Karlsruher Institut für Technologie

Intelligente Industrieroboter

Implementierung eines Clusteringbasierten Verfahrens zur Visualisierung von Volumenmodellen

geschrieben von

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. B

23. August 2018

Liste der noch zu erledigenden Punkte

Zusammenfassung

Deutsche Zusammenfassung

Stickwörter: *Keywords, of, my, Thesis*

Inhaltsverzeichnis

1. Einleitung

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.
Ein Routineprozedur der Neurochirurgie ist die Punktion 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 Punktion durchzuführen, muss der Chirurg eine Bohrlochtrepanation am sogenannten Kocherpunkt durchführen. Der Arzt muss anhand äußerer anatomischer Landmarker diesen Punkt auf wenige Zentimeter genau finden und die Stichrichtung der Punktion ausmachen. Dieses Verfahren ist sehr fehleranfällig. So kommt es nur in zwei drittel aller Operationen zu einem optimalen Ergebnis, wofür oftmals mehrfach punktiert werden muss. In Abbildung 1 ist ein Bild eines solche Eingriffs zu sehen. Es ist klar zu erkennen wir der Chirurg mit dem linken Zeigefinger den inneren Augenwinkel zur Orientierung abtastet.

Abbildung 1: Fotografie einer Ventrikelpunktion
Quelle: ...

Im Rahmen des HoloMed Projektes wird daran gearbeitet den Chirurg bei diesem Eingriff zu unterstützen. Der Plan ist es, dass der Arzt mithilfe einer AR-Brille angezeigt bekommt wo sich das Ventrikelsystem befindet und somit mit einer niedrigeren Fehlerwahrscheinlichkeit die Operation durchführen kann.
Dazu muss zunächst ein CT-Scan des Kopfes gemacht werden. Dieser misst die Absorptionswerte aus verschiedenen Winkeln und erstellt daraus Volumendaten. Diese Absorptionswerte liegen in mehreren Schnittbildern vor.
Anhand der CT-Daten des Gehirns wird versucht das Ventrikelsystem hervorzuheben und zu visualisieren. Um diese Aufgabe zu lösen eignet sich eine Übergangsfunktion, auch Transferfunktion genannt.
Allgemein gesprochen ordnet eine Transferfunktion volumetrischen Daten optische Ei-

genschaften zu. Diese Aufgabe teilt sich in zwei Bereiche auf. Zum einen entscheidet die Funktion welche Daten gezeigt werden und zum anderen wie diese dargestellt werden, beispielsweise welche Farb- und Okklusionswerte diese erhalten. Dabei ist der erste Teil der wesentlich wichtigere und die Farbwahl kann als sekundär betrachtet werden.
Das Ziel dieser Bachelorarbeit ist es, mithilfe einer geeigneten Transferfunktion das Ventrikelsystem in CT-Daten des Kopfes hervorzuheben.
Dabei teilt sich der Rest dieser Arbeit folgendermaßen auf. In Kapitel 2, Stand der Wissenschenschaft und Technik, wird ein Überblick zu dem aktuellen Stand der Wissenschaft und Technik gegeben, während sich Kapitel 3, Methoden, mit der in dieser Arbeit verwendeten Methode beschäftigt. In Kapitel 4, Design, wird das Softwaredesign der Implementierung erklärt und in Kapitel 5, Implementierung, die Implementierung besprochen. Kapitel 6, Ergebnisse, zeigt die Ergebnisse des Verfahrens und evaluiert diese. In Kapitel 7, Fazit, wird ein Fazit gezogen und in Kapitel 8, Ausblick, werden mögliche Verbesserungen aufgezeigt.

2. Stand der Wissenschenschaft 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 kann ist einerseits zwischen dem Level an Automation eines Systems unterscheiden. Hierbei gibt es vollautomatische Verfahren, bei denen keine Interaktion mit den 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 das meiste selber machen muss.
Transferfunktionen unterscheiden sich weiterhin in ihrer Dimensionalität. Es gibt 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 Bildbasiert oder wenden Machinelearning an um zum gewünschten Ziel zu gelangen.
In diesem Abschnitt wird ein Überblick über die unterschiedlichen Vorgehensweisen von Transferfunktionen gegeben. Dabei werden diese im Folgenden in die Kategorien Eindimensionale, Zweidimensionale, Räumlichbasierte, Machinelearning, Bildbasierte und Clusteringbasierte Verfahren geteilt.
Dabei ist jedoch zu beachten, dass die Kategorien lediglich einer übersichtlichen Struktur dieses Abschnitts dienen. Die Kategorien hätten anders gewählt werden können. Des Weiteren können manche Verfahren nicht eindeutig in die hier genannte Unterteilung eingeteilt werden, da eine Mehrfachnennung möglich wäre.

2.1. Eindimensionale Verfahren

Die einfachste Form der Transferfunktionen, sind die eindimensionalen Transferfunktionen. Meistens werden in diesen, den Voxeln Farbe und Okklusion nur abhängig von deren Intensitätswerten zugeteilt.
Abgesehen von den niedrigen Berechnungszeiten, sind diese jedoch aus mehreren Gründen suboptimal. Medizinischen Daten werden gemessen und haben deshalb meist ein Rauschen, was die genaue Darstellung erschwert. Weiterhin sind die Intensitätswerte verschiedener Bereiche nah beieinander oder gar gleich und damit sind eindimensionale Transferfunktionen unpraktisch um verschiedene Materialien kenntlich zu machen. Trotzdem sind eindimensionale Transferfunktionen weit verbreitet und werden oft benutzt.
Eine etwas komplexere eindimensionalen Transferfunktion wird im Paper von Drebin [1] vorgestellt, bei der Voxel in verschiedene Materialien, anhand von Wahrscheinlichkeiten, die auf den Intensitätswerten basieren, klassifiziert werden.

2.2. Zweidimensionale Verfahren

Bei zweidimensionalen Transferfunktionen werden häufig als zweite Eingabegröße neben den Intensitätswerte die Länge der Gradienten hinzugezogen.
Ein frühes Beispiel für die Verwendung der Gradientenlänge, ist aus dem Jahr 1988 aus dem Paper von Levoy [2]. In diesem werden, mithilfe einer simplen Funktion Voxeln Opazitätswerte abhängig von deren Intensitätswerten und Gradientenlängen zugewiesen.

[3] SG-TF

Die Arbeit von Shouren Lan [4] befasst sich mit der Verbesserung von 2D Transferfunktionen die auf Skalarwerten und Gradienten(SG-TF) basieren. Genauer geht es darum Überlappungen von Bereichen die nicht zusammen gehören zu vermeiden.
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 andern 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 von einander getrennt. Strukturen der Klasse (iii) werden durch eine weitere niedrig dimensionale Transferfunktion getrennt. Anschließend werden Löcher, die bei dem Aufteilen der beiden Strukturen entstehen mithilfe von einem Dilatationsoperator geschlossen. Als letztes wird den unterschiedlichen Strukturen verschiedene Farben und Intensitäten zugewiesen.
Das Vorgehen von Lan führt zwar zu guten Ergebnissen und wurde sogar schon am Ventrikelsystem getestet. Jedoch wäre die Implementierung der Unterscheidung der Klassen und die verschiedenen Algorithmen zum Teilen, schließen von Löchern etc. zu Zeitaufwändig für den Rahmen dieser Bachelorarbeit.

2.3. Räumlichbasierte Verfahren

In der Arbeit von Wesarg und Kirschner [5, 6] wird das *Stucture-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 alle 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 in einer Tabelle nachgeschaut.
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 Regiongrowing.

Beispielsweise wird im Paper von Huang [7] ein solches Regiongrowingverfahren 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 3 verschiedene Kostenfunktionen bereit. 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 unpassende Elemente zu entfernen, wenn beispielsweise 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 Regiongrowing sehr zeitaufwändig ist, kann der Anwender auswählen, dass er zunächst nur eine gewisse Teil der Region sich errechnen lässt.

Im Vergleich zu Huan [7] benutzt Chen in seiner Arbeit [8] nicht nur ein *seed*basiertes Vorgehen, sondern fügt noch ein sketchbasiertes Verfahren davor ein.
Anfangs 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 vorherig vorgestellten Verfahren seinen *seed* setzen.
Dies erleichtert dem Benutzer die Anwendung, da er schneller zu seinem Punkt von Interesse gelangt ohne vorher durch diverse Querschnittbilder 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 [9] 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 mapt Farbe und Okklusion zu den entsprechenden Größen der Features des Volumens.
In einem weiteren Paper [10] beschreiben die beiden Forscher ein Verfahren, dass 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 [11] 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 umsetzten kann.
Aus diesem Grund wird eine Energiefunktion erstellt, die es zu minimieren gilt. Damit wird das Problem wie bei [12] zu einem Optimierungsproblem, welches mithilfe von progressiver Suche gelöst werden kann. 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 [13] nochmals verbessert. Sie führten multidimensionale Sichtbarkeitshistogramme 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. Machinelearning Verfahren

Die Arbeit von Tzeng [14] benutzt Machinelearning 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 Segmentierung 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 [15], bei dem der Anwender direkt im Volumen markieren kann, welche Gebiete für ihn von Interesse sind.
Dabei wird überwachtes maschinelles Lernen angewendet, um aus dieser Eingabe eine probabilistische Übertragungsfunktion abzuleiten. Dabei ist es wichtig, dass das ausgewählte Machinelearningverfahren eine probabilistische Klassifikation erlaubt. Des Weiteren darf es nicht nur zwischen zwei verschiedenen Klassen unterscheiden können, sondern multiple Klassen unterstützen.
In dem Paper wird das Verfahren mit fünf verschiedenen Machinelearningverfahren getestet und erklärt wie weit mit diesen das gegebene Verfahren umsetzbar ist. Es wurden *Gaussian Naive Bayes*, *k Nearest Neighbor*, *Support Vector Machines*, *Random Forests* und *Neural Networks* getestet.

2.5. Bildbasierte Verfahren

Eine weitere Art Transferfunktionen anzuwenden, sind Verfahren, die auf ?einem Bild? basieren. Hier hat der Benutzer die Möglichkeit, mit dem Programm zu interagieren. Dies ist für einen unerfahrenen User intuitiver und er kann durch ausprobieren ein gewünschtes Ergebnis erzielen.
Fang stellt in seinem Paper [16] ein Verfahren vor, bei dem die Transferfunktion eine Abfolge von verschieden 3D Bildverabteitungsverfahren ist, deren Parameter vom Benutzer angepasst werden können.
Es gibt auch Methoden, bei denen ein Hilfswerkzeug zum Einsatz kommt. Zum Beispiel kann der Benutzer im Paper von Reitinger: [17] 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 [12] ein intuitives Verfahren zum verändern von Features von 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. Das Zusammenführen wird somit zu einem Optimierungsproblem undzwar dem minimieren der Energiefunktion. Dieses Problem kann mithilfe eines stochastischen Suchalgorithmus gelöst werden. Der Anwender bekommt am eine Visualisierung mit allen gewünschten Features.

