + 2Fyz + 2Gzx + 2Hxy + 2Ix + 2Jy + 2Kz + D \quad (1)$$

approximiert, da es nicht möglich ist, eine solche Funktion zu bestimmen. Um den Gradienten zu erhalten, muss diese Funktion abgeleitet werden. Für die Berechnung des Gradientenvektor n ergibt sich dadurch die Formel:

$$n = 2 \begin{bmatrix} Ax + Gz + Hy + I \\ Hx + By + Fz + J \\ Gx + Fy + Cz + K \end{bmatrix} \quad (2)$$

Um den Gradienten zu berechnen, müssen folglich die Parameter $A, B, C, E, F, G, H, I, J, K$ gelöst werden. Dazu wird im Paper zunächst die *error distance* vorgestellt. Diese beschreibt die Entfernung zwischen einem berechneten Datenpunkt und seinem korrespondierenden Punkt in den tatsächlichen Daten. Die Funktion $E(A, B, C, E, F, G, H, I, J, K)$ berechnet die Summe der *error distance* aller 64 Nachbarpunkte. Diese Summe drückt aus, wie gut die Approximation die tatsächlichen Intensitätswerte des Volumens beschreibt. Deshalb müssen die Parameter so gewählt werden, dass die Summe E minimal wird und somit die approximierte Intensitätsfunktion das Volumen möglichst genau beschreibt.

$$\begin{aligned} E(A, B, C, D, F, G, H, I, J, K) = & \\ \sum_{k=1}^{64} w_k (Ax^2 + By^2 + Cz^2 + 2Fyz + 2Gzx + 2Hxy + 2Ix + 2Jy + 2Kz + D - f_k)^2 & \end{aligned} \quad (3)$$

Dabei ist $w_k = \frac{1}{d_k}$, $d_k = (x_k^2 y_k^2 z_k^2)^{1/2}$ die Distanz des k-ten Punktes zum Ursprung des Koordinatensystems, wie es in Abbildung 2 zu sehen ist, und f_k der Intensitätswert des

k-ten Punktes.

Die Parameter $A, B, C, E, F, G, H, I, J, K$ können mithilfe der Methode der kleinsten Quadrate bestimmt werden. Dies ist ein mathematisches Verfahren, das zu mehreren Punkten eine Funktion bestimmt. Die Funktion beschreibt eine Kurve, bei der die Summe der Abweichung aller Punkte zur Kurve minimal ist. Dies lässt sich auf die Funktion E übertragen. Mithilfe dessen können durch mehrere Umformungsschritte die Parameter berechnet werden. Anschließend kann der Gradient $[X, Y, Z]$ des Punktes $[x, y, z]$ mit folgender Formel, die sich aus der gezeigten Ableitung des Intensitätsfunktion ergibt, kalkuliert werden:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 2A & 2H & 2G \\ 2H & 2B & 2F \\ 2G & 2F & 2C \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} 2I \\ 2J \\ 2K \end{bmatrix} \quad (4)$$

3.2. LH-Werte

Als nächster Schritt folgt die Berechnung der Low- und High-Werte. Diese beschreiben die Grenzen zwischen zwei verschiedenen Materialien. Liegt ein Voxel innerhalb eines Materials, so sind seine LH-Werte gleich. Liegt ein Punkt jedoch an der Grenze zweier Strukturen, so beschreibt der Low-Wert den Intensitätswert des Materials mit dem niedrigeren und der High-Wert das Material mit dem höheren Intensitätswert. Die LH-Werte eines Voxels werden berechnet, indem in die Richtung des Gradienten integriert wird. Hierfür wurde Heun's Methode, eine modifizierte Euler Methode, verwendet. Dies ist ein numerisches Verfahren, um gewöhnliche Differentialgleichungen mit einem Startwert zu lösen. Dabei lautet die für die Integration benutzte Formel:

$$u_{i+1} = u_i + \frac{1}{2}d(\nabla f(u_i) + \nabla f(u_i + d\nabla f(u_i))) \quad (5)$$

Hierbei sind u_i und u_{i+1} die Positionen des aktuellen, beziehungsweise des nächsten Voxels. $\nabla f(x)$ beschreibt den normalisierten Gradienten bei der Berechnung der High-Werte und den normalisierten inversen Gradienten bei der Berechnung der Low-Werte des Punktes x . d steht für die Schrittweite, die in dieser Arbeit auf einen Voxel festgelegt wurde. Die Integration stoppt, wenn eine lokale Extremstelle oder ein Wendepunkt erreicht wird. Diese sind daran zu erkennen, dass die Längen der Gradienten an diesen Stellen null sind. Wird ein solcher Punkt erreicht, wird der Intensitätswert des aktuell besuchten Voxels als Ergebnis für den Low- beziehungsweise High-Wert des Startvoxel gespeichert.

Anschließend wird ein LH-Histogramm mit allen berechneten LH-Wertpaaren erstellt. Hierbei sind auf der x-Achse die Low-Werte und auf der y-Achse die High-Werte angeordnet. Die Werte der Achsen reichen von null bis zum jeweiligen Maximum der Low-beziehungsweise High-Werte.

Ein simples Beispiel der LH-Werte ist in Abbildung 4 zu sehen. Es ist zu erkennen, dass die Voxel innerhalb der Materialien als LH-Werte den Intensitätswert des jeweiligen Materials besitzen. Der Voxel an der Grenze hat als Low- und High-Werte die Intensitäten der beiden Materialien zwischen denen die Grenze verläuft.

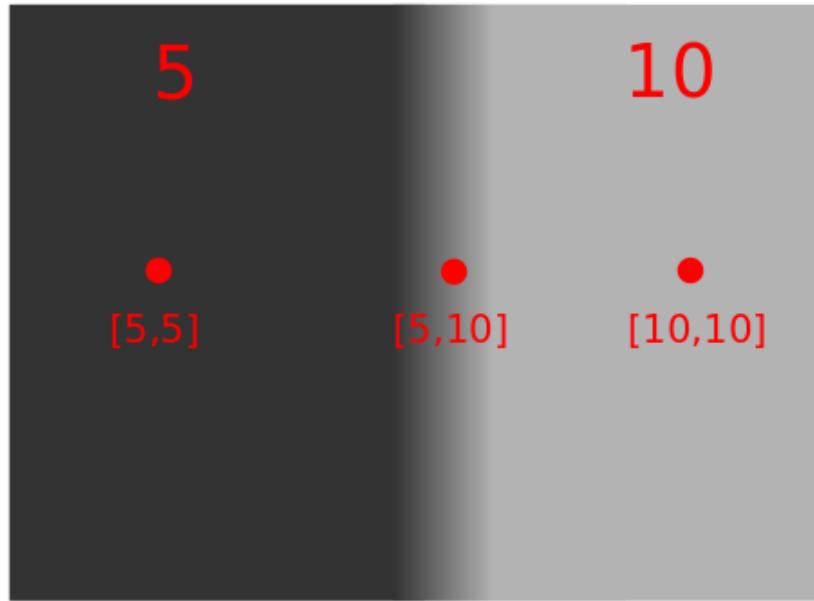


Abbildung 4: Beispiel LH-Werte

3.3. LH-Clustering

Als nächstes wird der erste Clusteringschritt durchgeführt. Dieser findet über dem LH-Raum, genauer gesagt auf dem zuvor berechneten LH-Histogramm statt. Dafür wird *Meanshiftclustering* verwendet.

Vor dem Berechnen der Cluster muss zunächst eine Bandweite sowie ein Thresholdwert bestimmt werden, welche die Sensitivität des Clusterings angeben. Die von Nguyen [1] verwendete Bandweite liegt bei 7% bis 9% des maximalen LH-Wertes und der Threshold bei 0,01. Anschließend kann die Kalkulation der Cluster beginnen.

Die Berechnung wird für jeden Punkt im Histogramm durchgeführt. Jede Iteration besitzt dabei einen Cluster mit einem Mittelpunkt. Der Mittelpunkt eines Clusters ist der jeweilige Mittelwert der Low- und High-Werte aller Punkte des Clusters. Wenn die Kalkulation für einen beliebigen Punkt startet, wird dieser zum Cluster hinzugefügt und als erster Mittelpunkt bestimmt. Anschließend werden alle Punkte die im Histogramm in einem Umkreis von der Bandweite um den Mittelpunkt liegen zum Cluster hinzugefügt. Danach wird der neue Mittelpunkt berechnet. Erneut werden alle Punkte, die innerhalb der Bandweite um den Mittelpunkt liegen und noch nicht zum Cluster gehören, hinzugefügt. Dies wiederholt sich so lange, bis die Distanz zweier aufeinanderfolgender Mittelpunkte kleiner als die Bandweite multipliziert mit dem Thresholdwert ist.

Nachdem diese Berechnung für jeden Punkt im Histogramm durchgeführt wurde, existieren mehrere Cluster, die oftmals aus vielen gleichen Punkten bestehen. Um dies zu unterbinden, werden in einem weiteren Schritt alle Cluster, die sehr nah beieinander liegen, verschmolzen. Dies betrifft jene Cluster, deren Mittelpunkte eine Distanz kleiner als die Hälfte der Bandweite zueinander haben.

3.4. Räumliches-Clustering

Anschließend wird der zweite Clusteringschritt durchgeführt, welcher erneut *Meanshift-clustering* verwendet. Dies wird mit dem gleichen Vorgehen, wie im vorherigen Schritt be-

schrieben, umgesetzt. Ein Unterschied besteht jedoch darin, dass diesmal das Clustering nicht auf dem LH-Raum, sondern auf dem kartesischen Raum des Volumens angewendet wird. Weiterhin wird das Clustering auf den zuvor entstandenen Clustern einzeln und unabhängig voneinander durchgeführt. Dadurch haben alle durch den räumlichen Clusteringschritt entstehende Ergebniscluster ähnliche LH-Werte und liegen auch im Volumen sehr nah beieinander.

Aus diesem Grund findet am Ende der Berechnung der Cluster keine Verschmelzung statt, da dies den Sinn der beiden von einander getrennten Clusteringschritte zerstören würde. Würden die Ergebniscluster anhand ihrer räumlichen Informationen verschmolzen werden, wäre der erste Clusteringschritt umsonst gewesen, da die Ähnlichkeit der LH-Werte verloren gehen würde.

3.5. Hierarchisches-Clustering

Im optionalen letzten Clusteringschritt, werden die berechneten Cluster hierarchisch miteinander verschmolzen. Dafür wird zunächst der paarweise Abstand zwischen allen Clustern berechnet. Der Abstand wird wie bei Sereda [25] anhand der Anzahl der direkten Nachbarschaften der Voxel beider Cluster bestimmt. Anschließend werden immer die beiden Cluster, die sich am nächsten sind, zu einem verschmolzen. Dieser Vorgang wiederholt sich so lange, bis nur noch ein Cluster existiert. Im Anschluss wird das Verfahren Schritt für Schritt rückgängig gemacht, bis eine vom Benutzer angegebene Anzahl an Clustern erreicht wird.

Der vorgestellte dritte hierarchische Clusteringsschritt wurde in dieser Arbeit nicht angewendet. Der Grund dafür ist, dass das Verschmelzen von Clustern abhängig von räumlicher Nähe für die Aufgabe das Ventrikelsystem hervorzuheben nicht zielführend ist. Wird das System als ein oder mehrere Cluster erkannt, liegen diese mitten im Gehirn neben sehr vielen anderen Clustern und würden mit anderen umliegenden Strukturen verschmolzen werden.

4. Design

Nachdem im vorherigen Kapitel die verschiedenen Methoden, die genutzt werden, vorgestellt wurden, beschäftigt sich dieser Abschnitt mit dem Softwaredesign.

Die Implementierung dieser Bachelorarbeit teilt sich dabei in zwei verschiedenen Programme auf. Zum einen den sogenannten *VolumeRenderHelper*, der für das Laden, Umwandeln, Verarbeiten und Speichern der Volumendaten zuständig ist. Zum anderen das Unityprogramm *VolumeRenderer*, welches zur Visualisierung der vom *VolumeRenderHelper* erzeugten Daten dient. Diese beiden Programme existierten bereits vor dieser Arbeit und sind in Vorarbeiten des IPRs entstanden.

In dieser Bachelorarbeit wurde der *VolumeRenderHelper* um die vorgestellte Transferfunktion erweitert und der *VolumeRenderer* geringfügig angepasst. Da beide Programme in C# geschrieben sind, wurden auch der Code, der im Laufe dieser Bachelorarbeit entstand, in C# geschrieben. Als Entwicklerumgebung wurde *Visual Studio 2017* von Microsoft verwendet. Die beiden Programme werden im Folgenden als Helper und Renderer bezeichnet. Weiterhin wurde ein Pythonskript *PlotHelper* geschrieben, um LH-Histogramme anzuzeigen.

MITK Anfangs liegen die CT-Daten der Volumen in mehreren Dateien als Schnittbilder im DICOM Format vor. Diese werden mithilfe der *Medical Imaging Interaction Toolkit* (kurz: *MITK*) *Workbench* [26] zu einer einzelnen Datei im .nrrd Format umgewandelt, da Volumen nur in diesem Format oder als binäre Datei vom Helper eingelesen werden können. *MITK* ist ein kostenloses open-source System zur Entwicklung von medizinischer Bildverarbeitungssoftware und wurde vom Deutschen Krebsforschungszentrum entwickelt. Dessen *Workbench* bietet zum einen das eben vorgestellte Laden und Umwandeln von DICOM Dateien, zum anderen auch diverse Segmentierungstools.

VolumeRenderer Die Darstellung von Volumendaten ist über den Renderer in Unity möglich. Der Benutzer hat hierbei mehrere Möglichkeiten Eingaben über Parameterfelder (wie in Abbildung 10 zu sehen), oder mithilfe der Tastatur zu machen und mit der Visualisierung zu interagieren. Zunächst muss er jedoch in das Parameterfeld *Binary Data* via drag and drop eine binäre Volumendatei ziehen. Anschließend kann die Visualisierung gestartet werden. Hierbei hat der Nutzer sowohl die Möglichkeit, die Kamera mit den Tasten W A S D zu bewegen als auch mit den seitlichen Pfeiltasten, sowie der Q und E Taste das Volumen um verschiedene Achsen zu drehen. Die Position der Kamera sowie die Drehung des Volumens kann ebenfalls durch entsprechende Parameterfelder verändert werden. Die Volumen werden als Graubilder dargestellt, wobei die Helligkeit der einzelnen Voxel abhängig vom Intensitätswert bestimmt wird. Dabei kann der User über ein weiteres Parameterfeld entscheiden, ob der Größte, der Kleinste, der Erste oder die Summe aller gefundenen Voxel als Wert für die Farbgebung benutzt wird. Der Standard ist hierbei, dass des maximale Wert genommen wird.

Des Weiteren kann der Nutzer einen Threshold setzen, bei dem alle Voxel, die einen Intensitätswert niedriger als diesen haben, für die Visualisierung ignoriert werden. Es existieren noch deutlich mehr Eingabefelder, mit denen der Benutzer die Darstellung des Volumens verändern und anpassen kann. Diese werden jedoch, da sie von geringer Interesse sind, im Kontext dieser Bachelorarbeit nicht vorgestellt.

