Isoflächenrekonstruktion aus Serienschnitten

Von der Universität Bayreuth
zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

von

Christine Ulrich

aus Lich

1. Gutachter: Prof. Dr. Michael Guthe
2. Gutachter: Prof. Dr. Eduard Gröller

Tag der Einreichung: 13. Dezember 2018
Tag des Kolloquiums: 17. Juli 2019

Abstrakt

This dissertation presents a pipeline for reconstructing 3d-meshes from serial sections. A process that should gives everyone involved a quick result. In immunohistological research the structure of interests are certain types of cells (e.g. cells of vessels) and for an examination the sections have to get an immunohistological staining. For the staining a tissue specimen will get cut in several very thin sections. A procedure that causes many distortions in each section. This means that once digitized, the immunohistologically stained sections must be aligned before a 3d-mesh can be reconstructed. The amount of digital data is large, because the techniques and the resolution of capture-systems has developed further through the years. Furthermore the structure of interests are very small (just several micrometers). This caused our team to contrive a new method of alignment. This uses Bi-Cubic B-Splines to minimize the distortion globally. Both, reconstruction and alignment methods, are accelerated by parallelizing for the handle of huge amount of data. For example, the marching cubes algorithm has been parallelized on the smallest level with CUDA. Although an adjustment of parameters (e.g. isovalue) is often needed in several passes (within the pipeline), the efficient run-time allows the medics to verify the 3d-model in an acceptable time and has already helped to understand the network of vessels within the spleen.

Diese Dissertation präsentiert eine Pipeline zur Rekonstruktion von 3D-Meshes aus Serienschnitten. Ein Prozess, der allen Beteiligten ein schnelles Ergebnis ermöglichen soll. In der immunhistologischen Forschung sind die Strukturen von Interesse bestimmte Arten von Zellen (z. B. Gefäße) und für eine Untersuchung müssen die Serienschnitte eine immunhistologische Färbung erhalten. Daher wird für die Färbung eine Gewebeprobe in mehrere sehr dünne Schnitte geschnitten. Eine Prozedur, die viele Verzerrungen in jedem Schnitt verursacht. Dies bedeutet, einmal digitalisiert müssen die immunhistologisch eingefärbten Schnitte eine Ausrichtung erfahren bevor ein 3D-Mesh rekonstruiert werden kann. Die Menge an digitalen Daten ist groß, da sich die Techniken und die Auflösung von Aufnahmesystemen über die Jahre hinweg weiterentwickelt haben. Weiterhin sind die Strukturen von Interesse sehr klein (nur einige Mikrometer). Dies veranlasste unser Team eine neue Ausrichtungsmethode zu entwickeln. Diese verwendet Bi-Cubic-B-Splines, um die Verzerrungen global zu minimieren. Sowohl die Rekonstruktions- als auch die Ausrichtungsmethode werden beschleunigt, indem eine große Menge von Daten parallel behandelt wird. So wurde zum Beispiel der Marching Cubes-Algorithmus auf kleinster Ebene mittels CUDA parallelisiert. Obwohl eine Anpassung der Parameter (z. B. Isowert) oft in mehreren Durchläufen (innerhalb der Pipeline) erforderlich ist, ermöglicht die effiziente Laufzeit den Medizinern, das 3D-Modell in einer akzeptablen Zeit zu verifizieren, und hat bereits dazu beigetragen, das Gefäßnetzwerk in der Milz zu verstehen.

Inhaltsverzeichnis

Danksagung

Zuallererst möchte ich gerne meinem Doktorvater Prof. Dr. Michael Guthe danken. Ohne seine vielseitige Unterstützung und seine Geduld wäre dies alles nicht möglich gewesen. Er war der ruhende Pol in abwechslungsreichen und aufregenden Zeiten.

Ebenso gilt mein Dank Herrn Dr. Oleg Lobachev. Seine Nerd-Power war bei der Registrierung eine große Hilfe.

Frau Prof. Dr. Steiniger danke ich für ihre ansteckende Begeisterung für die Thematik Immunhistologie, sowie für die Bereitstellung der nötigen Daten. Bei Letzteren geht natürlich mein Dank auch an die Mitarbeiterinnen Ihrer Arbeitseinheit.

Vielen Dank auch an alle Co-Autoren, die bei den eigenen Publikationen mitgewirkt haben.

Ebenso geht ein Dankeschön an die vielen kleinen unsichtbaren Helferlein außerhalb der Arbeit, die mich unterstützt haben. Sei es durch Passwort-Rücksetzung bei der Bayreuther Cloud, Bereitstellung eines kostenlosen Druckers oder (wie bei den netten Damen der Marburger Mathematik-Bibliothek) das Auffinden von Artikeln.

Als Letztes möchte ich besonders meiner Mutter Karin Ulrich danken, die immer an mich geglaubt und mich während der ganzen Zeit unbeirrt unterstützt hat.

1 Einleitung

In den letzten Jahrzehnten erhielt die Informationstechnik (IT) immer mehr Einzug in die Medizin. Alltäglich geworden sind die 3D-Darstellungen von medizinischen Daten. Am bekanntesten ist wohl die Computertomographie (CT), welche aus mehreren Röntgenschnittbildern 3D-Rekonstruktionen erstellt.

Es verwundert nicht, dass hier die ersten Entwicklungen im Bereich der Darstellung von Knochen und Zähnen stattfanden, da diese gut sichtbar und damit leicht zu segmentieren sind. Die Auflösung und die Größe der Struktur, die von Interesse ist, haben hier ein akzeptables Verhältnis. Ein Unterarmbruch ist beispielsweise auf dem CT gut zu diagnostizieren; der Kariesbefall eines Zahnes gut zu erkennen. Bei Diagnosen, die dagegen das Weichgewebe betreffen, wird es schwieriger.[1] Die anatomischen Strukturen lassen sich hier nicht mehr leicht von einander abgrenzen. Möchte man den Blutdurchfluss bei einem Thrombose-Patienten untersuchen, wird sogar auf die Magnetresonanztomographie (MRT) zurückgegriffen und die Adern werden mit Hilfe von Kontrastmitteln gegenüber anderen Strukturen hervorgehoben. Wenn nun aber die Struktur von Interesse deutlich kleiner ist und sich nicht mehr auf makroskopischer, sondern sich auf mesoskopischer oder mikroskopischer Ebene befindet und somit unter $500\mu m$ klein sein kann[2] [3] und diese Struktur zusätzlich auch noch immunhistologisch eingefärbt werden soll, da ihre Funktion oder die vorhandenen Zelltypen ebenfalls von Interesse sind, sind Nano-MRT und Nano-CT nur bedingt hilfreich. Die Auflösung der Geräte ist zwar beachtlich, so ist das Nano-CT [Rul] von Franz Pfeiffer (Technische Universität München) in der Lage $100nm$ feines Kanal-Netzwerk innerhalb einer $25\mu m$ großen Knochenprobe darzustellen und das Nano-MRT [Sch] kann Objekte unter $5nm$ detektieren, sie ersetzten aber nicht die immunhistologische Einfärbung, weil diese im Gegensatz zum Nano-CT oder -MRT in der Lage ist unterschiedliche Zelltypen zu differenzieren und so Gewebeeigenschaften nachzuweisen.

Die Immunhistologie scheint auf den ersten Blick nur mit 2D-Mikroskopschnitten zu arbeiten. Tatsächlich aber haben dort 3D-Modelle bzw. 3D-Rekonstruktionen durchaus ihre Verwendung. Wenn Immunhistologen sich beispielsweise mit Gewebeproben von Milz oder gar Knochenmark befassen, haben sie es unter anderem mit einem Netzwerk kleinster Gefäße zu tun. Die einzelnen Gefäße sind aber nicht immer komplett innerhalb eines Schnittes vorhanden und ziehen sich oftmals durch mehrere Schnitte einer eingefärbten Serie. Ein derartiger Befund wird während der Betrachtung und Analyse von den Immunhistologen im Kopf zu einem dreidimensionalen Gebilde zusammengesetzt, wobei sich diese oft nur auf einen kleinen Teil des Befundes (beispielsweise auf ein einzelnes Gefäß innerhalb eines Gefäßnetzwerkes) konzentrieren. Geht es um die Präsentation der festgestellten Ergebnisse, wird im Anschluss häufig auf Zeichnungen oder exemplarische Schnitte zurückgegriffen.

[1] Das CT arbeitet mit Röntgenstrahlen. Die Abschwächung, die die Strahlung beim Durchdringen des Körpers erfährt, wird mittels Hounsfield-Einheiten (HE) oder auch Hounsfield-Units (HU) (vgl. [Han09] Kapitel 2.1.3.2 Seite 13) beschrieben. Wasser hat dabei einen Wert von 0 HU und Knochen weisen zwischen ca. $500 - 1500$ HU auf (je nach Literatur). Weichgewebe hat einen HU-Wert von ca. $20 - 90$, wobei Milz, Niere, Blut und Leber HU-Intervalle aufweisen, die alle den Wert 40 HU beinhalten (vgl. [Stü]).

[2] Zur Ergänzung: Ein Zentimeter (cm) sind gleich 10 Millimeter (mm). Ein Millimeter sind gleich 1000 Mikrometer (μm). Und ein Mikrometer sind gleich 1000 Nanometer (nm).

[3] Noch ist der Unterschied zwischen der mesoskopischen Ebene und mikroskopischen Ebene nicht erklärt. Dies wird aber im Kapitel 2.3 nachgeholt.

Mit der Verwendung von computergenerierten 3D-Modellen lässt sich dieser Prozess aber verbessern. Anstatt nur einzelne Teile zu analysieren, können die Immunhistologen nun den Befund als Ganzes betrachten und begutachten. Eine dreidimensionale Rekonstruktion ermöglicht eine kompaktere und klarere Darstellung des Befundes (zum Beispiel innerhalb einer Journalveröffentlichung, was auch zum besseren (Gesamt-)Verständnis beiträgt) und hilft bei der Verifizierung des imaginären 3D-Modells[4] der Immunhistologen.[5]

Die Dissertation behandelt die Arbeitsschritte, die benötigt werden um ein 3D-Modell in Form eines Flächenmodells aus immunhistologisch eingefärbten Schnittserien zu erstellen. Da diese Schnitte beim Anfertigen gewissen Manipulationen unterlegen sind und die eingescannten Schnittserien ein beachtliches Datenvolumen aufweisen und somit zu ebenso großen 3D-Rekonstruktionen beziehungsweise Flächenmodellen (Meshes) führen, müssen verbesserte Wege der Vorverarbeitung, Erstellung, Optimierung beziehungsweise Nachbearbeitung, sprich muss eine Verbesserung der Flächenkonstruktion gefunden werden. Grundlagen und bekannte Methoden mit denen man diese Aufgabe bewältigen kann werden in den folgenden Kapiteln vorgestellt. Da es sich auch um eine interdisziplinäre Arbeit handelt, soll der erste Blick auf die Histotechnik gehen, welche die zu bearbeitenden Daten erzeugt.

1.1 Histotechnik

Übersetzt man den Begriff Histologie, so bedeutet er Gewebelehre.[6] [7] Er bezeichnet das Wissenschaftsgebiet, welches sich mit biologischen Gewebeproben auf mikroskopischer Ebene befasst und diese auf ihre Funktion und ihren Aufbau hin untersucht. Für derartige Untersuchungen müssen die Gewebeproben in der Histotechnik aufbereitet werden. Daher sollen in diesem Kapitel kurz die Arbeitsschritte der histologischen Gewebeverarbeitung und Serienschnitterstellung dargestellt und zugleich die möglichen Komplikationen aufgezeigt werden.[8]

1.1.1 Das Fixieren, das Einbetten und das Einblocken

In der Histologie werden Gewebeproben zuerst fixiert, dann eingebettet und im Anschluss eingeblockt. Die Fixierung, also die Haltbarmachung einer Gewebeprobe, ist dabei von mehreren Faktoren abhängig. Laut [Lan06] sind das, neben der Wahl des Fixiergemisches, die Gewebeprobengröße, der ph-Wert, die Temperatur und die Zeit.[9] Beispielsweise ist beim Faktor Zeit zu beachten, dass die Fixierung nicht zu langsam stattfinden darf. Das Gewebe wird nicht mehr mit Sauerstoff versorgt und erfährt ein Einsetzen des Zellzerfalls (Nekrose). Dieser wird in der Regel mit einem Fixiermittel wie Formaldehyd

[4] Gemeint ist hier das Modell, dass im Kopf zusammengesetzt wurde.

[5] Als Beispiel, wie die 3D-Modelle zu einem besseren Verständnis beitragen, wäre die Pressemitteilung Nr. 038/2017 vom 19. April 2017 zu nennen [Wiß]. In dieser heißt es: "Die bisherigen Untersuchungen haben bereits zu überraschenden Erkenntnissen geführt. So hat sich herausgestellt, dass die feinsten Blutgefäße in der Milz offen enden und das Blut für eine kurze Strecke außerhalb von Blutgefäßen fließt. Im blutbildenden Knochenmark des Beckenkamms verlaufen die beiden bisher bekannten Arten feinster Blutgefäße (Kapillaren) vermutlich nicht hintereinander, sondern nebeneinander. Darüber hinaus zeigte sich, welche Antikörper man verwenden muss, um beide Gefäßarten gleichzeitig nachzuweisen und somit die feinsten Gefäße im Knochenmark vollständig darzustellen."

[6] Etymologie: griechisch *ιστός histos* 'Gewebe', griechisch *λόγος logos* 'Lehre'.

[7] Vgl. [Wikf] und [LP12] Kapitel 'Einleitung' Absatz 1 Seite 1.

[8] Diese Schritte können nicht nur in dem hier viel erwähnten Buch von Frau Lang [Lan06] nachgelesen werden. Sie sind auch unter [Wel06] Kapitel 1.3 Seite 4ff und [LP12] Kapitel 27 Seite 629ff erläutert.

[9] Vgl. [Lan06] Kapitel 7 Seite 40ff.

oder mit Paraformaldehyd unterbunden. Dabei kann es zu Fixierungsartefakten kommen, wenn die Fixierung zu langsam erfolgt (Nekrose) oder wenn dies zu schnell geschieht ('Substanzflucht').[10]

Wurde die Probe haltbar gemacht, wird anschließend die vorhandene Zellflüssigkeit durch eine Einbettungsflüssigkeit ersetzt. Dies geschieht im klassischen Fall mit Paraffin und erfolgt über eine aufsteigende Alkoholreihe. Dabei wird das Gewebe erst in Ethanol, dann in Isopropylalkohol, dann in Xylol und zum Schluss in Paraffin getränkt. Eine Einbettung kann neben Paraffin auch mit anderen Flüssigkeiten erfolgen. Gudrun Lang [Lan06] nennt hier Einbettungsflüssigkeiten wie Gelatine, Agar, Polyesterwachs und andere.[11] Bei der Einbettung von Knochenmarkproben (samt Knochen) ist die Einbettungsflüssigkeit in der Regel Kunststoff, wie beispielsweise Methacrylate (Technovit) oder Epoxidharze.[12] Die Wahl der Einbettungsflüssigkeiten wird unter anderem aus dem Gesichtspunkt der Stabilität gewählt. Sie sollten genauso stabil sein wie das einzubettende Gewebe.

Nach der Einbettung folgt das Einblocken. Dies kann mit einem Einkammer-Einbettautomaten geschehen oder auch per Hand. Im letzteren Fall wird die Probe per Hand in einen Block gegossen.[13]

Das Ergebnis ist ein Paraffin-Block (FFPE[14]-Block) oder ein Hartplastik-Block, welcher im nachfolgenden Schritt weiterverarbeitet werden kann.

1.1.2 Das Zuschneiden

Aus diesen eingeblockten Gewebeproben werden mit Hilfe eines Mikrotomes (siehe Abbildung 1) mikroskopische Präparate in Serie (Serienschnitte) erstellt. Dieser Herstellungsschritt wird auch Mikrosektion genannt und erweist sich als kritisch. Die Präparate sind meist recht kleine Stücke, deren Kantenlänge kaum größer als $1cm$ ist, und die FFPE- bzw. Hartplastik-Blöcke selbst gehen selten über $2cm$ Kantenlänge hinaus. Die Schnitte, die angefertigt werden, weisen nur eine Dicke von wenigen Mikrometern (μm) auf.[15] Dies bedeutet, dass es beim Schneiden schon aufgrund dieser Dünne zu Manipulationen kommen kann. Ist beispielsweise der Neigungswinkel (Inklination) des Mikrotommessers zu steil, kann es zu Rippeln in den Schnitten (chattering) kommen.[16] Werden dagegen die Paraffinschnitte, wie zum Beispiel bei den Milzschnitten in dieser Dissertation, mit einem Schnittrichtungswinkel (Deklination) von 90° geschnitten, erfahren sie eine gewisse Stauung. Aus diesem Grund werden solche Schnitte in ein Wasserbad von 40° Celsius eingelegt und erfahren so eine Glättung. Im Anschluss werden sie jeweils mit einem Glasobjektträgern aus dem Wasser herausgezogen und auf eine Heizplatte zur weiteren Glättung gelegt. Es ist leicht nachzuvollziehen, dass dies unvermeidlich zu Verzerrungen innerhalb der Schnittpräparate führt. Bei einer Mikrosektion von Plastik-Blöcken wird ein leistungsstarkes Mikrotom benötigt (eventuell mit einem größeren

[10] Vgl. [Lan06] Kapitel 5.A.2 Seite 42f.
[11] Vgl. [Lan06] Kapitel 8 Abschnitt A bis F Seite 88ff.
[12] Vgl. [Lan06] Kapitel 8 Abschnitt G 2 Seite 111ff.
[13] Eine Einblockung per Hand geschah auch bei den Milzproben, mit welchen diese Dissertation arbeitet. Bei einer Einblockung per Hand wird das Gewebe in ein Metallgefäß gelegt und hochgehalten. Das flüssige Paraffin wird dann eingegossen. Die Probe und das Paraffin müssen dabei die gleiche Temperatur haben.
[14] FFPE steht für *formalin-fixiert paraffin-eingebettet* (vgl. [Lan06] Kapitel 8 Abschnitt A Absatz 1 Seite 88). Wobei Formalin eine $3,7-8,0$%-ige Formaldehydlösung ist (vgl. [Lan06] Kapitel 5 Abschnitt C 1 Seite 49). Die unterschiedlichen Prozentangaben beruhen darauf wie stark man das Formaldehydlösung verdünnt. Gängig sind $3,7$%-ige Lösungen, unter 2% sollte diese aber nicht gehen.
[15] Laut [Lan06] (Kapitel 9 Abschnitt A Seite 124) können die Schnitte eine Dicke von $0,01\mu m$ bis $60\mu m$ erreichen. Wobei Schnitte für Lichtmikroskopie mindestens $0,5\mu m$ dünn sind. Für eine immunhistologische Färbung darf die Dicke der Schnitt nicht über $50\mu m$ hinausgehen (vgl. [Lob+17]).
[16] Vgl. [Lan06] Kapitel 9 Abschnitt D Seite 137f.

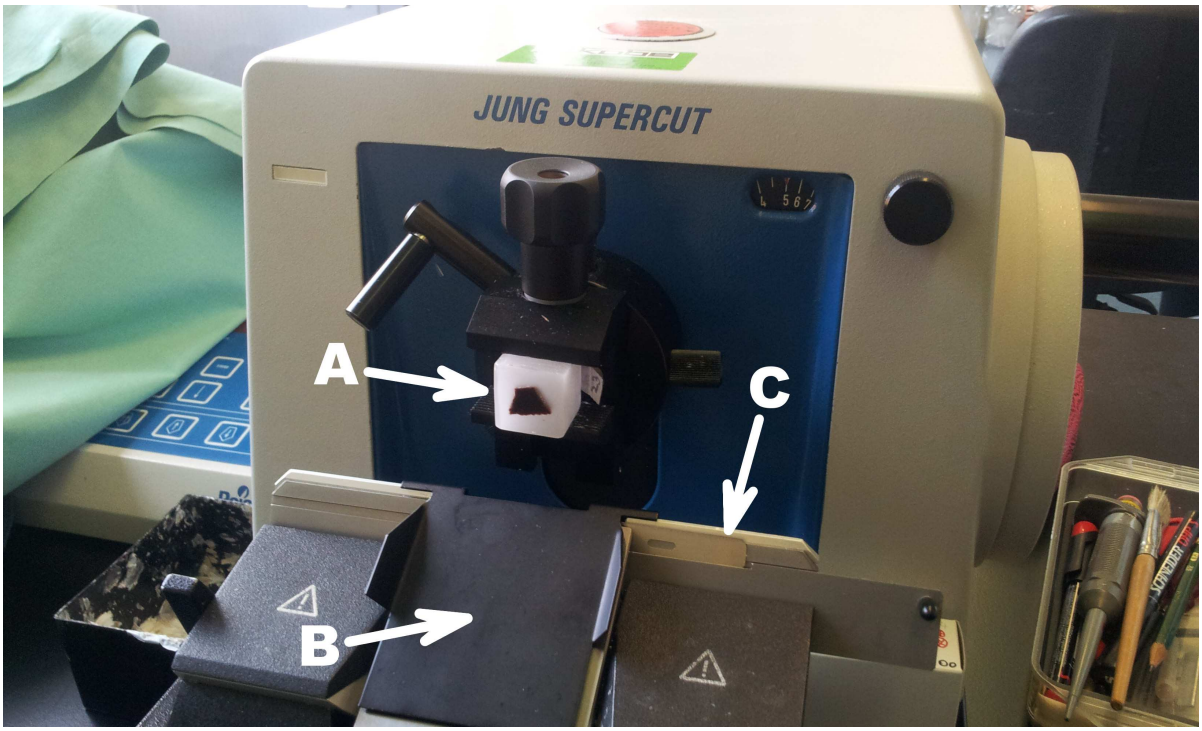


Abbildung 1: Ein Mikrotom ist eine Apparatur, welche speziell für die Erstellung von sehr dünnen Gewebeproben verwendet wird. Die Abbildung zeigt ein älteres Rotationsmikrotom, welches sich im immunhistologischen Labor von Frau Prof. Steiniger befindet. Zu sehen ist ein eingespanntes Paraffinstück (Pfeil A) und durch die Abdeckung (Pfeil B) nur zum Teil zu sehen die Einmalklinge aus Wolframcarbid (Pfeil C).

Deklinationswinkel). Dennoch sind auch hier die möglichen Manipulation durch den Vorgang der Mikrotomie die selben.

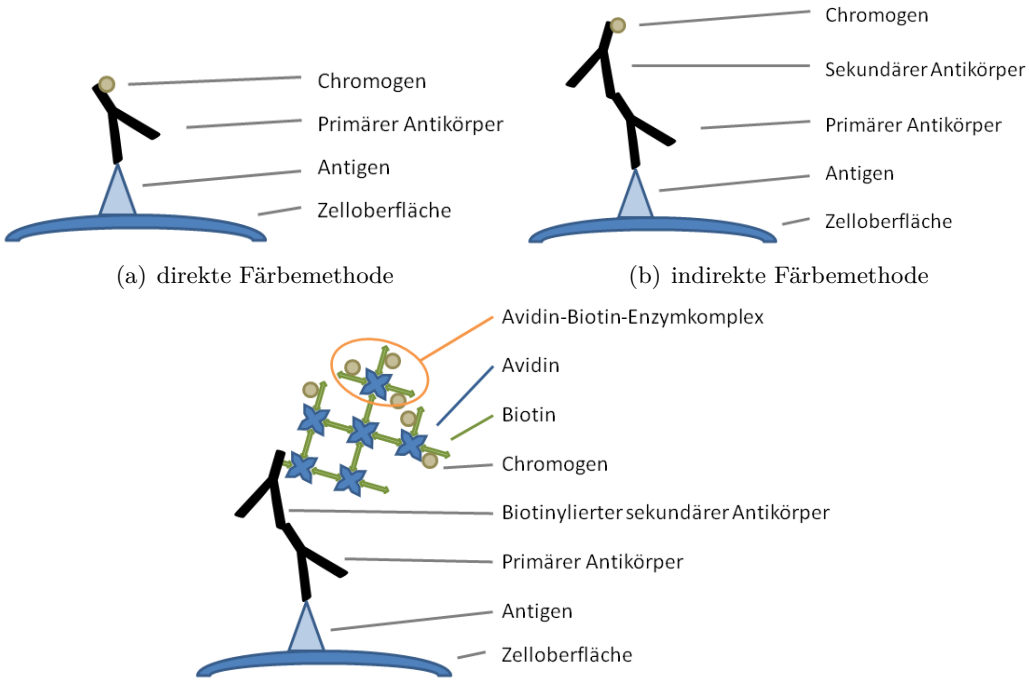
1.1.3 Das Einfärben

Wenn die einzelnen Paraffinschnitte auf Glasobjektträger aufgebracht sind, werden sie im Anschluss mittels Reagenzien eingefärbt, um so die Strukturen von Interesse sichtbar zu machen. Die klassische Histologie unterscheidet zwischen baso-, azido- und neutrophilen Strukturen. Entsprechend ist die Wahl der Reagenzien. So können basophile Zellkerne mit dem basischen Stoff Hämatoxylin blau gefärbt werden.[17]

Antigendemaskierung/Immunhistologische Einfärbung Die Gewebeproben mit denen diese Dissertation arbeitet sind immunhistologisch eingefärbt. Das heißt, hier sind die Strukturen von Interesse bestimmte Zelltypen beziehungsweise ihre *Antigene*[18]. Eine Einfärbung wird folglich mit Hilfe von Antikörpern umgesetzt. Dabei kommen ein bis vier Antikörper mit einer zusätzlichen Verwendung eines Enzym-Komplexes zum Einsatz (siehe Abbildung 2). Anschließend wird mit einem passenden Substrat beziehungsweise Chromogen eingefärbt.

[17] Vgl. [Lan06] Kapitel 6 Abschnitt C 3.5 Seite 79 und Kapitel 10 Abschnitt F Seite 183ff.

[18] Ein Antigen ist alles, was an die Antigenrezeptoren der Lymphozyten andocken kann. Lymphozyten gehören zu den Leukozyten (weiße Blutzellen) und erfüllen ihre Funktion in der adaptiven Immunabwehr des Körpers. Sie stehen daher im Zusammenhang mit den lymphatischen Organen, wie Milz oder Knochenmark (vgl. [LP12] Kapitel 12.1.13 Seite 287ff und Kapitel 13 Seite 304ff).



(a) direkte Färbemethode (b) indirekte Färbemethode

(c) ABC-Methode

Abbildung 2: Drei von mehreren verschiedenen immunhistochemischen Färbemethoden. In der direkten Methode (Bild (a)) setzt man für die Einfärbung zuerst einen Antikörper auf die Zellen beziehungsweise ihre Antigene an. Es kann optional ein zweiter Antikörper folgen (Bild (b)). Die ABC-Methode (Bild (c)) wird in Kapitel 2 noch genauer beschrieben.

Doch bevor eine solche Immunfärbung stattfinden kann, muss vorher eine Antigendemaskierung erfolgen. Denn durch die vorangegangene Fixierung und Einbettung ist wie zuvor erläutert alles fix (starr). Um wieder eine gewisse Beweglichkeit in die Strukturen zu bringen, werden die Schnitte üblicherweise mit Xylol, Ethanol (aufsteigende Alkoholreihe) und Hitze behandelt und so das Paraffin beziehungsweise das Hartplastik entfernt. Ist die Einfärbung erfolgt, werden die Serienschnitte wieder mit einer absteigenden Alkoholreihe in einen Zustand der längeren Haltbarkeit gebracht.

Zum Schluss werden die Glasobjektträger beschichtet. Es wird ein Deckglas aufgebracht und das Dauerpräparat ist fertig.

1.1.4 Die Begutachtung und die Digitalisierung

Die Dauerpräparate können nun mikroskopisch untersucht werden. Dies geschieht im Fall der hier in dieser Arbeit behandelten immunhistologisch eingefärbten Proben mit dem klassischen Lichtmikroskop.[19]

Damit die Serienschnitte am Computer zu einem 3D-Modell rekonstruiert werden können, müssen diese digitalisiert werden. Um die Vielzahl der Schnitte einer Serie nicht jeweils einzeln ablichten zu müssen, kann man diese mit Hilfe eines Objektträger-Scanners digitalisieren lassen. Die Gewebeproben, auf die in Kapitel 2 noch ausführlicher eingegangen wird, wurden zum Beispiel unter anderem mit

[19] Zwar gibt es auch andere Arten von Mikroskopen, wie beispielsweise das Elektronenmikroskop oder Magnetresonanzmikroskop (das Erstere hat eine Auflösung von etwa $0,1nm$; das Zweite und das klassische Lichtmikroskop haben eine Auflösung im Nanobereich), diese helfen aber nicht bei der Bestimmung der Funktion der Zellen und der Zelltypen.

einer Leica SCN 400 digitalisiert. Da der Vergrößerungsfaktor dabei das 20-fache betrug, arbeitet der Objektträger-Scanners derart, dass er jede einzelne Gewebeprobe im Rasterverfahren stückchenweise Spalte für Spalte einscannt. Die so gewonnenen Teilscans der Probe werden dann im Anschluss mittels Stitching[20] zu einem großen Scan zusammengefügt. Manchmal erkennt man das Stitching an den unterschiedlichen Helligkeiten im Großscan eines Schnittes. Die Objektträger-Scanner liefern als Ergebnis der Digitalisierung Dateien in BMP-, SCN- oder auch DICOM-Format.[21]

1.1.5 Fazit

Von den oben beschriebenen Arbeitsschritten der Histotechnik, die nur einen groben Überblick darstellen, sind das Zuschneiden der Präparate und das Aufbringen dieser auf die Glasobjektträger wohl die, die am meisten Einfluss auf die Qualität der eingescannten Schnittbilder haben. Es gibt viele Faktoren, die während der Histotechnik zu beachten sind. Das Wasserbad darf nicht zu warm sein (der Schnitt treibt sonst auseinander), die Mikrotomklinge muss nicht nur richtig angesetzt, sondern auch scharf sein (die Probe enthält sonst Scharten), die Antidemaskierung muss zeitlich stimmen (sonst findet zum Beispiel überhaupt keine Antidemaskierung statt). Selbst wenn während dieses Erstellungsprozesses der Serienschnitte alles richtig gemacht wurde, bleiben die Manipulationen, die durch das Auftragen auf die Glasobjektträger entstehen, unvermeidlich. Die bisherigen Arbeiten an der Promotion haben gezeigt, dass die Helligkeitsunterschiede, die während des Einscannens bei der Digitalisierung entstehen können, vernachlässigbar sind.

Beachtlich ist dagegen die Menge an Daten, die erzeugt wird. Ihr Speicheraufwand ist in den letzten Jahren enorm angestiegen. Die Gründe liegen hier bei der besser gewordene Auflösung der Mikroskope und dem schneller gewordenen Arbeitsprozess bei der Digitalisierung. Dies lässt aber auch einen Bedarf an mehreren Hilfsmitteln und Methoden entstehen, wenn es um die Erstellung von immunhistologischen 3D-Modellen geht. Die Schnitte müssen zuerst ausgerichtet werden. Sie alle liegen auf ihren jeweiligen Glasobjektträgern nicht an der gleichen Stelle und die Verzerrungen müssen ebenfalls so gut es geht korrigiert werden. Üblicherweise greift man hier auf Registrierungs- und Feature Detection-Algorithmen (Kapitel 1.3) zurück. Aus den ausgerichteten Schnitten, beziehungsweise aus der Region von Interesse (ROI) der Schnittserie, kann dann eine unkomprimierte Datei mit rohen Bilddaten (RAW-Datei) angefertigt werden. Aus diesen Volumendaten wird dann wiederum mittels eines Rekonstruktionsalgorithmus (Kapitel 'Rekonstruktion (Marching Cubes)' ab Seite 16) ein Volumenmodell erstellt. Das so gewonnene 3D-Modell, welches auch Dreiecksnetz oder Mesh genannt wird, bedarf eventuell noch einiger Optimierungen, wie Simplifizierung (Kapitel 1.5), Glättung, Rendering, etc. Um die Datenmenge bewältigen zu können, müssen hierfür einige der Algorithmen aus Registrierung, Feature Detection, Rekonstruktion und Simplifizierung entsprechend optimiert werden. Hauptsächlich geschieht dies durch Parallelisierung. Die Compute Unified Device Architecture (CUDA) von Nvidia aus Kapitel 1.6 stellt dabei, neben anderen Programmiersprachen (Kapitel 1.6.3), eine mögliche Herangehensweise dar.

[20] *to stitch* engl.: nähen. Gemeint ist damit ein Verfahren aus der digitalen Bildverarbeitung, welche mit Registrierungs- und Projektionsverfahren (ggf. auch mit Belichtungskorrektur) mehrere Bilder zu einem zusammensetzt. Genaueres lässt sich unter [Wikj] nachlesen. Der Wikipedia-Link enthält auch ein schönes Beispielbild, welches die Stadt Marburg im Panoramablick zeigt.

[21] Bei BMP (Bitmap) handelt es sich um ein Rastergrafikformat von Microsoft Windows. Bei den Bildern mit der Endung SCN liegt ein spezielles Format der Firma Leica vor. Dieses ist für virtuelle Mikroskopie angedacht. DICOM ist ein klassisches Format für medizinische Daten (Container-Format), welches von allen gängigen medizinischen bildverarbeitenden und -erzeugenden Geräten verwendet wird.

Doch zuerst sollen einige Begrifflichkeiten erklärt werden, da sie zum besseren Verständnis der folgenden Kapitel – vor allem Kapitel 3 – beitragen.

1.2 Mathematische Definitionen

Wenn in dieser Dissertation von *Volumendaten* gesprochen wird, dann ist damit eine Menge von dreidimensionalen Punkten gemeint, welche mit Attributen[22] versehen sind. Eines der bekannteren Verfahren, das 3D-Punkte mit Attributen zusammenführt, ist die Computertomografie (CT). Sie erzeugt aus mehreren 2D-Röntgenbildern 3D-Punkte und versieht diese dann mit HU-Werten (vgl. Seite 2 Fußnote 1). Die Anfertigung von digitalen Serienschnitten ist ebenfalls ein passendes Beispiel für ein solches Zusammenführen von attribuierten Punkten.[23] Gleich mit welchem Verfahren man eine attribuierte Punktemenge erzeugt, die Volumendaten können auch als eine Menge von n-Tupel betrachtet werden. Wobei die n-Tupel mindestens Quad-Tupel sind, da neben den x-, y- und z-Koordinaten mindestens ein Attribut vorhanden ist. Die Tupel-Menge ist endlich groß und ihre Größe ist abhängig von der Messgenauigkeit des Verfahrens (die Auflösung des CTs oder des Fotoapparates können als Einflussfaktor genannt werden). Wenn nun diese endliche Menge mit V bezeichnet wird (V steht für Volumendaten) und m die Anzahl der dreidimensionalen Punkte mit zusätzlichen Eigenschaften ist, dann lässt sich das Erzeugen von Volumendaten mathematisch wie folgt beschreiben:[24]

$$
\begin{aligned}
& \varphi \ : \mathbb{R}^3 \to \mathbb{R}^n \qquad (1)\\
\text{mit} \quad & M \ \subset \mathbb{R}^3 \wedge V \subset \mathbb{R}^n \wedge |M| = |V| = m \wedge n, i, m \in \{\mathbb{N} \cup 0\}\\
\text{und} \quad & V \ := \{v_i \ : \ v_i = (a_{i,1}, \ldots, a_{i,n}) \wedge (n > 3) \wedge (0 \leq i < m) \wedge\\
& \qquad \varphi^{-1}(a_{i,1}, a_{i,2}, a_{i,3}) = (x_i, y_i, z_i) = p_i \in M\}
\end{aligned}
$$

Anhand der letzten Zeile in Formel (1) lässt sich erkennen, dass die ersten Attribute des i-ten Elementes, also $a_{i,1}, a_{i,2}$ und $a_{i,3}$, als x-, y- und z-Koordinaten von v_i festgelegt sind.

Sieht man sich die Verfahrensweisen (wie CT, Objektträger-Scanners, etc.) mit denen Volumendaten erzeugt werden genauer an, stellt man fest, dass die attribuierten Punkte im dreidimensionalen Raum auf eine gewisse Art und Weise angeordnet sind. Diese Verteilung der Punkte im Raum kann dabei ganz willkürlich sein – in diesem Fall wären die Volumendaten eine ungeordnete Punktwolke – oder sie unterliegt einer gewissen Ordnung. Meistens sind die Punkte, welche auch als *Voxel*[25] [26] bezeichnet werden, als *kartesisches Gitter* angeordnet. Das heißt, die Punkte sind entlang der x-, y- und z-Achse im dreidimensionalen Raum derart aufgereiht, dass jede Reihe an Punkten jeweils parallel zur entsprechenden Achse verläuft. Die Abstände zwischen den Punkten entlang jeder Achse sind gleich groß oder anders ausgedrückt fix. Oftmals liegt bei der Verteilung der Punkte auch ein *regelmäßiges*

[22] Bei den Attributen handelt es sich um Zahlenwerte. Sollten sie es mal nicht sein, wie beispielsweise bei Farbangaben, dann können diese einfach in Zahlen umkodiert werden. Häufig handelt es sich dann um ganze Zahlen (in englisch Integers), welche bekanntermaßen eine Teilmenge der reellen Zahlen sind.

[23] Hier werden, wie gerade schon im Kapitel 1.1.4 erwähnt, entweder die Schnitte einzeln abfotografiert oder sie werden mittels Objektträger-Scanners eingelesen, per Stichingverfahren (vgl. Kapitel 1.1.4 Seite 6 Absatz 2) zusammengesetzt. Die aus diesem Digitalisierungsprozess entstandenen 2D-Bilder werden dann übereinander gestapelt.

[24] Ein laienhafteres Beispiel wären Messungen innerhalb eines Wohnzimmers. Definiert man in diesem einen dreidimensionalen Nullpunkt und misst dann zur gleichen Zeit an mehreren Stellen die Temperatur, die Helligkeit und die Luftfeuchtigkeit, so erhält man eine endliche Menge V, die aus n-Tupel besteht.

[25] Der Begriff Voxel leitet sich aus dem Volumen und Element ab.

[26] Siehe [Wikl] und [SW12] Kapitel 17.2.4.4 Seite 548 Absatz 1.

Gitter vor. Hier sind die Abstände in Richtung der x-, y- und z-Achse ebenfalls fix, aber sie unterscheiden sich bei mindestens einer Achse voneinander.