2.6. Clusteringbasierte Verfahren

Sereda baut seine Arbeit [18] auf den von Serlie [19] 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, aus dem Grund, dass 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 [20] 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 [21] stellt ebenfalls ein clusteringbasiertes Verfahren vor. Zunächst wird in einem Vorverarbeitungsschritt die Gradienten des Volumens berechnet. Hierzu wurde Hong's Methode [22] verwendet. Im Anschluss daran wird anhand der Gradienten die LH-Werte mithilfe von Heuns Methode, ein modifizierte Euler Methode, ermittelt.Hierbei wird des Weiteren 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 Benutzter steht dann ein zwei stufiges und ein drei stufiges 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 der Mittelpunkt zwischen zwei Iterationen nur minimal ändert. Der komplette Vorgang wird für jeden Punkt im LH-Histogramm wiederholt und in Folge werden Cluster, deren Mittelpunkt nah beieinander liegen zu einem Cluster verschmolzen.
Der zweite Clusteringschritt wird auf die Cluster des ersten Schrittes angewendet. Hierbei wird auch *Meanschiftclustering* 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, als 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 Visualisierung des Ventrikelsystems führen. Andere scheiden jedoch schon 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 [4] vorgeschlagen eignet sich schon 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 anzuzeigenden 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 zwar, sind aber wenig intuitiv und benötigen ein gewisses Fachwissen seitens des Nutzers. Darum wurde sich gegen einen Räumlichbasiertes Vorgehen entschieden.
Aus diesen Gründen, wurde sich in dieser Arbeit auf die Implementierung des clusteringbasierten Verfahren von Nguyen [21] festgelegt. Dieses eignet sich mit der Kantenerkennung gut zum Segmentieren von gezielten Strukturen.

3. Methoden

Diese Arbeit orientiert sich an dem zweistufigen Clusteringverfahren aus dem Paper von Nguyen [21]. In diesem Kapitel werden dabei die verschiedenen verwendeten Methoden und Verfahren vorgestellt.

3.1. Gradient

Zuerst müssen die Gradienten aller Voxel berechnet werden. Wie im Paper beschrieben wurde auch in dieser Arbeit Hong's Methode [22] dafür gewählt. Diese ist ein approximationsbasiertes Verfahren zur Berechnung von Gradienten eines Volumens.
In Hong's Verfahren wird zur Berechnung der Gradienten die lokale 4x4x4 Nachbarschaft des betrachteten Punktes hinzugezogen. 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 zwei Punkten berechnet werden. Deshalb liegt er im Falle eines dreidimensionalen Volumens im Zentrum eins 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 4x4x4 Nachbarschaft in Form der durchnummerierten Punkte zu sehen.

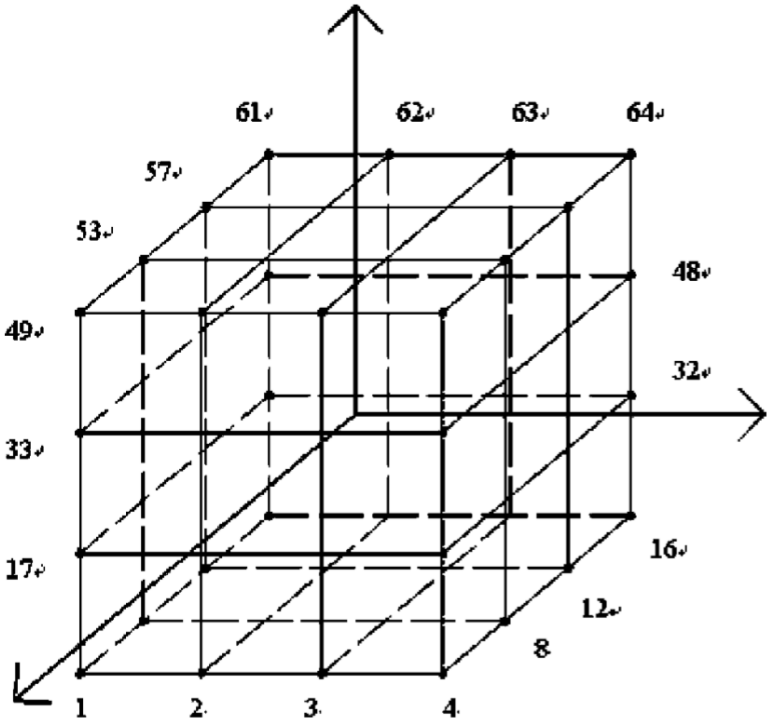

Abbildung 2: Darstellung der lokalen 4x4x4 Nachbarschaft
Entnommen aus Hong's Paper

Die Funktionen für die Intensitätswerte wird im Paper mit:

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

approximiert. Da der Gradient die Ableitung der Intensitätsfunktion ist, muss diese abgeleitet werden, um dreidimensionalen Gradientenvektor n zu erhalten:

$$n = [Ax + Gz + Hy + I, By + Fz + Hx + J, Cz + Fy + Gx + K] \tag{2}$$

Um den Gradienten zu Berechnen müssen die Parameter $A, B, C, E, F, G, H, I, J, K$ berechnet werden. Dies wird mithilfe der Methode der kleinsten Quadrate und der im Paper vorgestellten *error distance*, die die Entfernung zwischen einem berechneten Datenpunkt und seinem korrespondierenden Punkt in den tatsächlichen Daten beschreibt, umgesetzt. Wurden die Parameter berechnet kann der Gradient $[X, Y, Z]$ des Punktes $[x, y, z]$ mit folgender Formel kalkuliert werden:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 2A & 2H & 2G \\ 2H & 2B & 2F \\ 2G & 2F & 2C \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} 2I \\ 2J \\ 2K \end{bmatrix} \tag{3}$$

3.2. LH-Werte

Als nächster Schritt, nach der Kalkulation der Gradienten, folgt die Berechnung der Low- und High-Werte. Dazu wird für jeden Voxel im Volumen in Richtung des Gradienten integriert. Hierfür wurde, wie auch von Nguyen, Heun's Methode, eine modifizierte Euler Methode, verwendet. Die für die Integration benutzte Formel lautet:

$$u_{i+1} = u_i + \frac{1}{2} d(\nabla f(u_i) + \nabla f(u_i + d\nabla f(u_i))) \tag{4}$$

Hierbei sind u_i und u_{i+1} die Positionen des aktuellen, beziehungsweise des nächsten Voxels. $\nabla f(x)$ beschreibt den normalisierten Gradienten für die High-Werte und den normalisierten inversen Gradienten für die Low-Werte an Stelle 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. Dies ist daran zu erkennen, dass die Länge des Gradienten an dieser Stelle null ist. Bei MRT-Daten muss ein Grenzwert festgelegt werden, da Gradienten nie null werden. Bei CT-Daten ist dies jedoch kein Problem. Wird ein solcher Punkt erreicht, wird der Intensitätswert dieses 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 angesiedelt. Die Werte der Achsen reichen von null bis zum jeweiligen Maximum des Low- beziehungsweise High-Werte.

3.3. LH-Clustering

Als nächstes, wird der erste Clusteringschritt berechnet. Dieser findet im LH-Raum statt, genauer wird er auf dem eben berechneten LH-Histogramm angewendet. Dabei kommt *Meanshiftclustering* zum Einsatz. Der Ablauf davon geschieht wie folgt. Vor dem Clustering muss eine Bandweite und ein Thresholdwert bestimmt werden, welche die Sensitivität des Clusterings festlegen. Die Bandweite liegt im Paper von Nguyen [21] bei 7% -

9% des maximalen LH-Wertes und der Threshold bei 0,01. Anschließend kann das LH-Clustering auf jeden Punkt im Histogramm angewandt werden.
Für einen beliebigen Punkt, werden alle Punkte die innerhalb des Radius der Bandbreite liegen gespeichert. Diese Punkte bilden nun den gefundenen Cluster. Von diesem Cluster wird der neue Mittelpunkt, der jeweilige Mittelwert der beiden Koordinaten, berechnet. Um diesen Punkt wird erneut mit selben Radius alle Punkte die bisher nicht zu dem Cluster gehören gesucht und hinzugefügt. Dies geschieht solange, bis der Abstand des neu kalkulierte Mittelpunkt zum alten weniger als der Thresholdwert multipliziert mit die Bandweite ist. Nachdem dieses Prozedere für jeden Punkt im Histogramm ausgeführt wird, gibt es viele verschiedene Cluster. In einem nächsten Schritt, werden 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

Als letzten Schritt, wird erneut auf jedem eben entstandenen Cluster *Meanshiftclustering* angewendet. Dabei wird auf jedem Cluster einzeln und unabhängig von den anderen Clustern geclustert. Außerdem werden diesmal die räumlichen Informationen der Punkte in Betracht gezogen. Hierzu müssen zunächst erneut die beiden Parameter Bandweite und Threshold definiert werden. Das Clustering läuft wie zuvor im LH-Raum ab, nur wird statt im zweidimensionalem Raum mit einem zweidimensionalem Kreis im dreidimensionalem Raum des Volumens mit einer dreidimensionalen Kugel geclustert. Des Weiteren findet am Ende der Berechnung der Cluster keine Verschmelzung statt, da dies den Sinn der beiden verschiedenen Clusteringschritte zerstören würde. In den finalen Cluster haben alle Punkte jeweils ähnliche LH-Werte und liegen im Volumen nah beieinander. Würde man diese Cluster anhand ihrer räumlichen Informationen verschmelzen, haben die Punkte keine ähnlichen LH-Werte mehr und der erste Clusteringschritt wäre umsonst gewesen.

Der vorgestellte dritte hierarchische Clusteringsschritt wurde in dieser Arbeit nicht angewendet. Dies geschah aus dem Grund, dass bei diesem immer die beiden Cluster die sich am nächsten sind verschmolzen werden. Dies ist für die Aufgabe das Ventrikelsystem hervorzuheben nicht zielführend, da dieses mitten im Gehirn neben sehr vielen andern Cluster liegt. Folglich würde es relativ schnell mit andern Clustern verschmolzen werden.

4. Design

Nachdem im vorherigen Kapitel die verschieden Methoden die genutzt werden vorgestellt wurde, beschäftigt sich dieser Abschnitt mit dem Softwaredesign der Implementierung dieser Methoden.
Die Implementierung dieser Bachelorarbeit teilt sich dabei in zwei verschieden 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 *csharp* geschrieben sind, wurden auch der Code, der im Laufe dieser Bachelorarbeit entstand in *csharp geschriebn*. Als Entwicklerumgebung wurde dementsprechend *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.

Christians Paper

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* [23] zu einer einzelnen Datei im .nrrd Format umgewandelt, da nur Volumen in diesem Format 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.