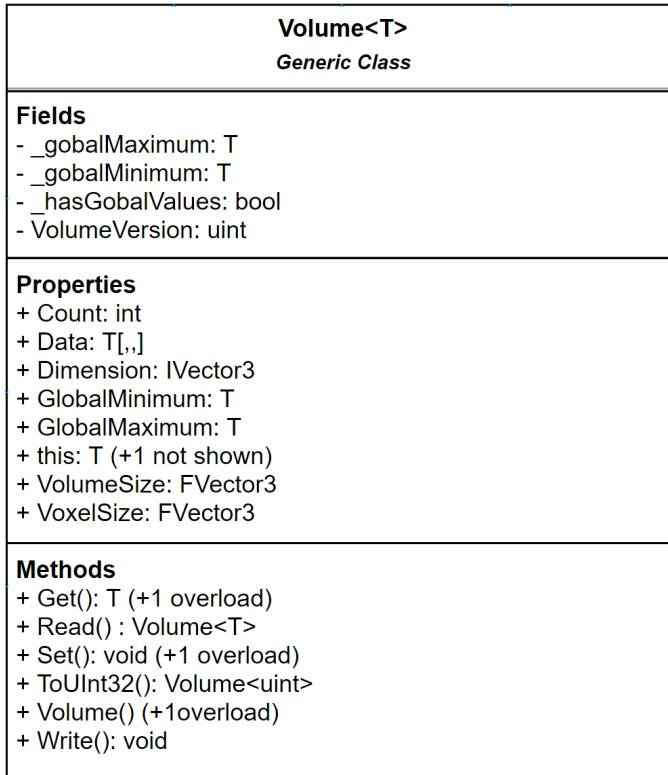


Abbildung 5: UML-Diagramm des Volumens

VolumeRenderHelper Die in der Vorarbeit bereits vorhandene interne Speicherung des Volumens im Helper wird mit der generischen Klasse *Volume* umgesetzt, deren Felder, Attribute und Methoden in Abbildung 5 zu sehen sind. Diese besitzt ein dreidimensionalen Array vom angegebenen generischen Datentyp sowie Informationen über das Volumen, wie zum Beispiel Größe der Voxel oder Anzahl der Elemente pro Achse. Des Weiteren bietet die Klasse verschiedene Funktionen, um Informationen auszulesen oder zu bearbeiten. Zur Darstellung von dreidimensionale Koordinaten werden die Klassen *IVector3* und *FVector3* benutzt. Diese stellen Vektoren dar, die entweder Integer, Ganzzahlen oder Float, Dezimalzahlen, als *x*, *y* und *z* Werte speichern.

Die Interaktion des Benutzers mit dem Helper findet über eine Kommandozeile statt. Hierbei hat der Anwender die Befehle *Load*, *Dump*, *Resample*, *Info*, *Write*, *LHHistogram*, *ClusterVolume* und *MergeCluster* zur Auswahl. Jedes Modul hat dabei spezifische Parameter, die beim Aufruf angegeben werden müssen. Wird ein Befehl mit einem Help dahinter aufgerufen, wird die richtige Benutzung des Moduls erklärt. Die vorgestellten Module erben dabei alle von der abstrakten *BaseModule* Klasse, die es ihnen vorschreibt, eine *Call* sowie eine *PrintHelp* Methode zu implementieren. Dies ist im UML-Diagramm in Abbildung 6 zu sehen.

Die Funktionen der Module entsprechen deren Namen und sind im Folgenden aufgelistet.

Load

Lädt eine .nrrd oder eine binäre Datei.

Dump

Speichert das aktuell geladene Volumen als binäre Datei.

Write

Speichert das aktuell geladene Volumen als .nrrd Datei.

4. Design

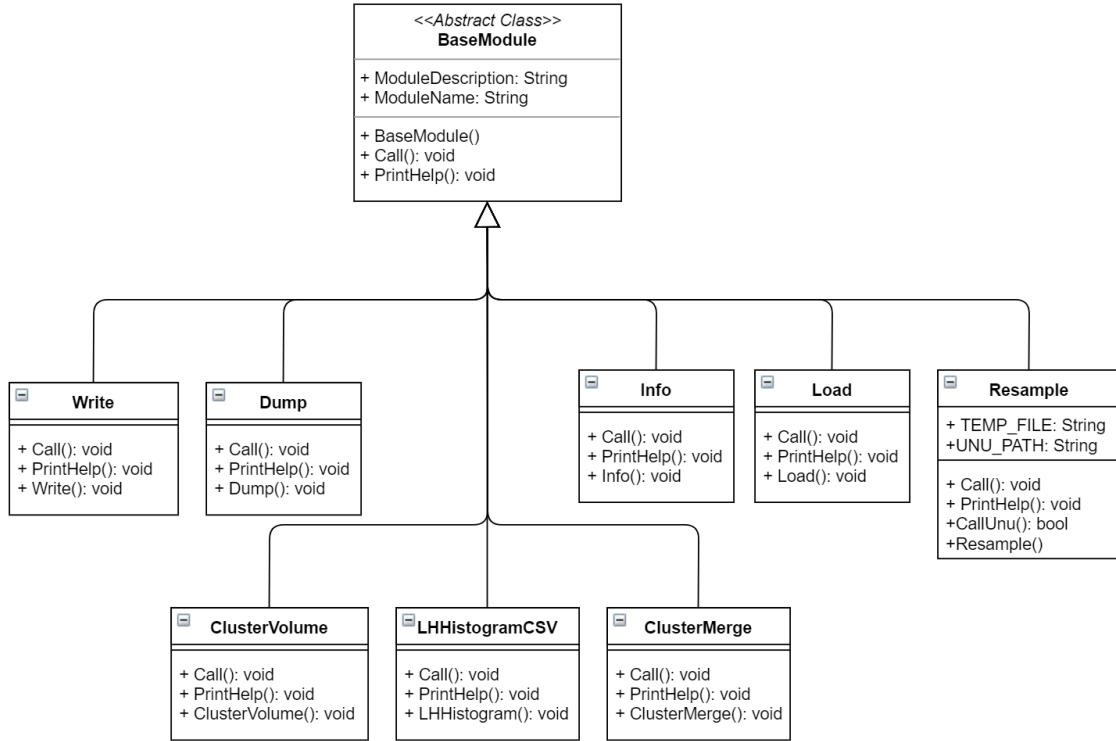


Abbildung 6: UML-Diagramm über die Module

Info

Gibt Informationen über das aktuell geladene Volumen, wie beispielsweise Größe, Minimum, Maximum, etc.

Resample

Verändert die Größe des aktuell geladenen Volumens.

Wird der *Dump* Befehl mit einem *u* am Ende aufgerufen, so wird das Volumen vor dem Speichern zum Typ *unsigned int* gecastet. Dies geschieht, indem auf alle Werte der Betrag des minimalen Wertes aufaddiert wird. Dadurch verschieben sich die Werte so, dass das Minimum bei null liegt, also nur noch positive Zahlen im Volumen vorhanden sind. Dies ist für die Darstellung im Renderer wichtig, da dieser nur mit positiven Zahlen funktioniert. Des Weiteren müssen alle binären Dateien mit dem Suffix *.bin.txt* gespeichert werden, da Unity die Dateien sonst nicht einlesen kann.

Im Folgenden werden hauptsächlich die Module *LHHistogram*, *ClusterVolume* und *MergeCluster* erläutert, da diese im Laufe dieser Arbeit entstanden sind.

LHHistogram

Berechnet das LH-Histogramm des geladenen Volumens und speichert dieses in einer *.csv* Datei ab.

ClusterVolume

Kalkuliert die LH-Werte des Volumens und führt hinterher die beiden Clusteringsschritte des Verfahrens aus. Als Ausgabe wird eine binäre Datei eines Volumens gespeichert, in welchem die verschiedenen IDs der Cluster gespeichert sind.

MergeCluster

Verschmilzt die gewünschten IDs mit dem ursprünglichen Volumen.

4. Design

Das Konzept hinter den IDs wird im Laufe dieses Kapitels erklärt. Des Weiteren ist das Ergebnis des *MergeCluster* Moduls eine finale binäre Datei, die im Renderer geladen und das Ventrikelsystem visualisiert werden kann. Ein Überblick über den gesamten Aufbau und Ablauf der Implementierung ist in Abbildung 7 zu sehen.

Als erstes muss mithilfe des *Load* Moduls ein Volumen geladen werden. Wird anschließend der *LHHistogram* oder *ClusterVolume* Befehl ausgeführt, beginnt der Ablauf der Berechnung in der statischen *Gradient* Klasse. Diese ist eine Implementierung des Verfahrens von Hong [3]. Zur Berechnung wird der Funktion *CalcGradientVolume* das Volumen der Intensitätswerte als Parameter vom Typ *Volume<int>* übergeben. Wie im vorherigen Kapitel besprochen, können die Gradienten nicht für einen Voxel direkt berechnet werden, sondern nur für die Punkte zwischen den Voxeln. Aus diesem Grund ist das Ergebnisvolumen um ein Voxel in jeder Achse kleiner als das Volumen der Intensitätswerte.

In Abbildung 8 ist dazu ein zweidimensionales Beispiel zu sehen. Die Daten werden hierbei jeweils in den Ecken der Gitter gespeichert. Die Gradienten werden, da es sich um einen zweidimensionalen Fall handelt, jeweils in der Mitte von vier benachbarten Punkten berechnet. Die Dimension der Intensitätswerte ist 5x5, in der Abbildung in Schwarz zu sehen. Nachdem die Gradienten berechnet wurden, ist zu erkennen, dass das Gitter der Gradienten eine Dimension von 4x4 besitzt, in der Abbildung in Rot zu sehen. Als Ergebnis der *CalcGradientVolume* Funktion wird ein Volumen vom Typen *FVector3* zurückgegeben.

Die Berechnung der LH-Werte findet in der statischen Klasse *LHValues* in der Funktion *LHValueVolume* statt. Als Parameter wird das *FVector3* Volumen der Gradienten aus dem Schritt davor entgegengenommen. Da für die Berechnung der LH-Werte die Intensitätswerte und die Gradienten am gleichen Punkt benötigt werden, müssen die Intensitätswerte für das verschobene Volumen der Gradienten berechnet werden. Dazu wurde eine einfache Interpolation durchgeführt, indem von allen 8 Nachbarn eines Punktes die Intensitätswerte aufaddiert und hinterher durch acht geteilt wurden. Hierbei muss jedoch beachtet werden, dass die durch diese Interpolation Informationen verloren gehen.

Als Ergebnis der Funktion wird ein Volumen von zweier Tupeln zurückgegeben, in denen die Low- und High-Werte gespeichert sind.

Hat der Benutzer das Modul *LHHistogram* aufgerufen, wird im Anschluss das LH-Histogramm in der Klasse *LHHistogramCSV* erstellt und wird von ihr als .csv Datei in einem vom Anwender angegebenen Pfad abgespeichert. An dieser Stelle kommt das Python Skript *PlotHelper* zum Einsatz. Dieses lädt die .csv Datei und visualisiert das LH-Histogramm mithilfe einer kalt-zu-heiß-Farbrampe in einem zweidimensionalem Koordinatensystem. Ein Beispiel ist in Abbildung 9 im Kapitel Implementierung zu sehen. Hierbei ist zu beachten, dass das Histogramm abhängig von der Häufigkeit des Vorkommens eines LH-Wertpaars im Volumen gebildet wird. Die Paare werden im jeweils dazu passenden Kästchen des Koordinatensystems gespeichert. Dies ist ein simplerer Vorgehen als das im Paper von Nguyen [1] benutzten Erstellen des Histogramms abhängig von einer für jeden Voxel berechneten Gewichtung.

Aus zeitlichen Gründen, wurde die Gewichtung, die einzig und allein einer genaueren Darstellung des für das Verfahren irrelevante LH-Histogramm dient, vernachlässigt. Die Gewichtung ist für das Clustering belanglos, da dort ein Histogramm wie eben beschrieben, abhängig von der Häufigkeit der LH-Werte verwendet wird. Bei der Erstellung des Histogramms im *LHHistogram* Modul wird weiterhin der Logarithmus der Anzahl der Einträge jedes Kästchens genommen. Dies geschieht aufgrund der großen Diskrepanz