Um mit den Elementen aus V besser arbeiten zu können, definiert man mit Hilfe der Abbildung ι eine weitere Menge $X \subset \mathbb{R}^3$, welche die x- y- und z-Koordinaten der Voxel aus V indexiert. In gewisser Weise werden somit den Elementen aus V drei weitere Attribute hinzugefügt. Hierfür wird zusätzlich ein *Selektor* benötigt. Sei $v_i \in \mathbb{R}^n$ und $i, s, n \in \{\mathbb{N} \cup 0\}$ mit $1 \leq s \leq n$ und $0 \leq i \leq (m-1)$, dann gilt:

$$\begin{aligned} &\sigma : \mathbb{R}^n \to \mathbb{R} \\ &\sigma_s(v_i) := a_{i,s} \qquad \text{mit } v_i = (a_{i,1}, \ldots, a_{i,s}, \ldots, a_{i,n}) \end{aligned} \tag{2}$$

Mit diesem lässt sich die Abbildung ι wie folgt beschreiben. Seien $X \subset \mathbb{R}^3 \wedge V \subset \mathbb{R}^n$ und idx_x, idx_y, idx_z die Indizes, welche aus der Indexierung von x, y und z entstehen. Weiterhin seien $dimX, dimY, dimZ$, $idx_x, idx_y, idx_z, i \in \{\mathbb{N} \cup 0\}$ und $x, y, z \in \mathbb{R}$, dann ist:

$$\begin{aligned} &\iota : \mathbb{R}^n \to \mathbb{R}^3 \\ \text{mit} \quad &X := \{(idx_x, idx_y, idx_z) : \ 0 \leq idx_x \leq dimX \wedge 0 \leq idx_y \leq dimY \wedge \\ &\qquad 0 \leq idx_z \leq dimZ \wedge \iota^{-1}(idx_x, idx_y, idx_z) = (a_{i,1}, a_{i,2}, a_{i,3}) = \varphi(x_i, y_i, z_i) \wedge \\ &\qquad \sigma_1(\iota^{-1}(idx_x, idx_y, idx_z)) \leq \sigma_1(\iota^{-1}(idx_x, idx_y, idx_z)) + 1 \Rightarrow idx_x \leq idx_x + 1 \wedge \\ &\qquad \sigma_2(\iota^{-1}(idx_x, idx_y, idx_z)) \leq \sigma_2(\iota^{-1}(idx_x, idx_y, idx_z)) + 1 \Rightarrow idx_y \leq idx_y + 1 \wedge \\ &\qquad \sigma_3(\iota^{-1}(idx_x, idx_y, idx_z)) \leq \sigma_3(\iota^{-1}(idx_x, idx_y, idx_z)) + 1 \Rightarrow idx_z \leq idx_z + 1\} \end{aligned} \tag{3}$$

Weiterhin gilt für $d_x, d_y, d_z \in \mathbb{R}$:

$$\begin{aligned} d_x &:= \Big(\sigma_1(\iota^{-1}(idx_x, idx_y, idx_z)) + 1\Big) - \sigma_1(\iota^{-1}(idx_x, idx_y, idx_z)) \\ d_y &:= \Big(\sigma_2(\iota^{-1}(idx_x, idx_y, idx_z)) + 1\Big) - \sigma_2(\iota^{-1}(idx_x, idx_y, idx_z)) \\ d_z &:= \Big(\sigma_3(\iota^{-1}(idx_x, idx_y, idx_z)) + 1\Big) - \sigma_3(\iota^{-1}(idx_x, idx_y, idx_z)) \end{aligned} \tag{4}$$

Gilt für die Abstände zwischen den Punkten $d_x = d_y = d_z$, liegt ein *kartesisches Gitter* vor. Im Beispiel der digitalen Serienschnitte, mit welchen diese Dissertation arbeitet und auf welche in den Kapiteln 2.2 und 4.1 Absatz 1 noch eingegangen wird, ist das Gitter *regelmäßig* und es gilt $d_x = d_z \neq d_y \wedge$ $d_x, d_z < d_y$. Die Werte $dimX, dimY$ und $dimZ$ stellen die Obergrenzen der Indizes dar und mit ihnen kann die *Dimension* (auch *Größe* genannt) der Volumendaten angegeben werden:

$$dim(V) := dim(X) = dimX \times dimY \times dimZ \tag{5}$$

Weiterhin lassen sich mit der eben einführten Menge X die in Formel (1) angegeben Variablen m und i genauer beschreiben.

$$m := dimX \cdot dimY \cdot dimZ \tag{6}$$

$$i := idx_x + idx_z \cdot dimX + idx_y \cdot dimX \cdot dimZ \tag{7}$$

Zu letzterer Formel sei erwähnt, dass die x-, y- und z-Achse wie in OpenGL üblich angeordnet sind.

Die z-Achse zeigt in Richtung des Betrachters, x zeigt nach rechts und die y-Achse in die Höhe. Somit definiert Formel (7) mit Hilfe der Indizes idx_x, idx_y und idx_z einen globalen Index i, der die Voxel von links nach rechts und von unten nach oben durchnummeriert.

Betrachtet man die Elemente der Menge V nicht als n-Tupel sondern als Vektoren, die über $\mathbb{R}^n$ kodiert sind, dann kann mit Hilfe eines Selektors σ_s (Formel (2)) und der Abbildung aus Formel (1) eine weitere definiert werden:

$$f : \mathbb{R}^3 \to \mathbb{R} \qquad (8)$$
$$f := \sigma_s \circ \varphi$$

Die Verkettung f wird auch als *Skalarfeld* bezeichnet, da sie jedem dreidimensionalen Punkt einen *Skalar* zuordnet. Im Beispiel der abgelichteten Gewebeschnitte sind den dreidimensionalen Punkten Rot-, Grün- und Blau-Werte zugeordnet. Die Elemente sind somit über $\mathbb{R}^6$ kodiert und es ließe sich beispielsweise mittels oben erwähnter Verkettung ein Skalarfeld bezüglich der Farbe Blau erstellen.[27] Für eine vereinfachte Schreibweise soll die Indexangabe beim Selektor entfallen und mit $p_i \in M \subset \mathbb{R}^3$ gilt nun:

$$\sigma(\varphi(p_i)) = a_i = f(p_i) \qquad (9)$$

Die Funktion φ ist streng genommen eine stetige Funktion und das Skalarfeld wäre auch ein *kontinuierliches*, da aber die Volumendatengewinnung durch Abtastung erfolgt, handelt es sich um ein *diskretes Skalarfeld*. Die Menge der Volumendaten ist folglich eine endliche.[28] Daher werden oftmals auch die Begriffe *diskreter Datensatz* oder *gesampelte Daten* verwendet. Im Beispiel der Schnittserien setzt der lichtempfindliche Sensor der Kamera beziehungsweise des Objektträger-Scanners mit seiner Abtastgenauigkeit wiederum die Genauigkeit der Approximation an φ fest.[29] Welche Werte zwischen den *diskreten Skalaren* liegen ist nicht bekannt. Sie werden daher bei Bedarf durch Interpolation ermittelt.

Mit Formel (9) lässt sich das Endprodukt des Marching Cubes-Algorithmus [LC87] (Kapitel 1.4) genauer beschreiben: die *Iso-Oberfläche*[30] (auch *Isofläche* genannt). Die Iso-Oberfläche ist ein dreidimensionales Konstrukt innerhalb eines Skalarfeldes. Ihre Fläche definiert eine Menge an dreidimensionalen Punkten, deren Skalare alle den gleichen Wert haben. Oder etwas mathematischer ausgedrückt: Sei ein $\rho \in \mathbb{R}$ ein gesuchter gemeinsamer Wert für eine Menge an Skalaren (ρ (rho) wird auch *Schwellenwert* oder *Isowert* genannt), dann ist eine Isofläche I_ρ wie folgt definiert:[31]

$$I_\rho := \{p : \ p \in \mathbb{R}^3 \wedge f(p) = \rho\} \qquad (10)$$

Es entstehen somit drei Punktemengen. Einmal die Isofläche I_ρ, die nicht zwangsläufig zusammenhängend sein muss, aber ohne Löcher und somit *watertight* also *geschlossen*, ist. Dann die

[27] Man könnte bei den Gewebescans aufgrund der Basisfarben Rot, Grün und Blau auch davon ausgehen, dass mit einem Vektorfeld gearbeitet wird. Dies ließe sich aber leicht in ein Skalarfeld umkodieren. Zudem wird beispielsweise in Java der Farbwert in `BufferedImage` als `int`-Wert kodiert. Siehe [Jav].

[28] Vgl. Kapitel 1.2 Absatz 1 Seite 8.

[29] In dieser Dissertation wird f, um eine gewisse Lesbarkeit zu erhalten, auch einfach nur als Datensatz, Volumendaten oder Voxelgitter bezeichnet. Man beachte hierbei, dass f sich auch als Tupelmenge darstellen lässt: $\{(p, f(p))|\ p \in X\}$.

[30] Etymologie: griechisch *ίσοσ* 'gleich'.

[31] p sollte nicht mit ρ (rho) verwechselt werden. Das Weglassen des Indexes i (p anstatt p_i) ist absichtlich geschehen, da man Iso-Oberflächen nur durch Interpolation erhält.

Menge mit all den Punkten, deren Skalar größer als ρ ist: $\{p : \ p \in \mathbb{R}^3 \wedge f(p) > \rho\}$. Und schließlich die Menge mit allen Skalaren unter dem Schwellenwert: $\{p : \ p \in \mathbb{R}^3 \wedge (f(p) < \rho)\}$.

Bei Isoflächen muss in Bezug zum Sampling stets mit Störungen und Verzerrungen gerechnet werden. Dennoch wird in beiden Fällen, sowohl bei der stetigen Funktion als auch beim Sampling, der Begriff Iso-Oberfläche oder Isofläche verwendet. Eine andere Bezeichnung wäre *implizite Fläche*, im englischen Sprachraum begegnet man auch dem Begriff *level surface* (*Niveaufläche*). Letztendlich werden durch die Approximation von φ nicht alle Daten erfasst. Die Position der impliziten Fläche im Raum muss daher bei einem diskreten Datensatz geschätzt werden. Die verloren gegangenen Daten werden, wie schon oben erwähnt, mittels Interpolation rekonstruiert. Dies kann ganz klassisch wie Lorenz und Cline [LC87] linear oder wie bei Chernyaev [Che95] trilinear geschehen.

Um zu verstehen wie der Marching Cubes arbeitet, werden zunächst Begriffe wie *Slice*, *Layer*, *Cubecode*, *Mesh* und *Gradienten* eingeführt werden. Diese werden später im Kapitel 3 verwendet. Ein Slice enthält alle Voxel, die den gleichen Index bezüglich der y-Achse besitzen. Sei V ein gleichmäßiges Voxelgitter und $j \in \{\mathbb{N} \cup 0\} \wedge (0 \leq j \leq dimY - 1)$, so ist das j-te Slice wie folgt definiert:

$$slice_j \quad := \quad \{v_i : v_i \in V \wedge \iota(\sigma_2(v_i)) = idx_y = j\} \tag{11}$$

Mit der Definition von Slices lassen sich nun auch die Layers genauer beschreiben. Für das j-te Layer gilt:

$$layer_j \quad := \quad slice_j \cup slice_{j+1} \tag{12}$$

Ein *Cubecode* wird aus acht Voxel generiert, wobei vier Voxel aus $slice_j$ sind und die anderen vier aus $slice_{j+1}$ stammen. Die Voxel werden, je nachdem ob sie in beziehungsweise auf der Isofläche liegen oder sich außerhalb befinden, mit Eins oder Null kodiert. Weiterhin sind sie, wie auf Seite 16 erläutert wird, als Eckpunkte eines *Würfels* (auch *MC-Würfel*) angeordnet und liegen direkt nebeneinander. Mit Hilfe eines lokal beschränkten Index $k \in \mathbb{N} \wedge 0 \leq k < 8$ und einem beliebigen aber festen $i \in \{\mathbb{N} \cup 0\} \wedge 0 < i < m$ nach Formel (7) können mit dem Voxel $v_i = v_{idx_x, idx_y, idx_z} = v_k \wedge k = 0 \wedge v_i \in V$ die übrigen Voxel adressiert und errechnet werden. So sei der *lokale Index* wie folgt definiert:

$$v_k = \begin{cases} \iota(\sigma_{(1,2,3)}(v_k)) = (idx_x \quad\quad , idx_y \quad\quad , idx_z \quad\quad) & \text{falls } k = 0 \\ \iota(\sigma_{(1,2,3)}(v_k)) = (idx_x + 1, idx_y \quad\quad , idx_z \quad\quad) & \text{falls } k = 1 \\ \iota(\sigma_{(1,2,3)}(v_k)) = (idx_x + 1, idx_y \quad\quad , idx_z + 1) & \text{falls } k = 2 \\ \iota(\sigma_{(1,2,3)}(v_k)) = (idx_x \quad\quad , idx_y \quad\quad , idx_z + 1) & \text{falls } k = 3 \\ \iota(\sigma_{(1,2,3)}(v_k)) = (idx_x \quad\quad , idx_y + 1, idx_z \quad\quad) & \text{falls } k = 4 \\ \iota(\sigma_{(1,2,3)}(v_k)) = (idx_x + 1, idx_y + 1, idx_z \quad\quad) & \text{falls } k = 5 \\ \iota(\sigma_{(1,2,3)}(v_k)) = (idx_x + 1, idx_y + 1, idx_z + 1) & \text{falls } k = 6 \\ \iota(\sigma_{(1,2,3)}(v_k)) = (idx_x \quad\quad , idx_y + 1, idx_z + 1) & \text{falls } k = 7 \end{cases} \tag{13}$$

Die Umkodierung der Isowerte der jeweiligen Voxel wird folgendermaßen umgesetzt:

$$code(v_k) = \begin{cases} 1 & \text{falls } f(v_k) \leq \rho \\ 0 & \text{falls } f(v_k) > \rho \end{cases} \tag{14}$$

Mit Formel (13) und (14) lässt sich nun der Cubecode für den Würfel, der mit v_i adressiert wird, wie folgt beschreiben:

$$cubecode_i := \Big((code(v_7), code(v_6), code(v_5), code(v_4), code(v_3), code(v_2), code(v_1), code(v_0)\Big) \tag{15}$$

Da, wie oben schon erwähnt, die vom Marching Cubes erzeugte Iso-Oberfläche implizit ist, kann diese auch als ein Tupel aus zwei Mengen aufgefasst werden. Einmal die Menge der *Vertices* (oder auch Schnittpunkte von *Würfelkante* und Iso-Oberfläche) und einmal die Menge der Dreiecke (*Faces*). Dieses Tupel wird auch als Mesh oder *Dreiecksnetz* bezeichnet:

$$Mesh := (Vertices, Faces) \tag{16}$$

Mit $l \in \{\mathbb{N} \cup 0\}$ sei die Menge der *Vertices* wie folgt definiert:

$$Vertices := \{p_l : \ p_l \in \mathbb{R}^3\} \tag{17}$$

$$Faces := \{(l_1, l_2, l_3) : \ p_{l_1}, p_{l_2}, p_{l_3} \in Vertices\} \tag{18}$$

Die *Faces* werden gegen den Uhrzeigersinn definiert, um so Rückseite und Vorderseite zu beschreiben. Zudem sollten die mit dem Mesh definierten dreidimensionalen Objekte *mannigfaltig* sein. Das heißt, zu einer Dreieckskante gehören höchstens zwei Faces.

Bei der Formel (16) handelt es sich um die einfachste Definition eines Meshes.[32] [33] Das Tupel kann noch dahingehend erweitert werden, dass weitere Mengen hinzugefügt werden. So könnten in einer zusätzlichen Menge *Shading* Informationen für Schattierungs- oder Simplifizierungsalgorithmen bereitgehalten werden. Derartige Informationen wären beispielsweise die Farbe oder die Normalenvektoren der jeweiligen Meshvertices sein. Letztere sind Vektoren, die orthogonal (senkrecht) auf einer Gerade oder Ebene stehen. Mit ihnen kann die Richtung angegeben werden, in welche die Außenseite eines Faces zeigt. Dies findet wiederum beim Rendern[34] eines Meshes (des 3D-Modells) Verwendung. Ein Beispiel wäre der Schattierungsalgorithmus Gouraud-Shading[35], bei welchem für die Einfärbung der Faces die Werte der jeweiligen drei Vertices verwendet werden. Die dafür wiederum benötigten Normalenvektoren der *Meshvertices* erhält man, indem man die Normalenvektoren der anliegenden Faces mittelt.

Beim Marching Cubes werden die Normalenvektoren für die Schnittpunkte (Meshvertices) mit Hilfe von *Gradienten* interpoliert. Ein Gradient ist ein Operator aus der Mathematik, welcher auf ein Skalarfeld angewandt werden kann. Das Ergebnis einer solcher Operation ist ein Gradientenfeld, dass jedem Skalar einen Vektor zuweist. Dieser zeigt jeweils in die Richtung, in welche die Skalare in der Umgebung die größte Veränderung aufweisen. Ein Merkmal des Gradientenvektors ist, dass dieser orthogonal auf der Niveaufläche (Isofläche) steht, in welcher sich der Skalar befindet.[36]

[32] Diese Definition ist recht nahe an das Indexed Face Set (IFS) angelehnt. Vgl. [Roe].
[33] Das Tool MeshLab [Cig+08], welches für Meshmanipulationen verwendet wird, arbeiten mit OBJ-Dateien, die ähnlich strukturiert strukturiert sind. Auch diese arbeiten mit Indizes.
[34] Etymologie: englisch *to render* 'machen'. Gemeint ist die Erzeugung eines Bildes mit der Hilfe der Computergrafik.
[35] Vgl. [Zep04] Kapitel 5.5.2 Seite 201.
[36] Vgl. [Wikd].

Die herkömmliche Berechnung eines Gradienten geschieht durch partielle Ableitung und die Normalenvektoren der Meshvertices könnte man aus dem Mesh selber erhalten.[37] Da das es sich aber beim Voxelgitter um ein diskretes Skalarfeld handelt und dieses für die Berechnung des Cubecode bereits im Arbeitsspeicher geladen ist, bietet sich die von Lorensen und Cline vorgestellte Verfahrensweise an.[38] Wenn $i \in \{\mathbb{N} \cup 0\} \wedge 0 < i < m$ nach Formel (7) beliebig aber fest ist und für $v_i \in V$ gilt $v_i = v_{idx_x,idx_y,idx_z}$, dann kann mit den Abständen d_x, d_y und d_z des Voxelgitters V (vgl. Formel (4)) der Gradient für v_i wie folgt berechnet werden:

$$grad(v_i) := (grad_x(v_i), grad_y(v_i), grad_z(v_i)) \tag{19}$$

$$grad_x(v_i) := \frac{f(v_{idx_x+1,idx_y,idx_z}) - f(v_{idx_x-1,idx_y,idx_z})}{d_x} \tag{20}$$

$$grad_y(v_i) := \frac{f(v_{idx_x,idx_y+1,idx_z}) - f(v_{idx_x,idx_y-1,idx_z})}{d_y} \tag{21}$$

$$grad_z(v_i) := \frac{f(v_{idx_x,idx_y,idx_z+1}) - f(v_{idx_x,idx_y,idx_z-1})}{d_z} \tag{22}$$

In den nachfolgenden Kapiteln wird nun ein kurzer Überblick über Registrierungs-, Feature Detection-Algorithmen, über Rekonstruktion (hier insbesondere über den Marching Cubes Algorithmus), über Simplifizierung und über CUDA gegeben. Dem Leser könnten die Kapitel 1.3, 1.4 und Kapitel 1.5 bekannt vorkommen, da sie auf den eigenen Veröffentlichungen der Autorin [Ulr+14a; Ulr+14b] basieren.[39]

1.3 Registrierung und Ausrichtung

Bisher existieren zur Registrierung und Ausrichtung von Serienschnitten nur wenige Methoden. Beispiele für manuelle Techniken wären Lösungsansätze von van Krieken et al. [Kri+85] und Steiniger et al. [SRB03]. Letztere Methode wurde acht Jahre später von Steiniger et al. [SBS11] durch die Verwendung eines Joysticks verbessert.

Eine der ersten praktikablen Herangehensweisen an die Problematik der Registrierung und Ausrichtung war der ICP-Algorithmus[40] von Besl und McKay [BM92]. Hierbei handelt es sich um eine Registrierungsmethode, die Daten eines starren Objektes miteinander verknüpft. Dabei können die Daten in Form von mehreren Punktewolken, Linienzüge[41], Isoflächen oder Isolinien[42], parametrisierter

[37] Dies ginge durch das Kreuzprodukt zweier Kanten.
[38] Vgl. [LC87] Kapitel 4 Seite 165 linke Spalte.
[39] Speziell sind hier die Kapitel *Related Work* aus [Ulr+14a; Ulr+14b] gemeint.
[40] ICP steht für Iterative Closest Point. Bei dieser Methode wird zu jedem Punkt der Quellpunktwolke der am nächstliegenden Punkt aus der Zielpunktwolke bestimmt. Im Anschluss wird – meisten mit einer Root Mean Square (RMS) Minimierungsmethode – die passende Rotation und Translation zur Anpassung der beiden Wolken geschätzt und auch ausgeführt. Dies wird solange wiederholt bis die Veränderung RMS-Abstandes unter einen vorher festgelegten Wert fällt.
[41] Gemeint ist hier ein Polygonzug oder auch Streckenzug. Ein Konstrukt bei dem mehrere Punkte mittels gerader Linien miteinander verbunden werden.
[42] Eine Isolinie ist gewissermaßen einen 'zweidimensionale Isofläche'. Auch sie teilt die Punkte (hier zweidimensional) ein drei Mengen auf. Bekanntestes Beispiel sind wahrscheinlich die Isobaren aus dem Wetterbericht.

Kurven oder Flächen aber auch Meshes sein. Diese Methode eignet sich aber nur für kleinere Datenmengen, welche zudem noch vorher eine manuelle starre (*rigide*) Anpassung zueinander benötigen.[43]

Auch wurden im Laufe der Jahre semi-automatische Ansätze vorgestellt. Wie beispielsweise von Gijtenbeek et al. [Gij+05] und Gilhuis et al. [Gil+06], bei welchen die eingefärbten Serienschnitte eine rigide Grobausrichtung mittels einfacher Rotation und Translation während der Aufnahme am Mikroskop erfahren. Diese Grobausrichtungen werden im Anschluss unter Einsatz von affiner Transformation[44] und Kreuzkorrelation (Cross-Correlation) verbessert. Der aufwändige Charakter bei diesen Methoden bleibt, da mit der manuellen Vorpositionierung noch ein manueller Bearbeitungsschritt vorhanden und die Ausrichtung mittels Kreuzkorrelation rechenintensiv ist. Die Lösungsansätze eigenen somit nur für kleinere Serienschnitte mit Bildern von geringer Auflösung.

Auch voll-automatisierte Ansätze sind zu dieser Thematik veröffentlicht worden. Zu nennen wären die Lösungsvorschläge von Ourselin et al. [Our+01] oder die von Nikou et al. [Nik+03]. Bei Ersteren werden Serienschnitte von Ratten- und Affengehirnen mittels Block-Matching[45] bearbeitet. Das Block-Matching übernimmt die Funktion der Merkmalserkennung[46] in den Bilder und paart die Blöcke entsprechend. Mit diesen gefundenen Blöcken wird dann die rigide Transformation zur Ausrichtung berechnet. Bei letzterer Methode wird eine globale Energiefunktion verwendet um Autoradiogramme[47] von Rattengehirnen auszurichten. Aber auch hier sind bei beiden Methoden die Datensätze wieder sehr klein.

Ma et al. [Ma+08] entwickelten eine Methode, die Schnitte von Mäuse-Lymphknoten[48] ausrichtet. Hierbei werden die Bilder mittels Vordergrund- und Hintergrundsegmentierung in Binärbilder umgewandelt und anschließend ein hierarchisches Verfahren angewandt, um diese rigide auszurichten. Eine ähnliche Methode stellten Tanács und Kato [TK11] für CT-Bilder vor. Weitere Methoden mit rigider Registrierung und im Kontext von MRT oder ähnlichen Aufnahmen wurden von Jenkinson und Smith [JS01], Malandain et al. [Mal+04] und Roche et al. [Roc+01] entwickelt. Ju et al. [Ju+06] nutzten wiederum die Stetigkeit des Materials (Mäusegehirn) aus, um die Verformungen zu reduzieren.

Wie in Steiniger et al. [SRB03] erwähnt und im Kapitel 1.1.5 bereits angedeutet, kommt es beim Schneideprozess unweigerlich zu (leichten) Verzerrungen, die zu einem unregelmäßigen Umriss der Gefäße führt. Daher ist für die Registrierung und Ausrichtung von Serienschnitten eine nicht-starre (*non-rigid*) Methode angebracht. Einige bereits veröffentlichte Herangehensweisen dieser Art sollen nun kurz dargestellt werden.

Unter der Annahme, dass biologische Strukturen keine nicht-stetigen Sprünge haben dürfen (Glätteannahme), entwickelten Cifor et al. [CBP11] eine Methode, die die 2D-Schnitte von Mäusegehirnen rekonstruiert. Ebenso wurden elastische Methoden [Cha+11; Gue+01] entwickelt. Gefen et al. [GTN03] verwenden ein 3D-Wavelet, um die histologischen Bilder elastisch zu transformieren. Bajcsy und Kovačič [BK89] praktizieren ihre elastische Registrierung von CT-Bilder

[43] Dies wird später im Kapitel 3.1.4 in Abbildung 13 auf Seite 45 noch veranschaulicht.
[44] Bei einer affinen Transformation handelt es sich um eine einfache Form der nicht-starren Transformation. Die affine Abbildung ist parallel- und linientreu. Während sich die Teilverhältnisse der Strecken untereinander nicht verändert, kann dies bei Winkeln, Längen und Flächen aber der Fall sein. Vgl. [Zep04] Seite 64.
[45] Beim Block-Matching handelt es sich um ein Verfahren aus der digitalen Videokompression. Sie soll Bewegungen detektieren und somit zur möglichen Kompression beitragen. Siehe [Wikc].
[46] Der Begriff wird in Kapitel 3.1.1 Seite 34 noch erläutert.
[47] Eine bildgebende Methode in der Medizin, die mit Hilfe radioaktiver Materialien chemische Komponenten sichtbar macht. Siehe [Wika].
[48] Siehe [LP12] Kapitel 13.3 'Lymphknoten'.

innerhalb einer Pyramide[49]. Wirtz et al. [Wir+05] verwenden Ableitungen des Bildes höherer Ordnung. Saalfeld et al. [Saa+12] bedienen sich wiederum des Block-Matchings, welche mit einer elastische Triangulierung und Hookschen Federn[50] arbeitete. Thirion [Thi98] benutzt ein Streuungsmodell, welches bei der Registrierung zweier Bilder die Konzepte der Thermodynamik verwendet.

Bağci et al. [BCU12] stellt den aktuellen Stand der Ansätze dar (State of the Art), die mit Bildpyramiden arbeiten. Dennoch behandeln alle eben vorgestellten hierarchischen Ansätze nur Strukturen von Interesse, die größer sind als die Strukturen in den immunhistologischen Serienschnitten mit denen diese Dissertation arbeitet.

Rueckert et al. [Rue+99] benutzten für ihre nicht-starre Methode, welche auf Mammographien angewendet wird, B-Splines-basierte Deformationen.[51] Wie bei Peng et al. [PLD05] und Well et al. [Wel+96] werden hier Transinformationen[52] verwendet, wodurch klare Kanten und große gleichförmige Bereiche nötig sind. Derartige Strukturen weisen die Serienschnitte in dieser Dissertation allerdings nicht auf. Schnabel et al. [Sch+01] erweiterten diesen Ansatz von Rueckert et al. mit einer hierarchischen Technik. Xie und Farin [XF04] haben ein Verfahren entwickelt, dass mit hierarchischen B-Splines arbeitet. Ihre Arbeit ist aber nur auf kleine Datensätze ausgelegt und enthält somit keine Merkmalserkennung (*Feature-Detection*).

Chui und Rangarajan [CR03] entwickelten einen nicht-starren Ausrichtungsmethode, der auf CT/MRT-Daten spezialisiert ist. Der Ansatz besteht in einer Umformulierung des Matching-Problems. Die Merkmale werden als Punktwolken aufgefasst, welche aneinander ausgerichtet werden müssen. Durch die Verwendung von Punktwolken verlieren die Merkmale, die mittels Robust Point Matchings (RPM) und Thin-Plate-Splines[53] (TPS) registriert und ausgerichtet werden, an Informationen.[54] Es handelt sich somit eher um eine dichte zufällige Punktmenge ohne strukturelle Informationen - was im Falle der Serienschnitte nicht hilfreich ist.

Wan et al. [Wan+13] verwenden für ihre nicht-starre Registrierung ebenfalls eine Merkmalserkennung, welche auf SURF[55] und TPS basiert. Diese benötigt bedauerlicherweise im ersten Schritt eine aufwändige Trainingsphase für die Merkmalserkennung und ist somit bei seltenen Follikelbefunden[56] nicht geeignet. Kim et al.'s [Kim+97] nutzen Transinformationen und TPS um Autoradiogramme auszurichten. Da es hierbei um Aufnahmen von Rattengehirnen handelt, liegt der Fokus wieder auf große Strukturen. Auer et al. [ARH05] verknüpfen eine hierarchische rigide Registrierung mit einem nicht-starren TPS-Schritt. Aber die in dieser Dissertation behandelten Schnitte haben eine wesentlich höhere Auflösung (20-fache Vergrößerung) als die in Auer et al. [ARH05]. Weiterhin verwendet die in Kapitel 3.1 vorgestellte Methode eine rigide Registrierung, die einmal komplett ausgeführt wird, und kombiniert diese mit einer anschließenden nicht-starren Transformation mit Hilfe von Bi-Cubic-B-Splines. Ebenfalls mit immunhistologischen Schnitten arbeitete Song et al. [Son+13]. Hier waren aber die Schnitte alternierend unterschiedlich eingefärbt.

[49] Gemeint ist eine Anpassung unter Verwendung von mehreren hierarchisch angeordneten Auflösungen der Bilder beziehungsweise der Modelle.
[50] Siehe [Wikg].
[51] Verwendet werden hier sogar dreidimensionale Spline-Kurven.
[52] Der Begriff Transinformation (mutual information, auch gegenseitige Information) kommt aus der Informationstheorie und bezeichnet die mittlere Informationsgehalt der beim Empfänger ankommt. Idealerweise ist dieser maximal. Siehe [Wikk]. Im Kontext der Registrierung und Ausrichtung handelt es sich um die Information zwischen 'Sender'- und 'Empfänger'-Bild.
[53] Siehe [Boo89; Lom; Elo].
[54] Eine Information, die beispielsweise verloren geht, ist die Größe eines Merkmals.
[55] SURF steht für *Speeded Up Robust Features* und ist ein Algorithmus zur Merkmalserkennung. Siehe [Bay+08].
[56] Was Follikel sind wird in Kapitel 2 noch genauer erklärt werden.

Alle bisherigen Methoden gehen von einem BRS (*best reference slice*) aus. Dies hat, wie in Bağci et al. [BB10] dargestellt, eine entscheidende Auswirkungen auf das Endergebnis. Da im Fall des Milz-Präparates alle Schnitte Verzerrungen aufweisen, ist es sinnvoll einen Lösungsansatz zu entwickeln, der eine globale Optimierung der Schnitte anstrebt.

Weitere Übersichten über unterschiedliche Ansätze bezüglich Registrierung und Ausrichtung sind in Werken von Brown [Bro92], Zitová und Flusser [ZF03] und Goshtasby [Gos05] zu finden. In [HDF12] stellen die Autoren Heinly et al. noch einmal BRISK, BRIEF und ORB – alles binäre Feature Detectors – gegenüber. Erstere Methode wird auch in Kapitel 3.1 beziehungsweise [Ulr+14b] verwendet.

1.4 Rekonstruktion (Marching Cubes)

Volumendaten können in unterschiedlichen Formen vorliegen. Sei es als Punktwolken oder jedmögliche Art von Gittern. Welche unterschiedlichsten Arten von Gitterstrukturen existieren lässt sich in der Veröffentlichung von Speray & Kennon [SK90] nachlesen. In dieser Arbeit sind die Volumendaten beziehungsweise die einzelnen Voxel der ausgerichteten Schnittserien gleichmäßig angeordnet.

Sollen diese Daten visualisiert werden, stehen dafür zwei Ansätze zur Verfügung. Eine Möglichkeit stellt das direkte Volumenrendering (DVR) dar. Bekanntere stellvertretende Methoden sind zum Beispiel das Ray-Tracing [SML88] (einschließlich der vereinfachten Form Ray-Casting), das Splatting [Wes91] oder die View-aligned Slices [EKE01]. Alternativ können Volumendaten mittels dem indirekten Volumenrendering (IVR) visualisiert werden. Hier gibt es neben der Konturverbindung [FMP77] eine weitere Methode, die sich sehr schnell durchgesetzt hat.[57] Bei dieser handelt es sich um den Marching Cubes-Algorithmus. Der Erfolg resultiert aus der einfachen und effizienten Herangehensweise, der er sich bedient: der divide-and-conquer-Strategie.

Die Methode der Marching Cubes wurde erstmals von Lorensen und Cline [LC87] vorgestellt. Sie ist darauf angelegt strukturierte Volumendatensätze – meist handelt es sich um kartesische Gitter – in Würfel zu unterteilen und diese einzeln, unabhängig und sequenziell abzuarbeiten. Hierfür werden acht Skalarwerte[58] – vier des unteren Slices[59] und vier des oberen Slices – zu einem Würfel zusammenfasst. Diese Skalarwerte werden dann mit einem vorher festgelegten Isowert verglichen und die Vergleichsergebnisse zu einem Cubecode[60] (8-bitigen Wert oder Byte) kodiert, welcher mit einer vordefinierten Mustertabelle für Oberflächendefinition, der look-up-Tabelle, abgeglichen wird.

Eine Hauptschwäche dieser Methode, das mögliche Vorkommen von Löchern innerhalb einer Iso-Oberfläche, wurde damals schnell erkannt und mehrere Verbesserungsvorschläge folgten zeitnah. Die Ursache für die Löcher lag an der von Lorensen und Cline [LC87] vorgestellten look-up-Tabelle, welche den möglichen Flächenverlauf innerhalb eines Würfel nicht vollständig berücksichtigte. Für die Isofläche gab es bei bestimmten Mustern mehr als eine Möglichkeit der Definition – es gab also ein Problem mit der Mehrdeutigkeit. Heiden, Goetze und Brickmann [HGB93], später auch Montani et al. [MSS94], ergänzten die fehlenden Flächendefinitionen durch weitere Muster. Während Chernyaev [Che95] dagegen die look-up-Tabelle gewissermaßen neu erstellte, indem er die approximierenden Flächen topologisch korrekt unter Zuhilfenahme tri-liniarer Interpolation beschrieb und definierte. Eine weitere Möglichkeit Löcher zu vermeiden stellt das Marching Tetrahedra-Verfahren

[57] Die Arbeit [Ulr08] zählt für DVR und IVR noch weitere Methoden auf. Nachzulesen sind diese auf Seite 11 in Tabelle 1.
[58] Siehe Formel (2) Seite 9.
[59] Siehe Formel (11) Seite 11.
[60] Siehe Formel (15) Seite 12.

dar. So wird in der Methode von Treece et al. [TPG99] ein Würfel in sechs Tetraeder[61] unterteilt, welche dann zur Iso-Oberfläche-Erstellung benutzt werden. Dies ist zwar eine recht geschickte Lösung, da die look-up-Tabelle für die Tetraeder, welche ja nur vier Ecken haben, wesentlich kürzer und eindeutig ist. Leider erzeugt dieses Verfahren ein Mesh[62] (Iso-Oberfläche), welches mehr Oberflächendreiecke aufweist als die Oberfläche des klassischen Marching Cubes-Algorithmus. Dies führt wiederum zu einer höheren Belastung des (Arbeits-)Speichers und des (Festplatten-)Speicherplatzes und bedeutet mehr Verwaltungsaufwand.

Neben diesen Korrekturen gab es auch Weiterentwicklungen in den Bereichen der Performanz und der Oberflächenapproximation. Verbesserungen der Laufzeit wurden beispielsweise erzielt, indem man die Volumendaten unter Verwendung eines branch-on-need-Octrees mit min-max-Selektoren [WV90] unterteilte. Oder wie im Fall der Span Spaces [She+96; Liv99], die Würfel anhand ihrer Minimum- und Maximumwerte als zweidimensional Punkte kodierte und somit eine schnelle Extraktion der Fläche durch Partitionierung oder k-d-Bäume ermöglichte. Bei der Verbesserung der Oberflächenapproximation widmete man sich zum einem den Artefakten, die beim Marching Cubes entstehen können. Zum Beispiel hat der Algorithmus Schwierigkeiten die glatten Kanten eines Legosteines zu approximieren. Es entsteht bei dem Versuch eine Art Treppeneffekt, wie man es beim Zeichnen einer glatten Linie auf dem Bildschirm beobachtet. Diese Artefakte werden entsprechend auch Aliasfehler genannt. Kobbelt et al. [Kob+01] haben diesen Approximationsfehler mit einem zusätzlichen edge-flip (Extended Marching Cubes) korrigiert. Da die Approximation des Marching Cubes spätestens dann an seine Grenzen stößt, wenn die Strukturen feiner sind als die Kantenlänge eines einzelnen MC-Würfels[63] – man stelle sich hier das Modell eines Sektglases vor, das zum oberen Rand hin immer feiner wird – wurden auch Verbesserungen entwickelt, die scharfe Merkmale besser approximieren können als der originale Marching Cubes. Am bekanntesten hierfür ist wohl der duale Marching Cubes von Schaefer und Warren [SW04]. Ihr Algorithmus legt ins bereits vorhandene Volumendatengitter[64] ein weiteres duales Gitter ein, mit dessen Hilfe das (duale) Mesh aufgebaut wird.[65] Die Oberflächenapproximation wurde zum anderen aber auch dahingehend versuchsweise optimiert, eine geringere Anzahl an Meshdreiecken zu erzeugen – und damit wiederum die Performanz zu verbessern. Hier sei exemplarisch der adaptive Ansatz von Müller und Stark [MS91] genannt, der den Volumendatensatz anhand eines Vorzeichenwechsels an den Kanten rekursiv bis zur kleinsten Einheit unterteilt und im bottom-up-Verfahren das Mesh generiert.

Es gab in jüngster Zeit auch Weiterentwicklungen, die sich mit dem Marching Cubes und der Verwendung von der GPU (Graphics Processing Unit) beziehungsweise von GPGPUs (General Purpose Computation on Graphics Processing Unit[66]) befassen. So präsentierten Reck et al. [Rec+04] einen Algorithmus, der unter Verwendung von GPGPU-Rechenleistung aus einem unstrukturierten Tretraeder-Gitter die Iso-Oberfläche extrahiert. Dabei wird eine Vorsortierung von CPU-Seite mittels Intervall-Baumes und Span Spaces vorgenommen und nur die Iso-Oberfläche schneidenden Tetraeder von der GPU bearbeitet. Tatarchuk et al. [TSD07] stellten eine Hybrid-Implementierung des klassischen

[61] Ein Tetraeder (Etymologie: griechisch *τετρα tetra* 'vier', griechisch *εδρα hédra* 'Seite, Fläche'. Singular: das Te · t · ra · eder, Plural: die Te · t · ra · eder) ist ein 3-Simplex mit vier Facetten (Seitenflächen) und vier Ecken (Punkten) und sechs Kanten. Oder umgangssprachlicher formuliert: Ein Tetraeder ist eine Pyramide, deren Grundfläche nicht viereckig sondern dreieckig ist.

[62] Siehe Formel (16) Seite 12.

[63] Siehe Seite 11, nach Formel (12).

[64] Vgl. Kapitel Mathematische Definitionen Seite 9.

[65] Interessanterweise verwendet diese Methode die Quadric Error Metric (QEM) von Lindstrom [Lin00] um die dualen Meshvertices zu setzen. Diese wird auch bei der Simplifizierung von Iso-Oberflächen (Kapitel 1.5 ab Seite 18) verwendet.

[66] engl.: Allzweck-Berechnung auf Grafikprozessoreinheit(en).

Marching Cubes und der Tetrahedra-Version vor. Hier wird die Iso-Oberfläche von der GPU erzeugt, hat aber durch die Unterteilung in Tetraeder wesentlich mehr Mesh-Dreiecke. Tatarchuk et al. greifen dabei auf das Vertex Programm von Reck et al. zurück. Johansson und Carr [JC06] verwenden ebenfalls Span Spaces, benutzen aber die GPU zur Interpolation und sind nicht mehr auf Tetraeder-Gitter beschränkt. Ähnlich verhält es sich bei einem Ansatz, welcher HistoPyramiden [Zie+06] verwendet. Bei den HistoPyramiden von Dyken et al. [Dyk+08] werden wie bei einer MipMap[67] die Skalarwerte bzw. das Skalarfeld unter Angabe eines Isowertes zu HistoPyramiden kodiert und verkleinert. Dies geschieht auf der GPU während die CPU den Anzahl der Vertices (Vertex Count) ermittelt, die für das zu erstellende Mesh benötigt werden. Das Rendering und somit die Darstellung werden mittels Vertex Count und HistoPyramiden-Textur auf GPU bewerkstelligt.