Die Darstellung von Volumendaten ist über den Renderer in Unity möglich. Der Benutzer hat hierbei mehrere Möglichkeiten Eingaben über Parameterfelder oder durch Drücken von Tasten 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 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 verschiedene Parameterfeldern 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 gefunden Voxel als Wert für die Farbgebung benutzt wird. Die Einstellung bei der der maximale wert genommen wird ist hierbei der Standard. Des Weiteren kann der Nutzer einen Threshold setzen, bei dem alle Voxel die eine Intensitätswert niedriger als diesen haben 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 im Kontext dieser Bachelorarbeit nicht von Interesse sind, nicht alle vorgestellt.

evtl. raycasting

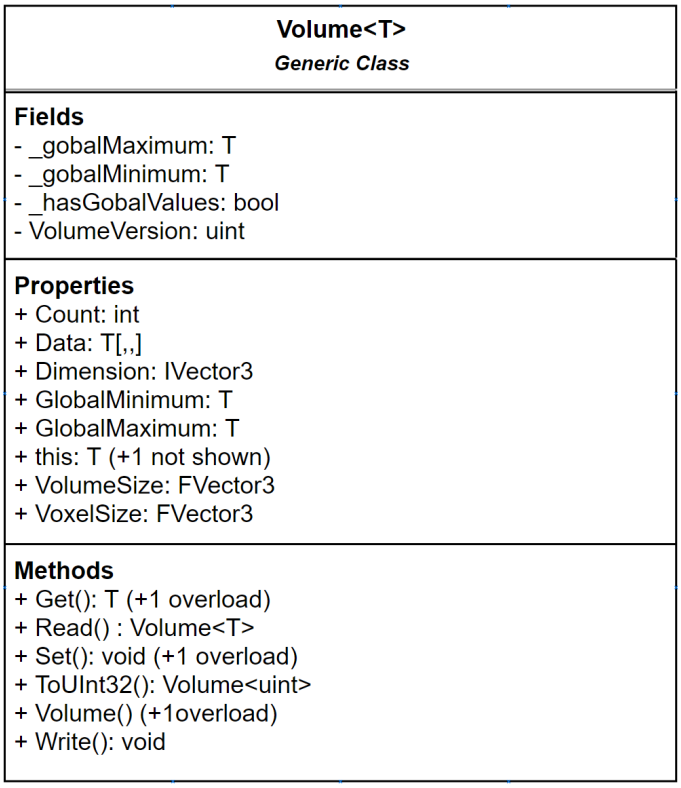

Abbildung 3: UML-Diagramm des Volumens

Die interne Speicherung des Volumens wird mit der generischen Klasse *Volume* umgesetzt, deren Felder, Attribute und Methoden in Abbildung 3 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 verschieden 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. Dies ist im UML-Diagramm in Abbildung 4 zu sehen. Jedes Modul hat dabei seine eigene Syntax die mithilfe eines Help Befehls angezeigt werden kann.

Die Funktionen der Module entsprechen deren Namen. So lädt *Load* bespielsweise eine .nrrd oder binäre Datei, *Dump* und *Write* speichern das geladene Volumen als binäre oder .nrrd Datei ab, *Info* gibt Informationen über das aktuell geladene Volumen zurück und *Resample* lässt den Anwender die Größe des Volumens verändern.
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 ebenfalls die LH-Werte des Volumens, führt jedoch hinterher noch die beiden Clusteringsschritte des Verfahrens aus. Als Ausgabe speichert das Modul eine binäre Datei eines Volumens, in welchem die verschiedenen IDs der Cluster gespeichert sind. Die Idee dahinter wird im laufe diese Kapitels erklärt. Das Modul *MergeCluster* dient der Verschmelzung der gewünschten IDs mit dem ursprünglichen Volumen. Das Ergebnis dessen, ist eine finale binäre Datei, die im Renderer das Ventrikel-

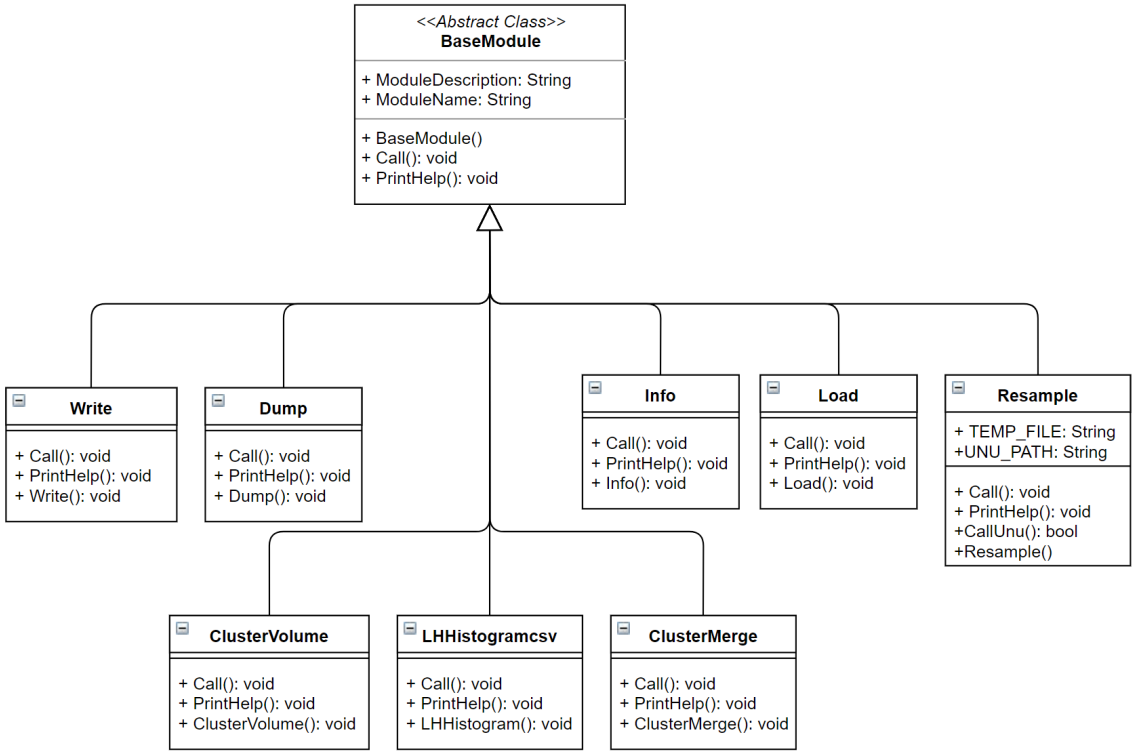

Abbildung 4: UML-Diagramm über die Module

system visualisiert. Ein Überblick über den gesamten Aufbau und Ablauf der Implementierung ist in Abbildung 5 zu sehen.

Der Ablauf der Berechnung startet in der statischen *Gradient* Klasse. Diese ist eine Implementierung des Verfahren von Hong [22]. Zur Berechnung wird der Funktion *CalcGradientVolume* das Volumen der Intensitätswerte, als Volumen aus Integern, als Parameter 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 6 ist ein zweidimensionales Beispiel zu sehen, bei dem die Gradienten berechnet werden und die Dimension der Achsen dadurch um eins kleiner wird. Die Dimension der Intensitätswerte ist ein 5x5 Gitter, in der Abbildung in schwarz zu sehen. Nachdem die Gradienten immer zwischen 4 Punkten berechnet wurde, ist zu sehen, dass die Dimension der Gradienten ein 4x4 Gitter ist. 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 vorhanden seien müssen, 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 Werte dadurch verändert werden, und Informationen verloren gehen.