4. Design

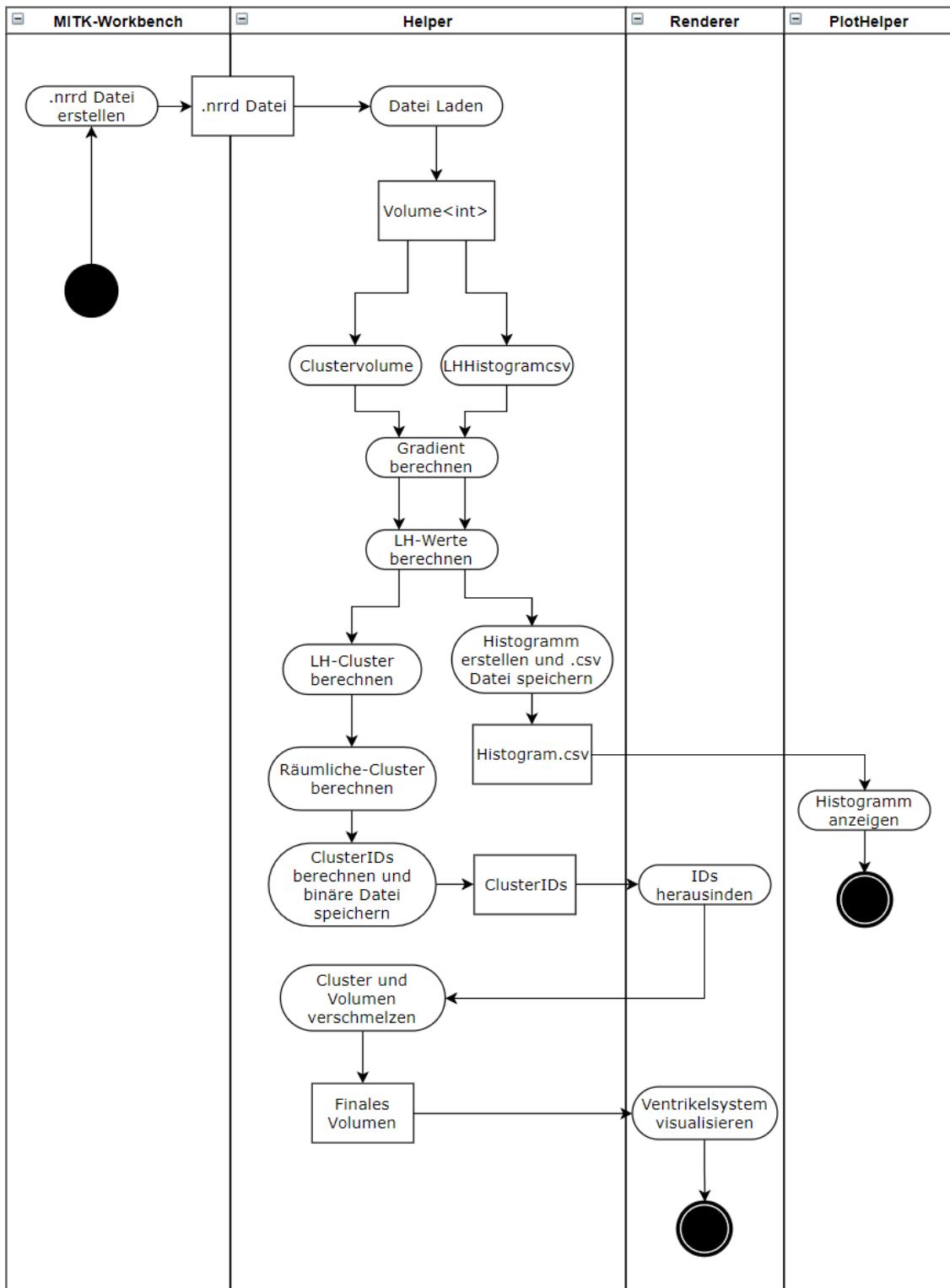


Abbildung 7: Aktivitätsdiagramm über den Prozessablauf

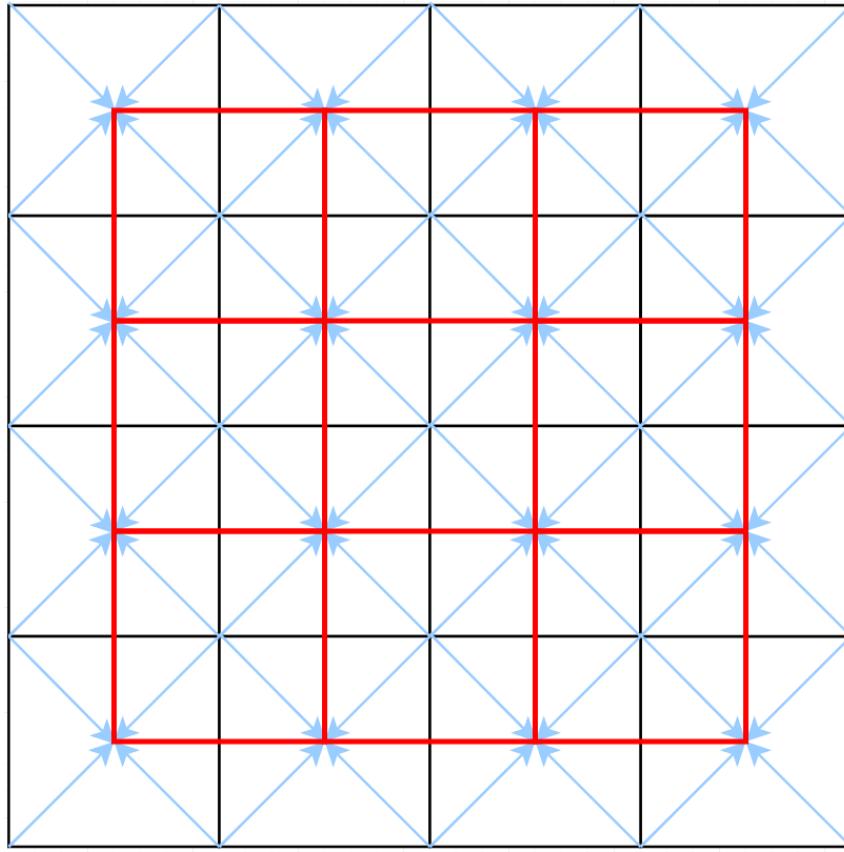


Abbildung 8: 2D Beispiel für die Verkleinerung des Volumens

zwischen der Anzahl der Einträge an den meisten Stellen im Histogramm im Vergleich zu dem Maximum. Wegen diesem Unterschied wird bei der Darstellung des Histogramms mit einer Farbrampe fast alles mit der Farbe des Minimums angezeigt. Da die Logarithmusfunktion für schnell wachsende Zahlen nur sehr langsam steigt, eignet sie sich dafür, die große Diskrepanz anzupassen.

Wurde das *ClusterVolume* Modul aufgerufen, wird mit den beiden Clusteringschritten fortgefahrene. Die Berechnung der LH-Cluster findet in der *LHClustering* Klasse statt. Hierbei nimmt die Funktion *ComputeLHClusters* das Volumen mit den LH-Werten entgegen und rechnet dieses aus Performancegründen mit dem gleichen Vorgehen, welches im *LHHistogram* Modul benutzt wurde, in ein Histogramm um. Jedoch wird nicht der Logarithmus der Einträge genommen. Dieser Schritt könnte gespart werden, wenn die Methode *LHValueVolume* direkt ein Histogramm als Rückgabewert liefern würde. Eine solche Änderung würde auch das *LHHistogram* Modul verbessern, da damit auch die Umrechnung in dieser Klasse gespart werden könnte.

Als Ergebnis der Clusteringfunktion wird eine Liste der Cluster zurückgegeben. Ein Cluster besteht hierbei aus einer Liste von *IVector3*. Die Cluster werden nur als Liste der räumlichen Informationen der Voxel gespeichert, da dies lediglich für den nächsten Clusteringschritt benötigt wird.

Anschließend geht es in der Funktion *ComputeIDs* der *SpatialClustering* Klasse mit der Berechnung der räumlichen Cluster weiter. Diese werden mit einer ähnlichen Implementierung wie in der *LHClustering* Klasse kalkuliert. Diesmal wird jedoch kein Histogramm für das Clustering erstellt, sondern die Berechnung direkt auf den übergebenen Listen durchgeführt. Des Weiteren wurde eine Mindestgröße für die entstehenden Cluster fest-

4. Design

gelegt, um kleine, unbedeutende Cluster zu entfernen. Als Ergebnis wird erneut eine Liste von Listen vom Typ *IVector3* zurückgegeben.

Nachdem alle Cluster kalkuliert wurden, werden sie in der *ClusterVolume* Klasse in einem Volumen gespeichert. Jeder Cluster bekommt dabei zunächst eine einzigartige ID. Die IDs beginnen bei eins und werden hochgezählt. Anschließend wird das Volumen mit den verschiedenen IDs gefüllt. Dies geschieht, indem für jeden Cluster an den Positionen der Punkte die jeweilige ID gespeichert wird. Allen anderen Voxeln des Clustervolumens wird der Wert Null zugewiesen. Dieses Volumen wird als binäre Datei gespeichert und ist das Ergebnis des *ClusterVolume* Moduls. Es ist bei diesem Modul sehr wichtig, dass es, wie es bei dem *Dump* Befehl möglich ist, mit einem *u* am Ende aufgerufen wird, da das Ergebnis sonst nicht vom Renderer eingelesen werden kann.

Das Ergebnis muss anschließend vom Nutzer in Unity geladen werden. Hierbei kommt die in dieser Arbeit vorgenommenen Anpassung am Renderer zum Einsatz. Hier hat der Benutzer durch die Erweiterung die Möglichkeit, einen speziellen Wert oder einen Wertebereich hervorzuheben. Da das Volumen lediglich die IDs als Werte gespeichert hat und sonst nur aus Nullen besteht, werden, wenn ein Wert vom User angegeben wird, die Voxel, die zum Cluster der jeweiligen ID gehören, hervorgehoben. Mithilfe dessen muss der Anwender alle Cluster, die zum Ventrikelsystem gehören, finden und sich deren IDs zu notieren.

Hat er dies getan, kann er mit dem *MergeCluster* Modul des *Helpers* das Ergebnis zusammenfügen und die finale Visualisierung erstellen. Beim Aufruf muss das Intensitätsvolumen bereits geladen sein und der Benutzer das Clustervolumen, die ausgewählten IDs und ein Zielpfad als Parameter übergeben. In der *Call* Methode der *MergeCluster* Klasse wird anschließend an den Stellen im Volumen, an denen die entsprechenden IDs gespeichert sind, die Werte im Intensitätsvolumen auf 5000 gesetzt. Nach dem beschriebenen Verschieben aller Werte ins Positive liegt der maximale Intensitätswert bei ungefähr 4500. Folglich erhöht das Hinzufügen des Wertes 5000 das Maximum des Volumens und damit die Darstellung im Renderer nur gering und lässt es zu, die segmentierten Bereiche klar vom Rest abzugrenzen. Dieses Volumen wird erneut als binäre Datei im Zielpfad abgespeichert.

Als letzten Schritt kann der Benutzer die finale binäre Datei in Unity laden. Anschließend muss er den *SpecificValue* Modus auswählen und den hervorzuhebenden Wert auf 5000 setzen.

5. Implementierung

Nachdem im vorherigen Kapitel das Softwaredesign erstellt wurde, wird im Folgenden die Implementierung beschrieben.

Ruft der Benutzer ein Modul des Helpers über den jeweiligen Befehl in der Konsole auf, so wird die jeweilige *Call* Methode, mit denen vom Benutzer gegebenen Parameter, ausgeführt. Es existiert für jeden der Berechnungsschritte der Gradienten, LH-Werte, LH-Cluster und Räumlichen-Cluster eine eigene statische Klasse, die eine öffentliche Methode besitzt. Diese berechnet für die gegebenen Parameter den jeweiligen Schritt und gibt das Ergebnis zurück. Beispielsweise wird in der *Call* Methode des *LHHistogram* Moduls, zuerst die *CalcGradientVolume* Funktion der *Gradient* Klasse mit dem Intensitätsvolumen als Parameter aufgerufen. Danach wird die *LHValuesVolume* Methode der *LHValues* Klasse mit dem Ergebnis der vorherigen Funktion als Parameter ausgeführt. Aus dessen Ergebnis wird in der *Call* Methode des Moduls direkt das LH-Histogramm erstellt und gespeichert. Im Modul *ClusterVolume* läuft die Berechnung ebenso über das Aufrufen von den Methoden der jeweiligen Klassen ab. Das Modul *MergeCluster* führt seine Berechnung komplett in der *Call* Funktion aus, da die Kalkulation nicht sehr aufwändig ist. Die Implementierung der Klassen und deren Methoden ist das Thema dieses Kapitels und wird im Folgenden genauer erläutert.

5.1. Gradient

Bei der Kalkulation der Gradienten wird parallel über jeden Voxel im Intensitätsvolumen iteriert und die Implementierung von [3] aufgerufen.

Für die Berechnung der Methode wird eine Gewichtung und die Koordinaten aller 64 Punkte im Koordinatensystem der lokalen Nachbarschaft benötigt. Da das gesamte Volumen die selbe Voxellänge hat und die Koordinaten in der lokalen Nachbarschaft immer gleich sind, können diese beiden Werte für alle 64 Nachbarn einmalig in einem Vorverarbeitungsschritt berechnet werden. Sie werden dabei in einem 64 Elemente großes Array, mit der gleichen Nummerierung gespeichert, wie in Abbildung 2 gezeigt. Bei der Kalkulation jedes Gradienten müssen lediglich die beiden Arrays durch iteriert werden, um die entsprechenden Gewichtung und Koordinaten jedes Nachbarn zu erhalten.

5.2. LH-Werte

In Listing 1 ist der Code zur Berechnung der High-Werte zu sehen. Der Code liegt innerhalb einer while-Schleife, die so lange aufgerufen wird, bis die Berechnung des Low- und des High-Wertes abgeschlossen, also *LisFinished* und *HisFinished* true sind.

Die Berechnung des Low-Wertes ist ebenfalls in der Schleife und sieht bis auf die Richtung der neuen *absVector* und *secondAbsVector* gleich aus. Folglich sind alle hier gegebenen Erklärung und Anmerkungen ebenso auf die Berechnung der Low-Werte zu beziehen.

Listing 1: Code zur Berechnung der High-Werte

```
1 if (!HisFinished)
2 {
3     FVector3 absVectorH = volumeVectors[xH, yH, zH];
4     FVector3 absVectorHNormalized = absVectorH.Normalize();
```

5. Implementierung

```
5     int tempNewXH = (((int)Math.Ceiling(xH + (stepSize *
6         absVectorHNormalized.X)))) ;
7     int tempNewYH = (((int)Math.Ceiling(yH + (stepSize *
8         absVectorHNormalized.Y)))) ;
9     int tempNewZH = (((int)Math.Ceiling(zH + (stepSize *
10        absVectorHNormalized.Z)))) ;
11    if (tempNewXH < volumeVectors.Dimension.X && tempNewYH <
12        volumeVectors.Dimension.Y && tempNewZH <
13        volumeVectors.Dimension.Z && !(tempNewXH < 0) && !(tempNewYH <
14        0) && !(tempNewZH < 0))
15    {
16        secondAbsVectorH = volumeVectors[tempNewXH, tempNewYH,
17            tempNewZH].Normalize();
18        highestIntensity.Add(gradientIntensity[xH, yH, zH]);
19        if (absVectorH.Length() < extremumLength)
20        {
21            HisFinished = true;
22        } else
23        {
24            if (highestIntensity[highestIntensity.Count - 1] ==
25                highestIntensity[highestIntensity.Count - 2] &&
26                highestIntensity.Count > 3){
27                HisFinished = true;
28            }
29            xH = (((int)Math.Ceiling(xH + 0.5 * stepSize *
30                (absVectorHNormalized.X + secondAbsVectorH.X)))) ;
31            yH = (((int)Math.Ceiling(yH + 0.5 * stepSize *
32                (absVectorHNormalized.Y + secondAbsVectorH.Y)))) ;
33            zH = (((int)Math.Ceiling(zH + 0.5 * stepSize *
34                (absVectorHNormalized.Z + secondAbsVectorH.Z)))) ;
35            if (xH > volumeVectors.Dimension.X-1 || yH >
36                volumeVectors.Dimension.Y-1 || zH >
37                volumeVectors.Dimension.Z-1 || xH < 0 || yH < 0 || zH
38                < 0)
39            {
40                HisFinished = true;
41            }
42        }
43    } else
44    {
45        HisFinished = true;
46    }
47 }
```

Am Anfang eines Durchlaufes wird der normalisierte Gradienten des aktuellen Punktes in *absVectorNormalized* gespeichert. Anschließend wird der Punkt des zweiten normalisierten Vektors in den Zeilen 5 bis 7 berechnet. Liegt dieser außerhalb des Volumens, so wird die Integration beendet. Liegt er jedoch innerhalb des Volumens, so wird der *secondAbsVector* ausgelesen und der Intensitätswert des aktuellen Punktes zu der Liste *highestIntensity* hinzugefügt. Ist der Gradient des aktuellen Punktes kleiner als die *extremumLength*, welche im Falle von CT-Daten bei null liegt, wird die Integration beendet. Andernfalls wird zunächst mithilfe der *highestIntensity* Liste nach Schleifen gesucht. Bei sehr kleinen Gradienten, die jedoch größer als null sind, kann es vorkommen, dass das Verfahren immer wieder den selben Punkt findet und somit in einer Endlosschleife feststeckt. Wie in Zeile 17s zu sehen ist, wird dabei erst getestet, ob die Liste schon mehr als 3 Einträge hat. Da anfangs vor der while-Schleife bereits der Intensitätswert des

Startvoxels zur Liste hinzugefügt werden muss. Dies geschieht aus dem Grund, dass sonst, würde das Verfahren bei dem ersten Abbruchkriterium bei der ersten Iteration schon stoppen, die *highestIntensity* Liste leer wäre, gäbe es kein Ergebnis für den High-Wert. Das Testen von Schleifen könnte jedoch über die Koordinaten der iterierten Punkte geschehen, für den unwahrscheinlichem Fall, dass zwei durch die Integration hintereinander besuchte Punkte genau den selben Intensitätswert haben. Anschließend wird der nächste Punkt der Integration mithilfe von *absVectorNormalized* und *secondAbsVecotr* gemäß Heun's Methode ermittelt, die im Kapitel Methoden vorgestellt wurde. Erneut endet die Integration, falls der neu berechnete Punkt außerhalb des Volumens liegt.