Einen grundlegenden Überblick über den Marching Cubes-Algorithmus, dessen Nach- und Vorteile sowie dessen Entwicklung, lassen sich in der Veröffentlichung von Newman und Yi [NY06] oder auch in der Diplomarbeit der Autorin [Ulr08] nachlesen.

1.5 Simplifizierung

Für eine schnelle Echtzeit-Darstellung von 3D-Modellen ist Mesh-Simplifizierung essentiell. Sie ermöglicht durch Reduktion der Dreiecke eine schnelle Rechenzeit und somit eine schnellere Darstellung. Das machte die Simplifizierung im Bezug des Real-Time Rendering in den letzten 20 Jahren zu einem aktiven Forschungsfeld. Ein detaillierter Überblick kann bei David Luebke [Lue01] erhalten werden. Da es aber in dieser Dissertation um eine effiziente und schnelle Darstellung großer Meshes und damit um deren Simplifizierung in Echtzeit geht, wird der Fokus des Überblickes auf einige Echtzeit-Lösungen liegen.

Rossignac und Borrel [RB93] stellten eine Simplifizierung vor, die die Bounding Box[68] des 3D-Modells in kleinere Unterboxen in Form eines regulären Gitters (vgl. Kapitel 1.2 Seite 9) unterteilt. Die in den Unterboxen enthaltenen Vertices werden 'gemittelt' und zu einem einzelnen Vertex zusammengefasst. Low und Tan [LT97] entwickelten diesen Ansatz weiter. In ihrer Methode werden die Vertices mit Gewichten bezüglich der nötigen Sichtbarkeit versehen und die Unterteilung in Unterboxen wird anhand dieser Gewichtungen ausgemacht. Diese Art der Simplifizierung ist recht schnell, hat aber eine Schwäche bei den größeren flachen Bereichen eines Meshes.

Die bekannteste Methode zur Simplifizierung, die *Vertex Pair Contraction*, wurde von Garland und Heckbert [GH97] beziehungsweise Popovic und Hoppe [PH97] entwickelt. Hier werden zwei Vertices, die verbunden sind durch eine Kante oder sehr nahe beieinander liegen, ersetzt durch einen neuen Vertex. Die Operation im ersten Fall wird auch *edge-collapse* genannt. Falls die beiden Vertices nicht mit einander verbunden sind, nennt man es *vertex contraction*. Die Position des neuen Vertex wird wiederum mit Hilfe einer Fehlerquadrik (*Quadric Error Metric* (QEM)) bestimmt. Bei dieser Verfahrensweise werden die angrenzenden Dreiecke eines Vertex als Teilstücke von Ebenen aufgefasst. Mit Hilfe der so gewonnen Ebenengleichungen, die quadriert und aufsummiert werden, wird ein dreidimensionales Skalarfeld gebildet. Letzteres repräsentiert den quadratischen Abstand (Quadric Error) eines dreidimensionalen Punktes zu den Ebenen.[69] Soll ein edge-collapse durchgeführt werden, so berechnet man die ideale Position des neuen Vertex indem man die Skalarfelder der beiden verbunden Vertices addiert und den Fehlerwert der Postion des neuen Vertex entsprechend unterhalb einer

[67] Dabei handelt es sich um eine AntiAlias-Technik für Texturen. Siehe [Wikh].
[68] Eine Bounding Box ist ein Quader, der das 3D-Modell minimal umschließt.
[69] Lässt man sich die Fehlerwerte als Isofläche anzeigen, so erhält in der Regel Ellipsoide.

vordefinierten Fehlertolerenzschwelle hält. Die *Quadric Error Metric*-Methode erlaubt zudem eine flexible Anpassung der Parameter.[70]

In [GH98] erweitern Garland und Heckbert dieses Verfahren mit der Möglichkeit Vertex-Attribute[71] zu behandeln und führen eine *contraint plane* ein, um Ränder eines 3D-Modells erhalten zu können. Peter Lindstrom [Lin00] kombiniert dagegen das zuvor erwähnte *Vertex Clustering* (von [RB93; LT97]) mit den Fehlerquadriken. Dieser Lösungsansatz eignet sich auch für große 3D-Modelle. Allerdings lässt diese Methode ebenfalls bei großen flachen Bereichen im Mesh noch viele Dreiecke zurück. Shaffer und Garland [SG01] verfolgten den adaptiven Ansatz von Low und Tan [LT97]. Sie verwenden anstatt *floating-cell clustering* eine binäre Raumaufteilung, auch BSP-Baum oder BSP-Tree genannt. Schaefer und Warren [SW03] benutzen dagegen einen Octree. Die letzten beiden Methoden liegen in ihrer Laufzeit zwischen der klassischen Simplifizierung mittels edge-collapse und der Vertex Clustering.

Auch bei der Simplifizierung gab es, wie beim Marching Cubes, in den letzten Jahren Weiterentwicklungen, die die Verwendung GPU mit einbezogen. DeCoro und Tatarchuk [DT07] beispielsweise implementierten eine parallelisierte Variante von Schaefer und Warren [SW03]. Sie benutzen einen *probabilistic octree* als passende GPU-Datenstruktur. Die Laufzeit ist recht hoch, aber das Verfahren liefert bezüglich des Meshes eine gleiche Qualität wie beim Vertex Clustering. Eine Methode die mit Parallelisierung arbeitet und GPU beziehungsweise GPGPU verwendet, ist die Methode von Grund et al. [GDG11]. Sie wird in der im Kapitel 3.2 vorgestellten Methode angewandt.

1.6 CUDA

1.6.1 Historie und Hardware

Auf den Webseiten des Grafikprozessoren-Entwicklers Nvidia [Nvia; Nvic] wird CUDA als eine Plattform für parallele Berechnungen und als Programmier-Modell beschrieben. Ursprünglich wurde CUDA als Akronym für *Compute Unified Device Architecture* eingeführt. Der Begriff *Unified* stellt dabei das Novum der damals im Jahre 2007 vorstellten Grafikprozessoren auf der Grafikkarte GeForce 8800 GTX [SW06] dar.

Zuvor musste ein 3D-Modell, sollte es für die Darstellung auf dem Computerbildschirm aufbereitet werden, eine starre Pipeline auf der Grafikkarte durchlaufen. Der Computer-Prozessor (CPU) steuerte die Grafikkarte über die *Host*[72]-Schnittstelle an und übermittelte die fest definierten Befehle und die Daten (des 3D-Modells). Die Daten wurden dann schrittweise weiterverarbeitet. Sie wurden zuerst für die Hardware aufbereitet und mit Attributen versehen. Dann wurden bezüglich der Meshdreiecke Gleichungen erstellt und die Bildschirmpixel entsprechend zugeordnet. Danach folgten Shader- und Raster-Operationen. Das heißt, den Pixeln wurden beispielsweise die endgültige Farbe zugeteilt oder ein AntiAliasing vorgenommen.[73] Im letzten Schritt erfolgt die Übergabe an die Bildspeicher-schnittstelle (Frame buffer interface) damit das Resultat am Computerbildschirm angezeigt werden kann. Zwar gab es vor dem Jahre 2007 schon ein Aufbrechen der starren Grafik-Pipeline, da einige Schritte bereits programmierbar waren, aber die Daten mussten noch umständlich als Texturen-Bilder aufbereitet werden.[74]

[70] Gemeint ist zum Beispiel die Fehlertoleranz.
[71] Dabei kann es sich um Informationen bezüglich der Textur, der Farbe oder um Normalenvektoren handeln.
[72] Mit *Host* ist der Computer selbst beziehungsweise die CPU gemeint.
[73] Vgl. [KH10] Kapitel 2.1.1 Seite 22ff.
[74] Vgl. [KH10] Kapitel 2.1.4 Seite 31f.

Mit der GeForce 8800 GTX wurde die Datenverarbeitung auf der Grafikkarte, dem *General-Purpose Programming on Graphics Processing Unit* (GPGPU), wesentlich einfacher. Grafikkarten mit Nvidia-Hardware stellten nun pro Karte mehrere parallel arbeitende *vereinheitlichte* Prozessoren zur Verfügung, welche *allgemeine* (numerische) Rechnungen ausführten und die IEEE Anforderungen für Fließkommazahlen-Arithmetik erfüllten.[75] Von da an konnten die Daten des 3D-Modells beliebig verarbeitet werden. Die Pipeline bestand nun darin, dass die Daten diese Grafikprozessoren mehrmals durchliefen und dazwischen die entsprechende Operationen (beispielsweise Vertex- oder Shader-Operationen) stattfanden.[76]

Ein Hauptunterschied zwischen CPU und CUDA-fähigen Grafikprozessoren ist wahrscheinlich die Größe des Cache. Dieser ist bei einer CPU pro Rechenkern wesentlich größer, da dessen Konstruktion für die Ausführung von sequentiellen Programmen optimiert wurde. Der Cache beherbergt die vielen unterschiedlichen Befehle, die nacheinander ausgeführt werden. Anders verhält es sich bei einem CUDA-fähigen Grafikprozessor. Er hat einen wesentlich kleineren Cache. Die Befehle behandeln hauptsächlich Fließkommazahlen-Operationen und die Recheneinheiten (*cores*) sind entsprechend für diese Operationen angepasst. Als Ausgleich können Daten von und zur Grafikkarte schneller transferiert werden. Dies ist auch nötig für die Bildwiedergabe.[77] Folglich eignen sich die CPUs für komplexe Programme (oder Programmabschnitte), die unterschiedliche Arbeitsschritte und wenige Daten beinhalten. Die CUDA-fähigen Grafikprozessoren sind dagegen dienlich für alle Aufgaben, in denen große Mengen an Daten verarbeitet werden und diese Daten zudem *unabhängig* voneinander und somit parallelisierbar sind.[78]

1.6.2 Heterogene Programmierung

CUDA oder genauer gesagt CUDA C (*CUDA C extends C*) ist kompatibel mit C++ und 64-Bit-Betriebssystemen wie Linux oder Windows.[79] Um voneinander unabhängige Daten parallel und optimal verarbeiten zu können, sind Kenntnisse über die Indexierung (Adressierung) der parallel arbeitenden Prozesse, die Organisation des Speichers und die Deklarierung von Vorteil. Daher soll in den folgenden Abschnitten darauf eingegangen werden.

Indexierung Für die Parallelisierung werden die Daten auf *Threads* (Fäden) verteilt. Die Threads werden zu *Blocks* (Blöcken) zusammengefasst, welche wiederum zu *Grids* (Gittern) zusammengefasst werden. Blocks können ein-, zwei- oder dreidimensional sein. Ebenso die Grids. Die Dimension eines Grids oder Blocks wird mit `dim3` angegeben und unterliegt gewissen Vorgaben. So kann ein Block nur maximal 1024 Threads – früher waren es 512[80] – zusammenfassen. Ein Grid konnte früher nur zweidimensional sein. Heute ist dagegen auch eine dreidimensionale Anordnung der Blocks möglich.[81] Weitere Vorgaben sind dem Programming Guide [Nvi18] von Nvidia unter Anhang H zu entnehmen.

[75] Gemeint sind die Anforderungen IEEE 754-1985 [85] und IEEE 754-2008 [08]. Diese definieren die Darstellung von Fließkommazahlen mit Binärzahlen. Vgl. auch [KH10] Kapitel 7.
[76] Vgl. [KH10] Kapitel 2.1.3 Seite 29 Absatz 1.
[77] Vgl. [KH10] Kapitel 1.1. Seite 4 und 5.
[78] Vgl. [KH10] Kapitel 1.3 Seite 10ff.
[79] Vgl. [Nvi18] Kapitel 3 (Programming Interface).
[80] Siehe [SK10] Kapitel 5.2.1 Abschnitt 'GPU SUMS OF A LONGER VECTOR' Seite 63.
[81] Dies hat sich seit der CUDA-Version 3.0 (Compute capability) geändert.

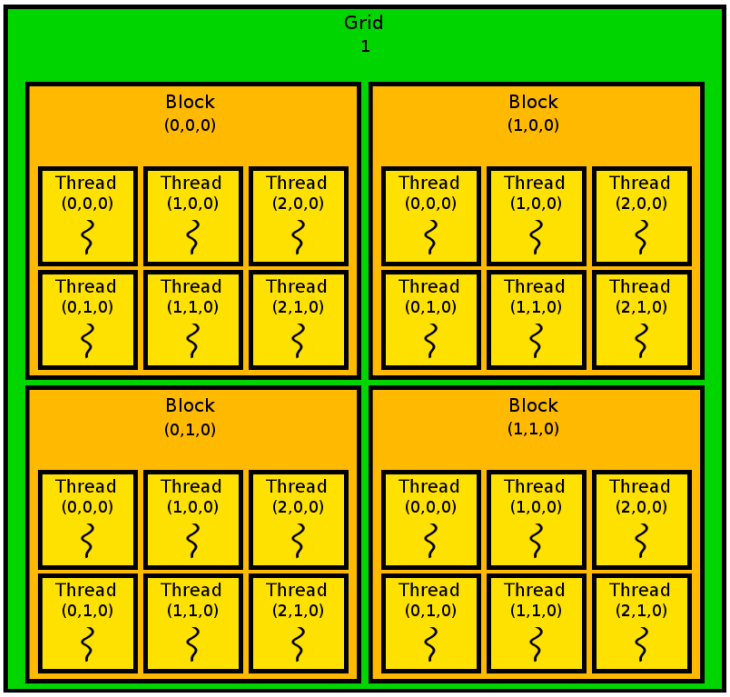


Abbildung 3: Indexierung der Threads und der Blocks. Es gilt: `gridDim = dim3(2,2,1)` und `blockDim = dim3(3,2,1)`.

Dort lässt sich beispielsweise auch nachlesen, dass die y-Dimension eines Gitters auf maximal 65535 Blocks begrenzt ist.[82]

Bei `dim3` handelt es um einen `uint3`-Vektor-Datentyp.[83] Wurden mit diesem die Dimensionen für die Grids und Blocks angelegt, können sie über die Build-In-Vektor-Datentypen `gridDim` und `blockDim` abgerufen werden. Hier stehen die entsprechenden Felder `x`, `y` und `z` zur Verfügung. Für y-Dimension eines Gitters muss somit folgender Ausdruck wahr sein:

```
gridDim.y < 65535+1.
```
(23)

In Abbildung 3 ist Organisation von Threads, Blocks und Grids schematisch dargestellt. Hier sind das Grid und die Blocks zu Anschauungszwecken zweidimensional gehalten. Die Threads als einzelne Fäden lassen sich über die dreidimensionale Indexangabe `threadIdx` ansprechen. Für die Blocks gibt es den reservierten Bezeichner `blockIdx`. Beide Bezeichner sind ebenfalls ein `uint3`-Vektor-Datentyp und können wieder mit den Feldern `x`, `y` und `z` verwendet werden. `threadIdx.x` kann einen Wert zwischen `0` und `blockDim.x-1` annehmen. Die selben Bedingungen gelten für `threadIdx.y`, `threadIdx.z`, `blockIdx.x`, `blockIdx.y` und `blockIdx.z`.

Oftmals ist es zweckmäßig Threads durchzunummerieren. So lassen sich parallel Daten eines eindimensionalen Arrays einlesen, bearbeiten und wieder in das Array schreiben, ohne dass die Threads sich per *race condition*[84] im Wege stehen. Eine Durchnummerierung könnte dabei wie folgt aussehen.

[82] Das Blocklimit wird auch in [SK10] Kapitel 5.2.1 Abschnitt 'GPU SUMS OF A LONGER VECTOR' Seite 63 erwähnt.

[83] Der erste Buchstabe von `uint` steht für *unsigned*. Das `int` steht für das englische Wort *integer*. Bei `uint3` handelt es sich also um einen Vektor, der aus drei ganzen Zahlen ohne Vorzeichen besteht. Das heißt, der Vektor besteht nur aus positiven Werten.

[84] Der Begriff *race condition* beschreibt im Falle der Threads eine Konkurrenzsituation, in der mehrere Threads

Die ersten drei Threads aus Block (0,0,0) erhalten die Indizes 0, 1 und 2. Die Threads in der oberen Reihe aus Block (1,0,0) erhalten die Werte 3, 4 und 5. Die nächsten drei Threads in der unteren Reihe aus Block (0,0,0) erhalten wieder die Nummern 0, 1 und 2. Und so weiter. Umsetzen lässt sich dies mit Formel (24). Für eine spaltenweise (anstatt zeilenweise) Durchnummerierung müsste diese entsprechend angepasst werden.[85] [86]

```
int nummer = threadIdx.x + blockIdx.x * blockDim.x
```
(24)

Diese Formel kann später dazu verwendet werden, um die Formel (7) (Seite 9) zu implementieren.

Deklarationen für Funktionen Die Bezeichner `gridDim`, `blockDim`, `blockIdx` und `threadIdx` stehen nur zur Verfügung, wenn der Programmcode auf der Grafikkarte ausgeführt wird. Um festzulegen, wo welcher Programmcode ausgeführt wird und wo welche Daten gespeichert werden, müssen die von CUDA C bereitgestellten Deklarationen verwendet werden. In CUDA wird zwischen dem Prozess, der von der CPU ausgeführt wird, und den Prozessen, die auf der Grafikkarte ausgeführt werden, unterschieden. Im ersteren Fall spricht man vom *Host*[87], im letzteren von *Device*[88]. Funktionen, die auf der Grafikkarte (dem Device) ausgeführt werden, bezeichnet man auch als *Kernels*.[89] Diese werden entweder mit `__device__` oder mit `__global__` deklariert. Eine `__device__`-Funktion kann nur von einem Device-Prozess aufgerufen werden. Eine Deklarierung mit `__global__` ermöglicht dagegen sowohl ein Aufruf durch einem Device- als auch von einem Host-Prozess. Sollen Funktionen vom Host ausgeführt werden, stellt man vor den Funktionsnamen ein `__host__`. Diese Angabe ist auch als Default-Wert definiert. Das heißt, wird keine Deklaration angegeben, wird angenommen, dass `__host__` gemeint ist.[90]

Da das CUDA-Programm vom Host aus gestartet wird, müssen auch die Kernels entsprechend vom Host aufgerufen werden. Dies geschieht mit drei spitzen Klammern rechts und links um die Dimensionsangaben von Grid und Blocks. Wenn also der Name des Kernel `NameDesKernels` ist und die Gitter und Blockgröße, wie in Abbildung 3 veranschaulicht, definiert wurden, dann sieht ein Kernelaufruf (ohne Parameterübergabe) wie folgt aus:[91]

```
NameDesKernels<<<gridDim,blockDim>>>();
```
(25)

Organisation des Speichers Bevor jedoch ein Kernel aufgerufen werden kann, müssen zuerst die Daten, die es zu bearbeiten gilt, auf die Grafikkarte transferiert werden. Dafür wird mittels `cudaMalloc` der benötigte Speicherplatz in Bytes auf der Grafikkarte reserviert und mit dem Befehl `cudaMemcpy` die

versuchen einen Speicherplatz im Array als Erster zu lesen beziehungsweise zu schreiben. Dies kann bei mehrmaligen Ausführen eines Code zu jeweils unterschiedlichen Ergebnissen führen, da die Reihenfolge der Threads beim Lesen und Schreiben nie eindeutig ist.

[85] Eine ähnliche Durchnummerierung ist in Kapitel 2.2 des Programming Guide [Nvi18] zu finden.
[86] Siehe [SK10] Kapitel 5.2.1 Abschnitt 'GPU SUMS OF A LONGER VECTOR' Seite 63.
[87] Etymologie: englisch *host* 'Gast'.
[88] Etymologie: englisch *device* 'Gerät'.
[89] Vgl. [SK10] Kapitel 3.2.2 Seite 23f.
[90] Vgl. [KH10] Kapitel 3.5 Seite 51ff und [Nvi18] Anhang B.2 'Variable Memory Space Specifiers'.
[91] Vgl. [Nvi18] Kapitel 2.1, [KH10] Kapitel 3.6.2 Seite 56. und [SK10] Kapitel 3.2.2 Seite 23f.

Daten vom Host zum Device kopiert.[92] Durch das Kopieren gelangen die Daten in den *global memory*. Dieser ist von allen Threads (in allen Blocks und Grids) und vom Host schreib- und lesbar.

Der *global memory* ist nicht der einzige Speicher in der Speicher-Hierarchie einer CUDA-fähigen Grafikkarte. Im CUDA Programming Guide [Nvi18] – unter anderem im Kapitel 5.3.2. 'Device Memory Accesses' – werden weitere Arten aufgeführt. Es sollen hier jedoch nur die wichtigsten Speicher-Arten kurz vorgestellt werden.

Neben dem globalen Speicher (*global memory*) gibt es noch den konstanten und den gemeinsamen Speicher. Der *constant memory* kann von Threads nur gelesen werden. Der Host hat dagegen Lese- und Schreibrechte.[93] Anstatt des `cudaMemcpy`-Befehls bedarf es einer `cudaMemcpyToSymbol`-Anweisung, um die Daten auf den konstanten Speicher hochzuladen.[94] Pro Block existiert zudem ein *shared memory*. Alle Threads des jeweiligen Blocks können auf den jeweiligen gemeinsamen Speicher lesen und schreiben.[95] Der Host hat darauf keinen Zugriff. Weiterhin besitzt jeder Thread seinen eigenen Register-Speicher, der nur von diesem schreib- und lesbar ist. Von den eben beschriebenen Speicher-Arten hat der *global memory* die langsamsten Zugriffszeiten und die Register die schnellsten.

Die Grafikkarten-Hardware setzt bei der Gestaltung des Programmcodes und bei der Reservierung des Speichers Grenzen. So ist bei der Indexierung, wie im [KH10] Kapitel 4.4 veranschaulicht, der Organisation durch die Anzahl der Streaming-Prozessoren (SP)[96] Grenzen gesetzt. Threads können beispielsweise in der GeForce GTX 1080 auf maximal 20 Streaming-Mehrprozessoren (SM) verteilt werden, die jeweils mit 128 SPs bestückt sind. Bei der GeForce GTX 680 sind es maximal acht SMs mit jeweils 192 SPs und bei der GeForce GTX 580 sind es maximal 16 SMs mit jeweils 32 SPs. Auch beim Speicher und dessen Aufnahmefähigkeit gibt es von Grafikkarte zu Grafikkarte Unterschiede. Der globale Speicher beträgt bei der GeForce GTX 1080 8192 MB, bei der GeForce GTX 680 2048 MB und bei der GeForce GTX 580 nur 1538 MB.[97]

Deklarationen für Variablen Wie bereits beschrieben, wird mit Hilfe der Deklarationen für Funktionen festlegt, wo welcher Code ausgeführt wird. Ähnliche Deklarationen existieren auch für Variablen. Mit diesen wird angegeben, in welcher Speicher-Art sie untergebracht werden sollen. Auch hier gibt es wieder die Deklaration `__device__`. Eine derartig deklarierte Variable verbleibt im Grafikkartenspeicher. Genauer gesagt im *global memory*. Die Deklaration `__constant__` muss außerhalb einer Funktion benutzt werden. Sie ist für Variablen bestimmt, die im *constant memory* untergebracht werden. Dann gibt es noch den Bezeichner `__shared__`. Er kennzeichnet Variablen, die im *shared memory* verortet werden. Wird kein Bezeichner verwendet, bedeutet dies, dass die Variable im Register gespeichert wird. Es kann jedoch sein, dass sie in den lokalen Speicher ausgelagert wird.[98] Beim *local memory* handelt es sich um Grafikkartenspeicher. Er wird nur von Threads verwendet und

[92] Vgl. [KH10] Kapitel 3.4 Seite 46ff, [Nvi18] Kapitel 3.2.2 und [SK10] Kapitel 3.2.3 Seite 24ff.
[93] Vgl. [Nvi18] Anhang D.3.1.6.4. 'Symbol Addresses', Anhang D.2.2.1.3. 'Constant Memory' und [KH10] Kapitel 5.2 Absatz 1 Seite 79.
[94] Vgl. [Nvi18] Kapitel 3.2.2. 'Device Memory'.
[95] Ein praktisches Beispiel kann in [SK10] Kapitel 5.3 Seite 75ff nachgelesen werden.
[96] Diese werden mittlerweile auch CUDA Cores genannt.
[97] Die GeForce GTX 1080 kann noch eine interessante Neuerung aufweisen. Sie verfügt über die Möglichkeit einen *unified memory* zu benutzen. Dieser kann vom Device und vom Host benutzt werden. Dies reduziert die Verwendung von zwei Pointer-Variablen auf eine. Die Reservierung des Speichers erfolgt hier mit dem Befehl `cudaMallocManaged()`. Weiteres ist unter [Nvi18] Abschnitt K nachzulesen.
[98] Bei Array-Variablen ist die Auslagerung in den lokalen Speicher der Regelfall.

dann beansprucht, wenn beispielsweise für eine (automatische) Variable[99] die Speicherkapazität des Registers nicht mehr ausreicht. Dieser ist bezüglich der Reaktionszeit und Datenübertragungsrate dem globalen Speicher gleich.[100] [101] [102]

Es gibt noch weitere hier nicht vorgestellte Deklarationen für Funktionen und Variablen. Diese wurden jedoch ausgespart, da nur ein kurzer Überblick gegeben werden soll.

1.6.3 Andere Schnittstellen/Plattformen

Die Nachfrage nach schnellerer und damit nach parallelisierten Anwendungen war und ist stets vorhanden. So gab und gibt es in den Bereichen der Medizin (bildgebende Verfahren, wie CT oder MRT), der wissenschaftliche Simulation (Wetter oder Tsunami), der Mensch-Maschine-Interaktion (Displays von Mobiltelefonen) und der Computerspielentwicklung ein Interesse an schneller Datenverarbeitung.[103]

Daher gab es schon vor CUDA Entwicklungen auf dem Gebiet der Parallelisierung. Das Message Passing Interface (MPI) [Mes15] ist ein solcher Ansatz. Es fasst Computer zu einem Cluster zusammen und führt so parallele Berechnungen aus. Solch ein Cluster verfügt aber nicht über einen gemeinsamen Speicher. Die Daten und Befehle müssen zwischen den Computern gesendet werden. Ein anderer Vorgänger ist die Programmierschnittstelle Open Multi-Processing (OpenMP) [Ope15; DM98]. Sie verfügt über einen gemeinsamen Speicher, ist aber auf Multiprozessor-Systeme beschränkt und somit nicht beliebig skalierbar.[104] Ein weiterer Ansatz ist OpenLava, welches früher LSF-HPC (Platform Load Sharing Facility - High Performance Computing) [IBM; Wiki] hieß. Mit diesem ist es möglich Hochleistungsrechner (mit Windows- oder UNIX-Betriebssystem) zu einem Cluster zu verbinden und deren Arbeitslast in Form von kleinen Batch-Jobs durch eine Zeitablaufsteuerung (job scheduler) zu organisieren. Auch hier fehlt der gemeinsame Speicher.

Der Vorteil von CUDA und des GPGPU liegt in deren leichten Zugänglichkeit. Gab es bisher bei Cluster und Multiprozessor-Systeme die Problematik, dass sie zu platzeinnehmend oder zu teuer waren, sind CUDA-fähige Grafikkarten mittlerweile recht erschwinglich geworden. Zudem sind Sie in der Größe kompakt und können mehrere tausende von Threads gleichzeitig starten.

Dennoch kann die Hardware, wie weiter oben schon beschrieben, ein Hindernis sein. Mit CUDA implementierte Programme laufen nur auf Rechner, die mit CUDA-fähigen Grafikkarten bestückt sind. Eine durch CUDA verbesserte Anwendung ist deswegen schwer auf andere Systeme übertragbar. Aus diesem Grunde wurde die Schnittstelle Open Computing Language (OpenCL) [Khr18] entwickelt. Diese ermöglicht es mit heterogenen Rechnern (zum Beispiel CUDA-Grafikkarten, Servern oder auch Handys) und mit parallel arbeitenden Recheneinheiten, wie zum Beispiel Multikern-CPUs, GPUs, DSP[105] oder auch FPGAs[106], übergreifend zu arbeiten. Dabei werden bekannte Konstrukte aus CUDA verwendet. So gibt es bei OpenCL ebenfalls einen Host und Kernels. Die Speicher-Organisation kennt

[99] Mit dem Begriff *automatische Variable* ist eine automatische Reservierung und Freigabe des Speichers für diese Variable während ihrer Verwendung im Programmablauf gemeint.
[100] Vgl. [Nvi18] Kapitel 5.3.2 Abschnitt 'Local Memory'.
[101] Vgl. [Nvi18] Abschnitt B.2. Variable Memory Space Specifiers.
[102] Variablen, die für den *unified memory* bestimmt sind, werden mit `__managed__` und `__device__` deklariert.
[103] Vgl. [KH10] Kapitel 1.3 Seite 10f.
[104] Vgl. [KH10] Kapitel 1.4 Seite 13ff.
[105] DSP steht für *Digital Signal Processor*. Diese Prozessoren sind für die Verarbeitung digitaler Signale ausgelegt und beispielsweise in Handys zu finden.
[106] Das Akronym steht für *Field Programmable Gate Array*.

auch die Arten *global* und *constant* und *shared memory*[107]. Die Threads heißen in OpenCL *work items*, welche in *work group* gruppiert werden. Erstere werden von einem *NDRange* global indexiert.[108]

1.7 Pipeline

Um aus einer immunhistologischen Probe ein computergrafisches 3D-Modell zu erstellen, sind eine Reihe von Schritten nötig. Wie in Abbildung 4 zu sehen ist, muss die Probe zuerst aufbereitet werden. Wie dies geschieht, ist zwar nicht Gegenstand dieser Dissertation, wurde aber in Kapitel 1.1 dennoch kurz umrissen. Sind im Schritt 'Histologische Vorbereitung' die Glasobjektträger fertiggestellt worden, werden sie digitalisiert und im darauffolgenden Schritt 'Vorverarbeitung der Bilder' gegebenenfalls bezüglich Helligkeitsunterschiede, Kontrast, Farbkanäle, und ähnlichem angepasst.

Die Ausrichtung selbst erfolgt mit einer eigens entwickelten Methode. Wie aus dem vorherigen Kapitel 1.3 ersichtlich ist, gibt es bereits mehrere Ansätze zur Registrierung und Ausrichtung von medizinischen Daten. Doch keine kann den typischen Verzerrungen, die die Schnitte nach der Präparierung aufweisen, gerecht werden. Daher wird in Kapitel 3.1 eine Methode vorgestellt, die mit Bi-Cubic-B-Splines arbeitet und die bei den medizinischen Daten, die noch im Kapitel 2 vorgestellt werden, sehr gute Ergebnisse bei der Entzerrung und Ausrichtung liefert.

Während diesem oder im nachfolgenden Schritt 'Weiterverarbeitung der Bilder' können die Bilder gegebenenfalls auf eine Region von Interesse – einer Region of Interest (ROI) – reduziert werden.

Nach der erfolgreichen Ausrichtung werden die Bilder ebenfalls weiterverarbeitet. Dabei erfolgt typischerweise eine Konvertierung nach Grau und eine Umwandelung in eine RAW-Datei. Aber auch in diesem Schritt können Farbkanäle, Schwellenwertbilder, binäre Bildoperation und ähnliches zum Einsatz kommen. Im eben genannten Dateiformat wird im Schritt 'Rekonstruktion (ggf. Simplifizierung)' mit Hilfe einer weiteren eigenen Entwicklung – einem Hybrid-Algorithmus, der aus einen Marching Cubes- und Simplifizierungs-Modul (Kapitel 1.4 und 1.5) besteht – ein Iso-Oberflächen-Modell erstellt. Auf den Hybrid-Algorithmus wird in Kapitel 3.2 genauer eingegangen.

Ist ein 3D-Modell erstellt, kann es zu einer 'Nachbearbeitung des Meshes' kommen. Dies kann eine Glättung der Oberfläche, das Einfärben bestimmter Teilobjekte, das Schließen von Löchern *am Rand* des Modell, und ähnliches sein. Das Endresultat wird im Anschluss mit den Medizinern begutachtet und diskutiert. Im Beispiel der Histologie verifiziert der Histologe sein imaginäres 3D-Modell mit dem der Computergrafik. Es erfolgt im besten Fall eine erfolgreiche Verifikation. Dies ist allerdings nach der ersten Rekonstruktion nie der Fall. Oftmals müssen die Parameter für Ausrichtung und Iso-Oberflächen-Erstellung angepasst werden. Seltener sind die Rückschlaufen zu den Schritten 'Digitalsierung' oder 'Vorverarbeitung der Bilder'. Diese treten zum Beispiel in Erscheinung, wenn ein geeigneterer Scanner gesucht wird. Bis der letzte Schritt erreicht wird und die endgültigen Befunde in Form von Bildern und Video erstellt werden, kann es zu mehreren Schleifendurchläufe kommen.

[107] Dieser heißt im OpenCL allerdings *local memory*.
[108] Vgl. [KH10] Kapitel 11.

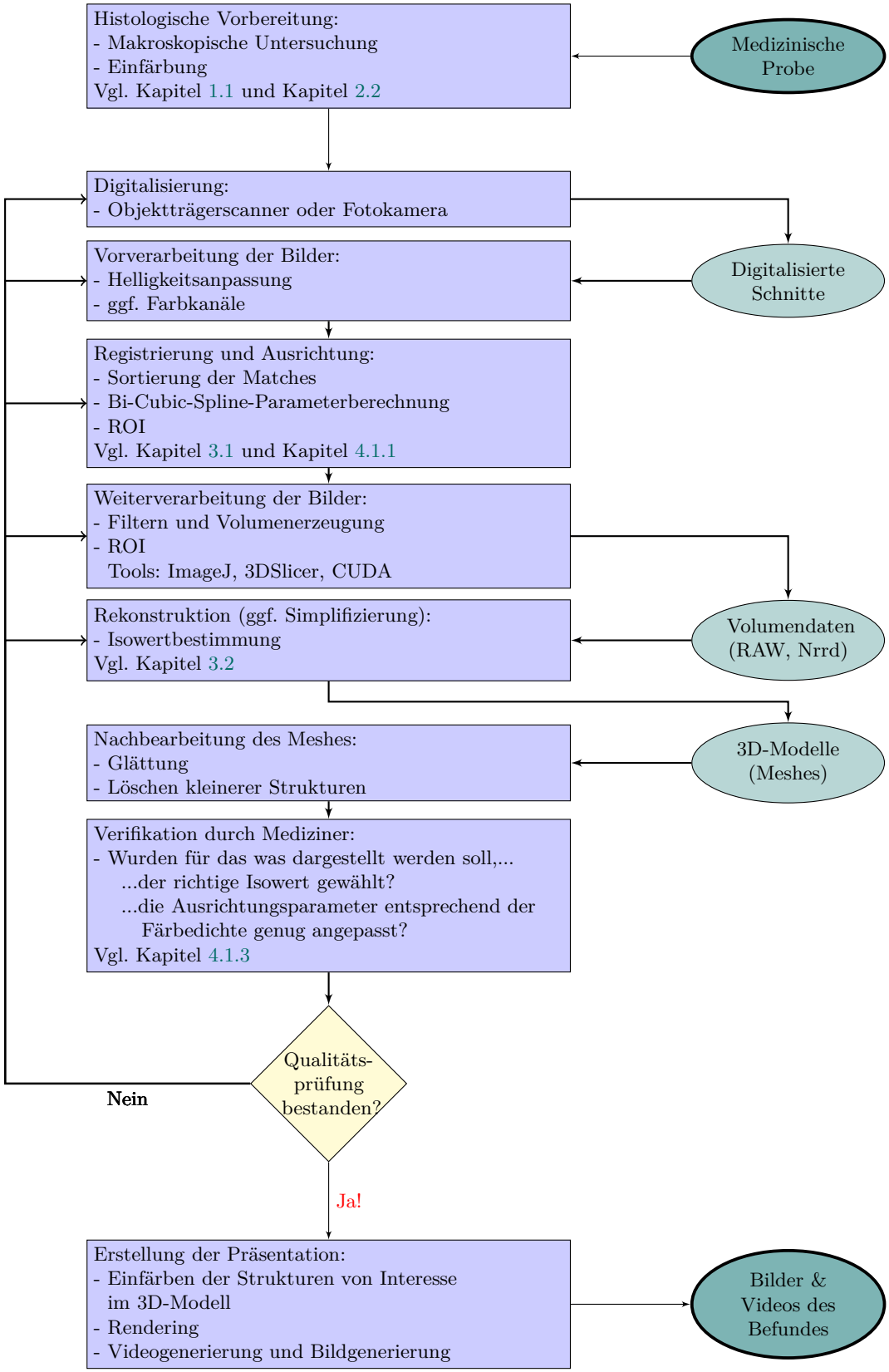


Abbildung 4: Die vollständige Pipeline zur Befundung histologischer Serienschnitte. Im Schritt 'Verifikation durch Mediziner' überprüfen die Mediziner, ob die Strukturen, die dargestellt werden sollen, auch vollständig sind. Dies kann, wie in [Wiß] nachzulesen, auch zu neuen Erkenntnissen führen.

2 Medizinische Daten

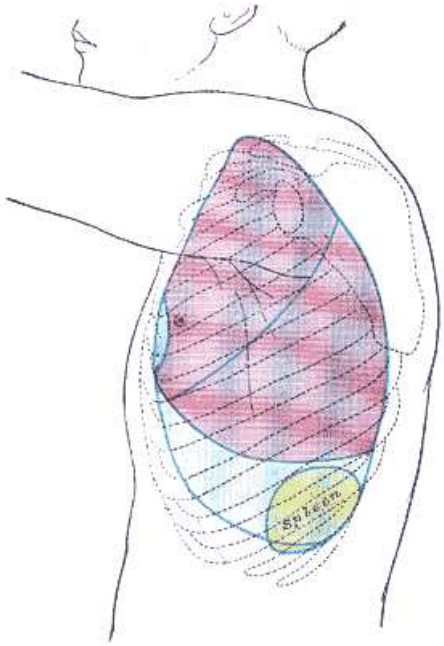


Abbildung 5: Die Lage der Milz. Zu sehen ist die Lunge (rosa) und die Milz (gelb). Blau eingezeichnet ist das Brustfell. Ebenfalls zu erkennen, die Rippen, die mit gestrichelten Linien eingezeichnet sind.

2.1 Anatomie der Milz im Überblick

Die Milz liegt in der linken oberen Ecke der Bauchhöhle in der Höhe der neunten bis elften Rippe (siehe Abbildung 5).[109] [110] Bei ihrer durchschnittlichen Größe wird auf die '4711-Regel' verwiesen und kann mit etwa $4 \times 7 \times 11$ Zentimeter angegeben werden. Das Gewicht liegt bei ungefähr 150-200 Gramm.[111] Die Milz ist das größte Organ innerhalb des lymphatischen Systems.[112] Während Knochenmark oder Thymus zu den primären lymphatischen Organen gehören, ist die Milz den sekundären, wie Lymphknoten, Mandeln und anderen, zuzuordnen.[113]

Eine Besonderheit der Milz ist, dass sie als lymphatisches Organ in den Blutkreislauf des menschlichen Körpers eingeschaltet ist.[114] Interessanterweise, bildet hier der Blutkreislauf nicht, wie beispielsweise

[109] Vgl. [Sta08] Kapitel 71 Absatz 1 Seite 1191, [LP12] Kapitel 13.4 Absatz 1 Seite 333 und [Wen12] Kapitel 7.10 Absatz 1 Seite 172.

[110] Die Lage der Milz innerhalb der Bauchhöhle ist auf DocCheck [Fleb] sehr anschaulich in einer interaktiven 3D-Grafik dargestellt.