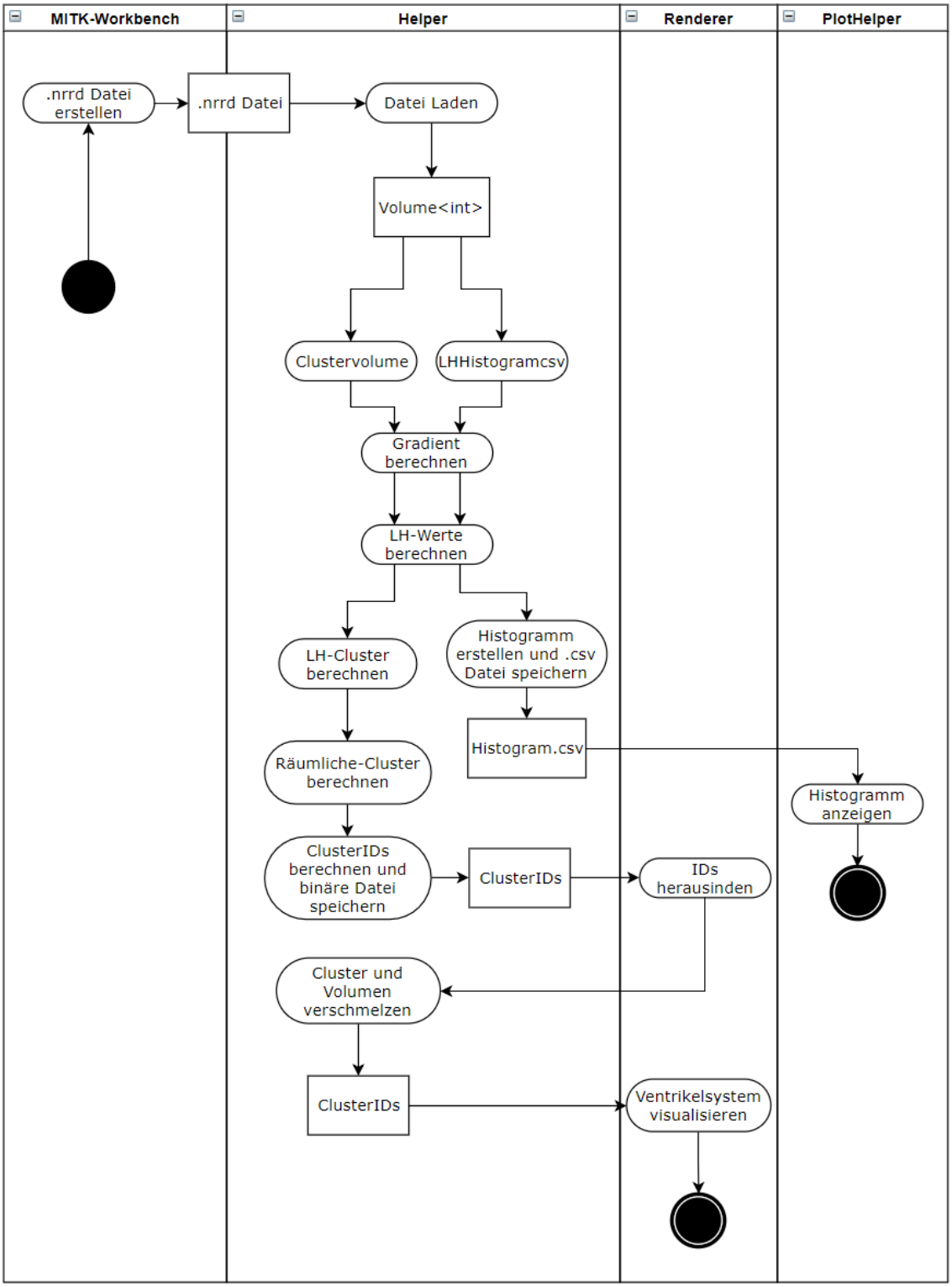

Abbildung 5: Überblick über den Ablauf

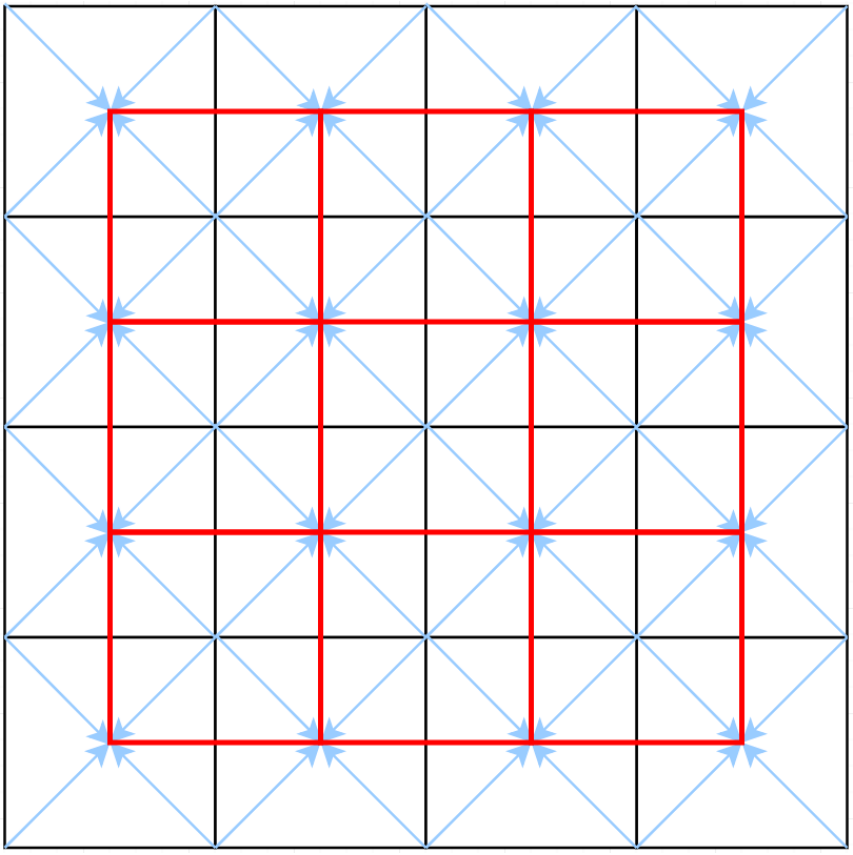

Abbildung 6: 2D Beispiel warum das Volumen kleiner wird

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 Pythonskript *PlotHelper* zum Einsatz. Dieses lädt die .csv Datei und visualisiert das LH-Histogramm mithilfe einer kalt-zu-heiß-Farbrampe in einem zweidimensionalem Koordinatensystem.
Hierbei ist zu beachten, dass das Histogramm abhängig von der Häufigkeit des Vorkommens eines LH-Wertpaares im Volumen gebildet wird. Diese werden im jeweils dazu passenden Kästchen des Koordinatensystems gespeichert. Dies ist ein simplerer Vorgehen, als das im Paper von Nguyen [21] benutzte Erstellen des Histogramms abhängig von einer für jeden Voxel berechneten Gewichtung. Da die Arbeit an der Implementierung zeitlich beschränkt war, wurde diese 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 oben 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 aus dem Grund der großen Diskrepanz zwischen der Anzahl der Einträge an den meisten Stellen im Histogramm im Vergleich zu dem Maximum. Dies lässt bei der Darstellung mit einer Farbrampe fast alles mit der Farbe des Minimums anzeigen. Da die Logarithmusfunktion für schnell wachsende Zahlen nur sehr langsam steigt, eignet sie sich dafür, diesen großen Unterschied anzupassen.

Wurde jedoch das *ClusterVolume* Modul aufgerufen, wird mit den beiden Clusteringschritten fortgefahren.
Die Berechnung die in der *LHClustering* Klasse geschieht nimmt das Volumen mit den LH-Werten entgegen und rechnet dieses aus Performancegründen, wie oben beschrieben, in ein ein Histogramm um. Dieser Schritt könnte gespart werden, wenn die Methode *LHValueVolume* direkt ein Histogramm als Rückgabewert liefern würde. Dies würde auch das *LHHistogram* Modul verbessern, da damit die Umrechnung in dieser Klasse ebenso hinfällig wird. Als Ergebnis der Clusteringfunktion *ComputeLHClusters* wird eine Liste der Cluster zurückgegeben, wobei ein Cluster aus einer Liste von *IVector3* besteht. Die Cluster werden nur als Liste der räumliche Informationen der Voxel gespeichert, da für den nächsten Clusteringschritt lediglich diese Information benötigt wird.

Anschließend geht es in der *SpatialClustering* Klasse mit der Berechnung der räumlichen Cluster weiter. Diese werden aus dem Grund, dass wieder *Meanshiftclustering* verwendet wird, mit einer ähnlichen Implementierung wie in der *LHClustering* Klasse kalkuliert. Diesmal wird jedoch nicht auf einem Histogramm, sondern direkt auf den übergebenen Listen die Berechnung durchgeführt. Folglich muss für jede Iteration eines Mittelpunktes die gesamte Liste durch iteriert werden.
Als Ergebnis wird erneut eine Liste von Listen vom Typ *IVector3* zurückgegeben.

Nachdem alle Cluster kalkuliert wurden, werden sie in einem Volumen gespeichert. Jeder Cluster bekommt dabei zunächst seine eigene ID beginnend bei eins. Das Volumen wird dann mit den verschiedenen IDs gefüllt. Dies geschieht indem für jeden Cluster an den Positionen der Punkte die jeweilige ID gespeichert wird. Alle anderen Voxel des Clustervolumen wird der Wert null zugewiesen. Dieses Volumen wird als binäre Datei gespeichert und ist das Ergebnis der 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 dargestellt werden kann.

Das Ergebnis muss anschließend vom Nutzer in Unity geladen werden.
Hierbei kommt die von dieser Arbeit vorgenommenen Anpassung am Renderer zum Einsatz. Dadurch ist es dem Benutzer möglich über ein Eingabefeld während der Visualisierung einen Wert, oder einen Wertebereich anzugeben. Dieser wird dann in einer gewählten Farbe, standardmäßig rot, hervorgehoben. Dies muss der Anwender nutzen, um die IDs, die das Ventrikelsystem beschreiben, zu finden.

Hat er dies getan kann er mit dem*MergeCluster* Modul des Helpers das Ergebnis zusammenfügen und die finale Visualisierung erhalten. Beim Aufruf muss der Nutzer das Intensitätsvolumen, das Clustervolumen mit den IDs und die ausgewählten IDs als Parameter übergeben. Anschließend wird an den Stellen der ClusterIDs die Werte im Intensitäsvolumen mit dem Wert 5000 überschrieben, da nach dem beschriebenen Werteshift ins positive der maximale Intensitätswert bei ungefähr 4500 liegt. Dies erhöht das Maximum des Volumens nur gering und lässt es zu die zu visualisierenden Bereiche klar vom Rest abzugrenzen. Dieses Volumen wird erneut als binäre Datei gespeichert.
Als letzten Schritt kann der Benutzer die finale binäre Datei in Unity laden. Stellt er den Wert der eben erklärten Erweiterung des Renderer auf 5000, wird das Ventrikelsystem in den ursprünglichen Volumendaten rot hervorgehoben.

5. Implementierung

Nachdem im letzten Kapitel das Softwaredesign besprochen wurde, beschäftigt sich das Kommende mit der Implementierung an sich.
Die vorgestellten Module erben alle von der abstrakten *BaseModule* Klasse, die es ihnen vorschreibt, eine *Call* sowie eine *PrintHelp* Methode zu implementieren.
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, da die Kalkulation nicht sehr aufwändig ist, aus.
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 für jeden die Implementierung von Hong's Methode [22] 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 wie in Abbildung 2 gezeigt, gespeichert. Bei der Kalkulation jedes Gradienten müssen lediglich die beiden Arrays durch iteriert werden um die entsprechenden Gewichtung und Koordinate des Nachbars zu erhalten

5.2. LH-Werte

In Abbildung 7 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.

```
if (!HisFinished)
{
    // the current vector is the gradient of the voxel
    FVector3 absVectorH = volumeVectors[xH, yH, zH];
    FVector3 absVectorHNormalized = absVectorH.Normalize();
    int tempNewXH = (((int)Math.Ceiling(xH + (stepSize * absVectorHNormalized.X))));
    int tempNewYH = (((int)Math.Ceiling(yH + (stepSize * absVectorHNormalized.Y))));
    int tempNewZH = (((int)Math.Ceiling(zH + (stepSize * absVectorHNormalized.Z))));


    if (tempNewXH < volumeVectors.Dimension.X && tempNewYH < volumeVectors.Dimension.Y && tempNewZH < volumeVectors.Dimension.Z
        && !(tempNewXH < 0) && !(tempNewYH < 0) && !(tempNewZH < 0))
    {
        secondAbsVectorH = volumeVectors[tempNewXH, tempNewYH, tempNewZH].Normalize();
        //add the intensity of the current voxel
        highestIntensity.Add(gradientIntensity[xH, yH, zH]);
        //check if the lengths of the current vector ist smaller than the abort criterion
        if (absVectorH.Length() < extremumLength)
        {
            HisFinished = true;
        }
        else
        {
            //stop if it is caught in a loop
            if (highestIntensity[highestIntensity.Count - 1] == highestIntensity[highestIntensity.Count - 2] && highestIntensity.Count > 3)
            {
                HisFinished = true;
            }

            // calculate the new x,y,z values
            xH = (((int)Math.Ceiling(xH + 0.5 * stepSize * (absVectorHNormalized.X + secondAbsVectorH.X))));
            yH = (((int)Math.Ceiling(yH + 0.5 * stepSize * (absVectorHNormalized.Y + secondAbsVectorH.Y))));
            zH = (((int)Math.Ceiling(zH + 0.5 * stepSize * (absVectorHNormalized.Z + secondAbsVectorH.Z))));
            // stop if the new x,y,z values are out of bounce
            if (xH > volumeVectors.Dimension.X-1 || yH > volumeVectors.Dimension.Y-1 || zH > volumeVectors.Dimension.Z-1
                || xH < 0 || yH < 0 || zH < 0)
            {
                HisFinished = true;
            }

        }
    }
    else
    {
        HisFinished = true;
    }

}
```

Abbildung 7: Implementierung der Berechnung der High-Werte

Am Anfang eines Durchlaufes wird der normalisierte Gradienten des aktuellen Punktes in *absVectorNormalized* gespeichert. Anschließend wird der Punkt des zweiten normalisierten Vektors 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 Abbildung 7 zu sehen ist, wird dabei jedoch erst getestet ob die Liste schon mehr als 3 Einträge hat. Dies geschieht aus dem Grund, dass 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, und es kein Ergebnis für den High-Wert gäbe. Diese Kontrolle könnte jedoch über die Koordinaten der iterierten Punkte geschehen, um dem 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, die im Kapitel der Methode vorgestellt wurde, ermittelt. 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 waren, also der Low- größer als der High-Wert war. 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 am Ende jeder Spalte deren Ergebniscluster soweit wie möglich verschmolzen.
Weiterhin speichert der temporäre Cluster *AktuellerCluster* lediglich die Koordinaten der zum Cluster dazugehörigen Punkte im Histogramm, jedoch nicht die räumlichen Informationen der darin gespeicherten Voxel, ab. 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 [21] über das gesamte Histogramm mit einer Bandweite von 7% des maximalen LH-Wertes für jeden Kasten berechnet. Dies ist aus mehreren Gründen schlecht. Zum einen ist der maximale LH-Wert sehr hoch, über 4000, obwohl in etwa nur 0,3% Voxel einen Wert von über 2400 haben. Zum anderen liegen über 5% 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 sehe viele Cluster gefunden und hinzugefügt werden mussten.