Dieser Vorgang wiederholt sich für den High- als auch für den Low-Wert solange, bis die Integration aus einem der gegebenen Abbruchkriterien stoppt. In diesem Fall wird der letzte Eintrag der *highestIntensity* Liste ausgelesen und als Ergebnis für den High-Wert gespeichert. Ebenso passiert dies mit der für die Low-Werte äquivalenten *lowestIntensity* Liste. Der Fall, dass ein Iterationsschritt in einer der gezeigten Formen außerhalb des Volumens liegt, und deshalb das Verfahren gestoppt wird, kommt in ungefähr 2%-3% der Fälle vor. Des Weiteren kommt es bei zirka 25% aller Berechnung dazu, dass die LH-Werte vertauscht sind, also der Low- größer als der High-Wert ist. Dem wird entgegengewirkt, indem bei einem Vorkommen dieses Problems die beiden Werte vertauscht gespeichert werden. Es war noch nicht möglich, den Grund für diese Verwechslung herauszufinden.

5.3. LH-Clustering

Die Implementierung des LH-Clusterings wurde mit einer parallelen for-Schleife realisiert, die über die L-Werte mit der Schrittweite von 5 iteriert. Für jede Spalte i wird dann die in Algorithmus 1 beschriebene Funktion aufgerufen.

Die erste for-Schleife iteriert über die H-Werte. Der Parameter j beginnt mit dem Wert i und wird wie dieser mit der Schrittweite 5 hochgezählt. Dies hat den Grund, dass es im LH-Histogramm keine Einträge gibt, bei denen der Low- höher als der High-Wert ist. Alle durch die beiden Schleifen entstehenden Punkte (i, j) sind jeweils die Startpunkte einer Clustersuche. Um nicht zu viele Cluster zwischenspeichern zu müssen und erst ganz am Ende alle ähnlichen Cluster zu verschmelzen, werden bereits nach der Berechnung jeder Spalte deren Ergebniscluster soweit wie möglich verschmolzen. Weiterhin speichert der temporäre Cluster *AktuellerCluster* lediglich die Koordinaten der zum Cluster dazugehörigen Punkte im Histogramm ab, jedoch nicht die räumlichen Informationen der darin gespeicherten Voxel. Erst am Ende der parallelen Schleife, wenn alle Ergebnisse gesammelt und die Cluster verschmolzen wurden, werden diese Daten ausgelesen. Dies spart Speicherplatz und Berechnungszeit, da in einem einzigen Kästchen im LH-Histogramm mehrere Tausend Voxel gespeichert sein können.

Das LH-Clustering wurde anfangs wie im Paper von Nguyen [1] über das gesamte Histogramm mit einer Bandweite von 7% des maximalen LH-Wertes für jeden Kasten berechnet. Dies ist aus mehreren Gründen suboptimal. Zum einen ist der maximale LH-Wert sehr hoch, über 4000, obwohl nur zirka 0,3% der Voxel einen Wert über 2400 haben. Zum anderen liegen über 50% der LH-Werte unter 5. Die Kombination aus diesen Gründen machte das Clustering extrem langsam. Im Bereich von 0 bis 50 wurde in einem sehr großen Radius geclustert, wodurch mit jeder Iteration sehr viele Cluster gefunden und hinzugefügt werden mussten.

5. Implementierung

```

input : LH-Histogramm, Spalte i
output: LH-Clusters

AlleErgbnisCluster;
Mittelpunkt;
AlterMittelpunkt;
AktuellerCluster;
for  $j \leftarrow i$  to Rand des Histogramms, Schrittweite: 5 do
    Mittelpunkt  $\leftarrow (i, j)$ ;
    while Abstand(NeuerMittelpunkt, AlterMittelpunkt)  $> Threshold * Bandweite$ 
        do
            for  $(k, l) \leftarrow$  Alle Punkte im Radius der Bandweite um den Mittelpunkt do
                if NochNichtTeilDesClusters( $(k, l)$ ) then
                    | AktuellerCluster.Hinzufuegen( $(k, l)$ );
                end
            end
            AlterMittelpunkt  $\leftarrow$  Mittelpunkt;
            Mittelpunkt  $\leftarrow$  NeuenMittelpunktBerechnen(AktuellerCluster);
        end
        AlleErgbnisCluster.Hinzufgen(AktuellerCluster);
        Leeren(AktuellerCluster);
    end
    VerschmelzeNaheCluster(AlleErgbnisCluster);
return AlleErgbnisCluster;

```

Algorithmus 1: Pseudocode der Implementierung der LH-Cluster

Eine erste Maßnahme um das Problem zu beheben, war es, alle Low- und High- Werten, die über 2400 waren, jeweils auf 2400 zu setzen. Dadurch wurden nur wenige Werte verändert und der Clusteringradius wurde deutlich kleiner. Dies konnte das beschriebene Problem jedoch nicht alleine lösen, da immer noch viele Punkte gefunden und dies für jeden Kasten im Histogramm durchgeführt werden mussten. Also wurde als nächste Verbesserung eine Schrittweite beim Clustern eingeführt. Dies geschah aus der Beobachtung heraus, dass zwei bis drei oder eventuell sogar mehr direkt nebeneinanderliegende Kasten meist zum selben oder einen ähnlichen Cluster führten, und dass diese im letzten Schritt des Clusteringsverfahrens verschmolzen wurden. Diese Änderung verbesserte die Berechnungszeit erneut, jedoch dauerte die Berechnung immer noch relativ lange und es entstanden sehr viele Cluster. Diese durchzuschauen benötigte viel Zeit und das Gehirn wurde dabei meist als wenige, sehr große Cluster erkannt. Daraufhin wurde das Clustering auf einen LH-Wertbereich von 1025 bis 1075 begrenzt und die Bandweite auf 0,1% des maximalen LH-Wertes gesetzt. Diese Änderung führte zum Erfolg und das Ventrikelsystem war innerhalb der paar hundert Clustern mit ein wenig Aufwand zu erkennen.

Im LH-Histogramm eines CT-Datensatzes in Abbildung 9 sind viele dieser Beobachtungen zu erkennen. Es ist deutlich zu sehen, dass sehr wenige Werte über 2400 existieren. Des Weiteren ist die Ballungen an Kästchen mit sehr vielen Einträgen im linken unteren Eck mit LH-Werten zwischen 0 und 50 zu erkennen. Weiterhin in dem kleinen Bereich, auf den der schwarze Pfeil zeigt, ebenfalls eine große Ansammlung an Kästchen mit sehr vielen Einträgen zu sehen. Diese Ansammlung an Punkten stellt das Gehirn dar und es ist zu erkennen, dass die Intensitätswerte des Gehirns sehr nah beieinander liegen. Dies macht das genaue Clustern über diesen speziellen Bereich notwendig.

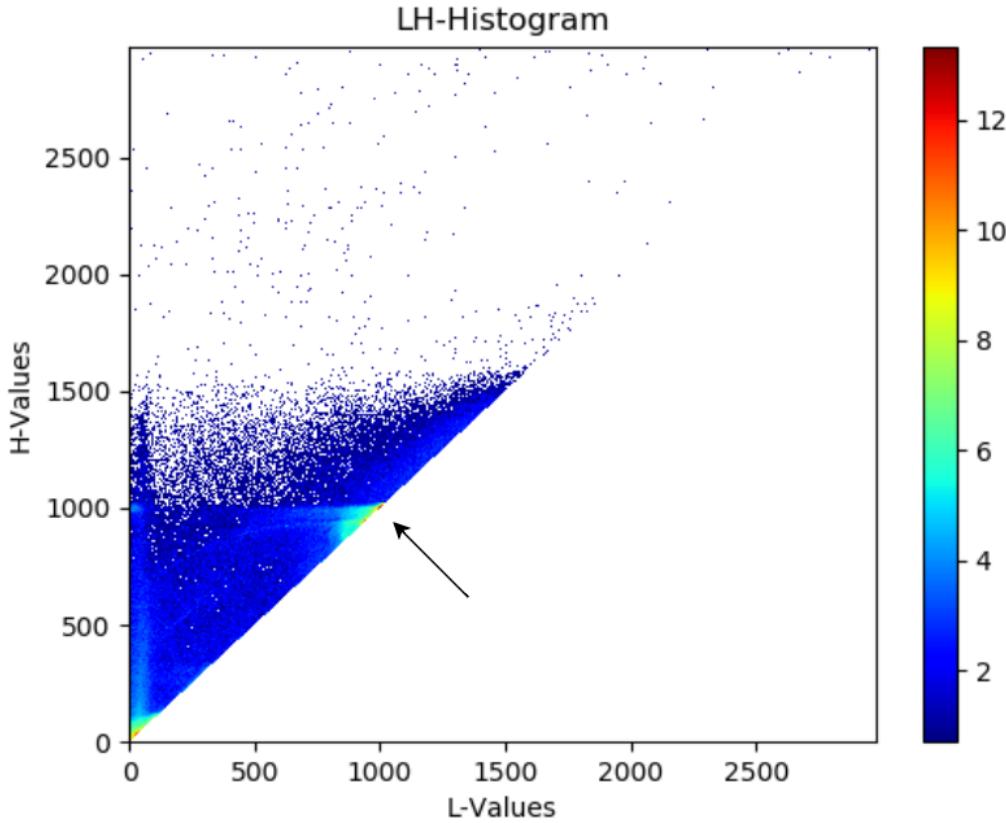


Abbildung 9: LH-Histogramm

5.4. Räumliches-Clustering

Die Berechnung der räumlichen Cluster und des Volumens, in dem die Cluster IDs gespeichert sind, wird in der statischen Klasse *SpatialClustering* ausgeführt.

Listing 2: Code zur Berechnung der räumlichen Cluster

```

1 List<IVector3> currentSpatialCluster = new List<IVector3>();
2     List<List<IVector3>> resultClusters = new List<List<IVector3>>();
3     FVector3 midPoint;
4     HashSet<IVector3> toRemove = new HashSet<IVector3>();
5     while (LHClusters2[i].Count > ClusterSizeMin)
6     {
7         midPoint = LHClusters2[i].ElementAt(0);
8         bool nothingMoreFound = false;
9         while (!nothingMoreFound)
10        {
11            foreach( IVector3 current in LHClusters2[i])
12            {
13                if (InDistance(midPoint, current))
14                {
15                    currentSpatialCluster.Add(current);
16                    toRemove.Add(current);
17                }
18            }
19            foreach(IVector3 remove in toRemove) {

```

5. Implementierung

```
20             LHClusters2[i].Remove(remove);
21         }
22         toRemove.Clear();
23         FVector3 newMidPoint =
24             NewMidpoint(currentSpatialCluster);
25         if (MinimumDistance > (SpatialDistance(newMidPoint,
26             midPoint)))
27         {
28             nothingMoreFound = true;
29         }
30         else
31         {
32             midPoint = newMidPoint;
33         }
34     }
35     if (currentSpatialCluster.Count > ClusterSizeMin)
36     {
37         resultClusters.Add(new
38             List<IVector3>(currentSpatialCluster));
39     }
40     currentSpatialCluster.Clear();
41 }
42 currentSpatialCluster.Clear();
43 return resultClusters;
```

Die Kalkulation der räumlichen Clustern wurde dabei, wie die Berechnung der LH-Cluster, mit einer parallelen for-Schleife realisiert. Diese iteriert über die LH-Cluster und ruft für jeden Cluster die in Listing 2 gezeigte Funktion auf. Die Cluster sind in der globalen Liste *LHClusters* gespeichert. Sie haben den Typ einer Liste von *IVector3*. Die Suche nach Clustern liegt in einer while-Schleife, die so lange ausgeführt wird, solange der LH-Cluster genug Elemente hat, damit ein Cluster mit der Mindestgröße an Elementen gefunden werden kann. Das Finden von Clustern läuft dabei folgendermaßen ab: Der erste Punkt im LH-Cluster wird als Startmittelpunkt gewählt. Danach wird durch alle Punkte iteriert und diejenigen, die innerhalb der Bandweite liegen, sowohl zum temporären Cluster *currentSpatialCluster* als auch zum HashSet der zu entfernenden Punkte *toRemove* hinzugefügt. Im Code ist dies in den Zeilen 7 bis 18 zu sehen.

Anschließend werden alle Elemente von *toRemove* aus dem LH-Cluster entfernt und wie beim LH-Clustering auch ein neuer Mittelpunkt für den aktuellen Cluster errechnet. Wenn das Abbruchkriterium der zwei naheliegenden aufeinanderfolgenden Mittelpunkte erfüllt ist, wird der temporäre Cluster *currentSpatialCluster* zur Liste der Ergebniscluster *resultClusters* hinzugefügt, falls er genug Elemente besitzt.

Für die Bandweite und die minimale Distanz des Abbruchkriteriums wurden im Paper von Nguyen [1] keine Werte angegeben. In dieser Bachelorarbeit wurde die Bandweite auf 15 und die Distanz auf 0,01 gesetzt. Diese Werte wurden durch Testen herausgefunden und erwiesen sich als zielführend. Anschließend wurde in der Funktion *ComputeIDs*, der die Cluster als Parameter übergeben werden, das Clustervolumen erstellt und zurückgegeben.

5.5. Unity

Die Erweiterung, die am Renderer vorgenommen wurde, wurde im Shader programmiert. Die dazugehörigen hinzugefügten Parameterfelder sind in Abbildung 10 zu sehen. Zuerst wurde eine drop-down-Liste *Volume coloring method* hinzugefügt, über die der Nutzer

5. Implementierung

zwischen den Modi *Default*, *SpecificValue* und *SpecificValueRange* wählen kann. Außerdem wurden drei Textfelder hinzugefügt.

Über das Parameterfeld *Specific value* ist es dem Anwender möglich, den spezifischen Wert für den Modus *SpecificValue* anzugeben. Weiterhin kann er über *Minimum volume value* die untere Grenze und über *Maximum volume value* die obere Grenze des Wertebereichs für den *SpecificValueRange* Modus angeben. Die Grenzwerte sind dabei jeweils inklusive. Des Weiteren wurde noch das Farbfeld *Color for specific value* hinzugefügt, welches die Farbe anzeigt, mit der hervorgehobenen Wert oder Wertebereiche dargestellt werden. Der Benutzer kann diese Farbe über das Parameterfeld verändern.

Der *Default* Modus steht für die schon vorher dagewesene und im Kapitel Design bereits beschriebene Implementierung der Farbgebung mit verschiedenen Grauwerten. Diese wurde in den beiden anderen beiden Modi übernommen. Der Unterschied besteht jedoch darin, dass allen Voxeln, die bei *SpecificValue* den spezifischen Wert oder bei *SpecificValueRange* innerhalb des Wertebereichs liegen, die Farbe des Farbfeldes *Color for specific value* zugewiesen wird. Allen anderen Voxel erhalten Grauwerte wie im *Default* Modus.