[111] Vgl. [Dre09] Kapitel 7.2 ('Lymph. Gewebe und Immunsystem') Absatz 2 Seite 54.

[112] Genauere Beschreibungen zum lymphatischen System können unter [LP12] Kapitel 13 Absatz 1 Seite 304 nachgelesen werden.

[113] Vgl. [LP12] Kapitel 13 Kasten Seite 304.

[114] Vgl. [LF98] Kapitel 6.3.2 Absatz 2 Seite 127, [Dre09] Kapitel 7.2 Absatz 1 Seite 54, [LP12] Kapitel 13.4 Kasten

bei der Lunge, ein kontinuierliches Kapillarnetzwerk[115]. Stattdessen existieren Bereiche innerhalb der Milz in der eine offene Zirkulation stattfindet.[116] Das heißt, die Kapillaren und Sinus, die das Ende der arteriellen Verbindung beziehungsweise den Anfang des venösen Blutgefäßsystems darstellen, sind nicht miteinander verbunden.[117] [118] Da sich zusätzlich die Gefäße von den Arterien[119] bis hin zu den Pinselarteriolen[120] stark verzweigen und die Blutzellen[121], sowie das Blutplasma[122], ihren Weg durch retikuläres[123] Bindegewebe[124] bahnen müssen und nicht mehr von Endothelzellen sondern von Fibroblasten[125] begrenzt werden, fließen diese Flüssigkeiten innerhalb des Organs recht langsam.[126] Dies ermöglicht der Milz als Blutfilter zu fungieren und in zwei Bereichen mitzuwirken. Zum einem hilft sie dem Körper bei der Immunabwehr. Zum anderem findet in ihr die Zellmauserung statt.[127]

Die Milz beherbergt damit auf makroskopischer Ebene zwei Arten von Reaktionsräumen (auch Kompartimente genannt):[128] die *rote Pulpa* und die *weiße Pulpa*.

Die rote Pulpa besteht überwiegend aus verschiedenen Arten von Fibroblasten, speziellen Makrophagen, Mastzellen[129] und Plasmazellen[130].[131] Sie stellt den Großteil der Milz dar. Entlang größere Pulpagefäße kommt es zu Ansammlungen von weniger wandernden Lymphozyten, welche zumeist B- und T-Lymphozyten sind. Hierdurch entstehen in der Nähe von Verzweigungen der Milzarterie (auch Zentralarterien genannt) Reaktionsräume, die als weiße Pulpa bezeichnet werden. Diese lassen sich weiter unterteilen in Bereiche, die man als periarterielle[132] Lymphozytenscheiden (PALS) bezeichnet. Diese Bereiche sind von begrenzter Länge und beherbergen eine Ansammlung von

Seite 332 und [Wen12] Kapitel 7.10 Absatz 1 Seite 172.

[115] Blutkapillaren (Etymologie: lateinisch *capillaris* 'zum Haar gehörend') sind unter anderem aus Endothelzellen aufgebaut und stellen die kleinste Verzweigungseinheit der Arterien und Venen dar. Sie haben somit den kleinsten Durchmesser unter den Blutgefäßen (vgl. [LP12] Kapitel 11.1 Kasten Seite 256 und Kapitel 11.1.3 Abschnitt 'Kapillaren' Seite 265 und [Wen12] Kapitel 7.6 Absatz 1 Seite 164 und Kapitel 7.7 Abschnitt 'Kapillaren' Seite 166). Endothelzellen (Etymologie: griechisch '*ένδο endo* örtlich: 'innen, innerhalb', griechisch *θηλη thēlē* 'Brustwarze' oder auch *θάλλω thállo* 'sprießen, reichlich vorhanden sein'.) wiederum sind platte Zellen, die das Endothel bilden. Letzteres kleidet das Innerste der Gefäße aus (vgl. [LP12] Kapitel 11.1 Kasten Seite 256).

[116] Vgl. [Sta08] Kapitel 71 Abschnitt 'Splenic microcirculation' Unterabschnitt 'Open and closed splenic circulations' Seite 1195 und Abbildung 71.6 auf Seite 1194, [LP12] Kapitel 13.4.1 Abschnitt 'Rote Pulpa' Seite 336. Ebenso zu beachten [SBS11] Kapitel 'Discussion' 1. Absatz Seite 648.

[117] Vgl. [Ste+18] Kapitel 1.

[118] Bei den Begriffen 'offen' und 'geschlossen' sollte man sich von der mathematischen Sichtweise dieser Begriffe lösen. Jede Kapillare bietet Möglichkeiten zum Durchdringen. Eine Kapillarwand gilt somit bei den Medizinern als geschlossen, wenn durch diese keine roten Blutkörperchen können.

[119] Arterien sind Blutgefäße, durch die sauerstoffreiches Blut fließt. Arterieolen sind kleine Arterien.

[120] Im Kapitel 4.1 in Abbildung 28 auf Seite 75 sind Pinselarteriolen innerhalb der PALS und Follikel zu erkennen.

[121] Gemeint sind die roten Blutkörperchen (Erythrozyten), Blutplättchen und die weißen Blutkörperchen (Leukozyten).

[122] Blutplasma besteht zum größten Teil aus Wasser, enthält aber auch Gerinnungsstoffe, Proteine und Harnstoff.

[123] 'retikulär' bedeutet 'netzartig verzweigt'.

[124] Im Falle der Milz handelt es sich dabei um ein dreidimensionales Geflecht, welches unter anderem Lymphozyten (welche zu den Leukozyten gehören) und Makrophagen (welche zu den Fresszellen gehören) beherbergt (vgl. [LP12] Kapitel 8.1.2 Seite 139 und Kapitel 13.2 Absatz 1 Seite 323 und Kapitel 13.4 Seite 332ff) .

[125] Fibroblasten sind Bindegewebezellen, die Kollagen und Kollagenfasern produzieren und bei der Reparatur von Bindegewebe eine Mitwirkung haben (vgl. [LP12] Kapitel 8.1.1 Seite 121f).

[126] Vgl. [Ste+18] Kapitel 1 und [Dre09] Kapitel 7.2 Seite 54.

[127] Vgl. [Dre09] Kapitel 7.2 Absatz 3 Seite 55 [LP12] Kapitel 13.4 Kasten Seite 332 und [Wen12] Kapitel 7.10 Abschnitt 'Rote Pulpa' Seite 172.

[128] Vgl. [LP12] Kapitel 13.4 Seite 332.

[129] Mastzellen sind freie Zellen, die im Bindegewebe zu finden sind. Sie spielen eine Rolle bei der Immunabwehr (Histamin) und Entzündungen (vgl. [LP12] Kapitel 13.1.2 Abschnitt 'Mastzelle' Seite 318f). Mastzellen gehören ebenfalls zu den Leukozyten. Sie wirken bei der Wundheilung und Allergien mit (vgl. [Flea]).

[130] Plasmazellen sind ehemalige B-Lymphozyten, die nun Antikörper produzieren.

[131] Vgl. [Ste+18] Kapitel 1.

[132] Etymologie: griechisch: *περί peri* örtlich: 'um. . . herum'.

T-Lymphozyten (auch T-Zellen). Am Rande der PALS kann es weiterhin zu Follikelbildungen[133] kommen. Diese Milzfollikel (auch Malpighi-Körperchen oder Milzknötchen) enthalten B-Lymphozyten (auch B-Zellen), sind oftmals kugelförmig und machen den Hauptbestandteil der menschlichen *weißen Pulpa* aus.[134] Weiterhin vorhanden sind sogenannte Hülsenkapillaren (auch Kapillarhülsen oder Schweigger-Seidel-Hülsen). Sie befinden sich in einiger Entfernung von den Follikeln und sind umgeben von Makrophagen und rezirkulierenden B-Zellen.[135] Weiterhin erwähnenswert, es existiert eine sogenannte Marginalzone (vor allem bei den Follikeln).[136] Sie liegt genau zwischen der weißen und der roten Pulpa und fungiert als Schnittstelle zwischen beiden.[137] [138]

2.2 Akquise

Die Milzprobe, mit der gearbeitet wurde, stammt von einem 22 Jahre alten Mann. Die Akquisition der Gewebeprobe fand im Jahre 2000 statt und erfolgte unter den damaligen Vorschriften der Ethikkommission der Philipps-Universität Marburg. Die Probe wurde in Paraffin eingeblockt.[139] Aus dem daraus resultierenden Block wurde eine Serie von 24 Schnitten erstellt.[140] Das Exemplare ist repräsentativ für eine große Zahl von erwachsenen Milzen mit kleinen Sekundärfollikeln, die in den letzten Jahren untersucht wurden.[141]

2.2.1 Einfach-Färbung

Die immunhistologische Einfärbung erfolgte nach der ABC-Methode.[142] Während die direkte Färbemethode (Abbildung 2 (a) auf Seite 6) manchmal in der Darstellung der Strukturen schwach sein kann und dadurch auch teuerer wird, da mehr Antikörper verwendet werden müssen, ist die indirekte Methode (Abbildung 2 (b)) geeigneter eine Hervorhebung zu bewirken.[143] Denn hier können gegebenenfalls mehrere Sekundärantikörper verwendet werden oder, wie im Falle der ABC-Methode (Abbildung 2 (c)), kann der sekundäre Antikörper mit Biotin – auch als Vitamin B_7 oder Vitamin H bekannt – markiert werden. Durch Letzteres wird die anschließende Anwendung eines Avidin-Biotin-Komplexes (ABC) ermöglicht. Das Protein Avidin wird aus Hühnereiweiß gewonnen und bindet mit höchster Affinität an Biotin.[144] Der Avidin-Biotin-Enzymkomplexes ist ein molekularer Komplex aus Avidin und biotinylierter[145] Peroxidase[146], der viele unbesetzte Biotinbindungsstellen in den Avidin-Molekülen besitzt.[147] Die Verwendung der ABC-Methode hat den Vorteil, dass die Anzahl der Peroxidase-Moleküle beliebig hoch sein kann und somit viele dieser Moleküle an der Stelle des

[133] Etymologie: lateinisch *folliculus* 'Hülle' oder 'Hülse'.
[134] Vgl. [Ste+18] Kapitel 1.
[135] Vgl. [LF98] Kapitel 6.3.2.2 Seite 129 und [Ste+18] Kapitel 1.
[136] Vgl. [LP12] Kapitel 13.4 Seite 335.
[137] Vgl. [Sta08] Kapitel 71 Abschnitt 'Marginal zone' Seite 1195.
[138] Für eine bessere Orientierung soll noch auf [Sch+09] hingewiesen werden. Unter diesem Weblink ist eine schematische Darstellung des histologischen Aufbaus der Milz zu sehen.
[139] Vgl. Kapitel 'Das Fixieren, das Einbetten und das Einblocken' auf Seite 3 und Steiniger et al. [Ste+18] Kapitel 4.1.
[140] Vgl. [Ulr+14b] Kapitel 1.1 Seite 70 und Kapitel 6 Seite 75. Sowie [Lob+17] Kapitel 4.2 Seite 293.
[141] Vgl. [Ste+18] Kapitel 3 Seite 9 und Kapitel 4.1 Seite 15.
[142] Vgl. Kapitel 'Das Einfärben' auf Seite 5.
[143] Vgl. [NS00] Kapitel 5.
[144] Vgl. [NS00] Kapitel 5 und [Ulr+14b] Kapitel 1.1.
[145] Der Begriff 'biotinyliert' bedeutet soviel wie 'mit Biotin markiert'.
[146] Peroxidasen sind Enzyme, die mit Hilfe von Wasserstoffperoxid die Oxidation von Substanzen beeinflussen. Sie sind in der Lage farblose Stoffe durch Oxidation in ihrer Farbe zu verändern (vgl. [LP12] Kapitel 11 Abschnitt G.4 und G.4.1 Seite 253f).
[147] [Ulr+14b] Kapitel 1.1.

Antigens vorhanden sind. Peroxidase ist ein Enzym, das eine oxidative Polymerisation von löslichen Chromogenen wie Diaminobenzidin zu einem unlöslichen Produkt in Gegenwart von Wasserstoffperoxid katalysiert. Dies führt zu einer Braun-Färbung an der Stelle wo der erste Antikörper sitzt.[148]

Die genaue Vorgehensweise dieser Färbung wird unter Steiniger et al. [Ste+18] Kapitel 4.2 beschrieben und hatte ihren Fokus auf das Oberflächenprotein CD34[149] verwendet. Die Besonderheit der hier vorgestellten Einfach-Färbung ist die Intensität des Brauns. Bei den verwendeten Milzproben werden deutlich mehr Strukturen hervorgehoben als es bei den Proben aus Kusumi et al. [Kus+15] der Fall ist.

Im Anschluss der Braun-Färbung wurden die 24 Milzschnitte mit einer Zeiss Mirax beziehungsweise mit einer Leica SCN 440, beides Objektträger-Scanner, bei 20-facher Vergrößerung digitalisiert. Das Ergebnis waren Dateien in BMP-, SCN- und DICOM-Format. In Abbildung 6 (a) auf Seite 31 ist ein in das Bitmap konvertierter DICOM-Scan der Leica SCN 440 dargestellt. Bild (b) zeigt einen stark vergrößerten Ausschnitt. Ein Teil der Ergebnisse des Zeiss-Scanners ist in (c) wiedergegeben. Zu sehen von links nach rechts: die Schnitte 1, 12, und 16. Das letzte Bild ganz rechts zeigt diese Schnitte nochmal übereinander gelegt. Man erkennt, dass die Platzierungen der Schnitte auf den Objektträgern nicht exakt übereinander liegen und die Schnitte auch leicht zueinander verschoben sind.

2.2.2 Zweifach-Färbung

Nach der ersten Digitalisierung wurden die Deckgläser von den 24 Einfach-Färbungen abermals entfernt und zur Wiederherstellung der Proteine (Demaskierung) mit Citratpuffer behandelt. Im Fokus der zweiten Färbung lag diesmal das Antigen CD271[150]. Dieses wurde mittels High Def Blue Peroxidase blau eingefärbt. Genauer wird die Vorgehensweise der Färbung unter Steiniger et al. [Ste+18] Kapitel 4.3 beschrieben.

Statt der Verwendung eines Scan-Mikroskops wurden diesmal die Schnitte 'per Hand' digitalisiert. Dies bedeutet, die Akquisition erfolgte mit einer Fotokamera (Canon 60D), welche auf einem Lichtmikroskop (Zeiss Axiophot) aufgebracht war. Die Regionen von Interesse (ROIs) wurden jeweils einzeln mit einer 10-fachen Vergrößerung abgelichtet.[151] Ein Teil der Resultate ist in Abbildung 7 auf Seite 31 illustriert. Man erkennt auch hier die noch notwendige Ausrichtung der Schnitte.

2.3 Makro, Meso, Mikro

Wie in Abbildung 6 (b) zu erkennen ist, handelt es sich um den linken oberen Abschnitt eines Follikels. Dies bedeutet, im rechten unteren Bereich des $7\mu m$ dicken Schnittausschnittes sind B-Lymphozyten (hier nicht eingefärbt und somit durchsichtig) sichtbar. Ihre Größe liegt bei ca. $6\mu m$ und hat zur Folge, dass diese oftmals nur in einem einzelnen Schnitt vorhanden sind. Neben diesen mikroskopischen Elementen sind ebenfalls – auffällig – die braun eingefärbten Strukturen sichtbar. So sieht man in der unteren rechten Ecke die Follikelarterie, die einen geschätzten Durchmesser von $75\mu m$ hat. Neben dieser bilden anscheinend die zusätzlichen eingefärbten Strukturen einen Kreis um diese Arterie. Die

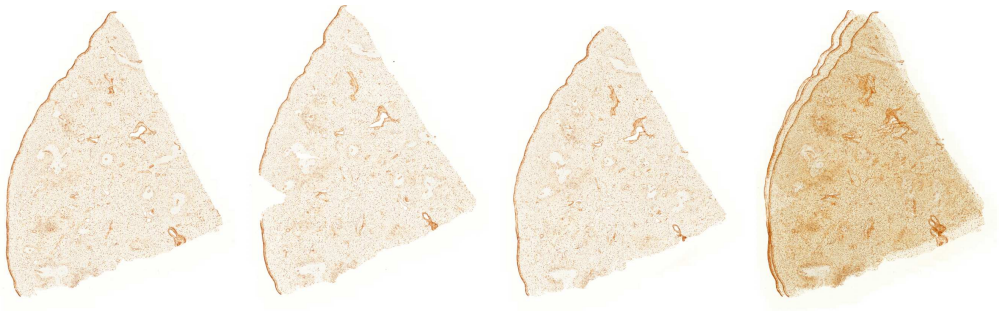
[148] [Ulr+14b] Kapitel 1.1.

[149] CD steht für 'Cluster of differentiation' und ist ein Namensverzeichnis für die gefundenen Antigene (vgl. [LP12] Kapitel 13.1.2 Seite 321). Wenn man in den im Internet verfügbaren Listen [Inc] nachschlägt, so erfährt man, dass CD34 unter anderem auf Endothelzellen von Kapillaren zu finden ist.

[150] Schlägt man in der CD Nomenklatur [Inc] nach, so sieht man, dass CD271 in Verbindung mit Follikuläre Dendritische Zellen (FDZ) steht.

[151] Die ROIs der Milzprobe werden in Abbildung 23 in Kapitel 4 auf Seite 69 noch deutlicher beschrieben.

(a) (b)



(c)

Abbildung 6: Die Digitalisierung der Einfach-Färbung. In (a) zu sehen ein mit der Leica SCN 440 eingescannter Objektträger (Schnitt 8). Das Bild (b) zeigt einen Ausschnitt des gleichen Schnittes in höchster Auflösung (20-facher Vergrößerung). Zu erkennen sind einzelne Zellen und Teile von angeschnittenen kleinsten Gefäßen in der roten Pulpa. In (c) sieht man die Ergebnisse der Zeiss Mirax (stark verkleinert). Man erkennt im Bild ganz rechts, dass die Schnitte, welche links nochmal einzeln dargestellt sind, noch nicht richtig zueinander ausgerichtet sind.

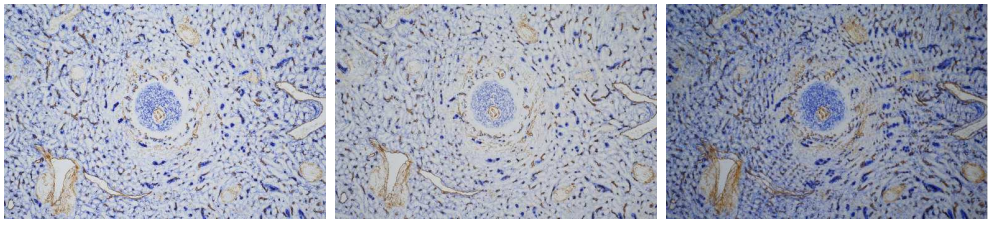


Abbildung 7: Die Digitalisierung der Zweifach-Färbung. Zu sehen ROI 1 aus Schnitt 1 und 2 der Serie. Im Bild ganz rechts sind die beiden Beispielschnitte wieder übereinander gelegt. Auch hier sieht man die noch nicht vorhandene Ausrichtung zueinander.

dunkelbraunen Elemente sind meist Kapillaren, während die heller eingefärbten Strukturen meist Sinus sind. Beide, wie auch die Follikelarterie, erstrecken sich über mehrere Schnitte und sind somit der mesoskopischen Ebene zuzuordnen. Der Follikel selbst, der in der menschlichen Milz einen Durchmesser von $300\mu m - 1mm$ haben kann, ist vorwiegend der mesoskopischen Ebene zuzuordnen.[152] Kapillarnetzwerke in der Milz, die sich über mehrere Millimeter erstrecken können, sowie Follikel mit einem $1mm$ Durchmesser oder größer, sind der makroskopischen Ebene zuzuordnen.

Es lässt sich somit zusammenfassend sagen: Zur makroskopischen Ebene gehören alle Strukturen ab $1mm$ oder größer. Mesoskopische Strukturen haben eine Größe zwischen $1mm$ und $500\mu m$ und Strukturen unter $500\mu m$ gelten als mikroskopisch.

2.4 Forschungsinteresse

Das medizinische Forschungsinteresse bei dieser Probe (Einfach- und Zweifach-Färbung) galt den Follikeln und der Gefäßstruktur(en), die die Follikel umschließt. Aber auch dem Netzwerk an Blutgefäßen an sich, welches sich über mehrere Millimeter erstrecken kann, galt das Interesse. Denn bisher ist die genaue Lage der Milzkapillaren innerhalb der weißen Pulpa nicht lokalisiert.[153] Gleiches gilt für die Bindegewebszellen des Follikels (Follicular Dendritic Cells (FDC) oder auch Follikuläre Dendritische Zellen (FDZ))[154] und die Lage der Kapillarhülsen.[155]

Wenn ein Blutgefäß als Teil einer mesoskopischen Struktur durch das Mikrotom zerlegt wurde, spiegelt das sich in den Schnitten je nach dessen Lage wie folgt wider: Wurde es orthogonal[156] angeschnitten, ergibt sich als Resultat ein Punkt oder bei größeren Gefäßen ein Kreis. War der Anschnitt des Gefäßes dagegen längs oder tangential, ergeben sich braune Strukturen in Form von Linien (meist Kapillaren) oder Doppellinien (meist Sinus).[157] Somit musste auf informatischer Seite nach einer geeigneten Methode gesucht werden, die die durch das Mikrotom zerlegten Gefäße Schnitt für Schnitt erkennt und unter den Schnitten richtig aneinander zuordnet und ausrichtet. Eine dreidimensionale Darstellung dieses Netzwerkes ist ebenso erforderlich, wie eine flexible und schnelle Anpassung der Methoden-Parameter für eine schnelle Verifizierung.

Strukturen von Interesse waren ebenfalls die Oberflächenbeschaffenheit von sämtlichen Zellen in den Follikeln sowie die Klärung der Frage, ob Kapillaren und Sinus miteinander verbunden sind. Wirft man einen Blick in die einschlägige Literatur [Wen12; Sta08], so scheinen zwei Verbindungsarten vorzuherrschen. Ein offener und ein geschlossener Blutkreislauf. Allerdings haben die Mediziner, mit denen zusammen gearbeitet wurde, bei den bisherigen Untersuchungen der Milzbefunde keine derartige geschlossene Gefäßverbindungen entdecken können. Sollte diese vorhanden sein, werden diese als anatomisch unbedeutend eingeschätzt und der Anteil dieser offenen Verbindungen mit mindestens 90% angegeben. Durch die (überraschende) Braun-Färbung der Sinus ergab sich zudem der Wunsch das umgebende Kapillarnetz inklusive des kapillararmen Follikelinneren dreidimensional darzustellen.[158]

[152] Vgl. [Ste15] und nach Aussagen von Frau Prof. Steiniger.
[153] Vgl. [Ste+18] Kapitel 1.
[154] Hierbei handelt es sich um spezielle Fibroblasten, die Lockstoffe für B-Lymphozyten aussenden.
[155] Vgl. [Ste+18] Kapitel 1.
[156] Hier bedeutet dies im Sinne der Ausrichtung der Volumendaten: die Gefäßachse zeigt in fast die gleiche Richtung wie die y-Achse.
[157] Vgl. [Lob+17] Kapitel 3.2.
[158] Frau Prof. Steinigers These zum Aufbau des Follikels ist, dass dies dazu dient die Lymphozyten abzukapseln.

3 Ausrichtung & Rekonstruktion

Wie im letzten Absatz des ersten Kapitels (Kapitel 'Einleitung' Seite 3) erwähnt, ist das Datenvolumen bei den heutigen Digitalisierungsmethoden immens. Da zudem die erstellten 3D-Modelle, wie in der Pipeline (Abbildung 4 Seite 26) angedeutet, die Qualitätsprüfung oft mehrmals durchlaufen, ist eine recheneffiziente Ausrichtung und Rekonstruktion erforderlich.

Die Ausrichtung der Serienschnitte ist aufgrund der spezifischen Verzerrungen eine besondere Herausforderung. Die Merkmale, die zur Anpassung der Bilder verwendet werden, müssen geeignet gewählt und deren Parameter zur Auswahl womöglich nachgebessert werden. Ebenso verhält es sich bei der Rekonstruktion. Die erstellten Modelle bestehen oftmals aus sehr vielen Dreiecken. Eine Reduktion der Faces während der Meshherstellung kann zu schneller verfügbaren Ergebnissen für die Verifikation führen. Auch hier muss gegebenenfalls der ideale Isowert für die Extraktion aus den Bildern noch gefunden und entsprechend mehrmals angepasst werden.

Es sollen nun einige Methoden vorgestellt werden, mit denen die medizinischen Daten bearbeitet und visualisiert wurden. Während einige Methoden bereits existierten und zum Teil schon implementiert waren,[159] wurden in dieser Arbeit diverse über den Stand der Technik hinaus gehende Verfahren entwickelt, um den Bearbeitungsprozess der Daten zu beschleunigen. Im letzten Kapitel wurde die Akquise und Digitalisierung der Daten beschrieben. Da die Vorverarbeitung der Bilder meist mit den üblichen Softwaretools (Irfanview, Fiji, oder ähnliche) umgesetzt wurde, wird dieser Schritt übersprungen und der Pipeline entsprechend nun der Schritt 'Registrierung und Ausrichtung' und die darin verwendete Herangehensweise dargestellt.

3.1 Registrierung, Ausrichtung und Deformation

Für die Registrierung, Ausrichtung und Deformation der Schnitte aus Kapitel 2.2 wurde ein eigenes Verfahren entwickelt, welches in Ulrich et al. [Ulr+14b] publiziert wurde. Der Ablauf des Algorithmus ist in Abbildung 8 veranschaulicht und lässt sich in drei Schritte unterteilen:[160]

1. **Merkmalsdetektion und Merkmalsextraktion (DF)**: Im Schritt 'Detect features' werden die markanten Merkmale des einzelnen Scan mittels BRISK extrahiert.
2. **Merkmalsanpassung und rigide Ausrichtung (FM & RFA)**: Im Schritt 'Rigid feature alignment' erfolgt eine starre Ausrichtung aller Bilder als initialer Startzustand mit Hilfe von RANSAC. Dieser Schritt soll die digitalisierten Schnitte grob aneinander ausrichten und so die nachfolgende Deformation unterstützen.[161] Da eine Deformation auch als Rotation und Translation interpretiert werden könnte, wird ein Schnitt (Slice) als Orientierung verwendet an welchem die restlichen Schnitte starr (rigide) ausgerichtet werden.[162]
3. **Minimierung der Deformation (FPC & NRFA)**: Die aus Schritt 2 gewonnene Ausrichtung wird dann im Anschluss iterativ nicht-rigide ausgerichtet und deformiert. In Schritt 3 wird die

[159] Unter anderem bietet MeshLab unter dem Menüpunkt 'Filters→Remeshing, Simplification and Reconstruction' nicht nur einen Simplifizierung-Algorithmus an.
[160] Vgl. [Ulr+14b] Kapitel 3.
[161] Vgl. [Ulr+14b] Kapitel 4.
[162] Vgl. [Ulr+14b] Kapitel 3.

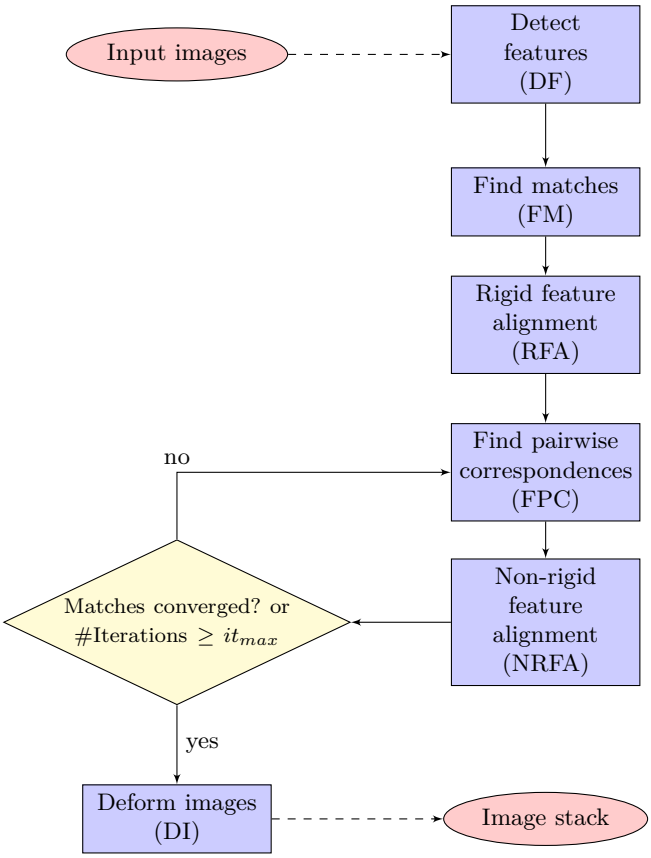


Abbildung 8: Die Arbeitsschritte des Verfahrens von Ulrich et al. [Ulr+14b]. 'Detect features' (DF) ist der erste Schritt. 'Find matches' (FM) und 'Rigid feature alignment' (RFA) bilden Schritt 2. Im dritten Schritt werden die Matches ('Find pairwise correspondences' (FPC)) in jeder Iteration neu bewertet und für die nicht-rigide Ausrichtung ('Non-rigid feature alignment' (NRFA)) verwendet. Der Schleifendurchlauf stoppt, wenn sich die Menge der Matches nicht mehr verändert oder die maximale Anzahl an Iterationsschritten (it_{max}) erreicht wurde. *(Das Bild mit wurde* `tikzpicture` *selbst erstellt und ist an Figure 2 aus [Ulr+14b] angelehnt.)*

Gesamt-Deformation der Schnitte minimiert. Es wird kein BRS[163] verwendet und die Verformungsminimierung geschieht global in *allen* Bildern, wobei als Randbedingung die Gesamtdeformation aller Slices minimiert wird.

3.1.1 Merkmalsdetektion und Merkmalsextraktion (DF)

Bilder werden aneinander ausgerichtet, indem man sie vergleicht. Vergleiche lassen sich zügig umsetzen, wenn man sich auf die herausstechenden Merkmale (markante Stellen) beschränkt. Im maschinellen Sehen stehen dafür mehrere Arten von Detektoren zur Verfügung, je nachdem ob diese Merkmale Regionen (Wasserscheidentransformation [BL79]), Linien (Canny Kantendetektion [Can86]) oder Punkte (Harris Corner Detector [HS88], Difference-of-Gaussian [Lin15]) sind. Markante Stellen können auch als Muster beschrieben werden (Templatematching).

Sind die Merkmale (*Features*) erkannt, müssen diese mit einem Deskriptor extrahiert (*Feature Extraction*) werden. Für die Beschreibung eines Merkmales durch einen Deskriptor wird meist die

[163] Siehe Kapitel 1.3 Seite 16.

nähere Umgebung des Merkmales mit einbezogen, ausgewertet und kodiert. Im Anschluss können dann die extrahierten Merkmale mit anderen extrahierten Merkmalen aus dem zu vergleichenden Bild zu Paaren zusammengefasst, also angepasst, werden. Diesen Vorgang nennt man auch *Feature Matching* oder *Merkmalsanpassung*.

Oftmals sind Detektion (*Feature Detection*) und Deskription (*Feature Description*) in einem Schritt zusammengefasst, wie beispielsweise beim SIFT (scale-invariant feature transform) [Vlf; Low99; Low04; CH09] oder SURF (speeded up robust feature) [Bay+08].[164] Dies ist ebenso bei BRISK der Fall. Dieser hält aber zusätzlich noch eine recheneffiziente Merkmalsanpassung bereit. Für die Merkmalserkennung verwendet BRISK einen angepassten FAST-Detektor [RD06; Mai+10]. Das heißt, die Punkte werden mittels einer $9-16$ Maske erkannt.[165] Weiterhin hilft der FAST-Detektor die Skalierungsgröße des Merkmals zu bestimmen.[166]

Beschrieben wird das Merkmal mit einem Muster aus 60 Kreisen.[167] Die Helligkeitswerte der Pixel, die sich im Kreisinneren befinden, werden Gauß-geglättet und nach einen bestimmen Muster paarweise verglichen. Dabei wird zwischen *short pairs* und *long pairs* unterschieden.[168] Die long pairs werden für eine Gradientenberechnung weiterverwendet und dienen dazu die Ausrichtung (Rotationswinkel) des Merkmals zu berechnen.[169] Bei den short pairs werden Helligkeitswerte miteinander verglichen und in 0 oder 1 kodiert.[170] Es entsteht ein binärer Code aus 512 Bits, welcher es ermöglicht die extrahierten Merkmale recht einfach mittels Hamming-Abstand zu vergleichen.[171]

Da die eingescannten Schnitte der Milz und des Knochenmarks recht kleine Strukturen enthalten, wie beispielsweise kleinere Zellen, die nur in einem Schnitt vorhanden sind,[172] oder kleinere Gefäßanschnitte, erkennt BRISK sehr viele Merkmale. Neben der dadurch erhöhten Wahrscheinlichkeit der Mehrdeutigkeit der Merkmale untereinander, erschwert und verlangsamt die hohe Anzahl die weiteren Berechnungen. Daher wird in der vorgestellten Methode [Ulr+14b] eine Selektion der Merkmale vorgenommen. Die Auswahl erfolgt anhand ihrer Durchmesser.[173] Dieser darf nicht kleiner als `100` Pixel sein sein.[174] Von dieser so entstandenen Menge werden wiederum nur N_{select} Merkmale pro Schnitt genommen und weiterverarbeitet.

3.1.2 Merkmalsanpassung und rigide Ausrichtung (FM & RFA)

Mit Hilfe des `BFMatcher` von OpenCV [BD08; BK08; KB17; Lib] werden als nächstes Merkmalspaare (*Matches*) gebildet. Da die Initial-Paare für die rigide Ausrichtung der Schnitte verwendet werden sollen

[164] SURF wird auch in Kapitel 4 verwendet. Diese Methode wurde erstmals im Jahre 2006 vorgestellt.
[165] Vgl. [LCS11] Kapitel 3.1.
[166] Bei einer 9-16 Maske muss der Kreis, welcher bei FAST für die Eckenerkennung verwendet wird und aus 16 Pixel besteht, mindestens 9 Pixel aufweisen, die deutlich heller oder dunkler sind als der Pixel (die Ecke) in der Mitte. Zusätzlich werden sogenannte *scale-space pyramid layers* verwendet. Dies bedeutet, die Bilder bzw. deren Pixel werden schrittweise herunterskaliert und ein maximaler *FAST-score* errechnet. Dieser Score dient dazu die Skalierungsgröße des Merkmals zu bestimmen.
[167] Vgl. [LCS11] Figure 3 Seite 2551
[168] Vgl. [KB17] Figure 16-30 Seite 559.
[169] Vgl. [LCS11] Kapitel 3.2.1 Seite 2551.
[170] Vgl. [LCS11] Kapitel 3.2.2 Seite 2551.
[171] Da der Hamming-Abstand [Ham50; Wike] lediglich eine XOR-Operation zweier Binärcodes ist, ist das Ergebnis wiederum ein Binärcode. Die Unterschiede sind in diesem als Einsen kodiert und werden in dem Verfahren von Leutenegger et al. zum Vergleich aufsummiert. Vgl. [LCS11] Kapitel 3.3 Seite 2551.
[172] Vgl. Kapitel 2.3 Seite 30 Abschnitt 'Makro, Meso, Mikro' und Kapitel 2.2.1 Seite 31 Abbildung 6.
[173] Unter OpenCV ist dieser mit `keypoints[i].size` abrufbar.
[174] Bei den Scans, die in [Ulr+14b] verwendet wurden, entspricht dies $60\mu m$. Hier wurden die Mirax-Scans um 50% verkleinert. Was zu einer Auflösung von $0,6\mu m$ pro Pixel führte.

und die Strukturen von Interesse gewisse Eigenschaften aufweisen, können die Matches weiter selektiert werden. Denn zwischen den beiden Schnitten kann die Skalierbarkeit der Merkmale eines Matches nicht besonders groß sein. Dies bedeutet, es sollten keine starken Größenunterschiede zwischen den Merkmalen vorherrschen. Gefäße laufen nicht gerade, machen keine plötzlichen 90-Grad-Wendungen und laufen gerade weiter. Daher werden nur die Matches genommen, deren Durchmessereigenschaften folgende Bedingung erfüllen:

$$\frac{1}{1+\varepsilon_l} \leq \frac{d_j}{d_{j-1}} \leq 1+\varepsilon_l \tag{26}$$

Dabei ist d_j der Durchmesser des Merkmales im $slice_j$ (auch Zielbild oder Referenzbild genannt) und d_{j-1} der Durchmesser des Merkmales im Quellbild $slice_{j-1}$ (auch Objektbild). Für beide gilt $d_j, d_{j-1} \in \mathbb{R}$. Weiterhin ist die maximale lokale Dehnung $\varepsilon_l \in \mathbb{R}$ beliebig aber fest und $j \in \{\mathbb{N} \cup 0\}$ mit $0 \leq j < dimY$. Abbildung 9 zeigt zur Veranschaulichung der Größenverhältnisse die Matches mit und ohne vorheriger Selektion nach Formel (26). Da eine rigide Transformation – auch euklidische Transformation genannt – Rotation und Translation beinhaltet, verändert sie nicht den Abstand und den Winkel der Punkte zueinander. Dies bedeutet, die Abstände können sich nur durch Deformation ändern. Dies hilft die Merkmalspaare weiter einzugrenzen. Für die Registrierung wird RANSAC [FB81] (random sample consensus) verwendet. Daher wird das folgende Selektionskriterium mit zwei Matches erklärt.