Eine erste Maßnahme um das Problem zu beheben, war es alle Low- und High- Werten, die über 2400 waren jeweils auf 2400 zu setzten. Dadurch wurden nur wenige Werte verändert und der Clusteringradius wurde deutlich kleiner. Dies konnte das beschriebene

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
    $AlleErgebnisCluster.Hinzufgen(AktuellerCluster)$;
    $Leeren(AktuellerCluster)$;
end
$VerschmelzeNaheCluster(AlleErgebnisCluster)$;
$return\ AlleErgebnisCluste$r;

Algorithmus 1: Pseudocode der Implementierung der LH-Cluster

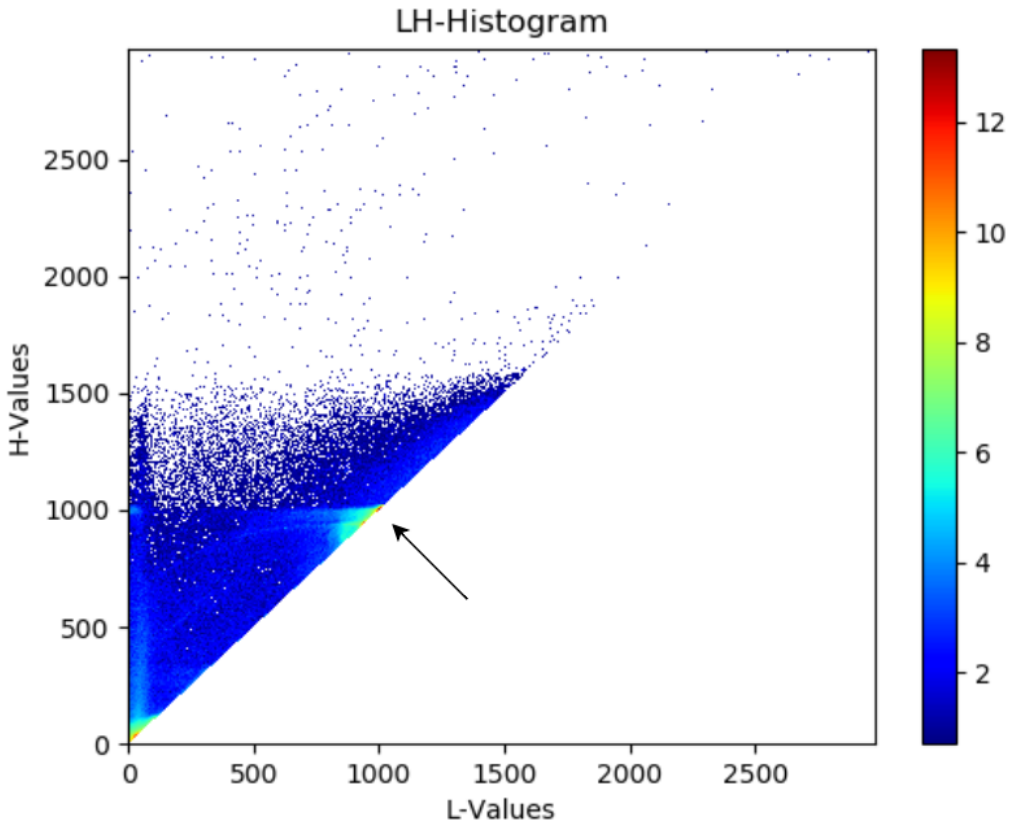

Abbildung 8: Beispiel eine LH-Histogramms

Problem jedoch nicht alleine lösen, da immer noch viele Punkte gefunden und dies für jeden Kasten im Histogramm durchgeführt werden musste.
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 so ähnlichen Cluster führen, dass diese im letzten Schritt des Clusteringsverfahrens verschmolzen wurden. Diese Änderung verbesserte die Berechnungszeit erneut, jedoch dauert die Berechnung immer noch relativ lange und es entstanden sehr viele Cluster. Diese durchzuschauen benötigte viel Zeit und das Gehirn wurde dabei meist als viele 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 hunderten Clustern mit ein bisschen Aufwand zu erkennen.
Im LH-Histogramm eines CT-Datensatzes in Abbildung 8 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 ist 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. Hierbei ist der Fakt zu sehen, dass die Intensitätswerte im Gehirn sehr nah beieinander liegen.

5.4. Räumliches-Clustering

In der statischen Klasse *SpatialClustering* ist die Implementierung zur Berechnung der räumlichen Cluster und dem Clustervolumen der IDs zu finden.

```
List<IVector3> currentSpatialCluster = new List<IVector3>();
List<List<IVector3>> resultClusters = new List<List<IVector3>>();
//List<IVector3> pickedLHCluster = LHClusters[i];
FVector3 midPoint;

HashSet<IVector3> toRemove = new HashSet<IVector3>();


while (LHClusters2[i].Count > ClusterSizeMin)
{
    midPoint = LHClusters2[i].ElementAt(0);
    bool nothingMoreFound = false;

    while (!nothingMoreFound)
    {

        foreach( IVector3 current in LHClusters2[i])
        {
            if (InDistance(midPoint, current))
            {
                currentSpatialCluster.Add(current);
                toRemove.Add(current);
            }
        }
        foreach(IVector3 remove in toRemove)
        {
            LHClusters2[i].Remove(remove);
        }
        toRemove.Clear();
        FVector3 newMidPoint = NewMidpoint(currentSpatialCluster);

        if (MinimumDistance > (SpatialDistance(newMidPoint, midPoint))) nothingMoreFound = true;
        else midPoint = newMidPoint;

    }
    // Only add clusters that are big enough
    if (currentSpatialCluster.Count > ClusterSizeMin) resultClusters.Add(new List<IVector3>(currentSpatialCluster));
    // Clear the current cluster for the next iteration
    currentSpatialCluster.Clear();
}

currentSpatialCluster.Clear();
return resultClusters;
```

Abbildung 9: Implementierung der Berechnung der Gewichte und Koordinaten

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 Abbildung 9 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 temporärem Cluster *currentSpatial-*

Cluster als auch zum HashSet der zu entfernenden Punkte *toRemove* hinzugefügt. Anschließend werden, alle Elemente von *toRemove* aus dem LH-Cluster entfernt und wie beim LH-Clustering auch ein neuer Mittelpunkt 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 für das Abbruchkriterium wurden im Paper von Nguyen [21] 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 führten zu den besten Ergebnissen.
Anschließend wird 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 wurden, wurden im Shader programmiert. Zuerst wurde eine drop-down-list hinzugefügt, über die der Nutzer zwischen den Modi *Default*, *SpecificValue* und *SpecificValueRange* wählen konnte. Weiterhin wurden drei Textfelder hinzugefügt, über die es dem Anwender möglich ist den spezifischen Wert als auch den Wertebereich anzugeben. Des Weiteren wurde noch ein Farbfeld hinzugefügt, mit welchem die Farbe, in der hervorgehobenen Bereiche dargestellt wird, anzeigt wird und vom Benutzer verändert werden kann.
Der *Default* Modus steht hierbei für die schon vorher dagewesene Implementierung der Farbgebung. Diese wurde weitestgehend in den beiden anderen beiden Modi übernommen. Der Unterschied besteht jedoch darin, dass wenn ein Voxel bei *SpeicifcValue*mit dem spezifischen Wert, oder bei *SpecificValueRange* innerhalb des Wertebereichs gefunden wird, dieser in der Farbe des Farbfeldes angezeigt wird. Alle andern Voxel bekommen die selbe Farbe, die sie auch im *Default* Modus erhalten würden.

6. Ergebnisse

Diese Kapitel beschäftigt sich mit den Ergebnissen und der Evaluation dieser Bachelorarbeit. Die Ergebnisse der Visualisierung sind in mehreren Abbildungen zu sehen. Die Evaluation teilt sich folgendermaßen auf. Als erstes werden die Ergebnisse der Visualisierung des Ventrikelsystems gezeigt und evaluiert. Dabei wurde für die Auswertung ein Interview mit einem Arzt durchgeführt. Danach wird eine Nutzerstudie vorgestellt, die die Benutzerfreundlichkeit des Systems testet. Als letztes wird die Berechnungszeit der Implementierung besprochen.

6.1. Visualisierung

Die in diesem Unterkapitel gezeigten Visualisierungen wurde alle auf Volumen mit einer Auflösung von 256x101x256 Pixeln berechnet. Aus dem Grund, dass dies ein gute Balance zwischen der Auflösung und der Berechnungszeit der Ergebnisse darstellt. Im Unterkapitel Berechnungszeiten wird dies genauer beschrieben. Bevor die Visualisierung ausgewertet werden kann, wird zunächst das Ventrikelsystem erläutert.
In Abbildung 10 ist das Ventrikelsystem zu sehen. Dieses besteht aus vier verschiedenen Ventrikeln. Zum einen der linke und der rechte Seitenventrikel, oberen beiden Bögen, die in der Abbildung zu sehen sind. Zum anderen der dritte und vierte Ventrikel, die zwischen den beiden Seitenventrikeln liegen und nach unten weggehen. Die Seitenventrikel bestehen aus einem Vorderhorn, Nr. 1, einem Hinterhorn, Nr. 2, und einem Unterhorn, Nr. 3. Der dritte Ventrikel ist mit der Nummer 4 und der vierte Ventrikel mit der Nummer 5 versehen.
Bei der Ventrikelpunktion, wird einer der beiden Seitenventrikel im vorderen Bereich punktiert. Dies macht vor allem die Darstellung der Seitenventrikel relevant.