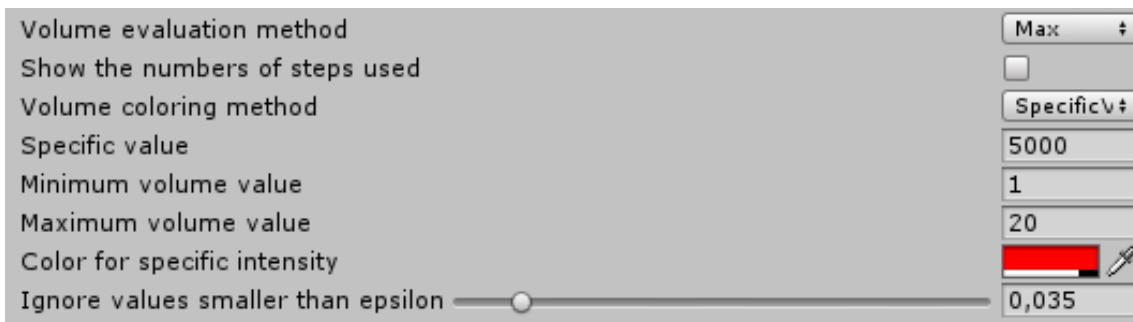


Abbildung 10: Parameterfelder in Unity

6. Ergebnisse

Dieses Kapitel beschäftigt sich mit den Ergebnissen der Segmentierung, sowie mit der Evaluation des Verfahrens. Der Abschnitt teilt sich dabei folgendermaßen auf. Als erstes werden die Ergebnisse der Visualisierung des Ventrikelsystems gezeigt und evaluiert. Dabei wurde zur Auswertung ein Interview mit einem Arzt durchgeführt. Danach wird eine Nutzerstudie vorgestellt, die die Benutzerfreundlichkeit des Systems testet. Abschließend wird die Berechnungszeit der Implementierung besprochen.

6.1. Visualisierung

6.1.1. Ventrikelsystem

Bevor die Visualisierungen ausgewertet werden können, muss zunächst das Ventrikelsystem erläutert werden. Das Ventrikelsystem besteht aus mehreren Hohlräumen im Gehirn, welche mit Hirnwasser, auch Liquor genannt, gefüllt sind. In Abbildung 11 ist eine Zeichnung des Systems zu sehen. Das Ventrikelsystem besteht aus vier verschiedenen Ventrikeln. Der linke und der rechte Seitenventrikel (die beiden oberen Bögen in der Abbildung), der dritte und vierte Ventrikel, die zwischen den beiden Seitenventrikeln liegen und nach unten weggehen. Die Seitenventrikel bestehen aus einem Vorderhorn (Nr. 1a) einem Hinterhorn (Nr. 1b) und einem Unterhorn (Nr. 2). Der dritte Ventrikel ist mit der Nummer 3 und der vierte Ventrikel mit der Nummer 4 versehen. Bei der Ventrikelpunktion, wird einer der beiden Seitenventrikeln im vorderen Bereich punktiert. Dies macht vor allem die Darstellung der Seitenventrikel relevant.

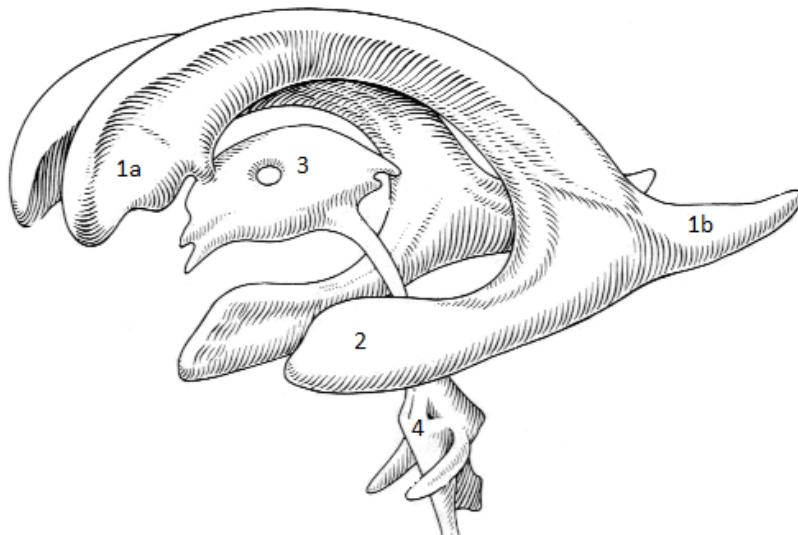


Abbildung 11: Zeichnung des Ventrikelsystems
Frei nach [4]

6.1.2. Visualisierungen

Die in diesem Unterkapitel gezeigten Segmentierungen wurde alle auf Volumen mit einer Auflösung von 256x101x256 Pixeln berechnet. In Abbildung 12 und Abbildung 13 ist die

6. Ergebnisse

Visualisierung eines normalen Ventrikelsystems, in Abbildung 14 und Abbildung 15 die Visualisierung eines weiteren normalen Ventrikelsystems und schließlich in Abbildung 16 und Abbildung 17 die Visualisierung des Ventrikelsystems eines Patienten mit Atrophie zu sehen. Die Datensätze werden im nächsten Unterkapitel erklärt. Die Abbildungen zeigen Screenshots aus der Darstellung in Unity, jeweils aus den Perspektiven von der Seite und von unten.

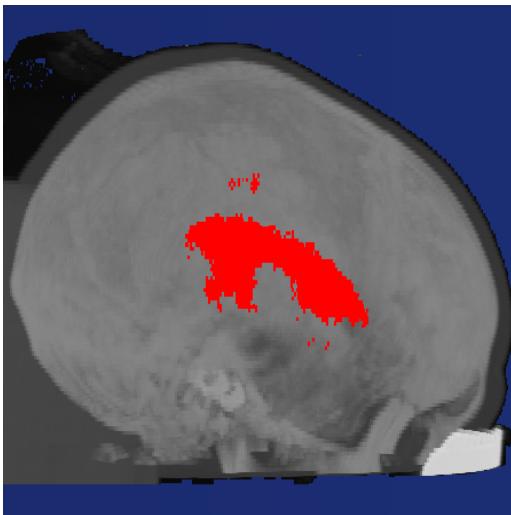


Abbildung 12: Visualisierung des ersten normalen Ventrikelsystems von der Seite

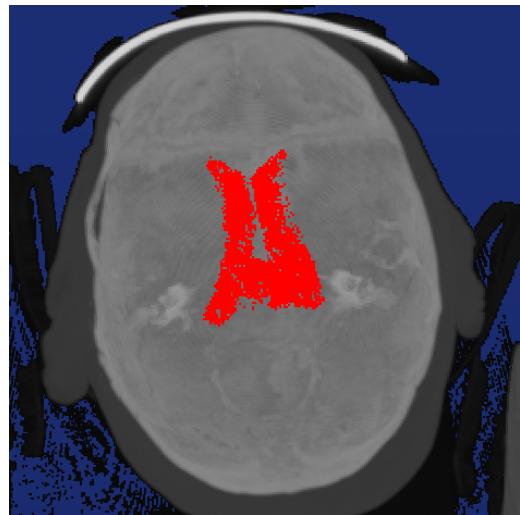


Abbildung 13: Visualisierung des ersten normalen Ventrikelsystems von unten

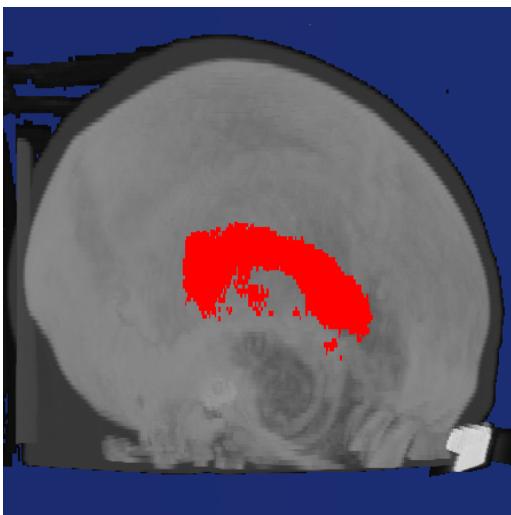


Abbildung 14: Visualisierung des zweiten normalen Ventrikelsystems von der Seite

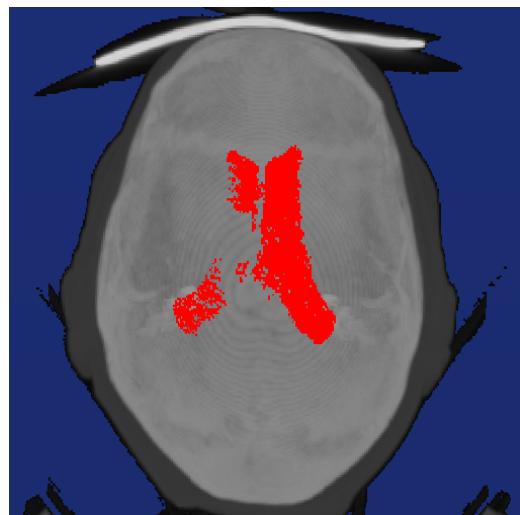


Abbildung 15: Visualisierung des zweiten normalen Ventrikelsystems von unten

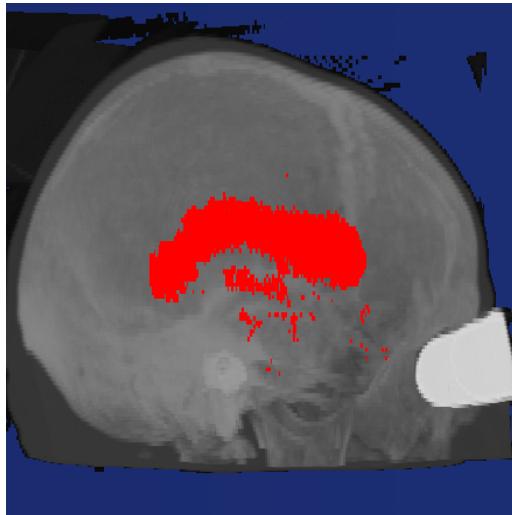


Abbildung 16: Visualisierung des Ventrikelsystems mit Atrophie von der Seite

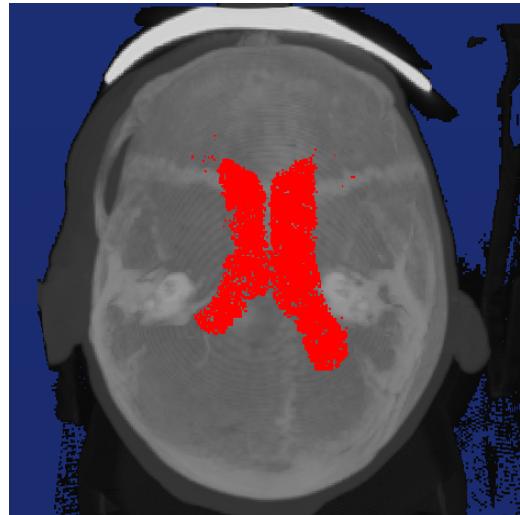


Abbildung 17: Visualisierung des Ventrikelsystems mit Atrophie von unten

6.1.3. Datensätze

Das Verfahren wurde an 15 verschiedenen CT-Datensätzen der Uni Ulm getestet. Darunter waren verschiedene Ventrikelsysteme. In Tabelle 1 ist eine Auflistung dieser zu sehen.

Datensatz (ID)	Anzahl
Normale Ventrikel (D1)	3
Schlanke Ventrikel (D2)	4
Mittellinienshift (D3)	2
Deformiertes (D4)	1
Atrophie (D5)	2
Hydrocephalus-Ventrikelblut (D6)	1
Subarachnoidale Einblutung (D7)	1
Blut-resorbiert (D8)	1

Tabelle 1: Verwendeten Datensätze

Da die Aufgabe das Ventrikelsystem anhand von CT-Daten automatisiert zu segmentieren sehr komplex ist, kam es bei Patienten veränderten Ventrikelsystemen, z.B. durch Verformung oder Krankheit, zu Problemen. Das Verfahren erkennt Strukturen anhand ihrer Grenzen. Deshalb war es nicht möglich, bei "D2: Schlanke Ventrikel" zu einer erfolgreichen Segmentierung zu gelangen. Sehr dünne Strukturen anhand von CT-Daten zu erkennen, ist kaum bis gar nicht realisierbar, da diese nicht hochauflösend genug sind. Dieses Problem tritt ebenfalls bei "D3: Mittellinienshift" und "D4: Deformiert" auf. Ein Mittellinienshift bezeichnet die Verschiebung der Mittellinie und eine Verformung des Ventrikelsystems, was beispielsweise durch Gewalteinwirkung auf den Kopf hervorgerufen werden kann. Bei "D4: Deformiert" wird das System aufgrund eines Tumors im Gehirn stark verformt. Durch diese Verformungen ist das Ventrikelsystem allgemein sehr schlecht bis gar

6. Ergebnisse

nicht von umliegenden Strukturen abgrenzbar.

Atrophie ist ein oft durch das Alter verursachter Schwund an Gehirnmasse. Bei diesem weiten sich alle Ventrikel und es ist im Allgemeinen mehr Liquor im Gehirn zu finden. Dadurch war es bei einem der beiden Datensätze von D5 nicht möglich, eine Segmentierung vorzunehmen, da durch die viele Flüssigkeit im Gehirn die Grenze zwischen dem System und der umliegenden Hirnmasse stark verschwommen. Ein ähnliches Problem entstand bei "D6: Hydrocephalus-Ventrikelblut" und "D7: Subarachnoidale Einblutung", bei denen die Ventrikel fast vollständig mit Blut vollgelaufen sind. Dadurch war es nicht möglich das System vom restlichen Gehirn abzugrenzen. Beim "D8: Blut-resorbiert" handelt es sich um den D6 Datensatz, nachdem das Blut entfernt wurde. Das Ventrikelsystem hat sich jedoch noch nicht wieder vollständig mit Liquor gefüllt, was erneut keine Segmentierung zuließ. Der dritte D1 Datensatz hat eine Verkalkung des liquorproduzierenden Plexus, weshalb auch hier weniger Liquor in den Ventrikeln zu finden war.

6.1.4. Auswertung

Um die Qualität der erzeugten Segmentierungen zu evaluieren, wurde ein Arzt von der Uniklinik Ulm interviewt. Diesem wurden die drei gezeigten Visualisierungen direkt in Unity vorgeführt. Während der Vorführung konnte der Mediziner selbst die Kamera durch die Darstellung lenken und das Ergebnis aus verschiedenen Blickwinkeln betrachten.

Anschließend bewertete er die Qualität der Ergebnisse, indem er drei verschiedenen Fragen zu den Darstellungen beantwortete. Er musste dabei seine Antwort immer auf einer Skala von 1 bis 5 angeben. Die Fragen lauteten:

- 1) Wie gut ist das Ventrikelsystem bei der Visualisierung zu erkennen?
sehr schlecht(1), sehr gut(5)
- 2) Wird das Ventrikelsystem in der Visualisierung vollständig dargestellt?
überhaupt nicht vollständig(1), vollständig(5)
- 3) Wie genau ist das Ventrikelsystem segmentiert?
überhaupt nicht segmentiert(1), ausschließlich das Ventrikelsystem ist segmentiert(5)

Die Antworten des Arztes werden in Tabelle 2 gezeigt. Allgemein merkte der Mediziner an, dass bei jeder Visualisierung, die durch das Verfahren erzeugt wurde, nur der linke und rechte Seitenventrikel zu sehen war. Die deutlich schmaleren dritten und vierten Ventrikel und die Hinterhörner der Seitenventrikel fehlten bei den Darstellungen. Er erklärte jedoch, dass diese bei einem gesunden Menschen sehr dünn und deshalb anhand von CT-Daten schwer bis gar nicht zu segmentieren seien. Weiterhin sind, wie schon erwähnt, die Seitenventrikel entscheidend für eine erfolgreiche Punktion.

Deshalb bezog sich die Beantwortung der zweiten Frage nach der Vollständigkeit des Ventrikelsystems lediglich auf die Vollständigkeit der beiden Seitenventrikel.