Wenn mp_1 und mp_2 zwei passende Merkmalspaare nach Formel (26) sind und P die Menge mit den jeweiligen vier Positionen im Objekt- ($slice_{j-1}$) bzw. im Referenzbild ($slice_j$) ist, dann wird ein *Match-Paar* nur genommen, wenn es eine gewisse globale Abstandsdehnung $\varepsilon_g \in \mathbb{R}$ nicht überschreitet:

$$\frac{1}{1+\varepsilon_g} \leq \frac{\|P_{mp_1,j} - P_{mp_2,j}\|}{\|P_{mp_1,j-1} - P_{mp_2,j-1}\|} \leq 1+\varepsilon_g \tag{27}$$

$$\|P_{mp_1,j} - P_{mp_2,j}\| \geq d_{min} \cdot s_{image} \tag{28}$$

$$\|P_{mp_1,j-1} - P_{mp_2,j-1}\| \geq d_{min} \cdot s_{image} \tag{29}$$

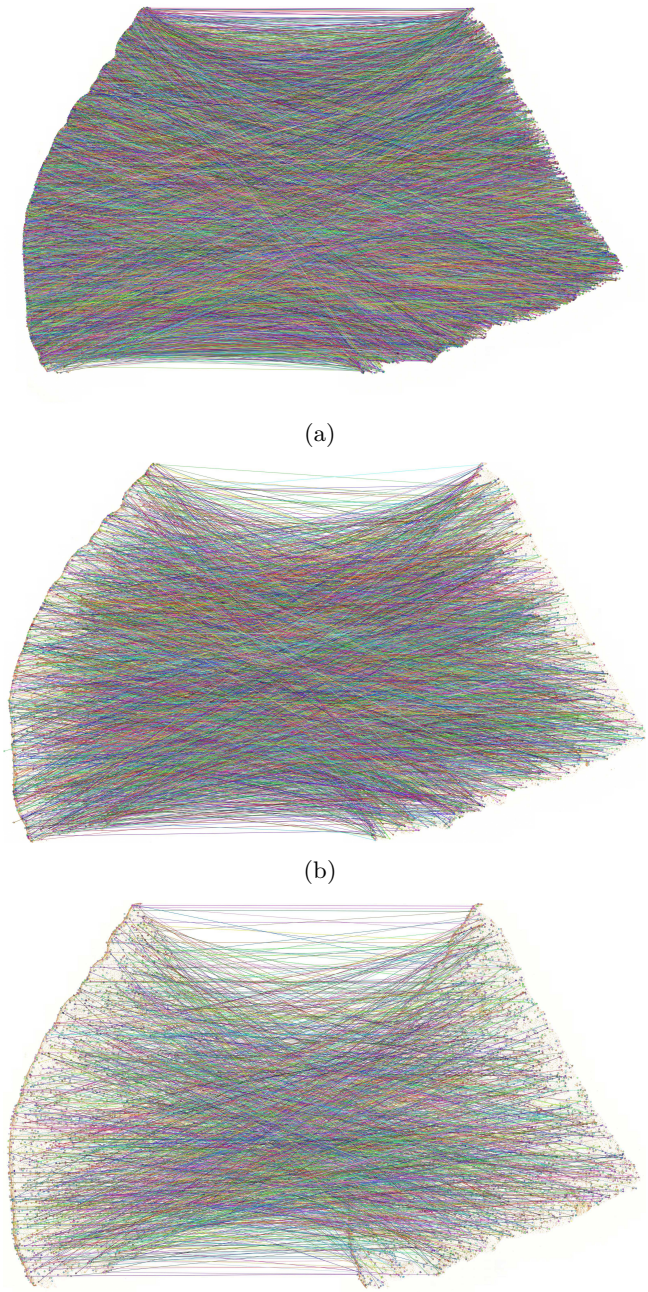
Dabei ist $d_{min} \in \mathbb{R}$ die relative minimale Distanz zwischen den Punkten und $s_{image} \in \mathbb{R}$ die Bilddiagonale des Schnittes. Weiterhin gilt aufgrund der rigiden Eigenschaft, dass mit einer gewissen Abweichung der Durchmesser sich auch eine maximale Abweichung der Positionen der Merkmalspositionen ergibt:

$$\varepsilon_g \leq \varepsilon_l \tag{30}$$

Mit den so selektierten Merkmalspaar-Paaren wird dann eine rigide Transformation T_j^R berechnet. Allerdings erfolgt die Qualitätsprüfung des Modells (der rigiden Transformation) nicht wie im klassischen RANSAC-Schritt. Die *Matches* werden nicht daraufhin geprüft, ob sie innerhalb einer gewissen Fehlertoleranz liegen und aufsummiert (Formel (31)).

$$\sum_{mp \in MP} g \tag{31}$$

$$\text{mit} \quad g := \begin{cases} 1 \text{ wenn } \|T_j^R P_{mp,j} - T_{j-1}^R P_{mp,j-1}\| \leq s_{image} \cdot \varepsilon_g \\ 0 \text{ sonst} \end{cases}$$



(a)

(b)

(c)

Abbildung 9: Die Auswahl der BRISK-Merkmale. Das Bild (a) zeigt die Registrierung der *aller* BRISK-Merkmale aus Quell- und Zielbild. In Bild (b) wird nur jeder 10-te Match dargestellt. Das letzte Bild zeigt ebenfalls nur jeden 10-ten Match. Hier wurden jedoch die Merkmale vorher gefiltert. Die Durchmesser betragen somit mindestens 100 und die Gesamtanzahl der Merkmale pro Bild beträgt nur $N_{select} = 20.000$.

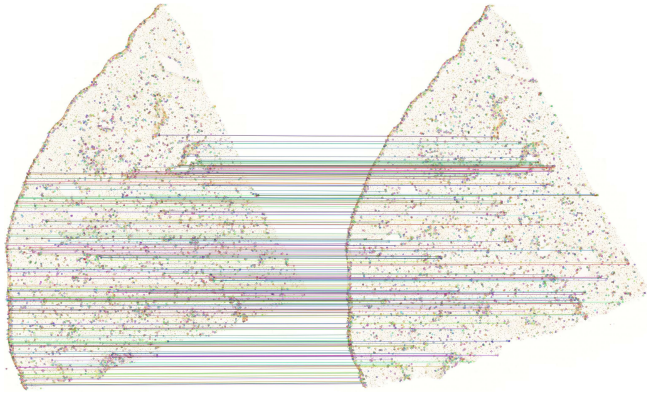


Abbildung 10: Eine Registrierung zweier Schnitte mittels RANSAC. Der obere Bereich der Schnitte ist deformiert und kann daher keine geeigneten *Matches* aufweisen.

Stattdessen werden die mittels des `BFMatcher` erzeugten $|MP|$ `DMatch`-Objekte und deren `distance`-Funktion verwendet. Mit diesen wird eine 'Qualitäts'-Gewichtung des Consensus-Sets[175] vorgenommen. Sei also $mp \in MP$ ein Merkmalspaar, erzeugt mit Hilfe des `BFMatcher`, und $\delta_{mp} \in \{0, \ldots, 512\}$ der Hamming-Abstand der beiden Merkmale dieses Paares, so ist mit einer beliebig aber festen Konstante $w_{match} \in \mathbb{R}$ die Gewichtung wie folgt:

$$\sum_{mp \in MP} \frac{1}{1 + w_{match}\delta_{mp}} \tag{32}$$

Die Suche nach einem geeignetem *Match*-Paar mp_1, mp_2 geschieht mit einem Schleifendurchlauf von 100.000. Ist ein geeignetes Modell gefunden, werden ein letztes Mal die *Inlier*[176] berechnet. Auch dies geschieht wieder mit Hilfe von Formel (32). Diesmal steht jedoch im Nenner der Least Squares Error. Hieraus resultiert die endgültige errechnete Transformation T_j^R. Ein Beispiel für das Endergebnis des Schrittes 'Rigid feature alignment' ist in Abbildung 10 zu sehen.

3.1.3 Minimierung der Deformation (FPC & NRFA)

Im dritten Schritt wird die übrig gebliebene Deformation in den Bildern und damit die Gesamt-Deformation minimiert. Hierbei handelt es sich um ein iteratives Verformungsverfahren, welches in jedem Durchlauf, die Matches erneut berechnet und diese für eine Bi-Cubic-B-Spline-Berechnung verwendet. Der Iterationsprozess endet, wenn sich die Menge der Merkmalspaare nicht mehr verändert oder die maximale Anzahl von it_{max} Iterationen erreicht wurde. Was in jedem Iterationsschritt passiert, soll nun beschrieben werden.

Erstellung der Menge an Matches Die detektierten Merkmale werden, wie in Schritt 2, layerweise gematch und vorsortiert (vgl. Formel (26) bis (29)). Ebenso erfolgt wieder eine Selektion. Dabei werden die Merkmale mk_1 aus Bild j und mk_2 aus Bild $j-1$ als Match mp genommen, wenn zwischen den

[175] Im RANSAC werden die Daten, die ein Modell unterstützen Consensus-Set genannt. Im unserem Fall gehören alle Matches zu diesem Set, wenn ihre Merkmale nach der Verformung nicht zu weit auseinander liegen.

[176] Der Begriff *Inlier* steht im Zusammenhang mit dem RANSAC-Algorithmus und bezeichnet ein Merkmalspaar, das durch das gefundene Modell erklärt wird beziehungsweise zur errechneten rigiden Transformation passt. Ein *Outlier* ist dementsprechend ein Ausreißerwert, der nicht zum Consensus-Sets passt.

beiden der Hamming-Abstand minimal ist und sie zudem folgende Bedingung erfüllen:

$$\|T_j^{NR} P_{mp,j} - T_{j-1}^{NR} P_{mp,j-1}\| \leq s_{image} \cdot \varepsilon_g \tag{33}$$

Man sieht, für die Selektion der nicht-rigiden Transformation wird der obere Term der selektiven Funktion g aus Formel (31) genommen und entsprechend angepasst. Das T^R wird durch ein T^{NR} ersetzt.

Parameterberechnung der Bi-Cubic-B-Splines Die Bi-Cubic-B-Splines sind als Gitter angeordnet und bestehen aus 9×9 Kontrollpunkten. An den Rändern des Gitters werden die Bi-Cubic-B-Splines um drei weitere Punkte erweitert.[177] Letztere liegen außerhalb des Bildes.

In jeder *it*-ten Iteration werden alle $dimY$ Bilder mittels einer for-Schleife durchlaufen. Über das erste Bild wird ein initiales Kontrollgitter gelegt. Dessen 9×9 Kontrollpunkte sind dabei so skaliert, dass sie das Bild in gleichen Abständen komplett abdecken. Für dieses und den nachfolgenden Bi-Cubic-B-Splines in den darauffolgenden Bildern müssen vorher für die Parameterberechnung noch einige *Minimierungsziele* definiert werden.

Sei mp ein Merkmalspaar aus dem Layer $layer_{j-1}$ und $j \in \{1, \ldots, dimY\}$ beliebig aber fest, dann gilt für die Kontrollpunkte des dazugehörigen Punktes $P_{mp,j}$ im Zielbild ($slice_j$) folgendes Minimierungsziel:

$$\sum_{l=0}^{14} \sum_{m=0}^{14} B_l(u_j) B_m(v_j) \alpha_{l,m}^j = T_j^R(P_{mp,j}) \tag{34}$$

$$\text{mit} \quad u_j = u\left(T_j^{NR}(P_{mp,j})\right) \text{ und } v_j = v\left(T_j^{NR}(P_{mp,j})\right) \tag{35}$$

In die Abbildungen u und v fließen die errechneten Parameter ein, die nötig sind, um die Pixel zu beziehungsweise den Punkt $P_{mp,j-1} = T_j^{NR}(P_{mp,j})$ aus Slice $j-1$ in Slice j richtig zu positionieren. Der Term $\alpha_{l,m}^j \in \mathbb{R} \times \mathbb{R}$ stellt den dazu passenden Kontrollpunkt dar. Die Terme $B_l(u_j)$ und $B_m(v_j)$ stehen für die beiden kubischen B-Spline-Basisfunktionen, mit denen der Kontrollpunkt $\alpha_{l,m}^j$ gewichtet wird.[178]

Die Formel (34) mit (35) ermöglicht die Registrierung des Slices j mit dem Slice $j-1$. Um größere Verformungen an den Stellen zu verhindern, an denen keine Merkmalspaare vorzufinden sind, bedarf es noch zwei weitere Zielvorgaben. Die Abstände der Kontrollpunkte sollen gegen d_x beziehungsweise gegen d_z minimieren. Es werden somit folgende Minimierungsziele definiert:

$$\forall j \in \{0, \ldots, dimY - 1\} \wedge \forall l \in \{0, \ldots, 13\} \wedge \forall m \in \{0, \ldots, 14\} : \left(\alpha_{l+1,m}^j - \alpha_{l,m}^j\right) w_\alpha = \begin{pmatrix} d_x \\ 0 \end{pmatrix} \tag{36}$$

$$\forall j \in \{0, \ldots, dimY - 1\} \wedge \forall l \in \{0, \ldots, 14\} \wedge \forall m \in \{0, \ldots, 13\} : \left(\alpha_{l,m+1}^j - \alpha_{l,m}^j\right) w_\alpha = \begin{pmatrix} 0 \\ d_z \end{pmatrix} \tag{37}$$

[177] Also gibt es 15 Kontrollpunkte pro kubischer B-Spline-Kurve und 30 kubische B-Spline-Kurven insgesamt.

[178] Die Implementierung erfolgt mit Hilfe von vier Methoden. Auf eine rekursive Umsetzung der klassischen Formulierung wurde verzichtet. Vgl. [Sal06] Kapitel 7.8 Seite 275f und [Boo02].

Hierbei sind $d_x, d_z \in \mathbb{R}$ die Abstände in x- und z-Richtung des Ergebnis-Slices j. $w_\alpha \in \mathbb{R}$ ist ein Gewicht zur Glättung und beliebig aber fest gewählt.

Die Nebenbedingungen (Formeln (34) bis (37)) ergeben zusammen ein dünn-besetztes lineares Gleichungssystem, welches mit dem *Verfahren der konjugierten Gradienten*[179] gelöst wird. Diese Methode erhält als Initialwerte die Werte der Kontrollpunkte in der vorherigen Iteration. In der allererste Iteration ($it = 0$) werden die Werte des noch nicht verformten Initial-Bi-Cubic-B-Splines wie folgt gesetzt:

$$\forall j \in \{0, \dots, dimY - 1\} \wedge \forall l, m \in \{0, \dots, 14\} : \alpha^j_{l,m} = \begin{pmatrix} (l-3)d_x \\ (m-3)d_z \end{pmatrix} \tag{38}$$

Da kein Slice als Referenz verwendet wird, bleibt die Gesamtdeformation als einziger Freiheitsgrad in den Volumendaten. Dies wird zur Minimierung der mittleren quadratischen Deformation aller Slices in jedem Iterationsdurchlauf benutzt. Wenn mit einem festen $l, m \in \{0, \dots, 14\}$ alle Kontrollpunkte in allen Slices ($slice_0$ bis $slice_{dimY-1}$) um einen gleichen Abstand verschoben werden, dann ändert dies nicht die Menge der Matches, die im Bilderstapel gefunden werden. Dies bedeutet im Umkehrschluss, dass die mittlere quadratische Deformation minimiert werden kann, indem die Verschiebung der Kontrollpunkte mit festen l und m über alle Bilder einen Mittelwert-Punkt bilden, welcher auf dem ursprünglichen Kontrollgitter liegt. Daher wird für die in den $dimY$ Bildern übereinander liegenden Kontrollpunkte folgendes Minimierungsziel formuliert. Sei $m, l \in \{0, \dots, 14\}$ beliebig, aber fest:

$$\frac{1}{dimY} \sum_{j=0}^{dimY-1} \alpha^j_{l,m} = \begin{pmatrix} (l-3)d_x \\ (m-3)d_z \end{pmatrix} \tag{39}$$
$$\text{mit} \quad \forall m, l \in \{0, \dots, 14\} \quad \text{beliebig, aber fest}$$

Dies heißt, am Anfang liegen die Kontrollpunkte $\alpha^j_{l,m}$ und $\alpha^{j-1}_{l,m}$ noch übereinander. Wenn nun zwei Merkmale in Bild $j-1$ und Bild j aufeinander abgestimmt werden sollen und das Merkmal in $slice_{j-1}$ links vom Kontrollpunkt $\alpha^{j-1}_{l,m}$ liegt und das Merkmal in $slice_j$ rechts vom Kontrollpunkt $\alpha^j_{l,m}$, dann muss $\alpha^{j-1}_{l,m}$ entsprechend nach rechts und $\alpha^j_{l,m}$ nach links verschoben werden.

Sind it_{max} Iterationsschritte erreicht oder ändert sich die Menge der Matches nicht mehr, werden im finalen Schritt 'Deform images' die Bilder aneinander ausgerichtet. In Abbildung 11 ist das Endergebnis dieser Iterationsausrichtung anhand der zwei Beispiel-Scans zu sehen. Man sieht die endgültige Menge an Matches. In jeder Region des Schnittes wurden zueinander passende Merkmale gefunden.

Verformen Für die Deformationsminimierung wird für jedes Pixel aus den zukünftigen Endbildern die passende Position im Quell-Bildern berechnet. Diese errechneten Positionen werden im Anschluss für die Ausrichtung der Bilder beziehungsweise für deren Verformung verwendet.

Da ein Bi-Cubic-B-Spline *uniform* (Formel (36) und (37)) ist und somit die Abstände der Kontrollpunkte gleich sind, kann dessen Berechnung eindimensional erfolgen. Zuerst wird jede kubische B-Spline-Kurve, dessen Kontrollpunkte ein festes aber beliebiges $l \in \{0, \dots, 14\}$ haben, bezüglich der x-Koordinaten ausgewertet. Danach erfolgt die Berechnung der z-Koordinaten. Das Verfahren der konjugierten Gradienten wird also zweimal angewandt.

[179] Vgl. Press et al. [Pre+07] Kapitel 10.8. Dieses wird auch CG-Verfahren oder *conjugate gradients method* genannt.

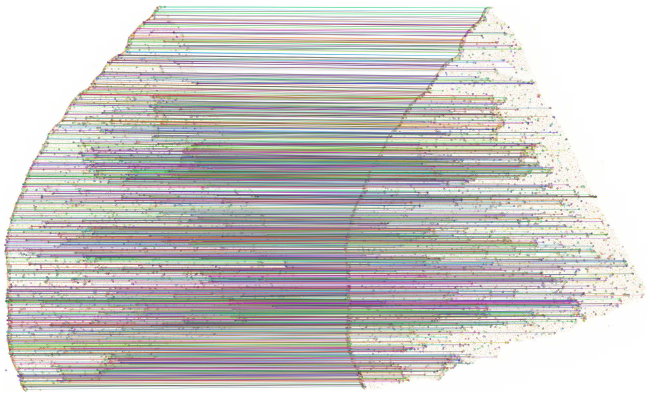


Abbildung 11: Das Endergebnis der Registrierungsmethode. Im Vergleich zur Abbildung 10 auf Seite 38 sieht man, dass nun auch für den oberen Bereich der Schnitte passende Merkmalspaare gefunden wurden. Hier, wie auch in der oberen Abbildung, sind nur die Matches mit einer hohen Gewichtung (Bewertung ≥ 0.2) dargestellt.

Da die B-Splines einen lokal kompakten Träger besitzen, kann bei der Auswertung der B-Spline-Kurven mit den in der Tabelle 1 angegebenen Parametern auf einen Bereich begrenzt werden, der von den vier Kontrollpunkten $\alpha^j_{l,m}, \alpha^j_{l+1,m}, \alpha^j_{l,m+1}$ und $\alpha^j_{l+1,m+1}$ als Eckpunkte umschrieben wird. Die Reduktion der Berechnung auf vier Basisfunktionen bedeutet eine ungefähre Zeitersparnis von 70% gegenüber einer vollbesetzten Matrix.[180]

Die berechneten Werte werden in einem `vector<Point2f>`-Array zwischengespeichert und mittels for-Schleife in zwei Matrizen für x- und z-Koordinaten transferiert. Beide Matrizen werden dann der OpenCV-Funktion `remap` übergeben. Diese verformt die Ausgangsbilder und sorgt somit für die Ausrichtung. Im abschließenden Schritt werden die finalen Bilder geglättet. Hierfür wird ein bi-linearer Filter[181] verwendet.

Während andere Methoden einen Referenzschnitt haben, an dem sie die Verformung und die Minimierung der Deformation genau anpassen, verwendet die eben vorgestellte Methode das erste Slice beziehungsweise Bild, um die *gemittelte Gesamt-Deformation* zu minimieren.

3.1.4 Resultate

Die Methode wurde in C++ und OpenMP[182] implementiert und verwendet zusätzlich die OpenCV-Bibliothek (Version 2.4.8). Die Rechenzeiten wurden auf einem Computer mit einem Intel Core i7 Prozessor, 16 GB Arbeitsspeicher und einem Windows 7 64-Bit Betriebssystem gemessen. Das Visual Studio-Projekt wurde als 64-Bit Code compiliert und in Tabelle 1 sind die verwendeten Parameter zu sehen, welche beim Testen der Methode verwendet wurden.[183] Zur Veranschaulichung werden in Abbildung 12 auf Seite 42 drei Bilder dargestellt, die zeigen wie der Wert w_α ermittelt wurde.[184]

[180] Vgl. [Ulr+14b] Kapitel 5.3 Seite 75.
[181] Vgl. [Wikb].
[182] Siehe Kapitel 1.6.3 Seite 24.
[183] Vgl. [Ulr+14b] Kapitel 6.1 Absatz 1 Seite 75.
[184] Vgl. Formel (36) und (37) auf Seite 39.
[185] Vgl. Auflistung in [Ulr+14b] Kapitel 6.1 rechte Spalte Seite 75.

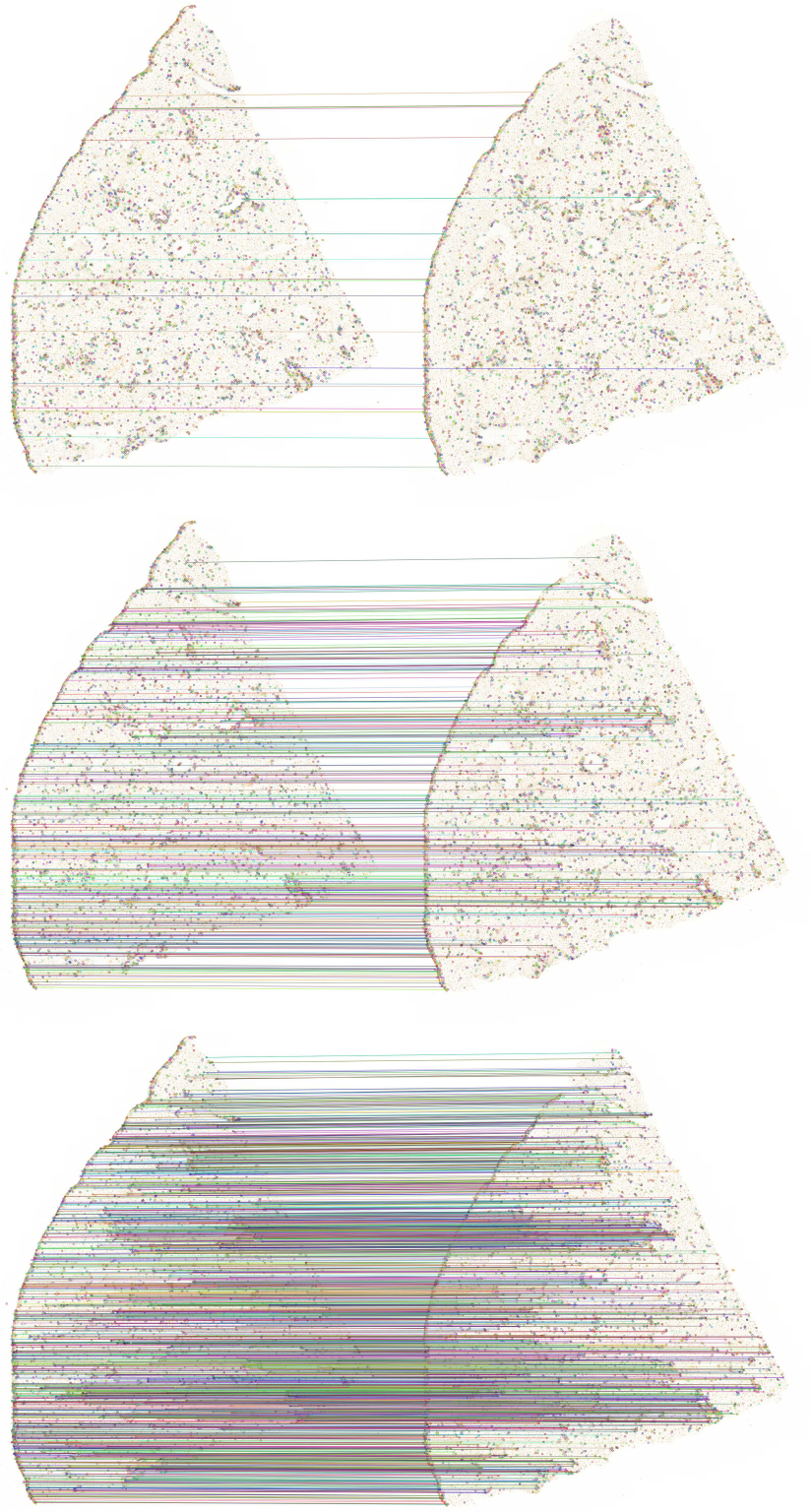


Abbildung 12: Die Ergebnisse bei unterschiedlichen Gewichtungen. Im oberen Bild sieht man die Menge an Merkmalspaaren bei einer Gewichtung von $w_\alpha = 0.5$. Darunter folgt die Darstellung mit $w_\alpha = 0.3$. Das letzte und unterste Bild zeigt den Wert der für die Ausrichtung der Schnitte genommen wurde. Hier ist $w_\alpha = 0.2$.

Parameter	Wert
N_{select}	20.000
s_{image}	41.140,4
d_{min}	0,1
ε_l	0,01
ε_g	0,005
w_{match}	10
w_α	0,2
it_{max}	20

Tabelle 1: Die verwendeten Parameter, die zur Ausrichtung der Schnitte verwendet wurden.[185]

Zeit (mm:ss)	**DF**	**FM RFA**	**FPC NRFA**	**DI**
Bild^a / *Bildpaar*^{b}	0:42^a	0:08^b	1:03^b	1:17^a
Gesamtdauer	16:58	3:20	24:02	30:35

Tabelle 2: Die gemessenen Rechenzeiten. Die obere Zeile zeigt die durchschnittliche Zeit und die untere Zeile die Gesamtzeit der Module bei der Verarbeitung der beiden 24 Bilder großen Milz-Scans. Die passenden Modulkürzel sind ebenfalls der Abbildung 8 auf Seite 34 zu entnehmen.

Als Testdaten dienten zwei eingescannte Serienschnitte. Neben der in Kapitel 2.2.1 als Einfach-Färbung beschriebenen Gewebeprobe der Milz, gab es eine weitere Doppelfärbung[186] mit CD34+ASM-1[187] und CD271. Beide Serien bestehen aus 24 Bildern, die jeweils eine Größe von 25.856 × 32.000 Pixel besitzen.[188] Diese wurden verwendet, um die Methode der Autorin mit der Iterative-Closest-Point-Methode (ICP) [BM92; Pom+13] und der nicht-starren (non-rigiden) Ausrichtungsmethode von Chui et al. [CR03] zu vergleichen.[189] Die Ergebnisse sind in den Abbildungen 13 und 14 auf Seite 45 und Seite 46 dargestellt.

In Abbildung 13 sieht man oben links das Ergebnis des ICP-Algorithmus mit zwei Original-Scans aus der Doppelfärbung als Eingabe. Es ist eine mangelhafte Ausrichtung zu erkennen. Die Bilder konvergieren hier 'nur' bis zum lokalen Minimum. Das Bild rechts daneben zeigt das Endresultat des gleichen Verfahrens. Hier wurden allerdings die zwei Schnitte vorher manuell grob aneinander ausgerichtet.[190] In der Abbildung unten links sieht man die Ausrichtung mit der Methode von Chui et al. [CR03] . Auch hier ist das Ergebnis nicht zufriedenstellend. Es scheint eine Art Fokus zu geben und der obere rechte Riss in beiden Schnitten konnte nicht richtig erfasst werden. Durch die Mehrdeutigkeit der Merkmale und der sich wiederholenden Strukturen stößt diese Methode an ihre Grenzen.[191] In Abbildung 13 unten rechts ist das Ergebnis der Ausrichtungsmethode von Ulrich et al. [Ulr+14b] zu sehen, welche ein deutlich besseres Endresultat liefert.[192] Trotz der geringen Merkmalsdichte im oberen rechten Bereich der Schnitte, wird diese Region richtig aneinander ausgerichtet. Ebenso stellt Abbildung 14 die Ergebnisse der vorgestellten iterativen Methode dar. Hier

[186] Gemeint ist nicht die Zweifach-Färbung aus Kapitel 2.2.2, welche nachträglich aus der Einfach-Färbung entstand.
[187] ASM-1 ist wie CD34 ein Antigen, welches zur Färbung - hier Braun - verwendet wird. Vgl. Seite 30 und [Mer].
[188] Vgl. [Ulr+14b] Kapitel 6 Absatz 1 Seite 75.
[189] Vgl. [Ulr+14b] Kapitel 6.1 letzter Absatz Seite 75f.
[190] Vgl. [Ulr+14b] Kapitel 6.1 letzter Absatz Seite 75f.
[191] Vgl. [Ulr+14b] Kapitel 6.1 linke Spalte Seite 76.
[192] Vgl. [Ulr+14b] Kapitel 6.1 linke Spalte Seite 76.

zu sehen sind die ersten Ergebnisse einer Ausrichtung der gesamten Schnittserien jeweils. Genaueres ist dem Abbildungstext zu entnehmen.

Da beide Schnittserien die gleiche Dimension aufwiesen, waren ihre Rechenzeiten gleich. Die Größe der Merkmalsmenge pro Bild, die im ersten Schritt von 'Detect features' (DF) gefunden wurde, betrug zwischen 430.000 und 450.000 Merkmale. Diese wurden durch den gewählten Wert N_{select} auf etwa $4 - 5\%$ reduziert.[193] In der Tabelle 2 sind die gemessen Zeiten wiedergegeben. Die erste Zeile gibt die durchschnittliche Dauer der entsprechenden Methodenschritte wieder. Bei den Modulen 'Detect features' (DF) und 'Deform images' (DI) bezieht sich der angegebene Wert auf die gemessene Zeit pro Bild. Bei den Modulen 'Find matches' (FM), 'Find pairwise correspondences' (FPC), 'Rigid feature alignment' (RFA) und 'Non-rigid feature alignment' (NRFA) bezieht er sich auf die gemessene Zeit pro Bildpaar. Die zweite Zeile gibt die durchschnittliche Gesamtdauer an. Man erkennt anhand der Tabelle 2, dass der Prozess der iterativen Deformungsminimierung der zeit-intensivste war. Dies liegt vor allem an der Berechnung der Pixelpositionen mittels der Bi-Cubic-B-Splines (Abbildungen u und v aus Formel 35) für das gesamte jeweilige Bild als Teil des finalen Bilderstapels.[194] Dennoch ist diese Vorgehensweise immer noch schneller als eine Korrektur der Bilder mittels Thin-Plate-Splines [Boo89; Lom; Elo], da bei diesen die Anzahl der Merkmale für zweidimensionale interpolierende Methode recht hoch ist. Weiterhin sollte angemerkt werden, dass die Laufzeit der eben vorgestellten Methode aufgrund des Matchings und des dünn besetzten linearen Gleichungssystems mit Anzahl der Schnittbilder linear ist.[195]

3.1.5 Fazit

Bei diesem Ansatz wurde eine Bi-Cubic-B-Spline-Interpolation verfolgt. Die zu interpolierenden Punkte sollten dabei in gleichen Abständen künstlich in das Bild eingefügt werden. Der Tangentenvektor eines jeden Punktes zeigt anfänglich in x-Richtung bzw. in die z-Richtung. Durch Vergleiche der Merkmale als Orientierung, lässt sich gegenüber dem vorherigen Bild feststellen wie dieser angepasst werden muss. Das Verwenden einer Spline-Approximation hat sich allerdings als geeigneter erwiesen. Die Punkte eines jeden zweidimensionalen Kontrollpolygons[196] werden durch das CG-Verfahren derart verschoben, dass mehr Merkmale zueinander passen. Die daraus resultierenden Bi-Cubic-B-Splines werden im Anschluss für die Position-Berechnung der Pixel verwendet.

Die Konstruktion des Kontrollnetzes samt seiner Minimierungsbedingungen passt gut zu den mechanischen Einwirkungen, die geschehen, wenn während der Mikrotomie[199] das Gewebestück geschnitten und im Anschluss auf dem Glasobjektträger aufgebracht wird. Die entstandenen Stauchungen und Verzerrungen sind alles andere als konzentrisch. Vielmehr kann die Intensität von Stauchung und Verzerrung variieren, da sich die Gewebeeigenschaft innerhalb der Probe ebenfalls ändert. Ein einfaches Beispiel ist ein kleiner Knorpel oder kleines Knochenstück, an dem das Mikrotommesser hängenbleibt und nach dem Durchdringen dieser Struktur wieder schneller durch das Gewebe gleiten kann. Dies kann zur Folge haben, dass das Gewebe an dieser Stelle vom Knochen gerissen wird. Die Einwirkungen der Mikrotomklinge sind somit alles andere als global. Ebenso erzeugt

[193] Vgl. [Ulr+14b] Kapitel 6.1 rechte Spalte Absatz 1 Seite 76.
[194] Vgl. [Ulr+14b] Kapitel 6.1 rechte Spalte Seite 76.
[195] Vgl. [Ulr+14b] Kapitel 6.1 rechte Spalte Seite 76.
[196] Gemeint ist das Kontrollnetz des Bi-Cubic-B-Splines.
[197] Vgl. Kapitel 4.1.3 Seite 79.
[198] Vgl. Kapitel 2.1 Seite 28.
[199] Vgl. Kapitel 1.1.2 Seite 4.

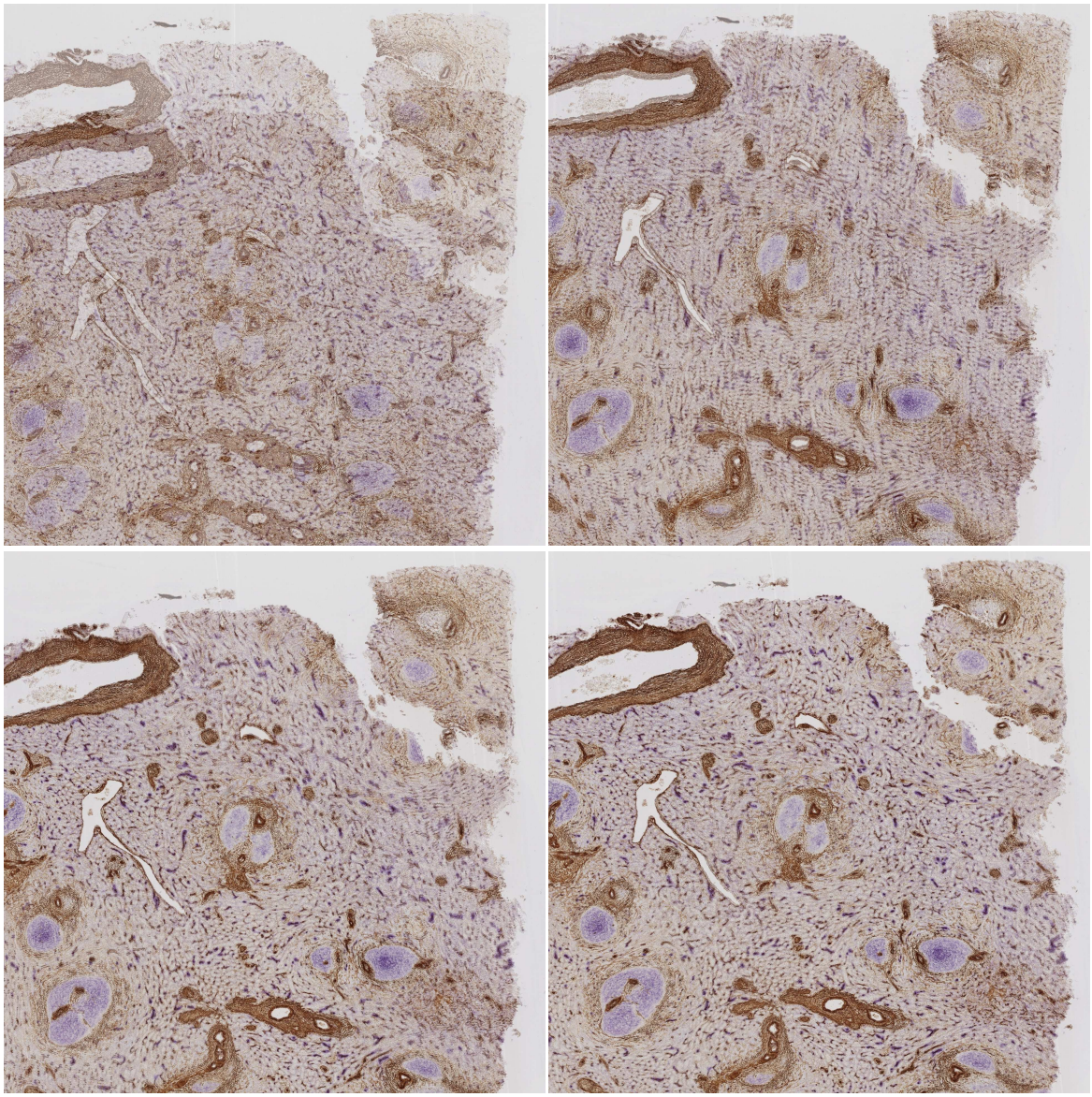


Abbildung 13: Der Vergleich zu anderen Methoden. Für die Versuche (obere Bilder) mit dem ICP-Algorithmus auf die Implementierung von [Pom+13] zurückgegriffen. Das Ergebnis der Methode Chui et al. [CR03] ist unten links zu sehen. Unten rechts ist die Methode der Autorin veranschaulicht.

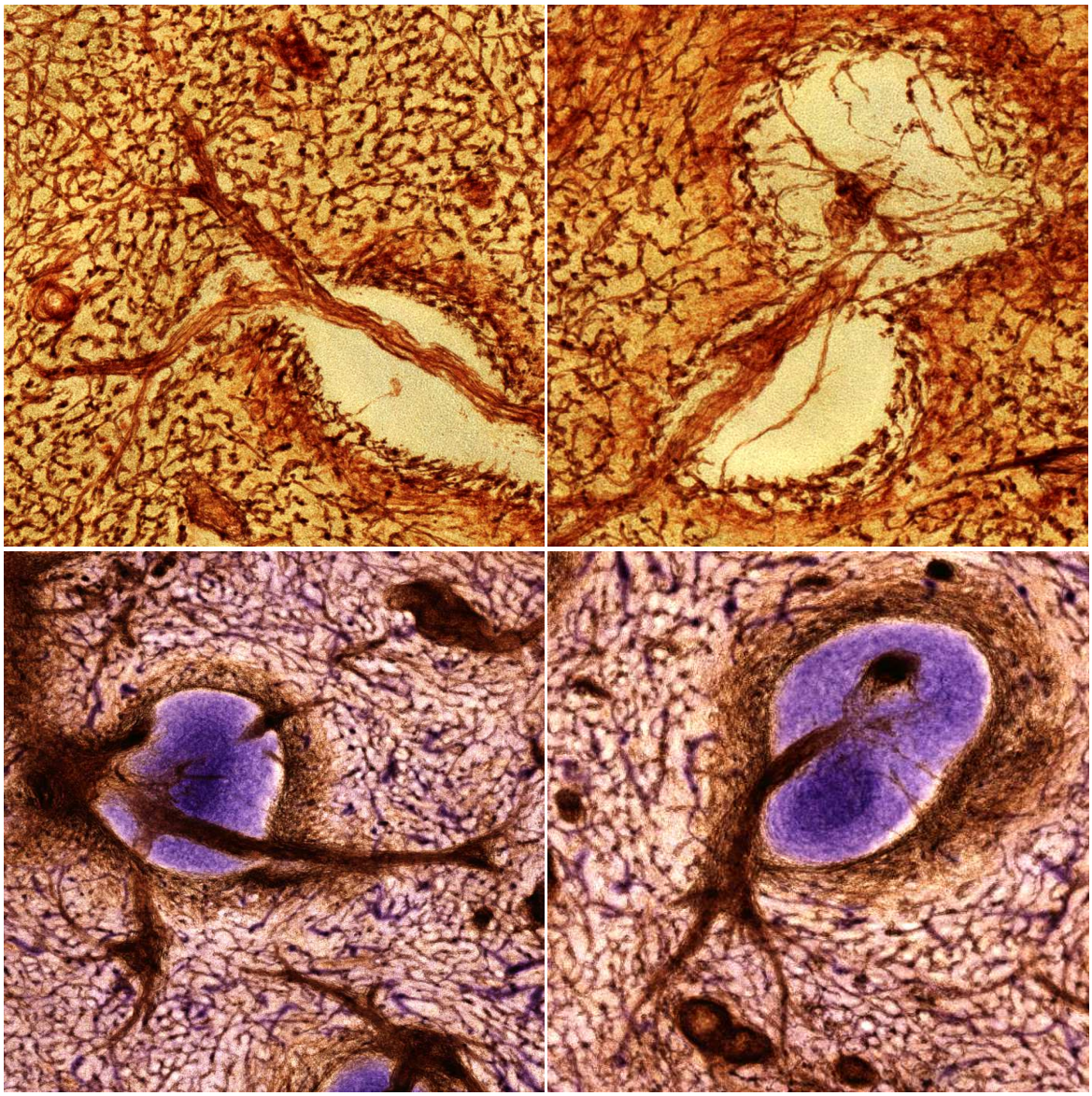


Abbildung 14: Die ROIs der Gesamtdeformationsminimierung der zwei Schnittserien. Die ausgerichteten Schnittserien wurden einem Volumenrenderer[197] übergeben. Zu sehen sind in dieser Abbildung Ausschnitte aus den Serien mit Strukturen von Interesse. Oben links ist eine etwas größere Arterie innerhalb der Milz veranschaulicht. Ebenso sind auf allen Bildern Teile der weißen und roten Pulpa[198] zu erkennen. Im Bild oben rechts sieht man einen Milz-Follikel samt seiner Blutversorgung durch eine Arterie. Die beiden unteren Bilder der Doppelfärbung zeigen ebenfalls Follikel mit Blutzufuhr. Eine derart räumliche und plastische Veranschaulichung war bisher bei digitalisierten Schnittserien nicht möglich.