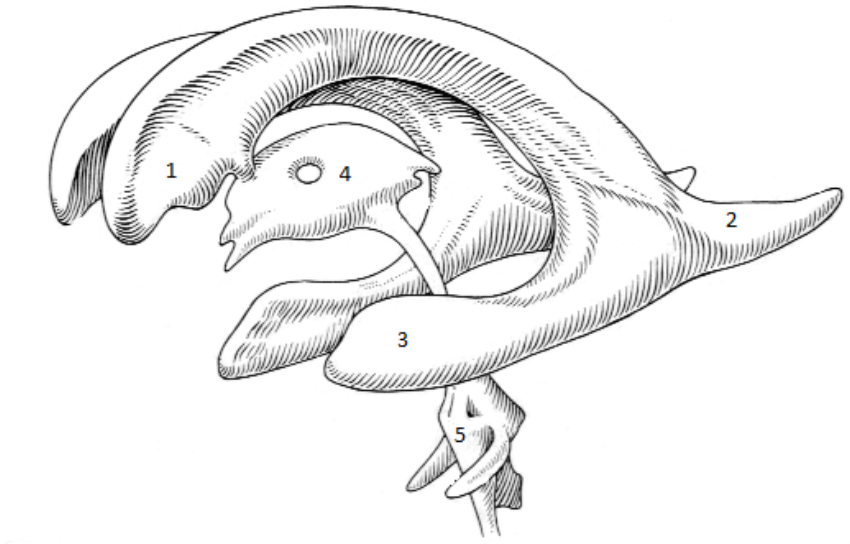

Abbildung 10: Zeichnung des Ventrikelsystems
Quelle:$http://www.kiefer-saarland.com/anatomie\_physiologie.htm$

Das Verfahren wurde an 15 verschiedenen CT-Datensätzen von der Uni Ulm getestet. Darunter waren verschiedene Ventrikelsysteme. Vier der Datensätze waren von Menschen

mit einem normalen Ventrikelsystem, vier andere wiesen ein sehr schlankes System auf. Weiterhin litten zwei Patienten unter Atrophie, einem, oft durch das Alter verursachtem, Schwund an Gehirnmasse. Zwei andere Datensätze waren von Leuten, die unter einem Mittellinienshift litten. Dies bezeichnet die Verschiebung der Mittellinie der Ventrikelsystems, was beispielsweise durch einen Schlag auf den Kopf hervorgerufen werden kann. Weiterhin gab es Daten, von einer Hirnblutung, und einem Hydrocephalus, einer Aufstauung von Nervenwasser im Kopf. Als letztes gab es noch Daten eines deformierten Ventrikelsystems, ausgelöst durch Wassereinlagerungen im Gehirn.
Eine erfolgreiche Visualisierung gelang in nur 3 der 15 Fälle und zwar bei zwei Datensätzen mit normalen Ventrikeln und einem von einem Patient, der unter Atrophie leidet. In Abbildung 11 und Abbildung 12 ist die Visualisierung des ersten normalen Ventrikelsystems, in Abbildung 13 und Abbildung 14 die Visualisierung des zweiten normalen Ventrikelsystems und schließlich in Abbildung 15 und Abbildung 16 die Visualisierung des Ventrikelsystems des Patienten mit Atrophie zu sehen. Die Abbildungen zeigen Screenshots aus der Darstellung in Unity, jeweils aus den Perspektiven von der Seite und von oben.

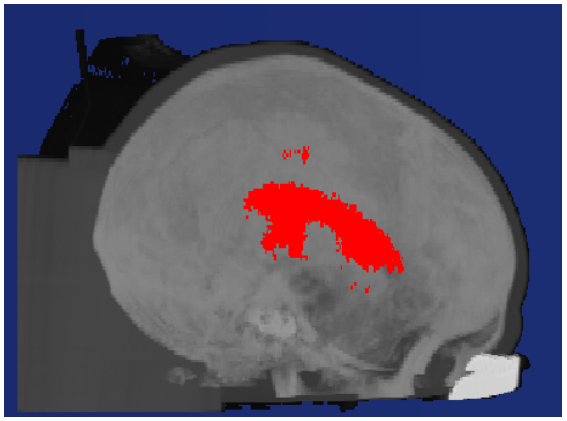

Abbildung 11: Visualisierung des ersten normalen Ventrikelsystems von der Seite

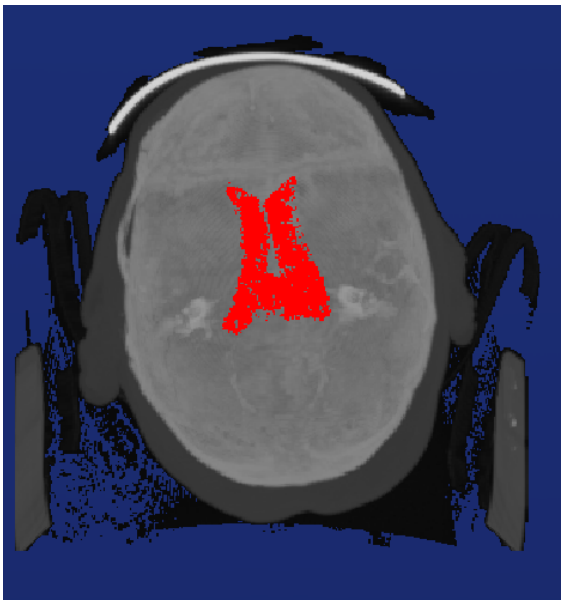

Abbildung 12: Visualisierung des ersten normalen Ventrikelsystems von Unten

In Abbildung 11, der Darstellung des ersten normalen Ventrikelsystems von der Seite, sind links über dem Hinterhorn der Seitenventrikel Punkte zu erkennen, die nicht zum Ventrikelsystem gehören. Auch bei den Visualisierungen der andern Ventrikelsysteme sind an mehreren Stellen solche Ausreißer zu erkennen. Diese können bei dem aktuellen Stand der Implementierung nicht entfernt werden und senken die Qualität der Darstellung.

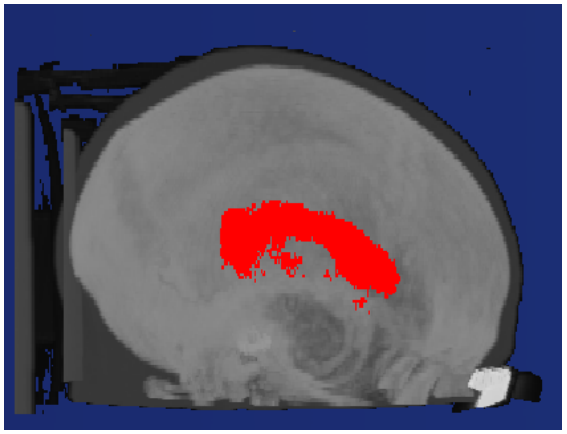

Abbildung 13: Visualisierung des zweiten normalen Ventrikelsystems von der Seite

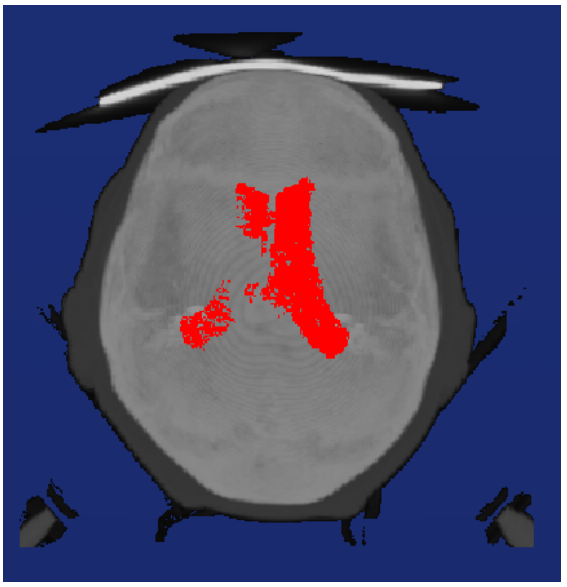

Abbildung 14: Visualisierung des zweiten normalen Ventrikelsystems von Unten

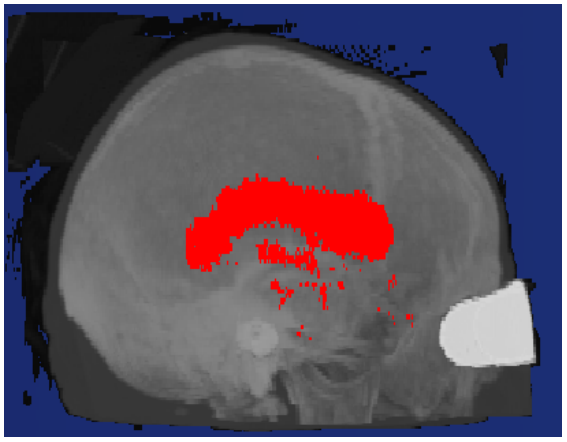

Abbildung 15: Visualisierung des Atrophie Ventrikelsystems von der Seite

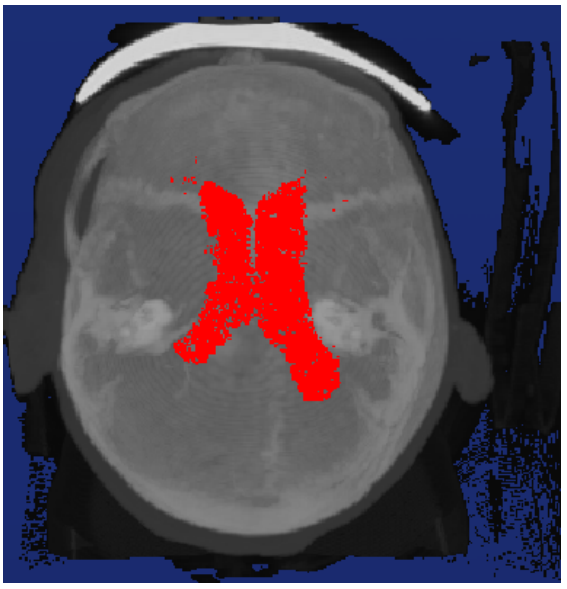

Abbildung 16: Visualisierung des Atrophie Ventrikelsystems von Unten

Für die drei erfolgreich visualisierten Daten wurde zur Evaluation ein Arzt von der Uniklinik Ulm interviewt. Diesem wurden die 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 - 5 sehr gut

- 2) Wird das Ventrikelsystem in der Visualisierung vollständig dargestellt?
  überhaupt nicht vollständig 1 - 5 vollständig
- 3) Wie genau ist das Ventrikelsystem segmentiert?
  überhaupt nicht segmentiert 1 - 5 ausschließlich das Ventrikelsystem ist segmentiert

Die Ergebnisse der Antworten des Arztes werden in Tabelle 1 gezeigt. Allgemein merkte der Mediziner jedoch 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 Unterhörner der Seitenventrikel fehlten bei den Darstellungen komplett. Er erklärte jedoch, dass diese bei einem gesunden Menschen jedoch sehr dünn und deshalb anhand von CT-Daten schwer zu segmentieren seien. Weiterhin sind, wie schon erwähnt, die Seitenventrikel entscheidend für eine erfolgreiche Punktion. In folge dessen, wurde sich darauf geeinigt, dass die Beantwortung der zweiten Frage, nach der Vollständigkeit des Ventrikelsystems, lediglich auf die Vollständigkeit der beiden Seitenventrikel bezogen ist.