Die Bewertung des ersten normalen Ventrikelsystems fiel positiv aus mit 11 von 15 möglichen Punkten. Das Ventrikelsystem war klar und deutlich zu erkennen, jedoch fehlte ein Teil der Hinterhörner. Bis auf diese waren die Seitenventrikel vollständig zu sehen. Allerdings war die Segmentierung nicht exakt, da es mehrere kleine Ausreißer gab.

Das zweite normale Ventrikelsystem war ebenfalls klar zu erkennen. Jedoch wurde fast ausschließlich der linke Seitenventrikel segmentiert. Die Bewertung der Vollständigkeit fiel mit einer vier hoch aus, da der eine Seitenventrikel sehr klar, deutlich und umfassend zu sehen war. Dieser war besser und glatter als die des ersten normalen Ventrikelsystems

6. Ergebnisse

Ventrikelsystem	1. Frage	2. Frage	3. Frage
Normal 1	4	4	3
Normal 2	4	2	3
Atrophie	4	3	2

Tabelle 2: Ergebnisse des Interviews mit einem Arzt

sein, da auch das Hinterhorn zu erkennen ist.

In einem Gehirn eines unter Atrophie leidenden Menschen ist deutlich mehr Liquor als bei einem gesunden Menschen zu finden. Diese Flüssigkeiten werden vom Verfahren ebenfalls erkannt, weshalb die Segmentierung etwas verschwommen erscheint und nicht die kompletten Seitenventrikel erfasst werden. Jedoch vermutete der Arzt, dass viele der Ausreißer zum dritten Ventrikel gehören könnten, da sich dieses, wie alle Ventrikel bei einer Atrophie, weitet. Trotz der Flüssigkeiten im Gehirn war das Ventrikelsystem in der Darstellung deutlich auszumachen.

Im Allgemeinen waren die Darstellungen nicht glatt genug und es existieren viele Ausreißer. Das Ventrikelsystem war jedoch bei allen Visualisierungen eindeutig zu erkennen und die Aufgabe das System zu segmentieren war gelungen.

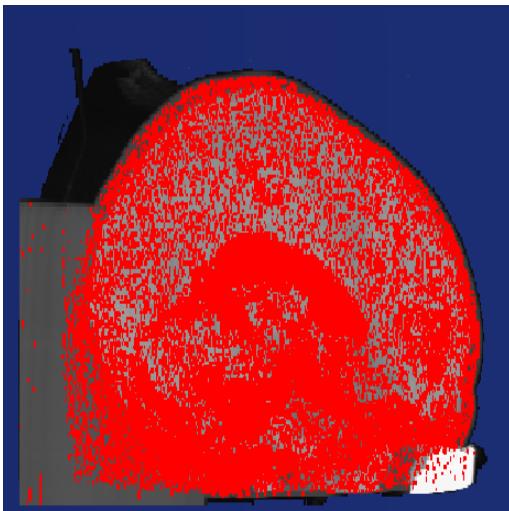


Abbildung 18: Visualisierung des ersten normalen Ventrikelsystems von der Seite mithilfe von Unity

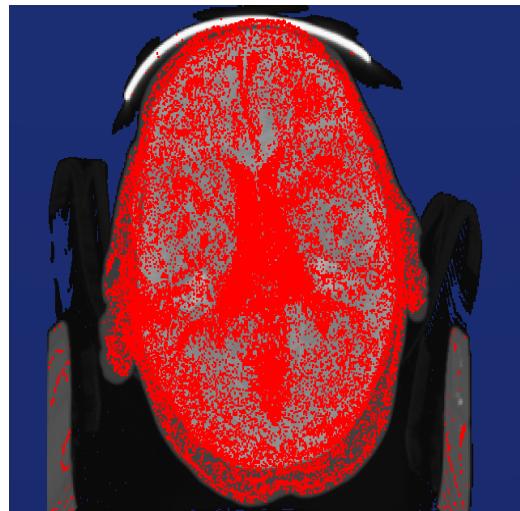


Abbildung 19: Visualisierung des ersten normalen Ventrikelsystems von unten mithilfe von Unity

Zur weiteren Auswertung der Ergebnisse wurde das erste normale Ventrikelsystem aus Abbildung 12 und Abbildung 13 in Unity und mit der MITK-Workbench, die auch zum Umwandeln der DICOM-Dateien benutzt wurde, visualisiert.

In Unity wurde dabei die Erweiterung zum Hervorheben von Wertebereichen benutzt. Diese wurde auf 1025 bis 1030 eingestellt, da das Ventrikelsystem Intensitätswerte in diesem Bereich hat. Dieses Vorgehen entspricht einer simplen eindimensionalen Transferfunktion, die abhängig von den Intensitätswerten Voxel, einfärbt. Die Ergebnisse sind in Abbildung 18 und Abbildung 19 zu sehen. Das Ventrikelsystem ist zwar zu sehen, es

6. Ergebnisse

gibt jedoch sehr viele Ausreißer, die es fast unmöglich machen, die Ventrikel eindeutig zu erkennen. Diese simple Vorgehensweise erzielt kein gewünschtes Ergebnis.

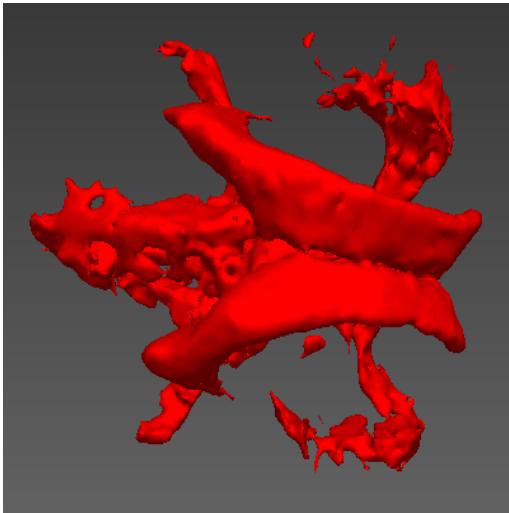


Abbildung 20: Visualisierung des ersten normalen Ventrikelsystems von oben mithilfe von MITK

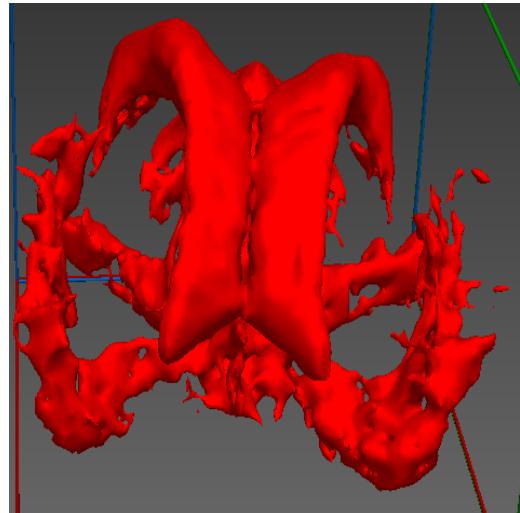


Abbildung 21: Visualisierung des ersten normalen Ventrikelsystems von vorne mithilfe von MITK

In der MITK-Workbench gibt es verschiedene Segmentierungstools. Darunter ist ein Region Growing Tool, bei dem der Nutzer einen *seed* Punkt in den Schnittbildern wählen kann. Über die verwendete Kostenfunktion gibt es keine Informationen. Anschließend können die Löcher der Auswahl mit einem *closing* Filter geschlossen und mit einem weiteren Tool ein geglättete 3D Ansicht des Ergebnis erzeugt werden. Eine genaue Beschreibung über das Anwenden der Tools ist im Anhang dieser Arbeit zu finden.

Screenshots der Ergebnisse sind in Abbildung 20 und Abbildung 21 zu sehen. In dem von der Workbench erzeugte Ergebnis ist das Ventrikelsystem gut zu erkennen. Die Seitenventrikel sind vollständig und besitzen eine glatte Oberfläche. Sogar große Teile des dritten und vierten Ventrikels sind Teil der Segmentierung. Allerdings sind auch ganze Bereiche hervorgehoben, die nicht zum Ventrikelsystem gehören. Diese könnten vom Benutzer manuell in jedem Schichtbild des Datensatzes entfernt werden, was jedoch sehr zeitaufwändig wäre.

6.2. Nutzerstudie

Im Rahmen der Evaluation der Benutzerfreundlichkeit des Verfahrens wurde eine kleine Nutzerstudie mit 5 Teilnehmern durchgeführt. Die Probanden waren alle Studenten aus verschiedenen Studiengängen im Alter zwischen 22 und 26 Jahren. Als erstes wurde den Teilnehmern der Ablauf und die vom Benutzer erforderlichen Schritte, um eine Visualisierung des Ventrikelsystems zu erhalten, mittels einer Vorführung durch den Interviewer gezeigt. Anschließend sollten die Probanden selbst das eben gelernte anwenden und das Programm selber ausführen. Dabei bekamen sie, wenn sie nicht weiterwussten, Hilfe vom Versuchsleiter.

Hinterher füllten die Teilnehmer einen NASA-TLX Bogen zu der Aufgabe aus. Dieser

6. Ergebnisse

besteht aus zwei Parts. Als erstes muss der Befragte in den Kategorien Geistige Anforderung, Körperliche Anforderung, Zeitliche Anforderung, Leistung, Anstrengung und Frustration die auszuführende Aufgabe jeweils auf einer Skala von 5 bis 100 bewerten. Die Werte der Skalen wachsen dabei in Fünfer Schritten an. Diese Bewertung ergibt hinterher die Gewichtung der Kategorien. Als zweites werden alle Kategorien paarweise miteinander verglichen und der Probanden muss angeben, welcher der beiden Kategorien für das Lösen der Aufgabe bedeutsamer war. Dadurch erhält jede Kategorie eine Anzahl an Klicks. Anschließend wird die Klickzahl durch 15, die Anzahl aller Klicks geteilt. Dies ergibt die die Wichtung der Kategorie. Diese Wichtungen werden mit deren jeweiligen Gewichtungen multipliziert und die Ergebnisse zur Gesamtbeanspruchung der Aufgabe aufaddiert. Die durchschnittlichen Ergebnisse der einzelnen Kategorien werden in Tabelle 3 gezeigt.

Kategorie	Gewichtung	Klicks	Wichtung
Geistige Anforderung	39	4,8	0,32
Körperliche Anforderung	23	1,8	0,12
Zeitliche Anforderung	29	1,2	0,078
Leistung	25	2,4	0,156
Anstrengung	36	2,4	0,156
Frustration	35	2,4	0,156

Tabelle 3: Durchschnittlichen Ergebnisse des NASA-TLX Bogens

Der durchschnittliche Wert für die Gesamtbeanspruchung lag bei 38,32.

Zunächst werden die Ergebnisse des ersten Teils betrachtet. Hierbei fällt als erstes auf, dass die bei Kategorien Geistige Anforderung, Anstrengung und Frustration durchschnittlichen die höchsten Bewertungen abgegeben wurden. Bei der Geistigen Anforderung und der Frustration gaben vor allem die Teilnehmer ohne oder mit nur wenig Programmiererfahrung höhere Werte an. Bei der Kategorie Anstrengung ist anzumerken, dass von zwei Probanden deutlich höhere Werte als vom Rest gegeben wurde. Alle anderen Kategorien wurden in etwa ähnlich bewertet.

Als nächstes werden die Ergebnisse des zweiten Parts diskutiert. Die höchste durchschnittliche Klickzahl wurde von der Kategorie Geistige Anforderung erzielt. Unabhängig von den Programmierkenntnissen der Probanden wurde diese Kategorie als die deutlich bedeutsamste angesehen. Die anderen Kategorien unterscheiden sich nur geringfügig und hatten meist Klickzahlen zwischen null und drei.

Als letztes wird die Gesamtbeanspruchung betrachtet. Dieser unterscheidet sich bei den Teilnehmern vor allem durch die unterschiedliche Bewertung der Kategorien Geistige Anforderung und Frustration sehr stark von einander. Die berechneten Gesamtbeanspruchungen lagen bei 14,6 27,3 40 40,6 und 63. Dies zeigt den großen Unterschied zwischen den Personen, die viel Programmiererfahrungen besitzen, und denen, die wenig oder noch nie programmiert haben. Die Benutzung einer Konsole fiel den Unerfahrenen vergleichsweise sehr schwer. Des Weiteren ist die Notwendigkeit die richtigen Suffixe und das *u* beim *ClusterVolume* Modul angeben zu müssen eine Hürde. Weiterhin wäre es einfacher, wenn nur ein Programm verwendet werden müsste.

6. Ergebnisse

Trotz der Schwierigkeiten gaben alle Probanden an, auch jene ohne Programmiererfahrung, dass sie die Aufgabe mit einer gut erklärten, ausführlichen Dokumentation alleine, ohne weitere Hilfe durchführen könnten.

6.3. Berechnungszeit

Die in diesem Unterkapitel vorgestellten Zeitmessungen wurden alle auf einem Computer mit einem 3.70GHz Intel Core(TM) i7-8700K CPU mit 32GB RAM ausgeführt. Um die Berechnungszeit des Systems evaluieren zu können, wurde die Kalkulation des gesamten Clusteringverfahrens sowie die Berechnung des LH-Histogramms auf drei Volumen verschiedener Größen durchgeführt. Damit der Vergleich nicht von unterschiedlichen Volumendaten verfälscht wird, wurden alle Volumen aus dem gleichen CT-Datensatz generiert.

Dabei wurde die Originalgröße, die ein Auflösung von 512x201x512 Voxeln besitzt, mithilfe des Resamplemoduls des Helpers verkleinert. Die beiden anderen Volumengrößen haben dabei die Hälfte, mit 256x101x256 Voxeln, und Dreiviertel, mit 384x151x384 Voxeln, der Auflösungen des Originalvolumens. Es war geplant, dass noch ein vierter Volumen zum Vergleich hinzugezogen wird. Jedoch war es aus einem unbekannten Grund nicht möglich, die beiden Berechnungen mit dem gevierten Volumen durchzuführen. Des Weiteren funktionierte beim Originalvolumen lediglich die Berechnung des LH-Histogramms. Die Kalkulation des Clusteringverfahrens war nicht möglich, vermutlich weil es bei dieser hohen Auflösung zu viele Daten für die aktuelle Implementierung zu berechnen gab. Die Berechnungszeit hängt stark von der Größe des Eingabevolumens ab. Dies ist in Tabelle 4 sehr gut zu erkennen. Sie gibt einen Überblick über die Berechnungszeiten der verschiedenen Volumengrößen.

Volumengröße	LH-Histogramm [s]	Komplettes Verfahren [s]
Halbes Volumen (256x101x256)	30	50
Dreiviertel Volumen (384x151x384)	90	380
Ganzes Volumen (512x201x512)	225	-

Tabelle 4: Überblick über die Berechnungszeiten der verschiedenen Volumengrößen

Dabei ist wichtig zu beachten, dass die Zeit zur Berechnung der LH-Histogramme die gleiche Zeit wie die Kalkulation der LH-Werte im gesamten Verfahren benötigt. Die Berechnungsdauer der Gradienten ist hierbei zirka doppelt so lange wie die der LH-Werte. Wird die Berechnungszeit des Histogramms von der Kalkulationszeit des gesamten Verfahrens abgezogen, ergibt dies die Zeit, die die beiden Clusteringschritte benötigen.