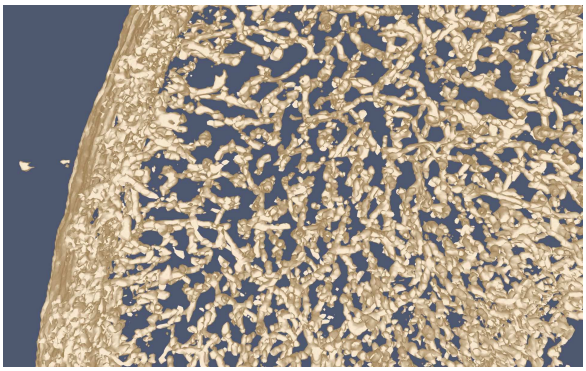


Abbildung 15: Die erste 3D-Rekonstruktion der Milzprobe.

das Aufbringen der einzelne Schnitte auf Glasobjektträger eine zusätzliche Verzerrung, die man als nicht-linear bezeichnen kann. Eine Deformungsminimierung durch ein Thin-Plate-Spline (TPS) ist für solche Verformungen weniger geeignet. Die im Gewebe verursachten Verzerrungen durch das Mikrotommesser besitzen keinen Ausgangspunkt, sondern neigen dazu eine 'Ereignis-Front' zu haben. Dies macht die Verwendung eines TPS weniger effizient, da hier auf ein vollständig besetztes Gleichungssystem zurückgegriffen wird und die hohe Merkmalsanzahl einen intensiven Rechenaufwand erzeugt. Da immer das gesamte Bild – also auf globale Weise – deformiert wird, ist es so schwieriger auf lokale anatomische Veränderungen innerhalb des Schnittes einzugehen. Ein TPS verfügt nicht über einen kompakten Träger und daher ist eine Reduktion auf 16 Kontrollpunkte (siehe Seite 41) nicht möglich.

Die Ausrichtung von Schnittbildern ist eine schwierige Herausforderung, aber die vorgestellte Methode bewältigt diese Aufgabe sehr gut.[200] Die so gewonnenen Darstellungen der Milzschnitte ermöglichten es den Medizinern, die Befunde in Art und Weise zu diskutieren und zu untersuchen, der bisher nicht möglich war.[201] Dies geschah unter anderen auch mit den 3D-Rekonstruktionen (Meshes). In Abbildung 15 ist eines der ersten Modelle zu sehen. Man erkennt, dass die Gefäße teilweise kurz sind. Der Grund dafür liegt in der Methode von Ulrich et al. [Ulr+14b] selbst. Diese arbeitet nur mit größeren Merkmalen. Dies ist in gewisser Weise ein Nachteil, welcher aber in Lobachev et al. [Lob+17] nachgebessert wurde.

3.1.6 Weiterentwicklung

Die medizinischen Proben aus [Ulr+14b] weisen besonders viele kleinere Merkmale auf, die sich zudem untereinander sehr ähnlich sind. Bei der Methode von Ulrich et al. [Ulr+14b] führt dies je nach Parameterwahl entweder zu einer sehr hohen Trefferquote an Merkmalen, die sich nicht eindeutig matchen lassen, oder zu einer künstlich gering gehaltenen Anzahl an Merkmalen, die größer sind aber auch eine ungenauere Ausrichtung bedeuten. Daher fokussiert sich die Weiterentwicklung von Lobachev et al. [Lob+17] auf die Performanz und die Merkmalserkennung.

Diese besteht nach wie vor aus drei grundlegenden Schritten (DF; FM, RFA; und FPC, NFRA), welche nun aber mehrmals durchlaufen werden. Wie aus Abbildung 16 ersichtlich ist, erfolgt nur im ersten Durchlauf eine rigide Ausrichtung, in den darauf folgenden Schleifen wird ein *Radius*

[200] Vgl. [Ulr+14b] Kapitel 7 Absatz 2 Seite 77.
[201] Vgl. [Ulr+14b] Kapitel 7 Absatz 1 Seite 77.

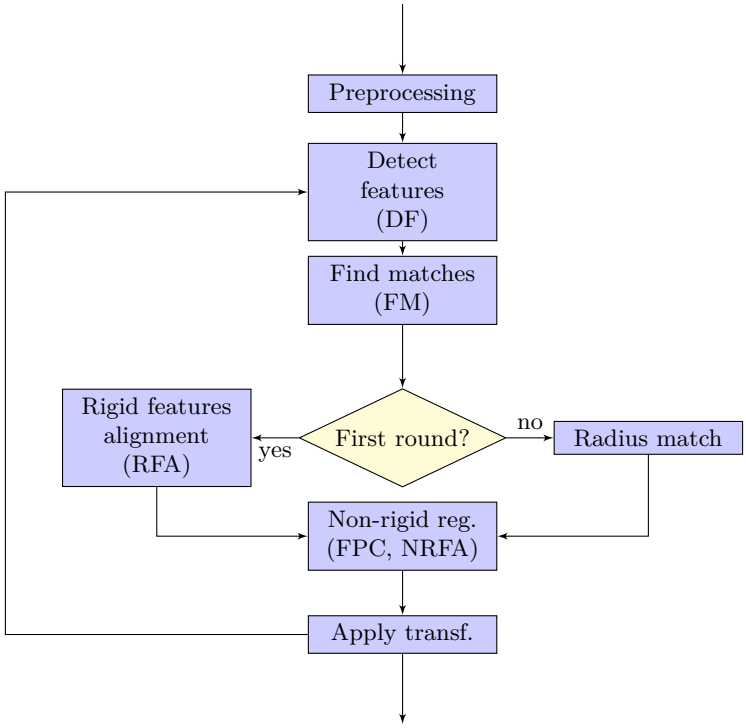


Abbildung 16: Die Arbeitsschritte des Verfahrens von Lobachev et al. [Lob+17]. *(Das Bild mit wurde* `tikzpicture` *selbst erstellt und ist an Figure 1 aus [Lob+17] angelehnt.)*

match angewandt.[202] Letzteres ist ebenfalls OpenCV zuzuordnen und sucht passende Merkmale nur innerhalb eines bestimmten Radius.[203] Während des Schleifendurchlaufs[204] wird der Suchparameter bezüglich der Mindestgröße der Merkmale immer wieder angepasst. Das heißt, in jedem Durchlauf wird nach kleineren Merkmalen gesucht. Das bereits beschriebene Kontrollpolyeder zur Entzerrung wird zusätzlich jedes Mal erweitert, indem die Kontrollpunkte in beide Richtungen (in x- und in z-Richtung) verdoppelt werden.[205] Anstatt des BRISK-Merkmalsdetektors wird nun SURF [Bay+08] verwendet. Dieser gehört auch zur OpenCV-Bibliothek (xfeatures2d) und ist in dieser mit OpenCL[206] umgesetzt. Laut Cameron Schaeffer [Sch13] hat er sich gegenüber BRISK als robuster und stabiler erwiesen. Weiterhin wird das Verfahren der konjugierten Gradienten durch die iterative Methode zur Lösung linearer Gleichungen LSMR [FS11] ersetzt. Diese hilft die Rechenzeit der neueren Ausrichtungsmethode erheblich zu verkürzen.

Durch Lobachev et al. [Lob+17] wurde die bisherige Methode Ulrich et al. [Ulr+14b] zu einen hierarchischen Ansatz in Bezug auf die Merkmalsgröße erweitert. Dieser beschränkt sich nicht nur auf große Merkmale für rigide und nicht rigide Ausrichtung, sondern geht in seiner zusätzlichen Schleife auf die kleineren Merkmale in den Serienschnitten ein.[207] Wie aus der Tabelle 4 von [Lob+17] zu entnehmen ist, werden die zusätzlichen Kontrollpunkte der Bi-Cubic-B-Splines und die höhere

[202] Vgl. [Lob+17] Kapitel 5.1 Seite 295 Absatz 2.
[203] Dieser war bei den Tests auf 0,5% der größeren Seitenlänge eines Scans gesetzt. Siehe [Lob+17] Kapitel 4.1 Seite 293 letzter Absatz.
[204] Gemeint ist die äußere Schleife in Abbildung 16, die von 'Apply transf.' nach 'Dectect Features' führt.
[205] Vgl. [Lob+17] Kapitel 4.1 Seite 293 letzter Absatz.
[206] Siehe Kapitel 1.6.3 Seite 24.
[207] Vg. [Lob+17] Kapitel 5 Seite 294f.

Anzahl der Merkmale gut mit dem LSMR aufgefangen.[208] Durch die Fokussierung auf kleinere Features schnitt die Weiterentwicklung auch bei der Registrierung und Ausrichtung besser ab. Ausgiebige Testergebnisse hierzu mit insgesamt sieben Qualitätsmaßen können in Lobachev et al. [Lob+17] und im dazugehörigen Supplementary [Lob+16] nachgelesen werden.

3.2 Rekonstruktion und Simplifizierung

Der Marching Cubes-Algorithmus eignet sich aufgrund seiner divide-and-conquer-Strategie (siehe Seite 16) hervorragend für die parallele Verarbeitung und somit für die Implementierung in CUDA. Nvidia selbst stellen in ihrer Beispielsammlung [Nvib] eine MC-Implementierung zur Verfügung. Allerdings kann diese nur mit würfelförmigen Volumendaten ($dimX = dimY = dimZ$) umgehen, während die in dieser Dissertation vorgestellte Methode auch quaderförmige Volumendaten, also $\neg(dimX = dimY = dimZ)$, zu bearbeiten vermag.

Da von Grund et al. [GDG11] ein parallelisierter Simplifizierungsalgorithmus veröffentlicht wurde, der ebenfalls unter Verwendung der GPU und CUDA arbeitet, bot es sich an diese beiden Methoden in einer Art Pipeline hintereinander zu schalten. Anstatt große Flächenmodelle zu erzeugen, auf eine Festplatte zu speichern und dann erst im Anschluss mittels Simplifizierung zu verkleinern, kann dies ebenso in einem Scan-Verfahren und unter Verwendung massiver Parallelisierung geschehen. Um dies zu bewerkstelligen werden eine eigene CUDA-Implementierung des MCs und das Simplifizierungsmodul von Grund et al. [GDG11] zu einem Hybrid-Algorithmus vereint. Ein zentraler Vorteil dieses Ansatzes ist, dass auch der Speicheraufwand massiv reduziert wird, da nie das vollständige hoch aufgelöste Dreiecksnetz im Grafikkartenspeicher gehalten werden muss.

Hybrid Algorithmen Die Idee eines *Hybrid-Algorithmus* dieser Art ist dabei nicht neu. Ähnliche Methoden wurden schon von Attali et al. [ACE05] und Dupuy et al. [Dup+10] vorgestellt. Auch sie reduzieren das Mesh direkt nach der Extraktion. Allerdings verwendet die Methode von Attali et al. [ACE05] noch den sequentiellen Marching Cubes. Ebenso wird eine sequenzielle Simplifizierung benutzt. Beide, Rekonstruktion und Simplifizierung, wechseln sich bei Attali et al. Layer für Layer im Tandem ab. Ein sogenanntes *time-lag* bewirkt eine Verzögerung bei den edge-collapses - dem Zusammenfallen von Meshdreiecken - und sorgt somit für eine bessere Qualität, als bei einer vollständigen Simplifizierung des Teilmodells.[209] des Meshes. Bei Dupuy et al. wird das Tandem-Verfahren von Attali et al. parallelisiert, indem sie einen *load-balanced-cluster* mittels der Plattform LSF-HPC-Software[210] [Wiki; IBM] umsetzen. Dabei wird die Extraktion des gesamten Volumens nicht mit einem Scan-Verfahren (also mit einer Sweepline) vollzogen, sondern der Datensatz wird wie bei Müller und Stark [MS91] in Blöcke unterteilt. Ein Schwellwert regelt die Unterteilung und somit ebenso die Größe eines Blattes[211], der kleinsten Unterteilungseinheit der Methode. Die Blätter werden dann mit einem modifizierten Tandem-Algorithmus behandelt.[212] Auch Cuccuru et al. [Cuc+09] haben einen Hybrid-Algorithmus vorgestellt. Dieser parallelisiert zwar die Rekonstruktion, tut dies aber indem er einen Daten-Stream (Daten-Strom) in Chunks (Daten-Blöcke) zerlegt. Auch diese

[208] Die Tabelle 4 aus [Lob+17] mit den aufgelisteten Rechenzeiten befindet sich auf Seite 300 oben.
[209] Gemeint sind unter anderem weniger Dreiecksplitter mit spitzen Winkeln, aber auch ein regelmäßigere Gitterstruktur.
[210] Siehe Kapitel 1.6.3 Seite 24.
[211] Vgl. [Dup+10] Kapitel 3.2 Seite 132 Absatz 4 - Splitting phase.
[212] Attali et al.'s Methode wurde dahingehend erweitert, dass bereits fertig bearbeitete Komponenten vom Speicher freigegeben werden.

Methode unterteilt die Volumendaten. Dies geschieht mit Hilfe eines Octrees. Die Meshherstellung bewerkstelligen Cuccuru et al. mittels MLS (Moving Least Square). Allerdings ist ihr Verfahren auf Punktwolken angelegt und wurde mit MPI (Message Passing Interface)[213] umgesetzt.

Parallelverarbeitung auf kleinster Ebene Im Gegensatz zu den anderen Hybrid-Methoden parallelisiert die vorgestellte Methode [Ulr+14a] den Marching Cubes und die Simplifizierung auf der kleinsten Ebene des jeweiligen Verfahrens selbst. Die CUDA-Threads[214] bearbeiten die einzelnen Cubes und im Anschluss die edge-collapses mit ihren (minimalen) Fehlerwerten parallel. Diese beiden Vorgehensweisen der Module (Rekonstruktion und Simplifizierung) soll nun detaillierter vorgestellt werden.

3.2.1 Das Parallele Marching Cubes Modul

Bei dem parallelen Marching Cubes wird der Datensatz für die Verarbeitung, wie bei Attali et al. [ACE05], in Slices und Layers unterteilt.[215] Mithilfe der Slices und einer gewählten Größe N werden Partitionen erstellt, und zwar so, dass sie in den jeweiligen Grafikkartenspeicher passen.[216] Hier sollte vorher noch ein Mal daran erinnert werden, was auf Seite 9 nach Formel (7) schon erwähnt wurde, die Höhe der Volumendaten (beziehungsweise der Schnittprobe) wird an der y-Achse angegeben. Die z-Achse zeigt von der Tiefe aus auf den Betrachter und die x-Achse zeigt nach rechts. Da die Slices mit 0-beginnend durchnummeriert sind, trägt das letzte und oberste Slice des Datensatzes bei einer Höhe von $dimY$ den Index $dimY-1$ und die Durchnummerierung der Layers geht von 0 bis $dimY-2$. Bevor der Hybrid-Algorithmus startet, werden dem Programmcode die globalen Konstanten als Parameter übergeben. Diese bestehen unter anderem aus dem Namen der Datei, welche die Volumendaten enthält und eingelesen werden muss, aus den Dimensionen der Volumendaten ($dimX$, $dimY$, $dimZ$) und aus der gewählten Partitionsgröße N. Weitere Parameter beziehungsweise globale Konstanten sind Angaben zum Pfadnamen und zu den Abständen, die zwischen den Voxeln im rektlinieären Volumengitters vorliegen.[217] Letztere definieren die Kantenlängen der Würfel und sind für die Mesherzeugung essentiell.[218]

Eine vereinfachte Darstellung des Marching Cubes-Moduls ist in Algorithmus 1 auf Seite 51 zu sehen. Hier wird, wie schon erwähnt, in der ersten Zeile aus den $dimY$ Slices die Anzahl der Partitionen ($Partitions$) und die Restanzahl an Slices ($Remainders$) erstellt werden. Jede Partition, bis auf die letzte, wird dann in der 'äußeren' for-Schleife per `cudaMalloc` und `cudaMemcpy` auf die Grafikkarte hochgeladen (Algorithmus 1 Zeile 3) und dort layerweise von den Kernels *kernel_cubecode* (Zeile 4) und *kernel_generate* (Algorithmus 1 Zeile 5) verarbeitet. Beide Kernel besitzen dafür jeweils eine ('innere') for-Schleife. *kernel_cubecode* erzeugt die Cubecodes, Schnittpunkte, Normalenvektoren und Gradienten. *kernel_generate* erhält die ersten drei der eben genannten Ausgaben als Eingabe und erzeugt mit ihnen und mit Hilfe der look-up-Tabelle von Paul Bourke [Bou94][219] die Isofläche, welche

[213] Siehe Kapitel 1.6.3 Seite 24.
[214] Siehe Kapitel 1.6.2 Seite 20.
[215] Siehe Kapitel Grundlagen Seite 11 Formel (11) und Formel (12).
[216] Es soll noch Mal darauf hingewiesen werden, dass die eingescannten und ausgerichteten Schnitte der medizinischen Probe vorher auf eine ROI begrenzt werden. Dieser Teilausschnitt des Volumendatensatzes passt aber meistens immer noch nicht komplett in den Grafikkartenspeicher. Eine Unterteilung in Layers und Slices ist daher sehr entgegenkommend.
[217] Hier sei noch mal bezüglich der Abstände auf Formel (4) auf Seite 9 hingewiesen.
[218] Vgl. Seite 9 Formel (4).
[219] Der Vorteil von Paul Bourkes Tabelle *triTable* ist, dass in ihr alle 256 Fälle definiert sind. Löcher wie beim

```
1:  Partitions, Remainders = gen_partition(dimY)
2:  kernel_cubecode_init()
3:  for each partition_index in range(Partitions-1) do
4:      kernel_cubecode(partition_index, N)
5:      kernel_generate(partition_index, N)
6:      call_simplification_module()
7:      kernel_arrangearrays()
8:  kernel_cubecode(Partitions − 1, Remainders + 1)
9:  kernel_generate(Partitions − 1, Remainders + 1)
10: call_simplification_module()
```

Algorithmus 1 : Das Parallel Marching Cubes-Modul. Es ist gegenüber der Darstellung in [Ulr+14a] in den Zeilen 3-5 und 7-9 etwas genauer. Das $Remainders+1$ rührt daher, dass das letzte Slice nicht mehr geopfert wird. Ein 'Aneinanderkleben' der Partition ist hier nicht mehr nötig (vgl. Abschnitt 'Übergang zwischen den Partitionen' Seite 54). N ist im gesamten Hybrid-Algorithmus von [Ulr+14a] eine globale Konstante.

an *call_simplification_module* (Algorithmus 1 Zeile 6) weitergereicht wird. Bevor es dann in den nächsten Schleifendurchlauf geht, werden die Arrays auf der Grafikkarte mit *kernel_arrangearrays()* neu arrangiert.[220] Zum Schluss werden in den Zeilen 8 bis 10 die verbliebenen *Remainders* Slices in der letzten Partition behandelt.

Bis auf die zweite Zeile ist das MC-Modul des Hybrid-Algorithmus von [Ulr+14a] vorgestellt worden. Was diese Zeile macht und wozu sie benötigt wird, erschließt sich, wenn man sich *kernel_cubecode* näher betrachtet. Dargestellt wird dieses Kernel in Algorithmus 2 auf Seite 52. Um dessen Funktion besser klären zu können, sollte die Abbildung 17 auf Seite 53 hinzugezogen werden. Sie hilft zu veranschaulichen, was die Threads jeweils parallel Lesen, Berechnen und Schreiben.

Kernel_cubecode im Detail Bei dem Kernel *kernel_cubecode* handelt es sich eigentlich um einen 4-fachen Aufruf desselben. Es arbeitet mit zweidimensionalen 16×16 (oder je nach Grafikkarte mit einem 32×32) Thread-Blöcken. Da es auch quaderförmige Volumengitter bearbeiten soll, zerlegt es die Dimension eines Slices in vier Teile. So gilt:

```
dim3 threads_main   =  (16, 16, 1) //oder (32, 32, 1)
dim3 blocks_main    =  (dimX/threads_main.x, dimZ/threads_main.y, 1)
dim3 threads_right  =  (dimX%threads_main.x, threads_main.y, 1)
dim3 blocks_right   =  (1,blocks_main.y,1)
dim3 threads_bottom =  (threads_main.x, dimZ%threads_main.y, 1)
dim3 blocks_bottom  =  (blocks_main.x,1,1)
dim3 threads_corner =  (threads_right.x,threads_bottom.y,1)
dim3 blocks_corner  =  (1,1,1)
```

Original-Algorithmus können so nicht entstehen (vgl. Abschnitt 'Rekonstruktion (Marching Cubes)' Seite 16). Zwei Anpassungen hat diese Tabelle aber allerdings erfahren. Zum einem wurde sie mit `__constant__` deklariert und so in konstanten CUDA-Speicher auf der Grafikkarte platziert. Zum Anderen wurde der Datentyp von `int` auf `unsigned char` gesetzt, da ein Würfel nun mal nur 12 Kanten hat. Folglich wurde in der Tabelle die `-1` durch `255` ersetzt.

[220] Dazu später mehr im Abschnitt Speichermanagement.

```
1: j = partition_index * (N − 2)
2: for each layer in range(N − 2) do
3:     for each cube ∈ j^th − layer  in parallel do
4:         calculate_gradients()
5:         generate_cubecode()
6:         if 0 < cubecode < 255
7:             calculate_intersections()
8:         shift_cubecode()
9:     j = j + 1
```

Algorithmus 2 : Das Kernel *kernel_cubecode*. Es erhält als ersten Parameter den Partitionsindex *partition_index* und als zweiten Parameter die Anzahl der Slices N, die hochgeladen und bearbeitet werden. Auch hier wurden, wie bei Algorithmus 1, die Zeilen 1 und 2 gegenüber dem Code aus [Ulr+14a] detaillierter dargestellt. Während in Zeile 2 die Layers mit den lokalen Indizes der auf der Grafikkarte befindlichen Arrays arbeiten, müssen bei den Berechnungen der Schnittpunkte (x-, y- und z-Koordinaten), Gradienten, etc. die globalen Positionen der Würfel berücksichtigt werden. Dies soll mit der Variable j in den Zeilen 1, 3 und 9 verdeutlicht werden.

Während die for-Schleife innerhalb des Kernels an der y-Achse entlang von Unten nach Oben durch die jeweilige Partition iteriert, berechnet jeder Thread die Koordinaten des Vertex v_0 eines Würfels und vollzieht anhand dieses Orientierungspunktes seine Berechnungen:

```
idx_x  =  (threadIdx.x + blockIdx.x*blockDim.x) + pos_x
idx_z  =  (threadIdx.y + blockIdx.y*blockDim.y) + pos_z
```

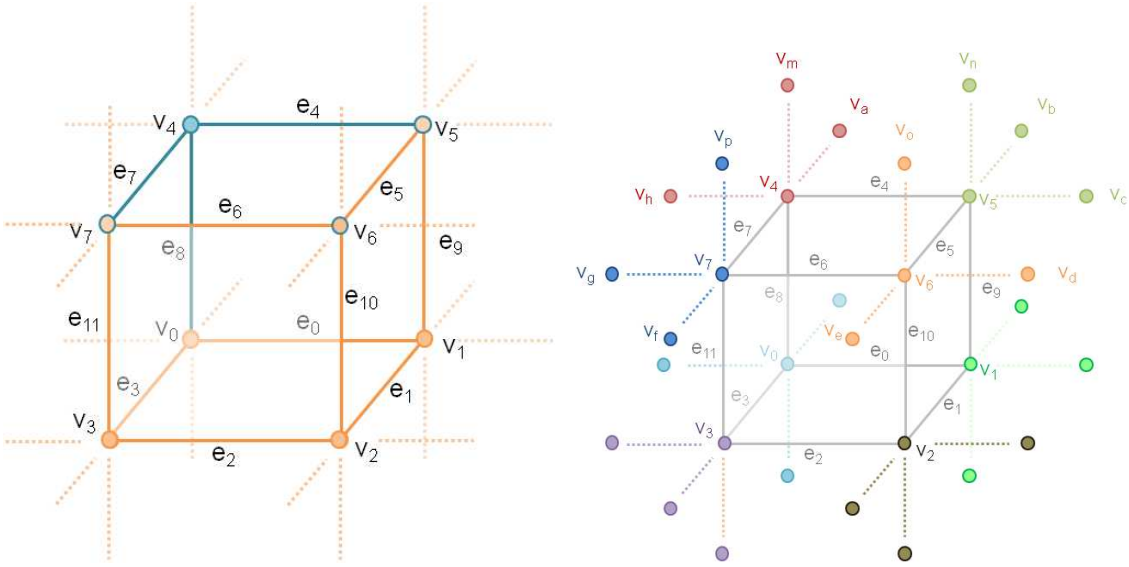
wobei

$$\texttt{pos\_x} = \begin{cases} \texttt{0} & \text{für } \texttt{threads\_main} \\ \texttt{threads\_main.x*blocks\_main.x} & \text{für } \texttt{threads\_right} \\ \texttt{0} & \text{für } \texttt{threads\_bottom} \\ \texttt{threads\_main.x*blocks\_main.x} & \text{für } \texttt{threads\_corner} \end{cases}$$

$$\texttt{pos\_z} = \begin{cases} \texttt{0} & \text{für } \texttt{threads\_main} \\ \texttt{0} & \text{für } \texttt{threads\_right} \\ \texttt{threads\_main.y*blocks\_main.y} & \text{für } \texttt{threads\_bottom} \\ \texttt{threads\_main.y*blocks\_main.y} & \text{für } \texttt{threads\_corner} \end{cases}$$

Wenn Vertex v_0 in Abbildung 17 im j-ten Slice des j-ten Layers liegt, dann berechnet Thread (`threadIdx.x`,`threadIdx.y`) den Vertex $v_0 = v_{\texttt{idx\_x},j,\texttt{idx\_z}}$.

Zu den Aufgaben eines jeden Threads gehört die Berechnung und Speicherung des Gradienten von v_4 (Algorithmus 2 Zeile 5). Dafür liest er die Isowerte von v_5, v_h, v_7, v_a, v_0 und v_m ein (siehe Abbildung 17 (b)). Ebenso berechnet und speichert er die oberen vier Bits des Cubes (Algorithmus 2 Zeile 6; Abbildung 17 (a)). Er kodiert also die Isowerte von v_4 bis v_7 und trägt diese in den Cubecode des Würfels ein. Die unteren vier Bits von v_0 bis v_3 wurden vom Thread selber im vorherigen Durchgang der for-Schleife berechnet. Damit ist der Cubecode vollständig und kann für die spätere Mesherzeugung ins Array geschrieben werden. Beinhaltet der Würfel laut Cubecode eine Isofläche,



(a) Die grundlegende Zuständigkeiten eines Threads. (b) Funktionsbeschreibung

Abbildung 17: Die Bearbeitung eines Cube durch das Kernel. Bild (a) zeigt die grundlegende Zuständigkeiten eines Threads. Er berechnet und speichert den Gradienten von v_4, die Schnittpunkte und deren zugehörigen Normalenvektoren der Kanten e_4,e_6 und e_7. So wird gewährleistet, dass die Threads beim Lesen und Schreiben der Array nicht kollidieren. Ebenso berechnet und speichert er, dargestellt durch die dunkelblauen Kreise, die oberen vier Bits des Cubecodes.
Das Bild (b) dient zur Funktionsbeschreibung. Es soll veranschaulichen, welche Daten der Thread liest und welche er schreibt. Beispielsweise werden für die Berechnung des Gradient von v_5 die Isowerte von v_c, v_4, v_6, v_b, v_1 und v_m eingelesen. In das passende Array wird der Gradient aber nicht geschrieben, da es nicht zu den Zuständigkeiten des Threads gehört. Dieser wird stattdessen temporär nur für die Normalenberechnung des Schnittpunktes von e_4 verwendet.
Die Vertices $v_0, \ldots, v_7$ sind mittels Formel (13) mit einem lokalen Index versehen und werden so vom Kernel adressiert.

so berechnet und speichert der Thread je nach Fall die Schnittpunkte der Kanten e_4, e_6 und e_7 (Algorithmus 2 Zeile 7 und 8; Abbildung 17 (a)). Hierfür müssen aber ebenfalls die dazugehörigen Normalenvektoren der Schnittpunkte berechnet werden. Dies bedeutet ein Einlesen des Gradienten von v_0 im Falle eines Schnittpunktes auf der Kante e_8. Der Gradient von v_0 wurde vom Thread selber ein Layer zuvor ermittelt. Bei den Kanten e_4 und e_7 müssen die Gradienten von v_5 und v_7 zusätzlich temporär berechnet werden. Sie werden aber nicht vom Thread gespeichert. Dies erledigen die anderen Threads im Block. Der Grund für diese Organisation des Berechnens und des Schreibens beziehungsweise Nicht-Schreibens der Gradienten unter anderem liegt im shared memory, welcher eine geringe Größe aufweist. Der globale Speicher hat mehr Kapazität und kann somit dem Speicherbedarf der vorgestellten Methode gerecht werden. Daher muss zusätzlich auf die klassischen Stolperfallen, wie beispielsweise die race condition,[221] geachtet werden. Sind alle Berechnungen geschehen, muss der Thread nur noch den Cubecode um vier Bits nach rechts shiften (und mit Nullen auffüllen) und ins Array an die entsprechende Stelle schreiben, damit dieser für das nächste Layer bereit steht. Dann kann der Thread sich in den darauffolgenden Schleifendurchlauf begeben.

[221] Siehe Kapitel 1.6.2 Seite 21.

Somit erschließt sich auch die Methode *kernel_cubecode_init()* in Zeile 2 aus Algorithmus 1 (Seite 51). Da das Kernel die Partitionen layerweise verarbeitet und in jedem Schleifendurchlauf davon ausgeht, dass der untere Teil des Layers (aus Sicht des Threads: der untere Teil des Würfels) schon bearbeitet wurde, muss, damit die Bearbeitung der Volumendaten starten kann, das allererste Slice, welches den Index 0 trägt, initialisiert werden.

Ausnahmebehandlungen Natürlich gibt es sprichwörtliche 'Randbedingungen'. So gibt es im Thread Ausnahmebehandlungen für Würfel, die am obersten und letzten Layer oder die an der linken, rechten, vorderen oder hinteren Seite des Volumendatensatzes liegen. Hier hilft es sich noch mal vor Augen zu führen, dass der Hybrid-Algorithmus mit globalen Konstanten gestartet wird und jeder Thread mit dem Orientierungspunkt v_0 arbeitet. Überprüft wird ob $x \in \{0, dimX - 2\}$ oder $y \in \{0, dimY - 2\}$ oder $z \in \{0, dimZ - 2\}$ ist. Je nach Fall werden vom Thread noch die Kanten e_5, e_9, e_6, e_{11}, e_{10} behandelt. Für die dann anstehenden Gradientenberechungen werden die benachbarten Isowerte mit den vorhandenen aufgefüllt (Zeile 4 und 7 in Algorithmus 2). So erhält der Vertex v_c, der am rechten Rand des rektliniearen Volumengitters nur theoretisch existiert, den Isowert von v_5. Gleiches passiert bei den Vertices v_b und v_n. Die Vertices v_d, v_e und v_o werden mit dem Isowert von v_6 befüllt und die Vertices v_f, v_g und v_p mit dem von v_7. Je nach Randlage des Würfels im Volumengitter dient der Isowert von Vertex v_4 als Ersatz für die nicht vorhandenen Isowerte von v_a, v_h und v_m.

Übergang zwischen den Partitionen Wie bewerkstelligt man den nahtlosen Übergang zwischen den Partitionen? Schließlich wird für die Gradientenberechnung der oberen Vertices ein Slice mehr als erdacht benötigt. Statt nun ein zusätzliches Slice mehr in den Grafikkartenspeicher zu laden – Grafikkartenspeicher ist knapp und somit als Programmier-Ressource kostbar – wird ein Slice weniger 'bearbeitet'. Man opfert es für die Gradientenberechnung und die Anzahl der Layers, die in einer Partition bearbeitet werden, sinkt von $N - 1$ auf $N - 2$. Die Partitionen müssen sich somit um zwei Slices überlappen. Was die Berechnung von $Partitions$ etwas erschwert. Sie ist nicht einfach mit $Partitions = (dimY/N)$ und $Remainders = dimY - (dimY/N) \cdot N$ zu bewältigen. Stattdessen muss für die Überlappung $N - 2$ anstelle von N genommen werden. Damit bei den erstellten Partitionen die letzte Partition diejenige ist, die die $Remainders$ enthält, wird in Formel (40) beim Zähler eine Eins abgezogen und dem Ergebnis eine Eins hinzuaddiert. Damit ist gewährleistet, dass gilt: $0 < Remainders \leq (N - 2)$.

$$Partitions = \frac{(dimY - 1)}{(N - 2)} + 1 \tag{40}$$

$$Remainders = dimY - (Partitions - 1) \cdot (N - 2) \tag{41}$$

Die äußere for-Schleife des MC-Moduls arbeitet also mit ganzen N Slices pro Partition (N Slices werden auch auf die Grafikkarte hochgeladen). Durch die Überlappung der Partitionen und der Opferung für die Gradientenberechnung werden aber im Kernel *kernel_cubecode* nur $N - 2$ Layer behandelt. Bei der letzten Partition werden dann nur noch $Remainders$ Slices hochgeladen.[222] Da das letzte Slice nicht mehr 'geopfert' werden muss, wird bei der Parameterübergabe eine Eins hinzuaddiert.

[222] Der Filereader registriert das Dateiende. Dieser geht übrigens für die Überlappungen der Partitionen nach jedem äußeren Schleifendurchlauf zwei Slices zurück und liest erneut N Slices ein.

Speichermanagement Weiterhin führt das 'Aneinanderkleben' der Partition zu einem höheren Verwaltungsaufwand der Arrays. Der Datentransfer zwischen Festplatte und Grafikkartenspeicher muss nur auf den Teil beschränkt werden, der wirklich aktualisiert werden muss. Wurden die Cubecodes in der for-Schleife des Kernels *kernel_cubecode* ein letztes Mal nach rechts geshiftet, können diese für die nächste Partition weiter verwendet werden. Es gibt also keine Freigabe und erneute Reservierung des Speichers. Stattdessen wird ein Transfer der Cubecodes des letzten Layers zum Anfang des Arrays und ein Memset am Arrayende getätigt. Gleiches geschieht bei den Arrays für die Schnittpunkte (samt Normalenvektoren) und Gradienten. Bei diesen wird das Memset mit dem Wert -1, bei den Cubecodes wird es mit Wert 0 ausgeführt. All diese Aufgaben werden im Kernel *kernel_arrangearrays()* erledigt (Algorithmus 1 Zeile 7).

Arrays	**Speicherbedarf (in Bytes)**
Slices	$(dimX \cdot dimZ) \cdot N$
Gradienten	$12 \cdot (dimX \cdot dimZ) \cdot N$
Schnittpunkte	$24 \cdot \big((3 \cdot N - 4) \cdot dimX \cdot dimZ - (dimZ + dimX) \cdot (N - 1)\big)$
Cubecodes	$(dimX - 1) \cdot (dimZ - 1) \cdot (N - 1)$
Meshdreiecke	$60 \cdot (dimX - 1) \cdot (dimZ - 1) \cdot (N - 1)$

Tabelle 3: Der Speicherbedarf der Arrays auf der Grafikkarte. $dimX$, $dimY$ und $dimZ$ ist die Dimensionsgröße des gerade bearbeitenden Volumens. N ist die gewählte Größe für die Partitionierung.

In der Tabelle 3 auf Seite 55 sind die benötigten Arrays und deren Speicherbedarf aufgelistet.[223] Die Angaben sind in Bytes. Dies ist der Tatsache geschuldet, dass die meisten Volumendaten, mit der der Algorithmus getestet wurde, aus der 'The Volume Library' [Roe12] stammen. Diese sind zumeist CT-Scans, deren Grau-Werte meist mit 8 Bits repräsentiert werden.[224] [225] Auch die Scans der Milz aus Kapitel 2, die einfach-gefärbt sind, können in Grau-Bilder konvertiert werden. Bei 16-Bit Datensätzen, wie höher-auflösende CT-Scans, muss die Speicherbedarfsangabe in Tabelle 3 für das Slices-Array verdoppelt werden. Der Datentyp für die Isowerte wird dann in der Implementierung des Parallel Marching Cubes-Modules von `unsigned char` auf `unsigned short` gesetzt. Das Slices-Array, das die Isowerte speichert, muss dann mit einem Speicheraufwand von $(dimX \cdot dimZ) \cdot N \cdot 2$ gekennzeichnet werden. Die übrigen Arrays müssen keine Anpassung erfahren. Deren Speicherbedarf bleibt gleich. Sind Schnittserien zweifach eingefärbt, können entweder mehrere 8-bitige Volumendaten (beispielsweise ein Rot-Kanal als eine Serie von Grau-Bildern) erzeugt werden oder es wird vorher beispielsweise vom RGB- in den HSV-Farbraum konvertiert.[226] [227]

Weiterhin ist in Tabelle 3 das Gradienten-Array aufgeführt. Dieses speichert die x-, y-, und z-Koordinaten als `float`-Datentyp. Daher werden insgesamt 12 Bytes pro Gradienten gebraucht.[228] Ähnliches gilt bei dem Array für die Schnittpunkte. Hier werden allerdings sechs `float`-Zahlen benötigt. Drei für den Schnittpunkt von Isofläche und Würfelkante und drei für dessen Normalenvektor. Das Array für die Meshdreiecke, welches von *kernel_generate* aufgefüllt wird, reserviert für jeden Würfel die

[223] Die Tabelle enthält gegen über den Angaben im [Ulr+14a] eine kleine Korrektur.
[224] Eigentlich werden die Grauwerte eines CT-Scans in 12 Bit kodiert, denn HU-Werte reichen typischerweise von -1024 HU bis 3071 HU und können so mit 12 Bit dargestellt werden (vgl. [Han09] Kapitel 2.1.3.2 Seite 13).
[225] Enthalten sind in der Bibliothek 36 8-Bit-Datensätze, 25 16-Bit-Datensätze und 2 RGB-Datensätze. Der Anteil der 8-Bit-Datensätze liegt somit bei 57%.
[226] Dabei ist der Kniff der OpenCV's `cvCvtColor()` [BD08; BK08; Lib] zu beachten: H geht nur von 0 bis 179.
[227] Neben OpenCV stehen auch Tools wie IrfanView, ImageMagick oder Fiji für eine Graukonvertierung zur Verfügung.
[228] Der Aufruf `sizeof(float)` gibt den Wert 4 zurück. Die Gradienten werden pro Voxel errechnet.