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

Tabelle 1: Ergebnisse des Interviews mit einem Arzt

Die Bewertung des ersten normalen Ventrikelsystems fiel positiv mit 11 von möglichen 15 Punkten aus. Das Ventrikelsystem war als solches klar und deutlich zu erkennen, jedoch fehlte ein Teil der Hinterhörner. Bis auf diese waren die Seitenventrikel jedoch vollständig zu sehen. Weiterhin 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 so hoch aus, da der eine Seitenventrikel sehr klar, deutlich und vollständig zu sehen war. Der Arzt sagte, dass dieser besser und glatter als die des ersten normalen Ventrikelsystems 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, da sich dieses, wie alle Ventrikel bei einer Atrophie, weitet. Trotz der Flüssigkeiten im Gehirn war das Ventrikelsystem in der Darstellung deutlich zu erkennen.
Im Allgemeinen sagte der Mediziner, dass das Ventrikelsystem bei allen Visualisierungen als solches eindeutig zu erkennen sei. Er bemängelte jedoch, dass die Darstellungen nicht glatt genug sei und zu viele Ausreißer besitzen.

Zur weiteren Auswertung der Ergebnisse wurde das erste normale Ventrikelsystem aus Abbildung 11 und Abbildung 12 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. Die Ergebnisse sind in Abbildung 17 und Abbildung 18 zu sehen. Dieses Vorgehen entspricht einer simplen eindimensionalen Transferfunktion, die abhängig von den Intensitätswerten Voxel einfärbt.
Das Ventrikelsystem ist zwar zu sehen, es gibt jedoch sehr viele Ausreißer, die es fast unmöglich machen die Ventrikel zu erkennen. Diese simple Vorgehensweise führt zu keinem gewünschtem Ergebnis.
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 werden keine Informationen genannt. Anschließend können die Löcher der Auswahl mit einem *closing* Filter geschlossen und mit einem weiteren Tool ein geglättete 3D Ansicht des Ergebnis erzeugt werden. Screenshots des Ergebnisses sind in Abbildung 19 und Abbildung 20 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.

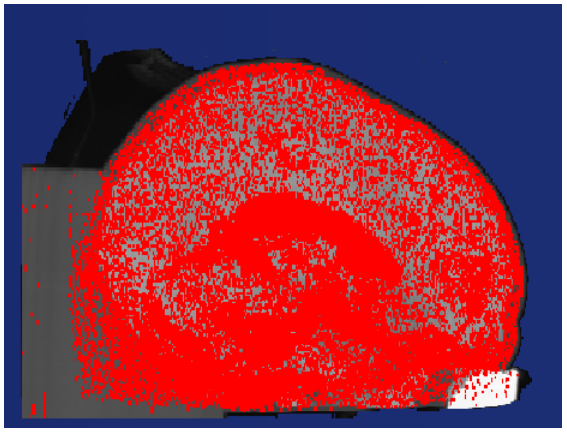

Abbildung 17: Visualisierung des ersten normalen Ventrikelsystems von der Seite mithilfe von Unity

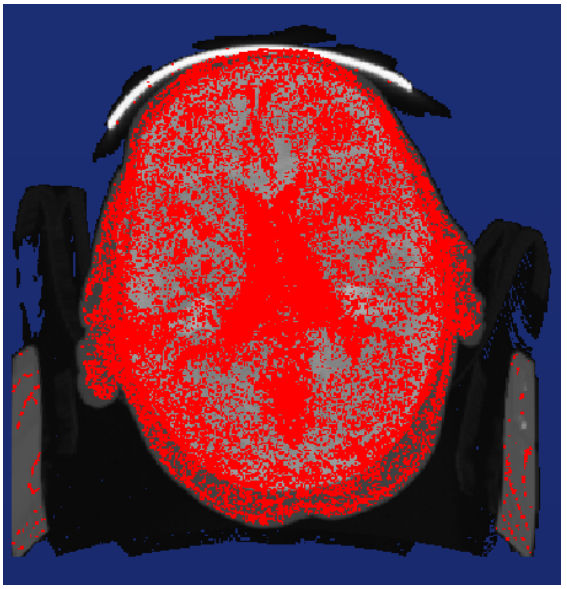

Abbildung 18: Visualisierung des zweiten normalen Ventrikelsystems von Unten mithilfe von Unity

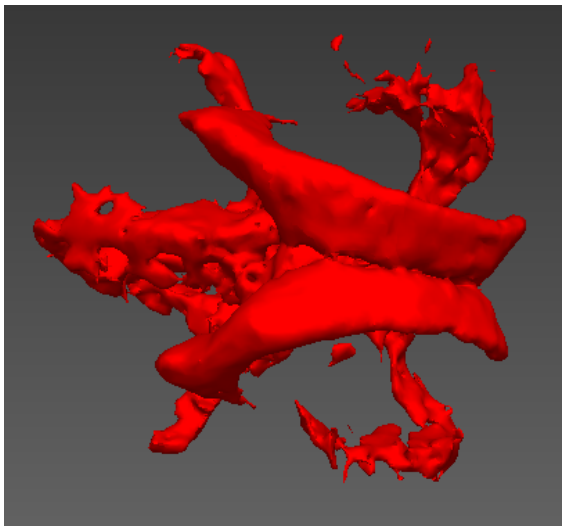

Abbildung 19: Visualisierung des ersten normalen Ventrikelsystems von Oben mithilfe von MITK

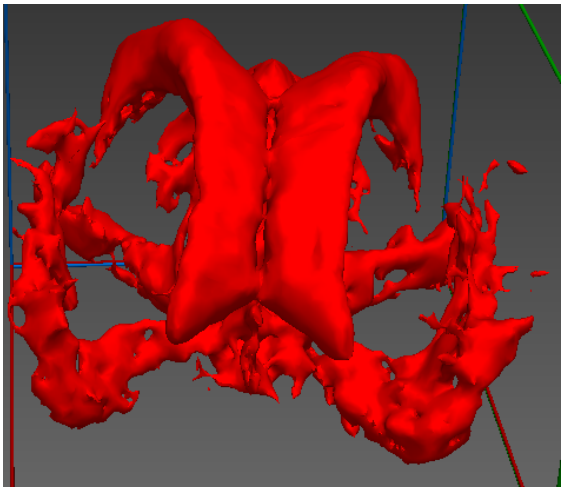

Abbildung 20: Visualisierung des ersten normalen Ventrikelsystems von Vorne mithilfe von MITK

6.2. Nutzerstudie

Im Rahmen der Evaluation der Benutzerfreundlichkeit des Verfahrens wurde eine kleine Nutzerstudie mit ... Teilnehmern durchgeführt. Bei dieser wurden den Probanden zunächst der Ablauf und die vom Benutzer erforderlichen Schritte um eine Visualisierung des Ventrikelsystems zu erhalten durch eine Vorführung durch den Interviewer gezeigt. Anschließend mussten die Teilnehmer selbst das eben gelernte anwenden und das Programm selber ausführen. Dabei bekamen sie, wenn sie nicht weiterwussten, Hilfe vom Versuchsleiter. Als Abschluss füllten die Probanden einen NASA-TLX Bogen zu der Aufgabe aus. Die durchschnittlichen Ergebnisse der einzelnen Kategorien wird in Tabelle 2 gezeigt.

Kategorie	Gewichtung	Klicks	Wichtung
Geistige Anforderung	0	0	0
Körperliche Anforderung	0	0	0
Zeitliche Anforderung	0	0	0
Leistung	0	0	0
Anstrengung	0	0	0
Frustration	0	0	0

Tabelle 2: Durchschnittlichen Ergebnisse des NASA-TLX Bogens

Der durchschnittliche Wert für die Gesamtbeanspruchung lag bei ... Personen ohne Programmiererfahrung ...

Trotz der Schwierigkeiten, gaben die Probanden an, dass sie die Aufgabe mit einer guten ausführlichen Dokumentation auch alleine ohne Hilfe hinbekommen würden.

6.3. Berechnungszeit

Die folgenden Zeitmessungen wurde 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 mit einer Auflösung von 512x201x512 Pixeln mithilfe des Resamplemoduls des Helpers verkleinert. Die beiden anderen Volumengrößen haben dabei die Hälfte, mit 256x101x256 Pixeln, und Dreiviertel, mit 384x151x384 Pixeln, der Auflösungen des Originalvolumens. Es war geplant, dass noch ein viertes Volumen zum Vergleich hinzugezogen wird.Jedoch war es aus einem unbekannten Fehler leider nicht möglich die beiden Berechnungen mit dem gevierteltem Originalvolumen durchzuführen. Des Weiteren funktioniert für beim Originalvolumen lediglich die Berechnung des LH-Histogramms. Die Kalkulation des Clusteringverfahrens war nicht möglich, vermutlich aus dem Grund, dass bei dieser hohen Auflösung es zu viele Daten für die aktuelle Implementierung zu berechnen gibt.
Die Berechnungszeit hängt stark von der Größe des Eingabevolumens ab. Die ist in Tabelle 3 sehr gut zu erkennen. Diese zeigt einen Überblick über die ungefähren 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 3: 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, kommt die Zeit, die die beiden Clusteringschritte benötigen heraus.
Eine interessante Beobachtung hierbei ist, dass die Berechnung der LH-Histogramme abhängig von der Anzahl der Pixel gesehen in etwa gleich schnell abläuft. Das halbe Volumen hat eine Gesamtpixelzahl von ungefähr 6,6 Millionen, das dreiviertel Volumen von zirka 22,2 Millionen und das Original von grob 52,6 Millionen Pixeln. Wird die Anzahl an Pixeln die pro Sekunde bei der LH-Wert Berechnung bearbeitet werden für diese drei Volumen berechnet, so ist zu beobachten, dass keine großen Unterschied zwischen den Zeiten existiert. Das Halbe bearbeitet etwa 220 Tausend, das Dreiviertel ungefähr 247 Tausend und das Ganze 234 Tausend Pixel pro Sekunde. Der kleine Unterschied in der Rate lässt sich einerseits durch Volumen 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 einen 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 Visualisierung des Ventrikelsystem basierend auf Volumendaten. Dieses Ziel wurde erfüllt, jedoch mit deutlichen Abstrichen.

Zunächst gilt es die Robustheit der Implementierung zu betrachten. Es konnte nur in 3 von 15 Fällen das Ventrikelsystem erfolgreich Visualisiert werden. Zwar waren diese Daten meist von einer besonderen Art, jedoch konnte auch bei normalen Ventrikeln nur in 50% der Fälle ein zufriedenstellendes Ergebnis erzielt werden. Weiterhin war es nur für bestimmte Auflösungen überhaupt möglich die Berechnung der Cluster durchzuführen.
Die Robustheit des Systems ist folglich verbesserungswürdig. Es funktioniert zwar, jedoch in nur wenigen Fällen.