Volumengröße	LH-Histogramm [Voxel/s]	Clustering [Voxel/s]
256x101x256	$220 * 10^3$	$330 * 10^3$
384x151x384	$247 * 10^3$	$76 * 10^3$
512x201x512	$234 * 10^3$	-

Tabelle 5: Überblick über die Berechnungsraten der verschiedenen Volumengrößen

Eine interessante Beobachtung hierbei war, dass die Berechnung der LH-Histogramme abhängig von der Anzahl der Voxel in etwa gleich schnell abläuft. Das halbe Volumen

6. Ergebnisse

hat eine Gesamtvoxelzahl von ungefähr 6,6 Millionen, das 75% Volumen von zirka 22,2 Millionen und das Original von ca. 52,6 Millionen Voxeln. Die Anzahl der bearbeiteten Voxel pro Sekunde bei der Berechnung des LH-Histogramms sowie der Clusteringschritte ist in Tabelle 5 zu sehen.

Dabei ist zu beobachten, dass beim der Kalkulation der LH-Histogramme keine großen Unterschiede zwischen den Raten existiert. Beim halben Volumen werden etwa 220 Tausend, beim dreiviertel Volumen ungefähr 247 Tausend und beim original Volumen 234 Tausend Voxel pro Sekunde bearbeitet. Der kleine Unterschied in der Rate lässt sich einerseits durch von der Volumengröße unabhängige Berechnungen und andererseits durch Messfehler erklären. Folglich kann die Aussage getroffen werden, dass die Berechnungszeit der LH-Werte bei dieser Implementierung in etwa linear mit der Anzahl an Eingabepixeln wächst.

Auf der anderen Seite ist jedoch auch zu erkennen, dass die beiden Clusteringschritte mit zunehmender Eingabegröße deutlich langsamer werden. Das Clustering des halben Volumens dauerte 20 Sekunden und hat damit eine Verarbeitungsrate von zirka 330 Tausend Pixeln pro Sekunde. Hingegen dauert es beim dreiviertel Volumen 290 Sekunden und erreicht damit gerade einmal eine Rate von 76 Tausend Pixeln pro Sekunde. Es benötigt also 14,5 mal so viel Zeit für die 3,3 fache Anzahl an Pixeln.

7. Fazit

Nachdem die Ergebnisse im vorherigen Kapitel gezeigt und evaluiert wurden, wird in diesem ein Fazit gezogen.

Das Ziel der Bachelorarbeit war die Implementierung eines Verfahrens zur erfolgreiche Segmentierung des Ventrikelsystem basierend auf Volumendaten. Dieses Ziel wurde erreicht.

Das Ventrikelsystem ist in den Visualisierungen deutlich zu sehen und als solches auch klar zu identifizieren. Die beiden für die Operation essentiell wichtigen Seitenventrikel werden hervorgehoben. Es fehlten zwar bei allen Visualisierungen der dritte und vierte Ventrikel, jedoch sind diese für die Ventrikelpunktion nicht notwendig. Das Hervorheben soll den Arzt im Operationssaal unterstützen. Dazu benötigt er nur für den Eingriff relevante Informationen. Das Anzeigen der, für die Operation irrelevanten, dritten und vierten Ventrikel könnte den Mediziner verwirren und die Punktion erschweren. Dies würde einen erfolgreichen Eingriff gefährden.

Es gab auch Datensätze, bei denen die Segmentierung nicht erfolgreich war. Diese hatten auffallende Besonderheiten, durch welche es schwer bis gar nicht möglich war mithilfe der CT-Daten eine Segmentierung zu erstellen.

Die Usability zur Durchführung des Prozesses zur Segmentierung ist zurzeit noch verbessерungswürdig. Der Nutzer benötigt Wissen darüber, welche Befehle wann im Helper ausgeführt werden müssen, deren Syntax und die passenden Dateiformate.

Trotzdem zeigte die Nutzerstudie, dass selbst Menschen ohne Programmiererfahrung die Aufgabe bewältigen können. Des Weiteren ist die Ausführung des Verfahrens mit einer guten Dokumentation auch ohne weitere Hilfe durchführbar. Nach einer kleinen Eingewöhnungsphase stellt das Durchführen für den Anwender kein Problem dar.

Schließlich muss noch die Verarbeitungsgeschwindigkeit des Verfahren betrachtet werden. Diese ist für die kleineren Auflösungen relativ schnell. Die benötigte Zeit zur Berechnung der Cluster wächst jedoch nicht linear mit der Größe der Eingabe. Hierbei ist jedoch unklar, ob dies an der Implementierung oder an dem Verfahren liegt, da über die Laufzeit des *Meanshiftclustering* keine Informationen vorliegen.

Zusammenfassend lässt sich sagen, dass die Bachelorarbeit zu einem positiven Ergebnis kam. Es war möglich, die im Kontext des HoloMed Projektes wichtigen Bereiche des Ventrikelsystems zu segmentieren und hervorzuheben.

In einer anschließenden Arbeit könnte das Verfahren noch weiter verbessert werden. Ein Ausblick, wie die verschiedenen genannten Probleme behoben und verbessert werden können, wird im nächsten Kapitel gegeben.

8. Ausblick

Innerhalb dieser Arbeit konnte ein Clustering-basiertes Verfahren zur Segmentierung erfolgreich umgesetzt werden.

Ein Versuch, die Ergebnisse der Segmentierungen zu verbessern, wäre ein noch genaueres Clustering anzuwenden. Dabei könnte einerseits die Bandweite und damit der Suchradius beim LH-Clustering verkleinert werden. Da dadurch deutlich mehr Cluster entstehen würden, die der Benutzer manuell nach dem Ventrikelsystem durchsuchen müsste, könnte weiterhin der LH-Wertebereich über dem geclustert wird, genauer abgesteckt werden. In Abbildung 18 und Abbildung 19 ist zu sehen, dass die beiden Seitenventrikel in einem Intensitätsbereich von 1025 bis 1030 bereits erkannt werden. Ob dieses Anpassen der Parameter jedoch zielführend wäre müsste getestet werden. Es ist völlig unklar, ob die CT-Daten hochauflösend genug wären, um ein präzises Clustering zu erlauben.

Ein weiterer Punkt für Verbesserungen ist die Benutzerfreundlichkeit. Der Anwender muss einen genauen Ablauf befolgen, um zu einem Ergebnis zu kommen. Dies könnte auf mehrere Weisen verbessert werden. Einerseits könnten die Befehle der Konsole vereinfacht werden, sodass der Benutzer weder das notwendige *u* noch den richtigen Suffix *.bin.txt* angeben müsste, da das Programm automatisch das Volumen umwandelt und im richtigen Dateiformat speichern würde.

Eine andere Idee wäre es, die zwei Programme zu einem einzigen Unity-Programm zu verschmelzen. Dadurch könnte der Benutzer direkt in Unity das Volumen laden, die IDs berechnen und das Verschmelzen vornehmen lassen. Dies würde weiterhin den Vorteil mit sich bringen, dass der Nutzer zwischen dem Volumen der IDs und dem finalen Volumen schnell hin und her wechseln könnte. Der Anwender müsste sich außerdem dadurch nicht mehr mit dem Speichern und Einlesen von Dateien beschäftigen. Weiterhin könnte die Nutzung durch das Erstellen einer GUI intuitiver gemacht werden, damit keine Eingabe von Befehlen in eine Kommandozeile mehr nötig wäre. Dies würde den Benutzern ohne Programmiererfahrung den Umgang mit dem Programm erleichtern.

Ein Problem, das vom Arzt benannt wurde, ist die nicht glatte Darstellung der Segmentierung, sowie die existierenden Ausreißer. Dies ließe sich lösen, indem auf das Endergebnis weitere Algorithmen zur Verbesserung der Segmentierung angewendet würden. Beispielsweise könnten mit Dilatations- und Erosionsfiltern die Oberfläche geschlossen und kleine Ausreißer eliminiert werden. Ein weiterer Verbesserungsvorschlag wäre, einen Region Growing Algorithmus zusätzlich am Ende des Verfahrens auf das Ergebnis anzuwenden. Dies könnte die Segmentierung deutlich glatter und ohne Löcher erscheinen lassen. Eine Implementierung solcher erweiterten Algorithmen könnte in einer weiterführenden Arbeit vorgenommen werden.

Des Weiteren könnte getestet werden, ob es möglich ist, die Berechnungszeit noch weiter zu verbessern. Aktuell laufen alle Kalkulationen auf der CPU. Die Clusteringschritte könnten jedoch womöglich auf der GPU deutlich schneller berechnet werden.

Literatur

- [1] NGUYEN, Binh P. ; TAY, Wei-Liang ; CHUI, Chee-Kong ; ONG, Sim-Heng: A clustering-based system to automate transfer function design for medical image visualization. In: *The Visual Computer* 28 (2012), Nr. 2, S. 181–191
- [2] *Forschungsprojekt Holomed.* https://www.ipr.kit.edu/projekte_2851.php
- [3] HONG, Di-hui ; NING, Gang-min ; ZHAO, Ting ; ZHANG, Mu ; ZHENG, Xiaoxiang: Method of normal estimation based on approximation for visualization. In: *Journal of Electronic Imaging* 12 (2003), Nr. 3, S. 470–478
- [4] *Kiefer-Saarland.* http://www.kiefer-saarland.com/anatomie_physiologie.htm
- [5] DREBIN, Robert A. ; CARPENTER, Loren ; HANRAHAN, Pat: Volume rendering. In: *ACM Siggraph Computer Graphics* Bd. 22 ACM, 1988, S. 65–74
- [6] LEVOY, M: *Display of surfaces from volume data: IEEE Comput. Graph. & Appl. Vol 8 No 3 (1988) pp 29–37.* 1988
- [7] KNISS, Joe ; KINDLMANN, Gordon ; HANSEN, Charles: Multidimensional transfer functions for interactive volume rendering. In: *IEEE Transactions on visualization and computer graphics* 8 (2002), Nr. 3, S. 270–285
- [8] LAN, Shouren ; WANG, Lisheng ; SONG, Yipeng ; WANG, Yu-ping ; YAO, Liping ; SUN, Kun ; XIA, Bin ; XU, Zongben: Improving separability of structures with similar attributes in 2D transfer function design. In: *IEEE transactions on visualization and computer graphics* 23 (2017), Nr. 5, S. 1546–1560
- [9] WESARG, Stefan ; KIRSCHNER, Matthias: Structure size enhanced histogram. In: *Bildverarbeitung für die Medizin 2009.* Springer, 2009, S. 16–20
- [10] WESARG, Stefan ; KIRSCHNER, Matthias ; KHAN, M F.: 2D histogram based volume visualization: combining intensity and size of anatomical structures. In: *International journal of computer assisted radiology and surgery* 5 (2010), Nr. 6, S. 655–666
- [11] HUANG, Runzhen ; MA, Kwan-Liu: RGVis: Region growing based techniques for volume visualization. In: *null IEEE*, 2003, S. 355
- [12] CHEN, Hung-Li J. ; SAMAVATI, Faramarz F. ; SOUSA, Mario C. ; MITCHELL, Joseph R.: Sketch-based Volumetric Seeded Region Growing. In: *SBM*, 2006, S. 123–129
- [13] CORREA, Carlos ; MA, Kwan-Liu: Size-based transfer functions: A new volume exploration technique. In: *IEEE transactions on visualization and computer graphics* 14 (2008), Nr. 6, S. 1380–1387
- [14] CORREA, Carlos ; MA, Kwan-Liu: The occlusion spectrum for volume classification and visualization. In: *IEEE Transactions on Visualization and Computer Graphics* 15 (2009), Nr. 6, S. 1465–1472
- [15] CORREA, Carlos D. ; MA, Kwan-Liu: Visibility-driven transfer functions. In: *Visualization Symposium, 2009. PacificVis' 09. IEEE Pacific IEEE*, 2009, S. 177–184

- [16] CORREA, Carlos D. ; MA, Kwan-Liu: Visibility histograms and visibility-driven transfer functions. In: *IEEE Transactions on Visualization and Computer Graphics* 17 (2011), Nr. 2, S. 192–204
- [17] TZENG, F-Y ; LUM, Eric B. ; MA, K-L: An intelligent system approach to higher-dimensional classification of volume data. In: *IEEE Transactions on visualization and computer graphics* 11 (2005), Nr. 3, S. 273–284
- [18] SOUNDARARAJAN, Krishna P. ; SCHULTZ, Thomas: Learning probabilistic transfer functions: A comparative study of classifiers. In: *Computer Graphics Forum* Bd. 34 Wiley Online Library, 2015, S. 111–120
- [19] FANG, Shiao-fen ; BIDDLECOME, Tom ; TUCERYAN, Mihran: Image-based transfer function design for data exploration in volume visualization. In: *Visualization'98. Proceedings IEEE*, 1998, S. 319–326
- [20] REITINGER, Bernhard ; ZACH, Christopher ; BORNIK, Alexander ; BEICHEL, Reinhard: User-centric transfer function specification in augmented reality. (2004)
- [21] WU, Yingcai ; QU, Huamin: Interactive transfer function design based on editing direct volume rendered images. In: *IEEE Transactions on Visualization and Computer Graphics* 13 (2007), Nr. 5, S. 1027–1040
- [22] BRUCKNER, Stefan ; GROLLER, M E.: *Volumeshop: An interactive system for direct volume illustration*. IEEE, 2005
- [23] SEREDA, Petr ; BARTROLI, Anna V. ; SERLIE, Iwo W. ; GERRITSEN, Frans A.: Visualization of boundaries in volumetric data sets using LH histograms. In: *IEEE Transactions on Visualization and Computer Graphics* 12 (2006), Nr. 2, S. 208–218
- [24] SERLIE, IWO ; TRUYEN, Roel ; FLORIE, Jasper ; POST, Frits ; VLIET, Lucas van ; VOS, Frans: Computed cleansing for virtual colonoscopy using a three-material transition model. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention* Springer, 2003, S. 175–183
- [25] SEREDA, Petr ; VILANOVA, Anna ; GERRITSEN, Frans A.: Automating transfer function design for volume rendering using hierarchical clustering of material boundaries. In: *EuroVis*, 2006, S. 243–250
- [26] *MITK*. [http://mitk.org/wiki/The_Medical_Imaging_Interaction_Toolkit_\(MITK\)](http://mitk.org/wiki/The_Medical_Imaging_Interaction_Toolkit_(MITK))

Anhang

A. Technische Dokumentation über die Benutzung der Segmentierungstools der MITK-Workbench

Als erstes muss in den DICOM Ordner der Ordner mit ungefähr 200 Elementen gefunden werden.

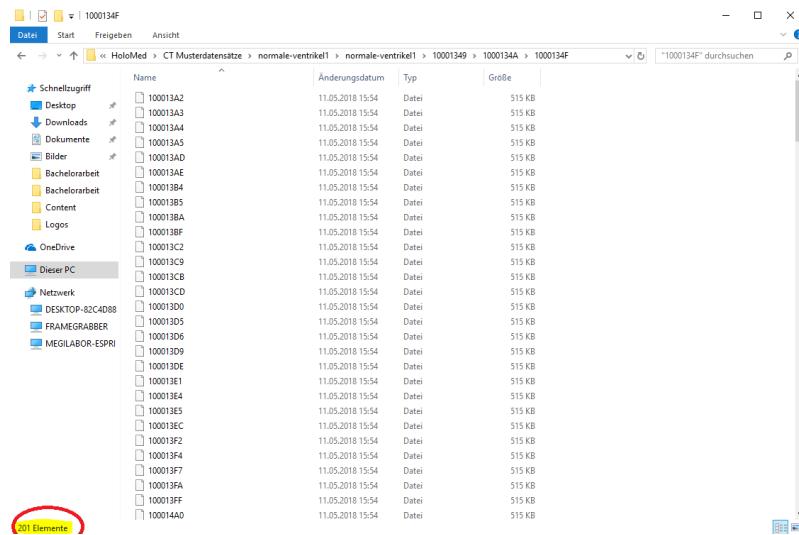


Abbildung A.1: Ordner finden

Anschließend kann eine der Dateien des Ordners via drag and drop in den *Data Manager* gezogen werden und das Fenster das erscheint bestätigt werden.