Möglichkeit fünf Dreiecke zu hinterlegen. Denn dies ist die maximale Anzahl, die eine Isoflächendefinition[229] innerhalb eines Würfel annehmen kann. Der Datentyp für dieses Array ist `unsigned int`, was ebenfalls aus vier Bytes besteht. Dieser Datentyp ist deshalb notwendig, weil die Meshdreiecke durch die Würfelkantenindizes definiert werden und letztere recht große Werte annehmen kann. Denn auf den Würfelkanten liegen gegebenenfalls errechnete Schnittpunkte (mit Isofläche), welche später die Meshvertices darstellen.

Adressierung Damit Arrays ausgelesen und befüllt werden können, müssen diese richtig adressiert sein. Dabei ist zwischen Arrays des Host und Arrays des Device zu unterschieden. Erstere beinhalten *alle* Daten und müssen mit einem globalen Index angesprochen werden. Letztere sind an die Partitionsgröße gebunden und benötigen einen lokalen Index.

Wie bei Formel (7) auf Seite 9 gesehen, lassen sich die Vertices der Volumendaten mit den globalen Index i durchnummerieren und mit Hilfe von Formel (24) in CUDA umsetzen. Dies kann auch mit den Würfeln geschehen. Seien $idx_x, idx_y, idx_z \in X$ (Formel (3)), dann wird der Index für den i_c-ten Cube wie folgt berechnet:

$$i_c := idx_x + idx_z \cdot (dimX - 1) + idx_y \cdot (dimX - 1) \cdot (dimZ - 1) \tag{42}$$

Diese Formel nummeriert nicht nur die Cubes innerhalb der Volumendaten durch, sie dient auch als Hilfe für die Adressierung innerhalb des Arrays in dem die Cubecodes gespeichert werden. Aus Formel (11) ist bekannt, dass $idx_y = j$ ist (mit $j \in \{\mathbb{N} \cup 0\} \wedge 0 \leq j \leq dimY - 1$). Daher lässt sich Formel (42) auch wie folgt definieren:

$$i_c := idx_x + idx_z \cdot (dimX - 1) + j \cdot (dimX - 1) \cdot (dimZ - 1) \tag{43}$$

Für das Cubecode-Array auf der Grafikkarte muss j durch einen *lokalen* Index j^l ersetzt werden. Wenn $j^l \in \{\mathbb{N} \cup 0\} \wedge 0 \ \leq j^l < N - 2$ des j^l-ten Layers innerhalb einer Partition ist, dann kann die Speicheradressierung des i_c^l-ten Cubecodes wie folgt definiert werden:

$$i_c^l := idx_x + idx_z \cdot (dimX - 1) + j^l \cdot (dimX - 1) \cdot (dimZ - 1) \tag{44}$$

Bei der Verwaltung der beiden Arrays für die Schnittpunkte verhält es sich ähnlich. Hierfür werden die Würfelkanten durchnummeriert, da die Schnittpunkte beziehungsweise die Meshvertices auf diesen liegen. Um die Indexierung der Würfelkanten innerhalb eines Array besser zu verstehen, hilft es die Abbildung 17 (Seite 53) zu betrachten. In der Abbildung, sei es (a) oder (b), ist die Indexierung als eine lokale zu verstehen. Die Kanten sind von e_0 bis e_{11} und die Vertices von v_0 bis v_7 durchnummeriert. Der Würfel wird mit dem Index des Vertex v_0 angesprochen. Wenn also v_0 den Index i hat und somit eigentlich v_i heißt, so wird der Würfel und damit auch die Kanten von e_0 bis e_{11} mit den zu i gehörenden Indizes idx_x, idx_z und j angesprochen. Für das Device-Array muss j nur noch entsprechend durch j^l ersetzt werden.

Bevor aber die Indexierung der Würfelkanten genauer erläutert werden kann, müssen vorher noch ein paar Variablen eingeführt werden. Diese helfen die Formel (49) etwas übersichtlicher zu gestalten. So soll $allx$ die Anzahl aller x-Kanten innerhalb eines Slices beschreiben:

$$allx := \Big((dimX - 1) \cdot dimZ\Big) \tag{45}$$

[229] Nach der look-up-Tabelle von Paul Bourke [Bou94].

Und $allxz$ die Anzahl aller x- und z-Kanten innerhalb eines Slices:

$$allxz := \Big(allx + \Big(dimX \cdot (dimZ - 1)\Big)\Big) \tag{46}$$

Die Anzahl aller y-Kanten innerhalb eines Layers (was gleich der Anzahl aller Vertices innerhalb eines Slices entspricht) soll mit $ally$ beschrieben werden:

$$ally := dimX \cdot dimZ \tag{47}$$

Weiterhin sei $allxyz$ die Anzahl aller x-, y- und z-Kanten innerhalb eines j^l-ten Layer (vgl. Formel (12) Seite 11), aber ohne den 'Deckel' $slice_{j^l+1}$:

$$allxyz := (allxz + ally) \tag{48}$$

Mit diesen Variablen, mit einem lokalen Index $j^l \in \{\mathbb{N} \cup 0\} \wedge 0 \leq j^l < N - 2$, den Indizies idx_x und idx_z und mit $h \in \{0, \ldots, 11\}$ lässt sich eine Funktion definieren, mit der die Kanten innerhalb der Partition adressiert werden können:

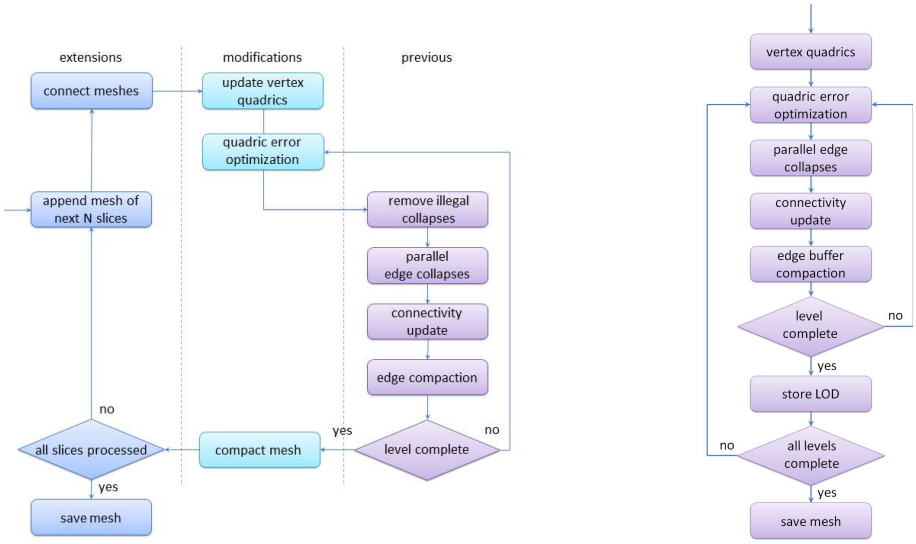
$$edges(idx_x, idx_z, h) = \begin{cases} idx_x + idx_z \cdot (dimX - 1) & \text{für } h = 0 \\ (idx_x + 1) + idx_z \cdot dimX + allx & \text{für } h = 1 \\ idx_x + (idx_z + 1) \cdot (dimX - 1) + j^l \cdot allxyz & \text{für } h = 2 \\ idx_x + idx_z \cdot dimX + allx + j^l \cdot allxyz & \text{für } h = 3 \\ idx_x + idx_z \cdot (dimX - 1) + ally + allxz + j^l \cdot allxyz & \text{für } h = 4 \\ (idx_x + 1) + idx_z \cdot dimX + allx + ally + allxz + j^l \cdot allxyz & \text{für } h = 5 \\ idx_x + (idx_z + 1) \cdot (dimX - 1) + ally + allxz + j^l \cdot allxyz & \text{für } h = 6 \\ idx_x + idx_z \cdot dimX + allx + ally + allxz + j^l \cdot allxyz & \text{für } h = 7 \\ idx_x + idx_z \cdot dimX + allxz + j^l \cdot allxyz & \text{für } h = 8 \\ (idx_x + 1) + idx_z \cdot dimX + allxz + j^l \cdot allxyz & \text{für } h = 9 \\ (idx_x + 1) + (idx_z + 1) \cdot dimX + allxz + j^l \cdot allxyz & \text{für } h = 10 \\ idx_x + (idx_z + 1) \cdot dimX + allxz + j^l \cdot allxyz & \text{für } h = 11 \end{cases} \tag{49}$$

Für das Host-Array muss natürlich j^l durch j ersetzt werden.

Das Kernel *kernel_cubecode* verwendet Formel (49), um die berechneten Schnittpunkte in das Schnittpunkte-Array zu schreiben. Das Kernel *kernel_generate* berechnet mit Hilfe dieser Formel die Würfelkantenindizes, welche die für die Definition der Meshdreiecke verwendet werden.[230] Bei den Arrays für die Meshdreiecke werden wieder die Formeln (42) und (43) für die Indexierung verwendet. Hier muss noch zusätzlich die Adresse entsprechend mit 15 multipliziert werden.

Übergabe Mit den Arrays für die Schnittpunkte und Meshdreiecke stehen die nötigen Informationen zur Verfügung, die in ein IFS (Index Face Set)[231] oder in eine OBJ-Datei geschrieben werden können. Man kann diese aber natürlich auch dem zweiten Modul zum Simplifizieren übergeben. Da aber nicht auf jeder Würfelkante auch ein Meshvertex liegt und nicht durch alle Würfel eine Isofläche geht (und wenn, dann nicht unbedingt der Art dass sie mit fünf Dreiecken definiert ist), sind die Arrays noch

[230] Zur Erinnerung: Schnittpunkte liegen auf Würfelkanten.
[231] Vgl. Seite 12.



(a) Das Simplifizierungsmodul (b) Instant LOD-Simplifizierung

Abbildung 18: Die Programmschritte im Vergleich. In Bild (a) erkennt man, dass die äußere for-Schleife über die Spalten 'extensions' (neu hinzugekommen), 'modifications' (angepasst) und 'previous' (gleich geblieben) geht. Die innere for-Schleife befindet sich nur in 'previous'. Eine innere und äußere for-Schleife sind auch in Bild (b) zu erkennen. *(Bild (a) wurde mit* `PowerPoint` *selbst erstellt und ist an Figure 4 aus [Ulr+14a] angelehnt. Bild (b) wurde mit* `PowerPoint` *selbst erstellt und ist an Figure 4 aus [GDG11] angelehnt.)*

recht aufgebläht. Daher werden sie vorher noch verdichtet und degenerierte Dreiecke entfernt, bevor sie zur Simplifizierung übergeben werden.[232]

3.2.2 Das Parallele Simplifizierungsmodul

Das Simplifizierungsmodul basiert auf der von Grund et al. [GDG11] vorgestellten Methode, welche wiederum die *Quadric Error Metric* (QEM) von Garland und Heckbert [GH97; GH98; Gar] verwendet. Wie schon in Ulrich et al. [Ulr+14a] Kapitel 5 ab Seite 364 beschrieben, ist es dem Simplifizierung-Modul egal auf welche Weise das Mesh erzeugt wurde. Sei es durch Marching Cubes, Marching Tetraeders oder Dual Marching Cubes, es erwartet als Eingabe ein Mesh (bestehend aus Vertices und Kanten).

Abbildung 18 (a) veranschaulicht die Vorgehensweise des Simplifizierungsmoduls. Das Mesh, welches aus den N Slices rekonstruiert wurde, wird dem Modul übergeben und falls schon ein bereits simplifiziertes Mesh aus der vorherigen Partition vorhanden ist, mit diesem verbunden.

Wie auch bei Grund et al. [GDG11] bestehen die ersten Schritte darin, die Datenstruktur aufzubauen und die Vertex-Quadriken zu berechnen. Das heißt, Kantenduplikate werden identifiziert (und später eliminiert) und Randkanten als solche markiert. Bei den Randkanten wird zusätzlich auch der gegenüberliegende Vertex notiert. Diese Information wird später für die Berechnung des QEM (siehe [GH98]) benötigt.[233] Im Anschluss werden die QEMs ihrer Lage im Mesh entsprechend berechnet beziehungsweise zu den bereits berechneten QEMs werden neue hinzugefügt. Die Vertices, welche

[232] Vgl. [Ulr+14a] Seite 364.
[233] Genaueres hierzu ist dem Kapitel 3.4 aus [Gru13] zu entnehmen. Vgl. [Gru13] Seite 102 Absatz 1, Seite 105 Absatz 3.

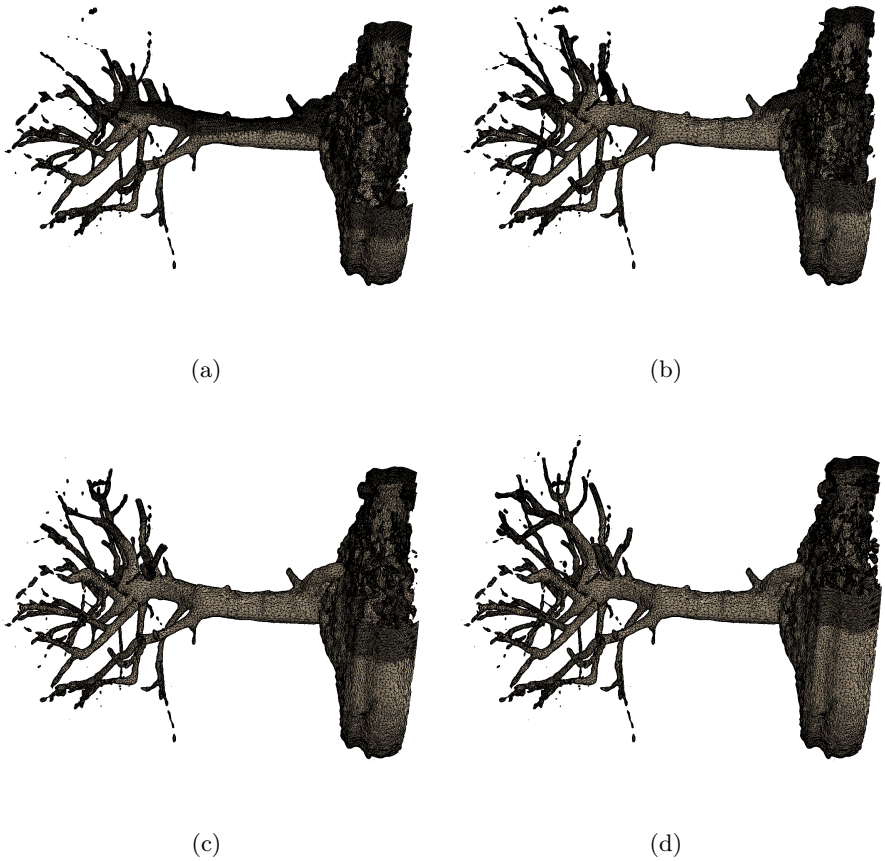
sich genau auf der Partitionsgrenze[234] befinden, werden markiert, damit sie von einem edge-collapse ausgeschlossen werden. Denn eine Kante darf nur kollabieren, wenn sie nicht mit der Partitionsgrenze verbunden ist und somit auch keine ihrer Vertices markiert ist. Diese Einschränkung ist notwendig, da ein Mesh aus der aktuellen Partition, das gerade behandelt wird, sich mit dem nachfolgenden Mesh aus der nächsten Partition ein Slice aus dem Volumendaten teilt. Die aller-obersten Kanten und Vertices des aktuellen Meshes sind also identisch mit den aller-untersten Kanten und Vertices des folgenden Meshes. Diese Struktur muss für das Zusammenfügen der Isoflächen erhalten bleiben. Dies bedeutet zusätzlich auch, dass direkte Nachbarn zur Partitionsgrenze nicht kollabieren dürfen, da ihre 'lokale minimale Fehlerwerte' noch nicht bekannt sind. Hierfür reicht es die Fehler der entsprechenden Kanten in einem äußeren Schleifendurchlauf zwischenzeitlich auf -1 zu setzen und so zu markieren, da negative Simplifizierungsfehler nicht möglich sind.[235] Auf diese Weise wird auch das Kollabieren von Nachbarkanten verhindert, da nur Kanten mit einem lokal minimalen Fehler kollabiert werden.

Durch die Markierung der Vertices zur Partitionsgrenze unterscheidet sich der Schritt 'update vertex quadrics' in Abbildung 18 (a) von dem Schritt 'vertex quadrics' in Abbildung 18 (b). Der gleiche Grund liegt beim Schritt 'quardic error optimization' vor.[236] Dieser ist nun in 'quadric error optimization' und 'remove illegal collapses' feiner aufgeteilt. Während in 'quadric error optimization' aus Bild (a) nun zusätzlich die Fehlerwertemanipulationen der anliegenden Kanten an der Scanline bei den Berechnungen mit beachtet werden, sind die Schritte von 'remove illegal collapses' bis 'edge compaction' mit dem *Instant Level-Of-Detail*-Algorithmus (Abbildung 18 (b)) identisch. Der innere Schleifendurchlauf beginnt mit dem Schritt 'quadric error optimization' auf dem der Schritt 'remove illegal collapses' folgt. Im Letzteren wird sichergestellt, dass in der direkten Nachbarschaft einer Kante, die möglicherweise kollabieren kann, keine weitere Kante mit niedrigeren Fehlerwerten existiert. Wäre dies so, würde solch eine Kante von einem edge-collapse ausgeschlossen.[237] Im nächsten Schritt werden die *full edge-collapses* parallel ausgeführt,[238] [239] ein neuer Vertex hinzugefügt und dementsprechend die Datenstruktur, mit der enthaltenen Verwaltung der Kanten, in 'connectivity update' aktualisiert. Der letzte Schritt, der sich in der Spalte 'previous' befindet und ebenfalls mit Hilfe von CUDA parallelisiert wurde, verdichtet die zuvor aufgebaute Datenstruktur. Hier werden „um Performanzeinbußen vorzubeugen [...] alle ungültigen Kanten aus der Datenstruktur entfernt".[240] Hat die Überprüfung im Anschluss ergeben, dass keine weiteren Kanten zur Simplifizierung vorhanden sind, kann die innere Schleife verlassen werden. Da aufgrund der edge-collapses in der Datenstruktur durch das Verschwinden der Vertices und der Kanten Lücken entstanden und ebenso noch Dreiecke definiert sind, die nicht mehr existieren, wird im Schritt 'compact mesh' die Datenstruktur diesbezüglich aufgeräumt. Ebenso erfährt die look-up-Tabelle, in der die markierten Vertices verwaltet werden, eine Aktualisierung. In Abbildung 19 auf Seite 60 ist die soeben beschriebene Verzahnung von Rekonstruktions- und Simplifizierungsmodul noch mal grafisch dargestellt.

Der in Ulrich et al. [Ulr+14a] vorgestellte Hybrid-Algorithmus benötigt gegenüber Attali et al. [ACE05] kein time-lag, da dieser nur die edge-collapses hinauszögert, die noch nicht ausführbar sind. Da die

[234] Man könnte diese Stelle auch als Sweepline bezeichnen.
[235] Hierbei handelt es sich, wie bei dem Schnittpunkte-Array im Kapitel 'Speichermanagement' auf Seite 54, um eine künstliche Markierung.
[236] Gemeint ist der Schritt aus Abbildung 18 (b).
[237] Vgl. [Gru13] Seite 106 Absatz 2.
[238] Im Gegensatz zum full edge-collapse wird bei einem *half edge-collapse* ein Vertex der kollabierenden Kante behalten und dessen Position auch nicht verändert. Diese Methode führt aber manchmal zu nicht-mannigfaltigen Meshstrukturen.
[239] Vgl. [Gru13] Seite 100 Kapitel 3.4.2.
[240] Siehe [Gru13] Seite 107 Absatz 3.

edge-collapses lokal beschränkt sind, ist das Mesh von gleicher Qualität, wie bei einer Simplifizierung mit globalen Operationsreihenfolge (solange die lokale Reihenfolge unverändert bleibt). Dies wird erreicht, indem alle Operationen an der Partitionsgrenze, wie oben erwähnt, durch den illegalen Fehlerwert von -1 blockiert werden. Mit zusätzlicher Erzwingung der lokalen korrekten Reihenfolge nimmt der Simplifizierungsfehler automatisch zu aktuellen Partitionsgrenze ab. Das Ergebnis-Mesh ist somit identisch zu einem Mesh, welches komplett und in einem Rutsch simplifiziert wurde und bei dessen Simplifizierung die Operationen zur Speicherbedarfsminimierung sobald als möglich ausführt.



(a) (b)

(c) (d)

Abbildung 19: Die Verzahnung von Rekonstruktion und Simplifizierung. Hier ist die Bearbeitung des Bonsai-Modells aus [Roe12] bei einem Isowert 50 zu sehen. In den Bilder (von a) nach d) gelesen) erkennt man, wie das Mesh allmählich mit voranschreitender Sweepline (Partitionsgrenze) simplifiziert wird, ohne dabei ein *time-lag* zu verwenden). Ebenfalls auf den Bilder zu erkennen, ist die Zunahme der Meshkantenanzahl in Richtung Sweepline.

Die obige Behauptung der identischen Meshes kann bewiesen werden, indem man die Operationen der sequenziellen Simplifizierung und die des parallel arbeitenden Simplifizierungsmoduls analysiert. Bei der klassischen sequenziellen Simplifizierung wird eine Kante solange mit dem kleinsten Fehlerwert kollabiert bis ein gewisser Schwellenwert erreicht ist. Als Folge erhöhen sich die Fehlerwerte der Nachbar-Kanten. Dies impliziert, dass zu einem gewissen Zeitpunkt der hier vorgestellte Hybrid-Algorithmus ebenfalls diese Kante kollabieren lässt. Wenn wiederum beim Simplifizierungsmodul eine Kante kollabiert, werden die Fehlerwerte der Nachbar-Kanten ebenfalls erhöht. Dies bedeutet, dass die sequenzielle Methode hier ebenfalls ein edge-collapse ausführt bevor

Nachbar-Kanten folgen können. Dabei verzögert das zeitlich begrenzte Setzen der Fehlerwerte von Kanten an der Sweepline auf -1 nur die Verarbeitung an dieser Stelle. Die Nachbar-Kanten an der Sweepline können *nur* vorerst nicht kollabieren. Zusammengefasst lässt sich sagen, dass die sequentielle Methode die lokalen Minima der errechneten Kantenfehlerwerte nacheinander bearbeitet, die parallele Methode dagegen bearbeitet diese alle gleichzeitig.

3.2.3 Resultate

Die Evaluierung wurde mit einem Rechner durchgeführt, der einen Intel Core i7 CPU, 6 GB Arbeitsspeicher und eine NVIDIA GTX 580 besaß. Die Implementierung der Module wurde mit CUDA 5.0 umgesetzt. Wie schon weiter oben erwähnt, sind die Modelle, welche für die Tests verwendet wurden, aus der 'The Volume Library' [Roe12] entnommen. Diese sind in Tabelle 4 nochmals aufgelistet. Das Modell Porsche ist das größte und verfügt über 198.434.379 Voxel, während die anderen beiden Modelle - bonsai #2 und CTA head - mittlerer Größe sind und aus ca. 3/4 weniger Voxel als das Porsche-Modell bestehen.

Modell	**dim(x,y,z)**	**# Voxel**
bonsai #2	$512 \times 189 \times 512$	49.545.216
CTA head	$512 \times 120 \times 512$	31.457.280
Porsche	$559 \times 347 \times 1023$	198.434.379

Tabelle 4: Die Test-Modelle aus [Roe12]. Zu sehen sind hier die Dimensionen und die Anzahl der Voxel. Als Vorverarbeitung für den Hybrid-Algorithmus wurde das pvm-Dateiformat mit Hilfe des Versatile Volume Viewers [Sou] in ein raw-Format umgewandelt. Dies bedeutet, ein Voxel kann einen Wert zwischen 0 und 255 annehmen und hat somit einen Bedarf an einem Byte des Speicherplatzes. *(Die Tabelle enthält gegenüber der Tabelle 2 aus [Ulr+14a] leichte Änderungen.)*

Die Wahl fiel auf diese drei Modelle, da zu einem gezeigt werden sollte, dass die Hybrid-Methode jede Art von Daten bearbeiten kann, gleich ob es sich um medizinische Inhalte handelt oder nicht (vgl. bonsai #2 und CTA head). Zum anderem sollte gezeigt werden, dass die Methode mit großen Volumendaten umgehen kann (vgl. Porsche).

In Tabelle 5 werden die Größen der ausgewählten Modelle noch etwas genauer beschrieben. Einmal sieht man die Spalte der 'crossed cubes'. Sie gibt den prozentualen Anteil und die Anzahl der Würfel wieder, die sich mit einer Isofläche schneiden. Weiterhin ist in der letzten Spalte die Anzahl der Dreiecke des Meshes beim jeweiligen Isowert vor dem Simplifizierungsprozess angegeben. Man erkennt zudem, dass die Anzahl der Würfel ungefähr halb so groß ist wie die Anzahl der Meshdreiecke.

Für die Zeitmessungen wurden 12 Slices ($N = 12$) pro Partition verwendet. Die Rechenzeiten werden in Tabelle 6 aufgelistet. Dort zusehen sind die Anzahl der Dreiecke *nach* der Simplifizierung und die Performanz der beiden Module. Letztere wird in den entsprechenden Spalten 'extr.' und 'simp.' angegeben. Man erkennt, dass die Rekonstruktionsdauer (Tabelle 6 2. Spalte) fast konstant zur Volumengröße ist und sich nur leicht mit der Anzahl erzeugten Dreiecke (Tabelle 5 letzte Spalte) erhöht. Die Performanz der Extraktion der Isoflächen reichte bei den Tests von 12,0 Millionen (CTA head, 60) zu 12,9 Millionen (Porsche, 14) Würfel pro Sekunde. Die Geschwindigkeit mit der die Dreiecke erzeugt wurden, reichte von 174.000 (bonsai #2, 25) bis 2,24 Millionen (CTA head, 60) Dreiecke pro Sekunde. Der Speicherbedarf hängt stark von der Volumengröße und den Gradienten der aktuellen Partition ab.

Modell	crossed cubes		# Dreiecke
	%	#	
bonsai #2 (20)	4,49	2.203.645	4.405.952
bonsai #2 (25)	2,29	1.125.808	2.252.046
bonsai #2 (50)	0,67	329.858	658.158
CTA head (50)	3,08	956.441	1.913.256
CTA head (60)	9,53	2.961.041	5.878.764
CTA head (250)	2,24	694.745	1.392.432
Porsche (14)	2,40	4.744.499	9.580.084

Tabelle 5: Die relative und absolute Anzahl an Würfel, die eine Iso-Oberfläche beinhalten. Der jeweilige verwendete Isowert zur Rekonstruktion ist hinter der Modellbezeichnung in runden Klammern angegeben. Weiterhin in der letzten Spalte zu sehen, die Anzahl der Dreiecke des Meshes nach der Rekonstruktion und *vor* der Simplifizierung. *(Die Tabelle enthält gegenüber der Tabelle 3 aus [Ulr+14a] leichte Änderungen.)*

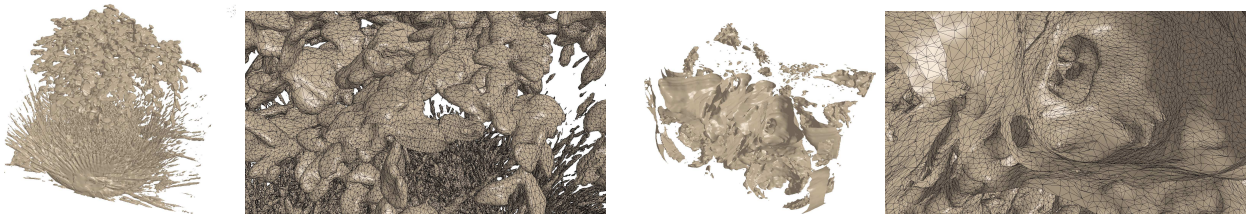
Modell	extr.	simp.	# Dreiecke	Speicher
bonsai #2 (20)	4,09s	5,74s	4.396.060	480,5
bonsai #2 (25)	3,99s	2,86s	2.245.412	430,6
bonsai #2 (50)	3,78s	0,95s	658.158	394,7
CTA head (50)	2,58s	1,19s	1.845.976	423,4
CTA head (60)	2,62s	5,31s	1.216.734	514,2
CTA head (250)	2,58s	1,37s	1.392.432	411,5
Porsche (14)	15,34s	21,12s	4.140.690	923,2

Tabelle 6: Die Rechenzeiten für die Rekonstruktion (extr.) und die Simplifizierung (simp.). Die zwei Spalten rechts enthalten die Anzahl der Dreiecken nach der Simplifizierung und den maximalen Speicherbedarf des Hybrid-Algorithmus in MBytes. *(Die Tabelle enthält gegenüber der Tabelle 4 aus [Ulr+14a] leichte Änderungen.)*

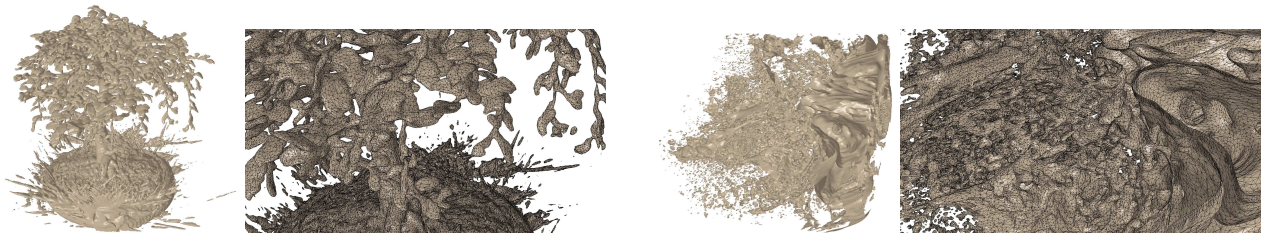
Ebenfalls zu erkennen ist, dass die Art der Daten keinen signifikanten Einfluss auf den Speicherbedarf oder auf die Rechenzeiten hatte. Man sieht aber auch, dass sowohl der prozentuale Anteil als auch die Verteilung der 'crossed cubes' in den Volumendaten einen Einfluss auf die Laufzeit und den Speicherbedarf haben.

Die erzeugten Isoflächen sind in Abbildung 20 auf Seite 63 und in Abbildung 21 auf Seite 64 dargestellt. Mit diesen Abbildungen und den Tabellen 5 und 6 lässt sich erkennen warum die Simplifizierung bei dem Modell bonsai #2 bei unterschiedlichen Isowerten unterschiedliche Performanz aufweist. Wie in Abbildung 20 zu sehen ist, sind die Artefakte, die von dem Sockel ausgehen, beim Isowert 20 am stärksten vorhanden. Hinzukommt, dass das Simplifizierungsmodul bei diesem Modell grundsätzlich mit sehr stark gekrümmten Flächen, welche vornehmlich durch Blätter zustande kommen, konfrontiert ist. Dementsprechend ist die Simplifizierung bei dem bonsai #2 in Abbildung 20 (a) am langsamsten und am wenigsten effektiv. Das Verhältnis zwischen dem simplifizierten Mesh und dem rekonstruiertem liegt hier bei 0,9978, also bei fast 100%.

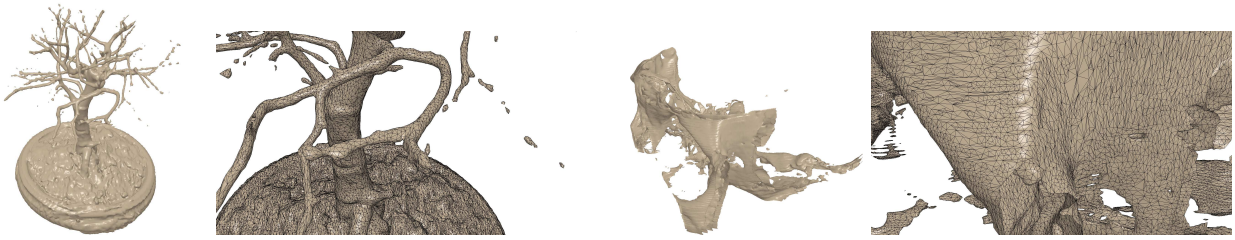
Bemerkenswert ist auch das Verhältnis zwischen Dreieckanzahl vor und nach der Simplifizierung bei dem Modell CTA head, welches mit einem Isowert von 60 extrahiert wurde (Abbildung 20 (e)). Diese liegt hier bei 0,21, also bei ungefähr 20%. Dies ist das beste Ergebnis des Simplifizierungsschrittes, während bonsai #2 (20) das schlechteste darstellt. Auch ist beim Modell CTA head anhand der



(a) bonsai #2 - iso 20



(b) bonsai #2 - iso 25



(c) bonsai #2 - iso 50

(d) CTA head - iso 50

(e) CTA head - iso 60

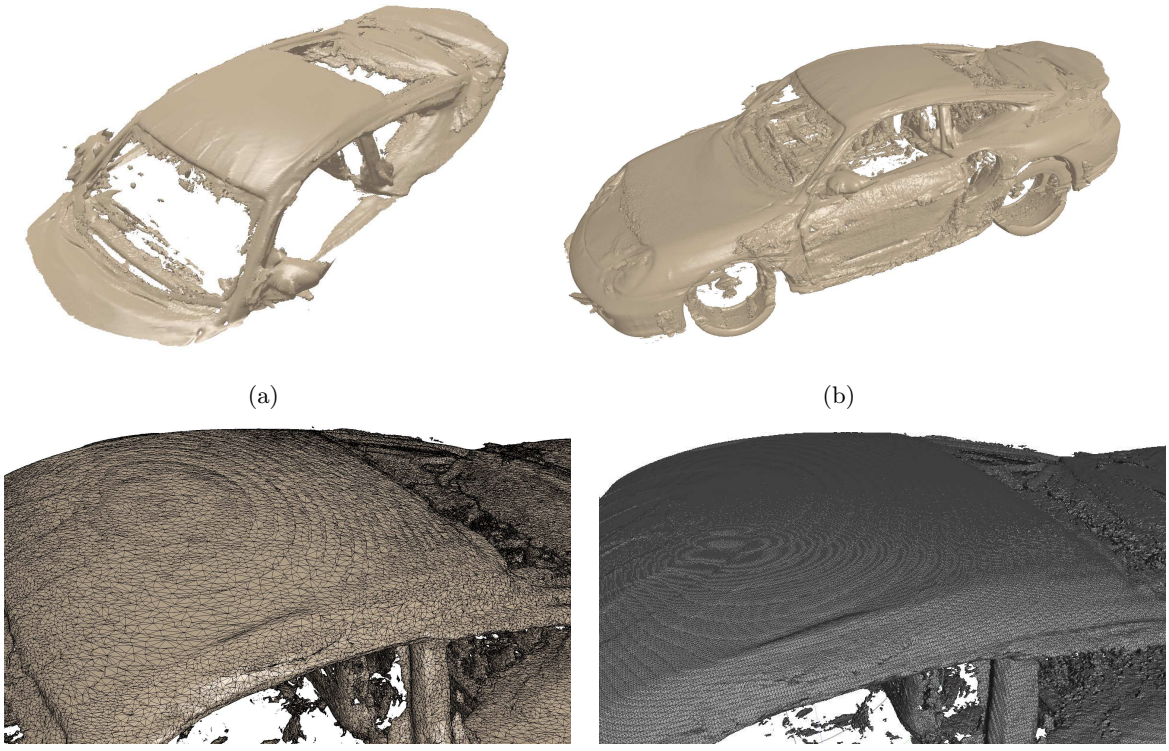
(f) CTA head - iso 250

Abbildung 20: Die Rekonstruktionsergebnisse von CTA und Bonsai. Links zu sehen, die generierten Bilder der extrahierten und simplifizierten Meshes des Modells bonsai #2 mit den Isowerten 20, 25 und 50. Auf der rechten Seite befinden sich die Bilder zum CTA head mit den Isowerten 50, 60 und 250. Neben jedem Modell in ganzer Ansicht ist ebenfalls rechts daneben ein größerer Ausschnitt zu sehen. In diesem ist zusätzlich das entsprechende Meshgitter (wire frame) sichtbar.

Tabellen zu erkennen, dass sehr stark gekrümmte Flächen Auswirkungen auf die Performanz des Simplifizierungsmoduls haben. So sind die Rechenzeiten des Simplifizierungsverfahrens bei den CTA head-Modellen mit dem Isowerten 50 und 250 deutlich kürzer.

Die Simplifizierung fand mit bis zu 258.000 edge-collapses pro Sekunde statt. Um die Hybrid-Methode gegenüber dem Tandem-Algorithmus von Attali et al. [ACE05] vergleichbar zu machen, wurde unter anderem das Modell bonsai #2 gewählt. Die Rekonstruktion mit dem Isowert 25 (Abbildung 20 (b)) weist ähnliche Charakteristika auf wie der Old Bone aus Attali et al. [ACE05] (siehe [ACE05] Seite 144 Abbildung 5 untere Zeile) und die Rekonstruktion mit dem Isowert 20 (Abbildung 20 (a)) gleicht dem Young Bone (siehe [ACE05] Seite 144 Abbildung 5 obere Zeile). Dabei ist das Bonsai-Modell dreimal so groß wie Young Bone oder Old Bone. Zwar waren die Modelle aus Attali et al. [ACE05] für einen direkten Vergleich nicht zugänglich. Aber, auf unserem System benötigte die nachgestellten Verfahrensweise von Attali et al. zirka 60 Sekunden für den bonsai #2 (25) und ungefähr 125 Sekunden für bonsai #2 (20), so dass unser Algorithmus 8,8 beziehungsweise 12,7 Mal schneller ist.[241] [242]

[241] Vgl. [Ulr+14a] Kapitel 6.1 Seite 367 letzter Satz.
[242] Die sequenzielle Simplifizierung wurde mit QSlim [Gar04] umgesetzt.



(a) (b)

(c) (d)

Abbildung 21: Die Rekonstruktion und Simplifizierung des Modells Porsche. In (a) ist die abwechselnde Rekonstruktion und Simplifizierung dargestellt. Daneben, unter (b), sieht man das Endergebnis. Während ganz rechts, unter (c), eine Nahaufnahme des Endresultates zu sehen ist. Hier ist ebenfalls das Meshgitter wieder sichtbar. Bild (d) wurde nachträglich hinzugefügt. Es zeigt zum Vergleich das Modell mit eingeblendeten Meshgitter ohne Simplifizierungsschritt.

Bei dem Modell Porsche (Abbildung 21) konnte ebenfalls ein gutes Ergebnis beobachtet werden. Es verfügt vorwiegend über glatte, große Flächen und zeigt, dass die vorgestellte Hybrid-Methode mit großen Volumendaten sehr gut umgehen kann. Der Speicherbedarf für die Erstellung dieses Modells ist in Abbildung 22 veranschaulicht. In Anbetracht, dass die NVIDIA GTX 580 über nicht genügend Grafikkartenspeicher verfügte, um das Modell komplett zu bearbeiten, wurde dieses entsprechend partitioniert. Die Partitionen mit 10 Layers sind auf der unteren Achse eingetragen. Zu jeder Partition wurde der maximale Speicherbedarf während der Rekonstruktion und Simplifizierung und die Anzahl der Dreiecke als fertiges Endergebnis der abgearbeiteten Partitionen aufgezeichnet. Die daraus resultierenden Linien geben zum einem wieder, dass der Gesamtspeicherbedarf mit der Anzahl der erstellten Meshdreiecke ansteigt und zum anderem, dass dieser wiederum von den Datenstrukturen, die zur Verarbeitung benötigt werden, dominiert wird.