Als nächstes wird die Qualität der erzeugten Ergebnisse betrachtet.
In dem Fall, dass die Visualisierung geglückt ist, waren die Ergebnisse von einer guten Qualität. Das Ventrikelsystem war deutlich zu sehen und als solches auch klar zu identifizieren. Es fehlten zwar bei allen Visualisierungen der dritte und vierte Ventrikel, jedoch sind diese für die Ventrikelpunktion nicht von Nöten. Dem Arzt dem das Hervorheben des Systems im Operationssaal assistieren soll, sollten womöglich nur die relevanten Informationen angezeigt werden, damit diese stets übersichtlich bleiben und ihn nicht im schlimmsten Fall zusätzlich verwirren.
Jedoch fehlten auch bei den Seitenventrikel teils kleine, teils große Stücke in der Visualisierung. Des Weiteren gab es viele Ausreißer, die vom interviewten Arzt als störend empfunden wurden. Weiterhin sind die Ergebnisse nicht glatt, sondern haben in der hervorgehobenen Struktur kleine Löcher und Unreinheiten.
Für die Zeit die für die Implementierung zu Verfügung stand, sind die Ergebnisse schon solide, benötigen jedoch an mehreren Stellen einer Verbesserung.

Im folgenden Abschnitt wird die Benutzerfreundlichkeit des Verfahrens diskutiert.
Die Schritte, die vom Benutzer ausgeführt werden müssen, um zu einem Ergebnis zu gelangen, sind wenig intuitiv. Er benötigt Wissen darüber welche Befehle wann im Helper ausgeführt werden müssen und deren Syntax. Weiterhin muss er selbst Dateien in speziellen Formaten speichern und laden. Außerdem muss der Anwender die passenden IDs manuell finden, was zeitaufwändig ist.
Im Bezug auf die Nutzerfreundlichkeit benötigt das Verfahren viel Überarbeitung. Das es bisher schwer zu bedienen ist, ist dem Fakt geschuldet, dass das Funktionieren der Methode oberste Priorität bei dem Programmierung hatte. Aus der schon mehrfach erwähnten Zeitknappheit wurde die Benutzerfreundlichkeit zunächst vernachlässigt.

Schließlich wird noch die Verarbeitungsgeschwindigkeit des Verfahren betrachtet.
Diese ist bei 50 Sekunden zur Berechnung der IDs schnell. Die benötigte Zeit wächst jedoch nicht linear mit der Größe der Eingabe. Dies liegt jedoch an der Berechnung der Cluster und dabei ist unklar, ob dies an der Implementierung oder an dem Verfahren an sich liegt, da über die Laufzeit der Meanshiftclustering keine Informationen vorliegen. (Da das Meanshiftclustering für jeden Punkt ausgeführt wird, und im worst-case bei jeder Iteration jedes Punktes ein Punkt gefunden wird liegt die Laufzeit bei $O(n^2)$)

Zusammenfassend lässt sich sagen, dass die Bachelorarbeit zu einem positiven Ergebnis gelangt ist. Zwar gibt es noch viele Stellen, an denen Verbesserungen vorgenommen

werden können. Aber es war möglich die im Kontext des HoloMed Projektes wichtigen Bereiche des Ventrikelsystems zu segmentieren und hervorzuheben.
Ein Ausblick wie die verschiedenen genannten Probleme behoben und das Verfahren an vielen Stellen verbessert werden kann wird im nächsten Kapitel gegeben.

8. Ausblick

Die Implementierung eines Clusteringbasierten Verfahren kam in dieser Arbeit zu einem Erfolg, jedoch gibt es noch sehr viele Verbesserungs- und Erweiterungsmöglichkeiten. Diese werden im folgenden Kapitel vorgestellt.
Die Visualisierung des Ventrikelsystems funktionierte bei nur wenigen Datensätzen. Bei den Restlichen wurde das System aus verschiedenen Gründen nicht erkannt. Es gab Probleme, wenn einerseits das Ventrikelsystem zu klein oder verformt war, oder andererseits die Hirnmasse im Kopf und um den Ventrikel herum andere Intensitätswerte als üblich hatten. Bei normalen Ventrikel funktionierte es teilweise auch nicht. Diese hohe Inkonsistenz gilt es zu beheben. Dazu können verschiedene Veränderungen vorgenommen werden. Da das Verfahren oft daran scheiterte, die Kanten von Ventrikeln zu erkennen, da sie zu klein oder der Unterschied zur Umgebung zu gering ist, wäre es eine Möglichkeit, einen noch kleinere Bandweite und damit ein noch genaueres Clustern über dem LH-Raum zu implementieren. Dadurch würden jedoch noch mehr Cluster entstehen, was die Benutzung noch schwieriger macht. Dies könnte jedoch behoben werden, indem einerseits Cluster die am Rande des Volumens liegen weggelassen werden. Da das Ventrikelsystem in der Mitte des Kopfes liegt, ist es klar, dass diese Cluster keine Teile eines Ventrikels enthalten können. Andererseits könnte der LH-Wertebereich über dem geclustert wird noch kleiner gemacht werden. Wie in Abbildung 17 und Abbildung 18 zu sehen ist, werden die Seiten im Intensitätsbereich von 1025 bis 1030 bereits erkannt. Allgemein würde ein weiteres Anpassen der Parameter auf das Ventrikelsystem zu besseren Ergebnissen führen.

Ein weiteres Problem war die Benutzerfreundlichkeit. Ein Anwender muss einen genauen Ablauf befolgen um zu einem Ergebnis zu kommen, indem er erst mithilfe der Konsole Daten laden, falls sie noch im .nrrd Format vorliegen formatieren, die Cluster erstellen lassen, diese in Unity laden, die passenden Cluster manuell herausfinden, und schließlich mit einem weiteren Befehl über die Kommandozeile zu einem Ergebnis verschmelzen lassen. Dieser Ablauf ist wenig Intuitiv und kann wie die Nutzerstudie aus Kapitel 6 gezeigt hat für Menschen ohne Programmiererfahrung zu einer Hürde werden.
Dies könnte an vielen Stellen verbessert werden. Beispielsweise können die Befehle der Konsole vereinfacht werden, sodass der Benutzer das notwendige u sowie das Suffix .bin.txt nicht mehr angeben muss, da diese vom Programm schon hinzugefügt werden. Eine andere Idee wäre es, die zwei Programme zu einem zu verschmelzen, sodass der Anwender direkt in Unity das Volumen laden und die IDs berechnen lassen könnte. Des Weiteren wäre es dann möglich, dass die Cluster direkt in Unity angezeigt, ausgewählt und verschmolzen werden könnten. Dadurch bestünde auch die Möglichkeit leicht zwischen der Cluster und der finalen Ansicht hin und her zu wechseln. Der Anwender muss sich dadurch nicht mehr mit Dateipfaden dem Speichern und Einlesen von Dateien im richtigen Format beschäftigen. Weiterhin könnte Nutzung durch das Erstellen einer GUI intuitiver gemacht werden, damit keine Eingabe von Befehlen in eine Kommandozeile mehr nötig ist.

Ein Problem, dass auch vom Arzt erkannt wurde, ist die nicht glatte Darstellung im Ergebnis, sowie die vielen existierenden Ausreißer. Dies macht die Visualisierung teilweise sehr ungenau.
Dies lies sich lösen, indem auf das Endergebnis weitere Algorithmen zur Verbesserung dieser angewendet werden. Beispielsweise könnten mit Dilatations- und Erosionsfiltern die Oberfläche geschlossen und kleine Ausreißer eliminiert werden. Eine weitere Verbesserung hierbei wäre, einen Regiongrowingalgorithmus am Ende des Verfahrens anzuwen-

den. Dieser könnte das Ergebnis deutlich verbessern, da dadurch ein vollständigeres Bild des Ventrikelsystems entstehen und eventuell sogar weitere Ventrikel, wie zum Beispiel das Dritte und Vierte, erkannt werden könnten.

Weiterhin könnte an vermutlich mehreren Stellen des aktuellen Codes kleine Optimierungen vorgenommen werden. Beispielsweise die Verbesserung der Suche nach Schleifen beim Berechnen der High-Werte, die im Kapitel Implementierung beschrieben wurde. Oder es könnte beim räumlichen Clustern ein Verschmelzen der im gleichen LH-Cluster gefundenen räumlichen Cluster hinzugefügt werden. Dies sind nur zwei Beispiele und es gibt vermutlich mehrere Stellen an denen diese kleinen Verbesserungen vorgenommen werden könnten.
Diese Änderungen hätten möglicherweise nur geringe Auswirkungen, sie könnten jedoch nützlich sein, um ein optimales Ergebnis zu erreichen.

Ein weiterer Punkt für Verbesserungen ist das Beheben der Bugs. Der Grund für das scheitern der Berechnung des Verfahrens auf dem ganzen, sowie auf den viertel Volumen muss gefunden und behoben werden.

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

Literatur

[1] DREBIN, Robert A. ; CARPENTER, Loren ; HANRAHAN, Pat: Volume rendering. In: *ACM Siggraph Computer Graphics* Bd. 22 ACM, 1988, S. 65–74

[2] LEVOY, M: *Display of surfaces from volume data: IEEE Comput. Graph. & Appl. Vol 8 No 3 (1988) pp 29–37.* 1988

[3] 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

[4] 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

[5] WESARG, Stefan ; KIRSCHNER, Matthias: Structure size enhanced histogram. In: *Bildverarbeitung für die Medizin 2009*. Springer, 2009, S. 16–20

[6] 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

[7] HUANG, Runzhen ; MA, Kwan-Liu: RGVis: Region growing based techniques for volume visualization. In: *null* IEEE, 2003, S. 355

[8] 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

[9] 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

[10] 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

[11] CORREA, Carlos D. ; MA, Kwan-Liu: Visibility-driven transfer functions. In: *Visualization Symposium, 2009. PacificVis' 09. IEEE Pacific* IEEE, 2009, S. 177–184

[12] 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

[13] 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

[14] 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

[15] 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

[16] FANG, Shiaofen ; BIDDLECOME, Tom ; TUCERYAN, Mihran: Image-based transfer function design for data exploration in volume visualization. In: *Visualization'98. Proceedings* IEEE, 1998, S. 319–326

[17] REITINGER, Bernhard ; ZACH, Christopher ; BORNIK, Alexander ; BEICHEL, Reinhard: User-centric transfer function specification in augmented reality. (2004)

[18] 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

[19] 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

[20] 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

[21] 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

[22] 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

[23] *MITK*. `http://mitk.org/wiki/The_Medical_Imaging_Interaction_Toolkit_(MITK)`

Anhang

A. First Appendix Section

ein Bild

Abbildung A.1: A figure

...