In der Mitte sind die verschiedenen Schnittbilder zu sehen, durch die mit dem Mausrad durchgegangen werden kann.

Die Zahlen von 1 bis 5 markieren verschiedene Tools. Werden diese angeklickt, öffnet sich an der Seite oder unter den Schnittbildern das jeweilige Tool. Um ein Tool auf ein Volumen anzuwenden, muss dieses immer im *Data Manager* ausgewählt sein.

Mit dem Regler am rechten Rand der Schnittbilder kann die Darstellung der Grauwerte eingestellt werden.

A. Technische Dokumentation über die Benutzung der Segmentierungstools der MITK-Workbench

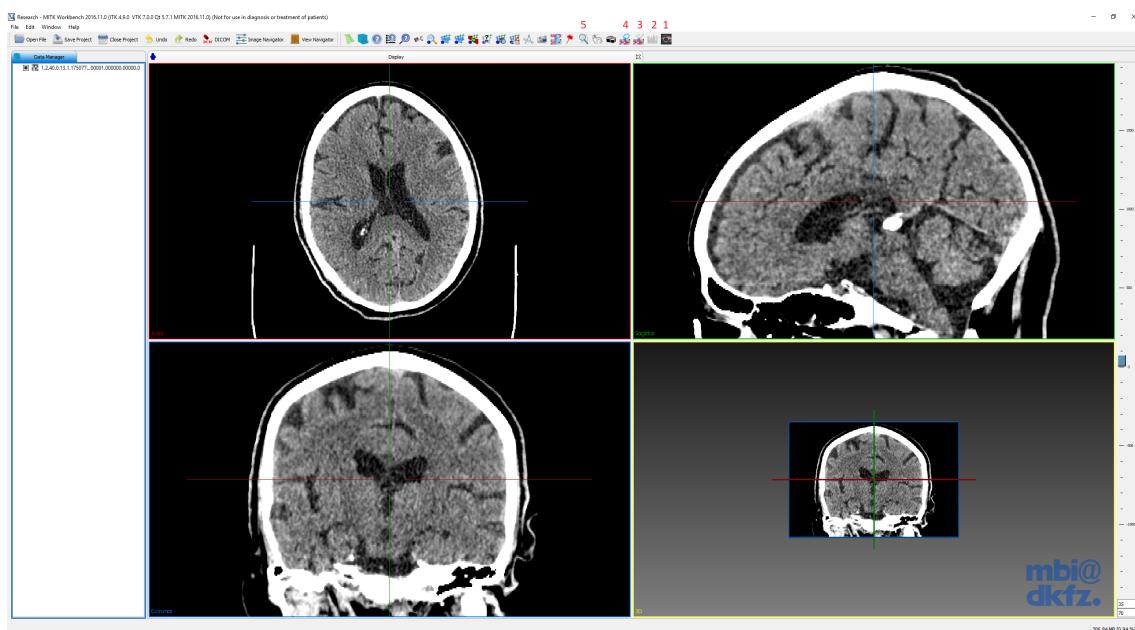


Abbildung A.2: Überblick über die Workbench

Das Tool mit der Nummer 1 macht es möglich eine Transferfunktion zu definieren. Anhand der Intensitätswerte, den Punkten über dem Diagramm und dem Farbstrahl darunter kann die Transferfunktion angepasst werden. Ist der Haken bei *Volumerendering* gesetzt, so wird die Visualisierung im Fenster links unten angezeigt.

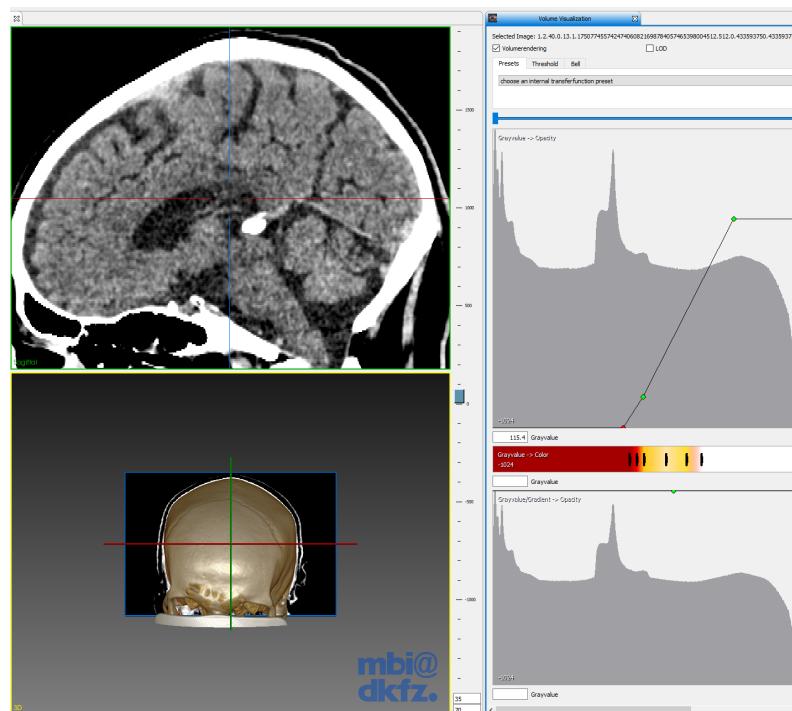


Abbildung A.3: Transferfunktionstool der MITK-Workbench

Das Tool mit der Nummer 2 zeigt Statistiken zum Volumen an, wie zum Beispiel Verteilung der Intensitätswerte, der Median dieser etc.

A. Technische Dokumentation über die Benutzung der Segmentierungstools der MITK-Workbench

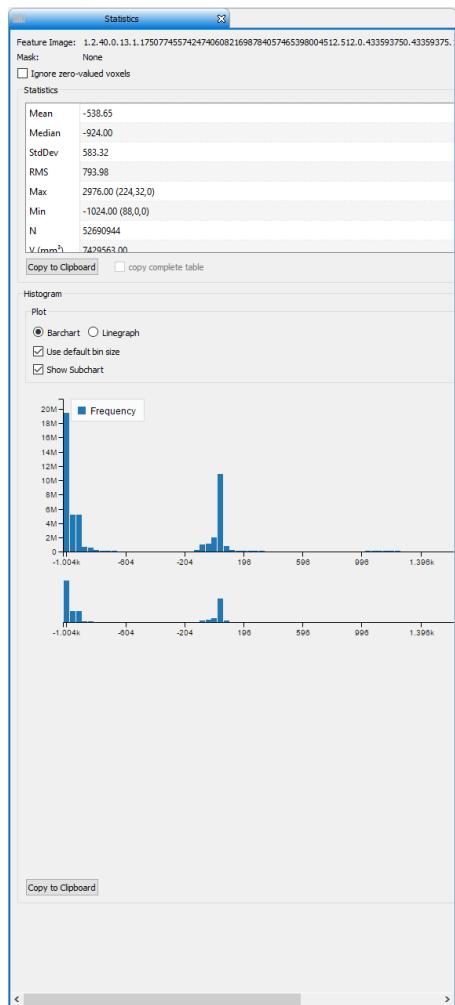


Abbildung A.4: Statistiktool der MITK-Workbench

Das Tool Nummer 3 bietet verschiedene 2D und 3D Segmentierungstools an. Zunächst muss jedoch eine neue Segmentierung über den rot markierten Knopf erstellt werden.



Abbildung A.5: Segmentierungstools der MITK-Workbench

Wurde dies getan, kann das 3D Tool *Region Growing 3D* ausgewählt werden.

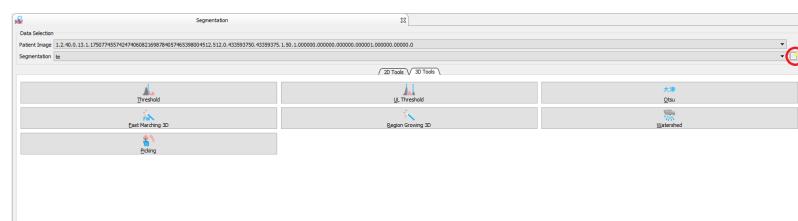


Abbildung A.6: 3D-Segmentierungstools der MITK-Workbench

A. Technische Dokumentation über die Benutzung der Segmentierungstools der MITK-Workbench

Anschließend kann mit Shift + Linke Maustaste eine Punkt im Volumen ausgewählt werden. Danach muss mit *Run Segmentation* die Segmentierung ausgeführt werden.

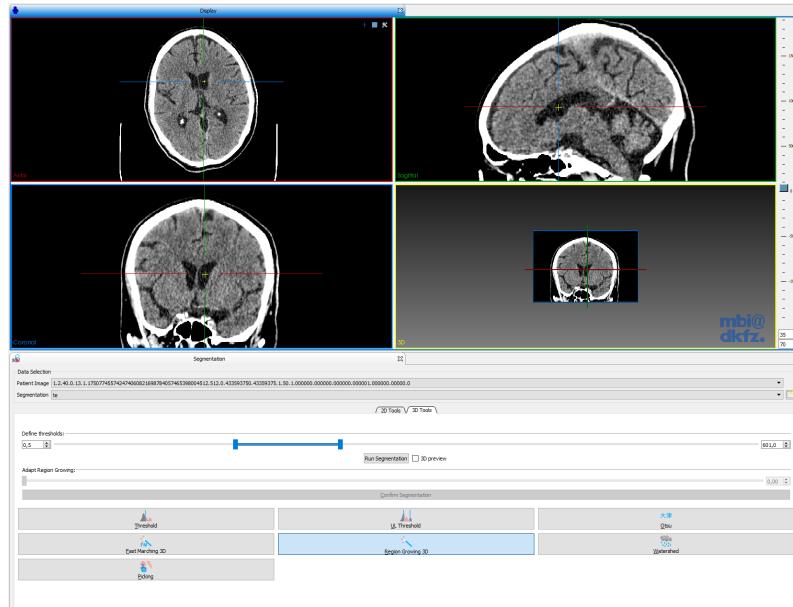


Abbildung A.7: Region Growing Tool der MITK-Workbench

Nun kann der Threshold angepasst und das Region Growing mit dem *Adapt Region Growing* Regler angepasst werden. Wird das gezielte Ergebnis erwünscht, kann es mit *Confirm Segmentation* bestätigt werden.

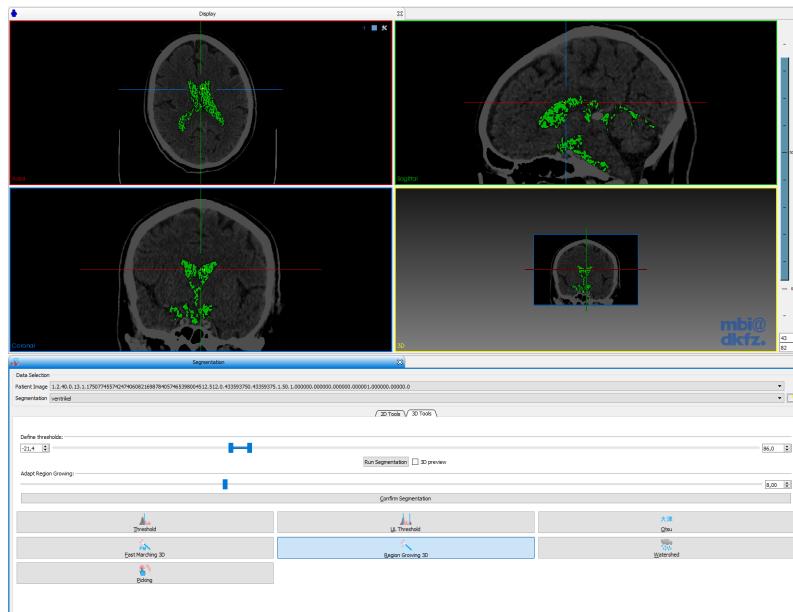


Abbildung A.8: Anpassen der Parameter des Region Growing Tools der MITK-Workbench

Mit Tool Nummer 4 kann das Ergebnis mit verschiedenen Operationen verbessert werden. Hierbei wird oft *Closing* benutzt. Anschließend kann mit Rechtsklick auf die Segmentierung im *Data Manager* der Befehl *Create smoothed polygon model* ausgeführt werden

A. Technische Dokumentation über die Benutzung der Segmentierungstools der MITK-Workbench

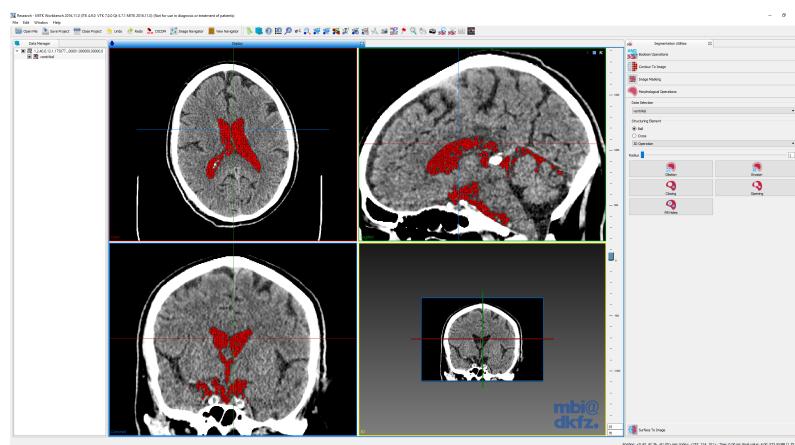


Abbildung A.9: Filtertool der MITK-Workbench

Nachdem alle diese Schritte befolgt wurden ist im linken Unteren Kasten die Segmentierung sehen. Mit Tool Nummer 5 kann die Darstellung der Segmentierung verändert werden.

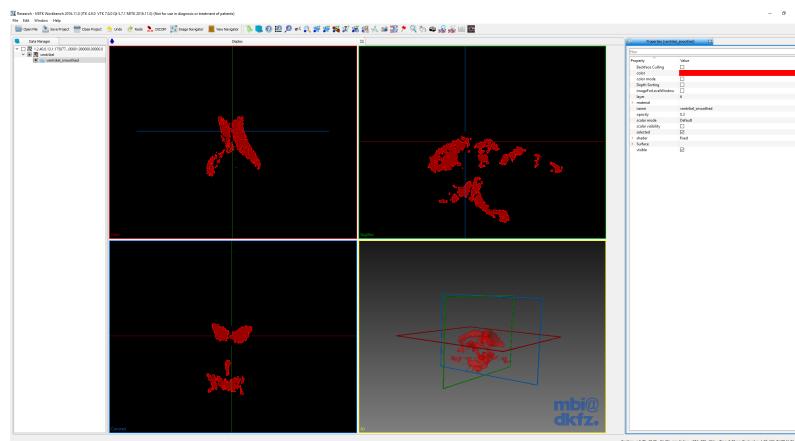


Abbildung A.10: Darstellungstool der MITK-Workbench