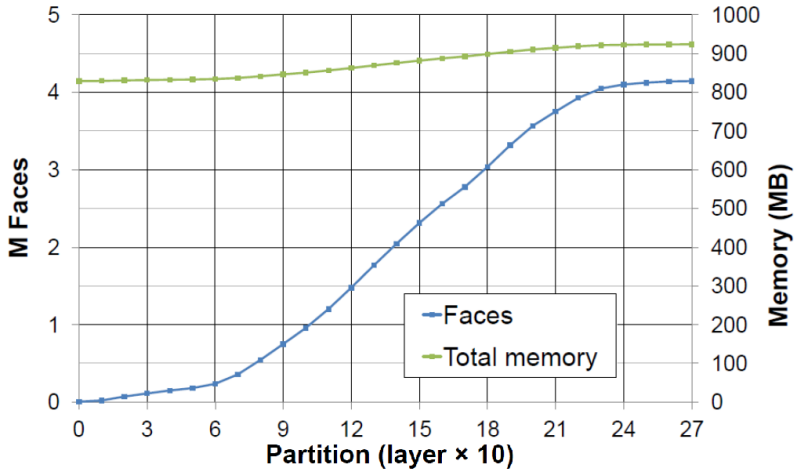


Abbildung 22: Zu sehen ist der Speicherbedarf und die Anzahl der Faces (der Dreiecke), die bei der Bearbeitung der Porsche-Modells benötigt werden.

Weiterhin soll angemerkt sein, dass die Hybrid-Methode [Ulr+14a] ein Mesh gleicher Qualität erstellt wie die LOD-Methode [GDG11]. Beide Simplifizierungsmethoden wurden damals verglichen. Für die kleineren Modelle war die Rekonstruktion, bis auf Floating Point-Rundungsfehler, gleich. Größere Modelle wie CTA head (60) und Porsche passten damals nicht in den Grafikkartenspeicher der GTX 580 und konnten daher von der Methode aus [GDG11] nicht bearbeitet werden. Auch wurde mit verschiedenen *error-threshold* experimentiert. Im Vergleich zu Attali et al. [ACE05] ist die *directional bias*[243] fast Null und bedarf keines time-lags.[244]

3.2.4 Fazit

Bei der soeben vorgestellten Methode ermöglicht das Ersetzen des time-lags durch das Blockieren beziehungsweise Verzögern der edge-collapses eine Reduzierung des Speicherbedarfs und der Rechenzeit. Durch die Parallelisierung auf kleinster Ebene bei beiden Modulen ist eine Bearbeitung großer Volumendaten in nur wenigen Sekunden möglich. Zusätzlich kann sie von den Weiterentwicklungen bei Grafikhardware oder anderen parallelen Systemen profitieren.

[243] Der Begriff 'directional bias' lässt sich mit gerichtete Verzerrung übersetzen.
[244] Vgl.[Ulr+14a] Kapitel 6.3 Seite 367.

Durch das Garantieren der lokalen Reihenfolge der edge-collapses, wie sie bei der sequenziellen Simplifizierung der Fall ist, wird die Erstellung eines Meshes von gleicher Qualität, wie bei einer nachgelagerten (Offline-) Simplifizierung, ermöglicht. Dies impliziert auch, dass keine Artefakte an den Partitionsgrenzen entstehen.

Durch die Hybrid-Eigenschaft der Methode von Ulrich et al. [Ulr+14a] kann das hier vorgestellte Simplifizierungsverfahren mit jeder anderer Art von Meshgenerierung arbeiten und ebenso umgekehrt kann das hier vorgestellte Rekonstruktionsverfahren das Mesh (zum Beispiel in Form einer IFS- oder OBJ-Datei)[245] an jede andere Methode (zum Beispiel in Form eines MeshLab-Filters oder der Methode von Lindstrom [Lin00]) zur Weiterverarbeitung geben. Letzteres ist für die Pipeline (Kapitel 1.7) sehr nützlich. Denn dies erlaubt eine sehr effiziente Isowertbestimmung, die oftmals mit Medizinern zusammen in mehreren Schleifendurchläufen ausgelotet werden muss. Weiterhin gestattet die geschickte Verwendung von Layers und Slices Serienschnitte im größeren Umfang zu bearbeiten.

Auch kann das Parallele Marching Cubes-Modul dahingehend erweitert werden, dass es sehr große gleichmäßige Volumendaten beliebig großer Dimension (beispielsweise geophysikalische Daten) verarbeiten kann. Hierfür müssen vom dem Kern-Modul die Volumendaten zuvor der x- bzw. der z-Achse entlang ebenfalls partitioniert werden.

[245] Vgl. Seite 12.

4 Anwendungen & Ausblick

Die im vorherigen Kapitel präsentierten Methoden [Ulr+14b; Lob+17; Ulr+14a] haben für die Herstellung der immunhistologischen 3D-Modelle mehrere Vorteile.

Wie im Kapitel 'Pipeline' (Seite 25) schon ersichtlich wurde, kann es passieren, dass der Prozess (Abbildung 4 Seite 26) mehrmals die Schleifen durchläuft. Hierfür sind das Ausloten des richtigen Isowertes und die übrigen Parameter[246] als Hauptursache zu nennen. Durch die Parallelisierung – mit CUDA, OpenMP, und OpenCL – und durch die geschickte Auswahl an mathematischen Methoden – in Form einer dünn besetzen Matrix und dem LSMR – wurde die Generierung der Meshes erheblich verkürzt. Dies ermöglichte es zusammen mit den Medizinern die Auswahl von mehreren ROIs, die gleich 'vor Ort' ausgerichtet wurden. Die Laufzeit solcher Software-Lösungen ist besonders kritisch für ein effizientes Arbeiten.

Ein weiterer Vorteil besteht in der Wahl der IVR-Methode Marching Cubes. Wurden die ROIs noch anfänglich durch direkte Volumenmodelle begutachtet, ermöglichte der Algorithmus aus Kapitel 'Rekonstruktion und Simplifizierung' (Seite 49) eine Begutachtung des medizinischen Präparates aus jedem beliebigen Sichtwinkel. Gegebenenfalls notwendige Anpassungen des Isowertes konnten so dank der Parallelisierung schnell vorgenommen werden. Eine Reduzierung der Meshdreiecke durch das Simplifizierung-Modul half den Speicherbedarf zu verkleinern und bedeutete eine geringere Ladezeit. Dies ermöglichte wiederum eine bessere Verifikation der 3D-Modelle durch die Immunhistologen, da eine schnellere und reibungslosere Betrachtung möglich war. Da die Verwendung medizinischer Daten *ex vivo* geschah und nicht zur Diagnose bestimmt war, waren und sind Mesh-Simplifizierung und andere nachträgliche Meshmanipulationen vertretbar.[247]

Im Kapitel 4.1 wird nun eine praktische Anwendung der eben vorgestellten Methoden präsentiert. Der in Kapitel 1.7 vorgestellte Herstellungsprozess wurde während der Entstehung des Artikels von Steiniger et al. [Ste+18] entwickelt.

4.1 Anwendungen

Durch das Kapitel 'Akquise' ist bekannt, dass die Milzschnitte insgesamt dreimal digitalisiert wurden. Zweimal wurden sie eingescannt und einmal abfotografiert. Bei den Scans von der Leica SCN 440 weisen die Bilder eine Auflösung von $0,5\mu m \times 0,5\mu m$ pro Pixel auf. Beim Objektträger-Scanner Zeiss Mirax besitzen die Scans eine Größe von 25.856×32.000 Pixel und eine Auflösung von $0,3\mu m \times 0,3\mu m$. Dies bedeutet, dass die Zeiss-Scans eine höhere Auflösung haben und somit für die Pipeline und die medizinische Begutachtung am besten geeignet waren. Die mit Blau nachgefärbten Milzschnitte haben eine Größe von 5184×3456 Pixel. Ihr Auflösung beträgt $0,416\mu m \times 0,416\mu m$.

[246] Als Beispiel sei hier der Glättungsparameter w_α aus Abbildung 12 (Seite 42) zu nennen.
[247] Sofern die Mediziner sie für vertretbar hielten.

Mit einer Dicke der Serienschnitte von $7\mu m$ gelten für die Volumendaten folgende Abstände innerhalb des Gitters (Formel (4) auf Seite 9):

- Zeiss Mirax: $d_x = d_z = 0,3\mu m \quad \wedge d_y = 7\mu m$
- Zeiss Axiophot (Canon 60D Kamera): $d_x = d_z = 0,416\mu m \quad \wedge d_y = 7\mu m$

Durch die Verwendung von Paraffin gibt es keinen Verlust an Material zwischen den Schnitten. Die Höhe der digitalisierten Volumendaten beträgt somit theoretisch $168\mu m$.

4.1.1 Ausrichtung der Milzschnitte

Die beiden Färbungen wurden normalisiert [Rei+01; Kha+14]. Dies war nötig durch die Helligkeitsunterschiede, die beim Scannen aber auch beim Fotografieren entstehen.[248] Danach wurden die Bilder mit der Methode aus Kapitel 3.1.6 registriert und ausgerichtet.

Es wurde das 'coarse-grain fine-grain'-Verfahren[249] und die Merkmalserkennung SURF verwendet. Zuerst wurden die Bilder komplett aneinander ausgerichtet, dann erfolgte im Anschluss eine weitere Entzerrung mit kleineren Merkmalen, die sich allerdings auf ROIs beschränkte. Die Regionen von Interesse waren am Anfang nur vage bestimmt. Die genaue Festlegung ihrer Lokalisation und Dimension erfolgte erst später (kurz vor der Publikation). Dennoch sollen mit Abbildung 23 auf Seite 69 die endgültigen Definitionen der ROIs schon vorweggenommen werden, da dies die weitere Beschreibung des Hergangs erleichtert. Während der Versuche variierten selbstverständlich Mittelpunkt und Größe der Regionen. Ebenso wurden auch die ROI-Bilder in ihrer Auflösung manchmal herunterskaliert.

Insgesamt wurde bei der Ausrichtung die äußere Schleife[250] vier Mal durchlaufen. Die verwendeten Werte für die Featuremindestgrößen betrugen $100, 50, 20$ und 10. Der Wert für ε_l wurde wie im Kapitel 3.1.4 gewählt und betrug $0,01$. Durch den Radius match und die unterschiedlichen Merkmalsgrößen variierten die Werte für N_{select} und ε_g. Die Größen für N_{select} gingen von 10.000 bis 40.000 und für den Parameter ε_g von $0,01$ bis zu $0,05$. Ursprünglich sollte sich der Radius match an der längsten Seite eines Bildes orientieren. Eine Festlegung des Startwerte von 80 (Pixel)[251] erwies sich jedoch als stabiler. Dieser Wert halbierte sich in jeder der vier Iterationen (vgl. Abbildung 16 Seite 48).

Da Endresultat der ausgerichteten Einfachfärbung ist in Abbildung 24 auf Seite 70 zur Ergänzung dargestellt.

4.1.2 Rekonstruktion der Milzschnitte (Einfach-Färbung und Zweifach-Färbung)

Einfach-Färbung Die ausgerichteten Daten wurden als erstes mittels OpenCV in Grau konvertiert und invertiert. Bei der Graukonvertierung wurde auch wieder die Methode `cvCvtColor()` verwendet.[252] Die Invertierung ist der Tatsache geschuldet, dass die Modelle Porsche, Bonsai #2 und CTA Head aus Kapitel 3.2.3 ebenfalls derartig kodiert waren. Dort galt je heller, desto mehr Massedichte ist vorhanden. Die Graubilder wurden direkt und ohne eine weitere Behandlung in eine RAW-Datei umgewandelt. Die Abbildung 25 veranschaulicht nochmal die eben genannten Schritte.

[248] Vgl. Kapitel 1.1.4 Seite 6.
[249] Vgl. [Lob+17] Kapitel 3.3.
[250] Vgl. Abbildung 16 in Kapitel 3.1.6 auf Seite 48.
[251] Das sind $24\mu m$ beziehungsweise $33,28\mu m$, je nach Bilderserie.
[252] Hierbei sollte erwähnt werden, dass eine Umwandeln der Bilder mit OpenCV eine andere Graukonvertierung erzeugt als eine Umwandlung mit IrfanView oder ähnlichen Tools.

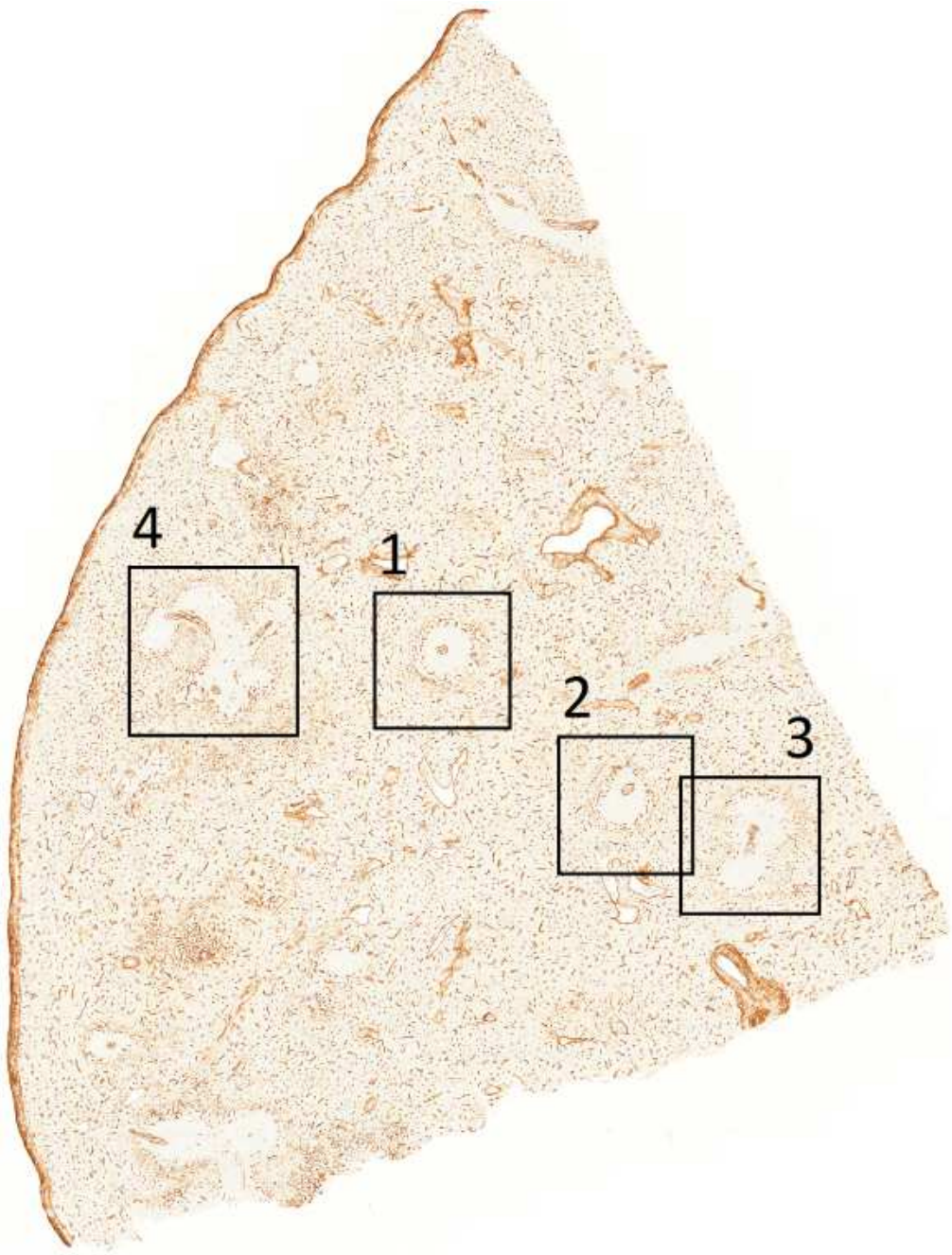


Abbildung 23: Die *endgültigen* definierten Regionen von Interesse (ROIs). Die Ausschnitte der Follikel wurden noch um 90° im Uhrzeigersinn gedreht, außer ROI 4.

Im Anschluss erfolgte eine Generierung des Meshes mit Hilfe des Parallelen Marching Cubes-Moduls [Ulr+14a]. Auf die Verwendung des Simplifizierungsmoduls aus Kapitel 3.2.2 wurde anfänglich verzichtet, da die Registrierungsmethode noch erst getestet werden musste und der endgültige und ideale Isowert für die Strukturen von Interesse noch nicht feststand. Der so erhalten gebliebene hohe Informationsgehalt in den Daten trug ebenfalls dazu bei, dass von den Medizinern die Rekonstruktionen mit den ausgerichteten Schnitten verglichen und die möglichen nächsten Schritte der erneuten verfeinerten Rekonstruktion oder der nächsten Meshmanipulationen erörtert werden konnten.

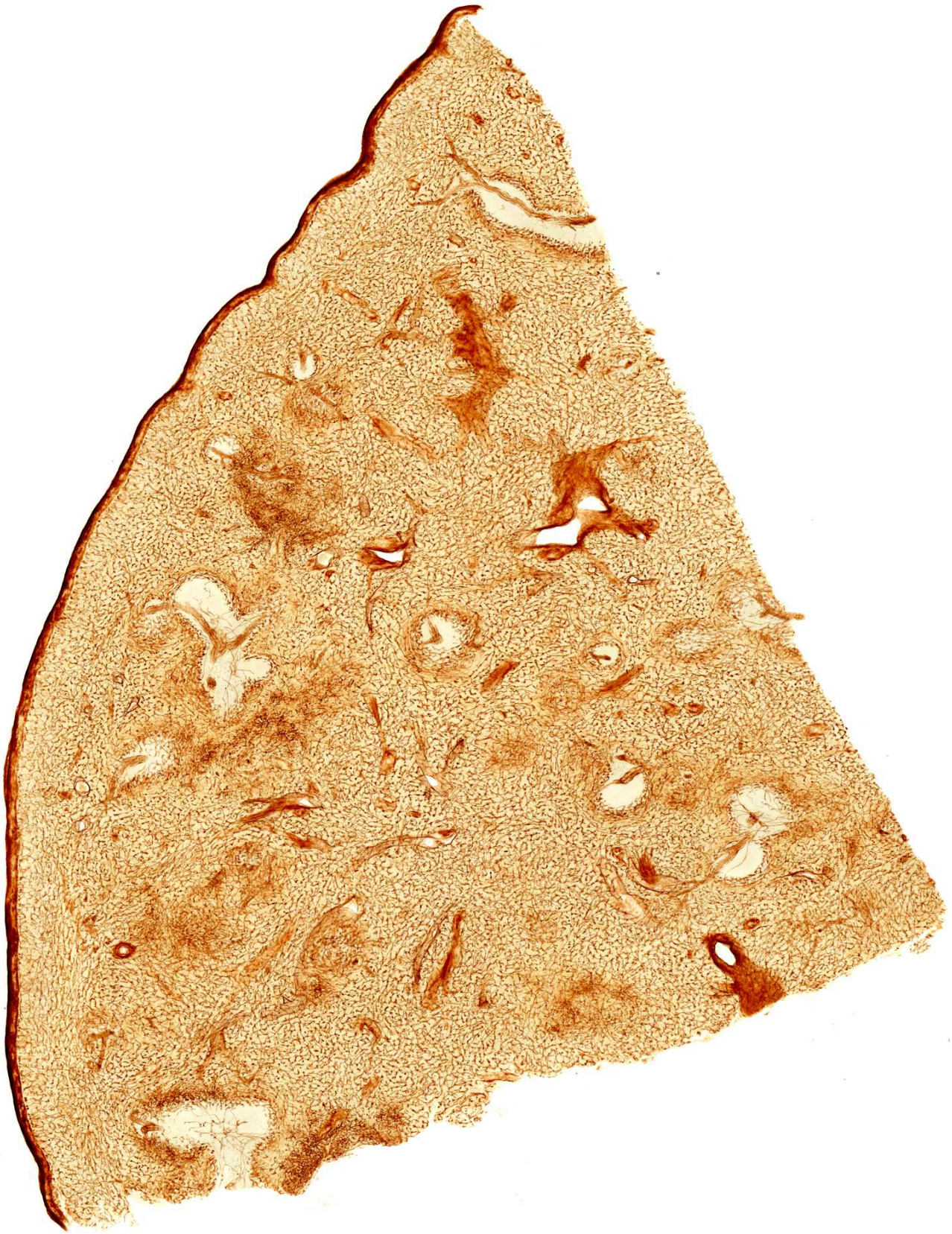
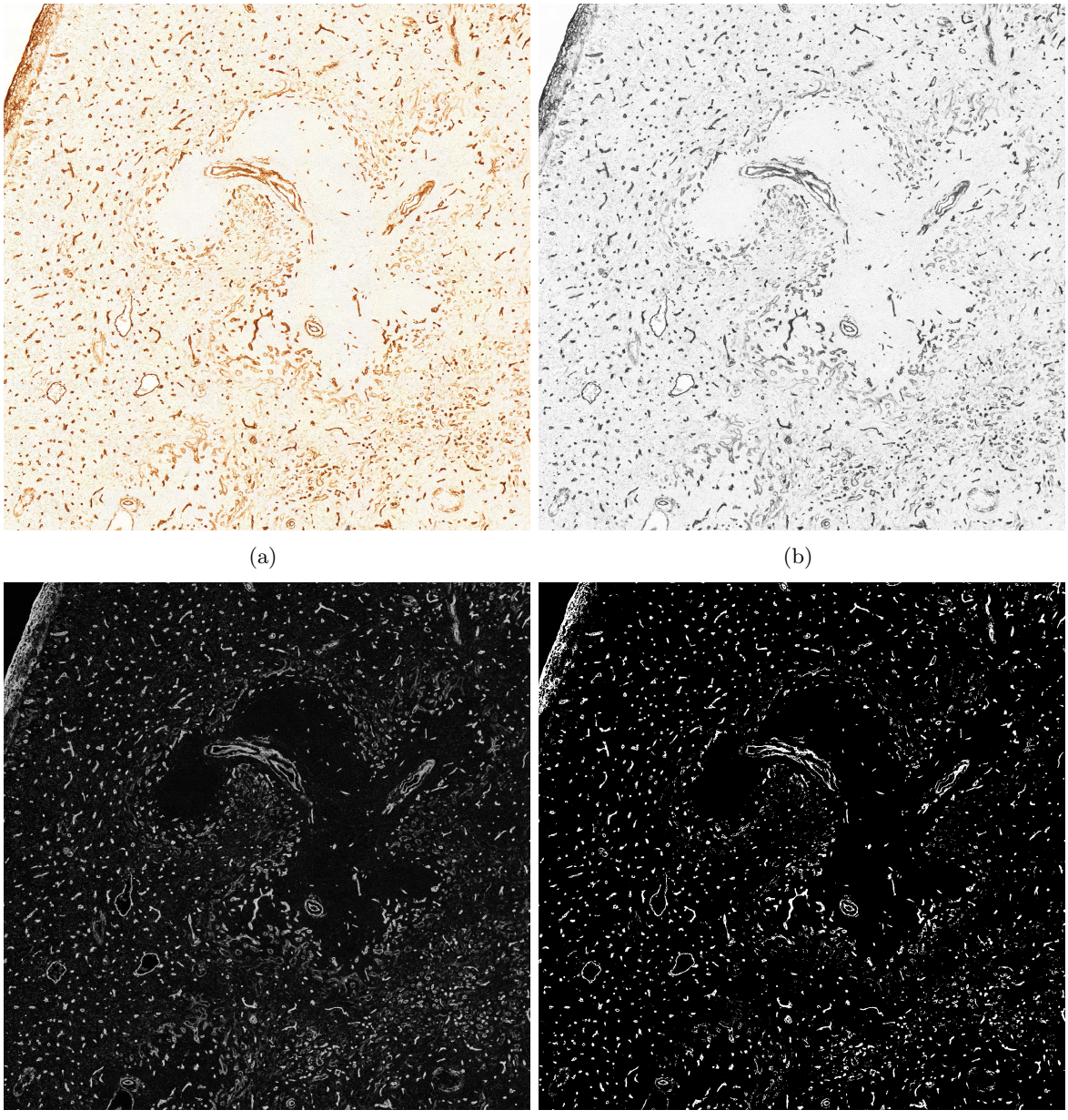


Abbildung 24: Die ausgerichteten Milzschnitte. Zu sehen ist das Ergebnis der erfolgten Ausrichtung der Einfach-Färbung mittels des vorgestellten Registrierungsalgorithmus.



(a) (b)

(c) (d)

Abbildung 25: Die Graukonvertierung der Serienschnitte. In Bild (a) sieht man das Originalbild. Es ist Schnitt 1 von ROI 4. Das Bild (b) zeigt die Umwandlung in die Graustufen. Die Invertierung ist im Bild (c) zu sehen. Das Bild (d) zeigt eine Selektierung der Pixel mit dem Isowert 119. Pixel, die darunterliegen, sind in Bild (d) schwarz dargestellt.

Die ersten Annäherungsversuche an den idealen Isowert und Bildausschnitt samt Auflösung erfolgten mit dem ROI 1 (siehe Abbildung 23). Nach den Rekonstruktionen der Meshes mit mehreren Schwellenwerten folgten immer Sitzungen mit den Immunhistologen. Es wurde über die Möglichkeiten der Darstellung und der Segmentierung diskutiert. Wie konnte die ideale Kapillardarstellung erreicht werden, wenn diese dunkelbraun eingefärbt sind, aber stellenweise auch hellbraun wie Sinus sein können und zwar dann, wenn diese tangential angeschnitten wurden?[253] Zeitweise war auch in Betracht gezogen worden einen künstlichen Kreis in ROI 1 einzuzeichnen oder den Follikel gar in der Mitte aufzuklappen. Ziel war es die nach der Rekonstruktion folgenden Schritte so zu wählen, dass zwar eventuelle Verbindungen von Blutgefäßen nicht angezeigt werden, aber auch keine durch Artefakte entstehen. Dies heißt, dort wo in den Endergebnissen (Abbildung 27) Verbindungen zu sehen sind, existieren sie auch.

In Abbildung 26 sind einige Versuche illustriert. Das Bild (a) zeigt eine erzeugte Isofläche bei einem Isowert von 30 und einer Dimension von $dim(1175, 24, 1175)$. Hier ist das Kapillarnetzwerk kaum zu erkennen. In Bild (b) wurde bereits der ideale Isowert gefunden, aber das Modell beinhaltet noch sehr viele kleine Komponenten. Man erkennt wie der Farbstoff vom Follikelinneren nach außen gestreut hat. In Abbildung 26 (c) wurde untersucht, ob eine optimale Darstellung des Follikel-prägenden Blutgefäßnetzes mit künstlicher Einfärbung gelingt.[254] Dies führte aber zu keinem befriedigenden Ergebnis. In Bild (d) wurde wieder der Isowert 119 genommen, doch zusätzlich wurden mit MeshLab (Version 1.33) [Cig+08] alle isolierten Komponenten entfernt, die kleiner als 10% bezüglich Raumdiagonale waren.[255]

Die Versuche führten zu dem Ergebnis, dass der Isowert 119 am geeignetsten für die Kapillardarstellung ist. Die CD34+ perifollikulären[257] Sinus sind zwar im Volumenmodell noch vorhanden, aber schwach genug für eine weitere Bearbeitung. Der Isowert wurde für ROI 1,2 und 4 angewandt. Bei ROI 3 musste er auf 110 angepasst werden. Wahrscheinlich liegt diese Abweichung an der leicht unterschiedlichen Immunfärbung der Follikel, die ihre Ursache wiederum im Fixierungsschritt (siehe Kapitel 1.1.1 Seite 3) hat.[258]

Für ROI 1 bis 3 wurden die Ausgangsdaten herunterskaliert und die Abstände betrugen in der Anfangsphase noch $d_x = d_z = 0,6\mu m \wedge d_y = 7\mu m$. Dann wurden die Graubilder der ROI 1 bis 4 mittels Lobachev et al. [LSG17] und Farnebäck [Far03] in y-Richtung interpoliert[259] und mit einem eigens angefertigten Programm in OpenCV zu vier RAW-Dateien verarbeitet. Die erzeugten Follikel-Modelle entstammen somit aus einem Volumengitter mit den Abständen $d_x = d_z = 0,6\mu m \wedge d_y = 1\mu m$ und haben ein Dimension von $dim(1600, 161, 1600)$. Die Ergebnisse sind in Abbildung 27 veranschaulicht. Die dort dargestellten Meshes wurden nach der Generierung mittels Polymender[260] [Ju04] bearbeitet

[253] Eine ähnliche Problematik gibt es bei dem CT. Hier gibt es den *Partialvolumeneffekt*. Dieser tritt zum Beispiel auf, wenn Knochen recht dünn sind oder nicht komplett von einem Voxel erfasst werden. Der typische Wert um 500 HU ist in solche Fällen dann nicht im CT-Bild wiedergegeben. Man vergleiche hierzu [Han09] Kapitel 2.1.3.2 Absatz 3 Seite 13 und Kapitel 2.2 Absatz 5 Seite 36.

[254] Hierfür wurden die Vertices des Meshes per MeshLab mit einem Qualitätsmaß versehen: $f(p) = (x-500)^2+(z-500)^2$. Alle Vertices, die einen kleineren Wert als 141.000 hatten, wurden dem Kreisinneren zugeordnet. Es wurden drei Meshes erstellt: ein Gesamtmesh, ein Mesh mit Kreis, ein Mesh ohne Kreis. Der Kreis erfuhr noch ein Entfernen vom Komponenten die 100 Dreiecke oder weniger hatten. Beim Mesh ohne Kreis wurden alle Komponenten entfernt die weniger als 500 Faces aufwiesen. Beim Gesamtmesh wurde nichts verändert. Im Bild ist dieses noch zu sehen und rot eingefärbt, wird aber von den anderen Meshes überdeckt.

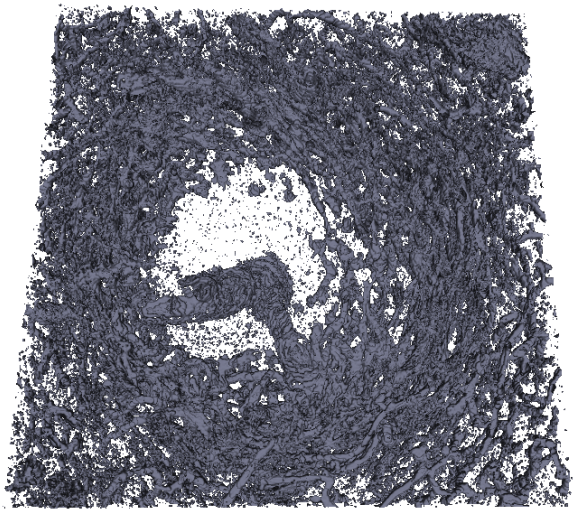
[255] Gemeint ist die Filterfunktion in MeshLab 'Remove Isolated Pieces (wrt Diameter)'.

[256] Bei der Milzkapsel handelt es sich um die umschließende Gewebehülle, die rote und weiße Pulpa zusammenhält.

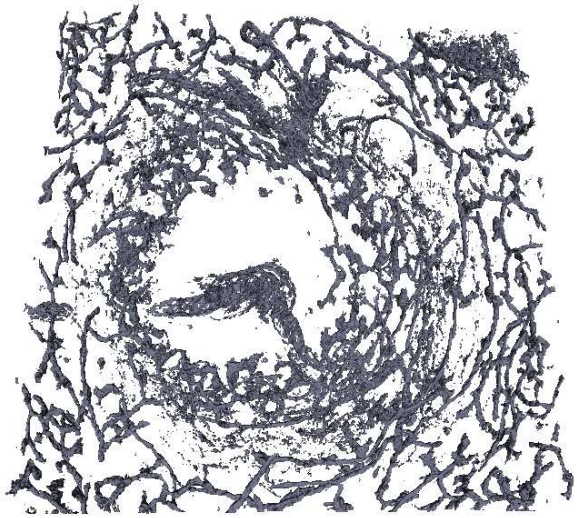
[257] Etymologie: griechisch $\pi\epsilon\rho\iota$ *peri* 'um ... herum', lateinisch *folliculus* 'Hülle' oder 'Hülse'.

[258] Vgl. [Ste+18] Kapitel 4.5.3.

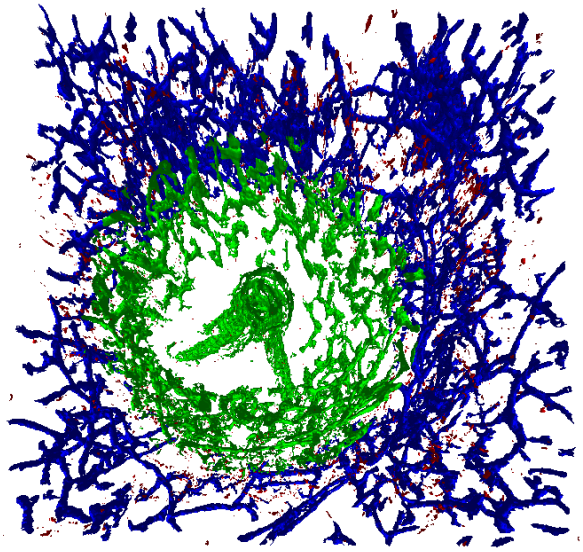
[259] Ziel dieser Vorverarbeitung war eine Glättung der Blutgefäße und eine Reduktion der Anisotropie, der Richtungsabhängigkeit bezüglich der y-Achse.



(a) Isowert 30, 1175 × 24 × 1175



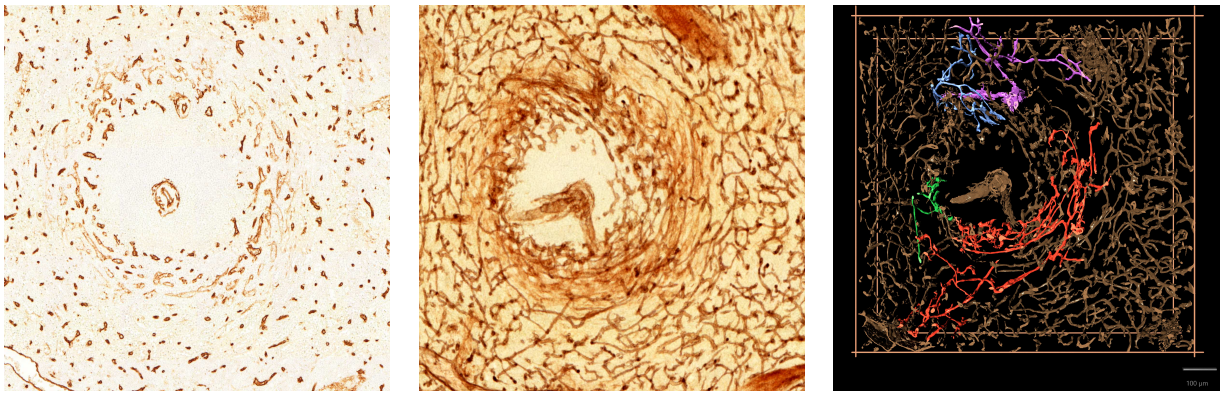
(b) Isowert 119, 1175 × 24 × 1175



(c) Isowert 109, 1175 × 24 × 1175

(d) Isowert 119, 1175 × 24 × 1175

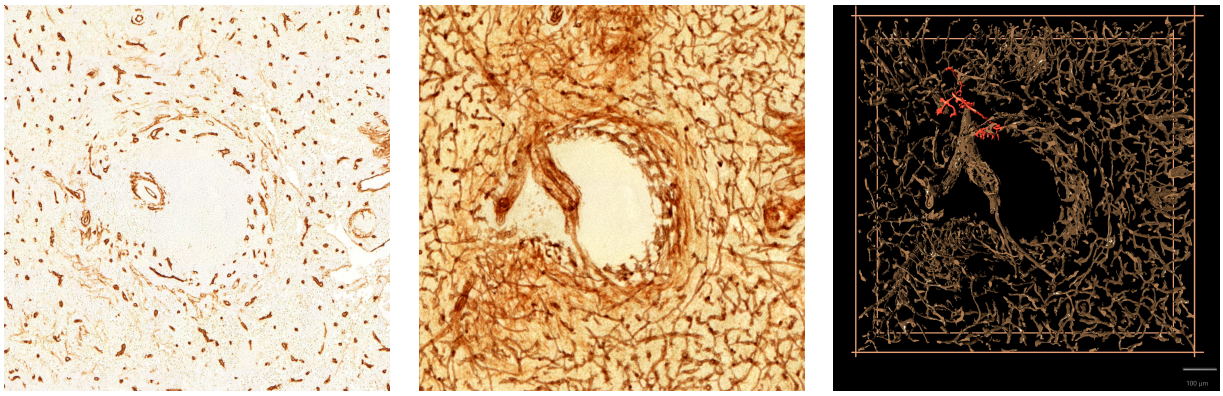
Abbildung 26: Die mehreren Rekonstruktionsvarianten.



(a) ROI 1 einfach gefärbt, Schnitt 1

(b) ROI 1 einfach gefärbt, Volume

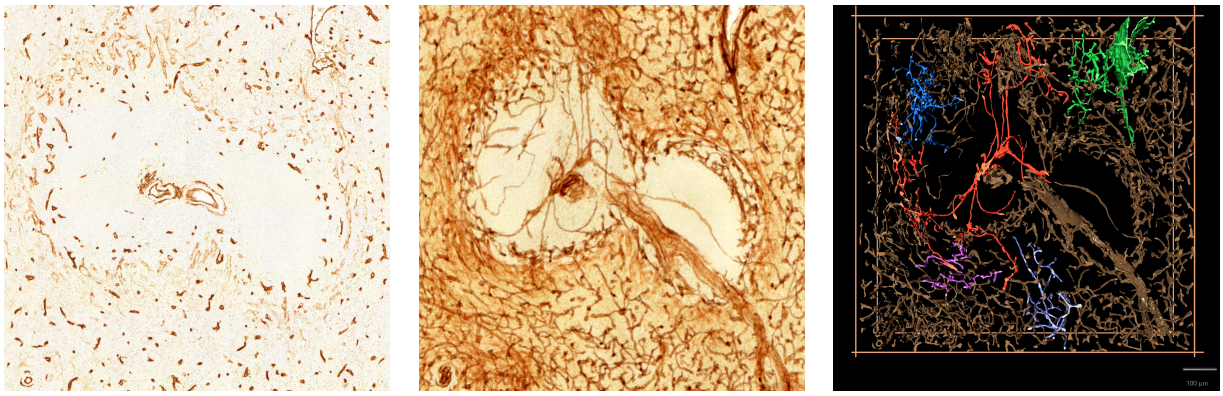
(c) ROI 1 einfach gefärbt, Mesh mit Isowert 119



(d) ROI 2 einfach gefärbt, Schnitt 1

(e) ROI 2 einfach gefärbt, Volume

(f) ROI 2 einfach gefärbt, Mesh mit Isowert 119



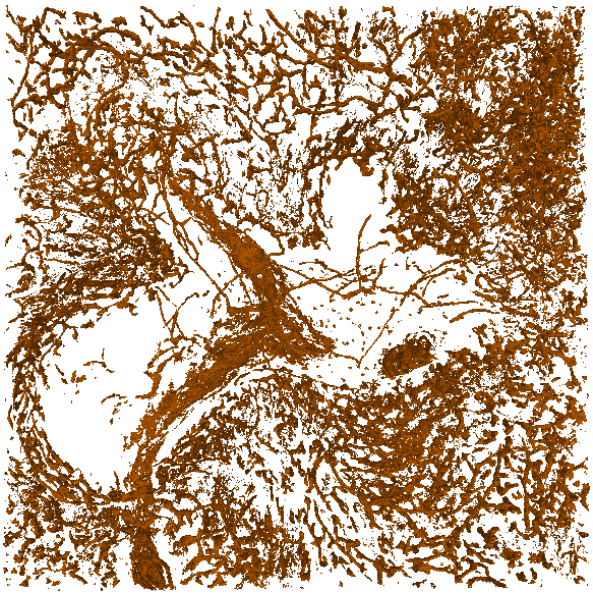
(g) ROI 3 einfach gefärbt, Schnitt 1

(h) ROI 3 einfach gefärbt, Volume

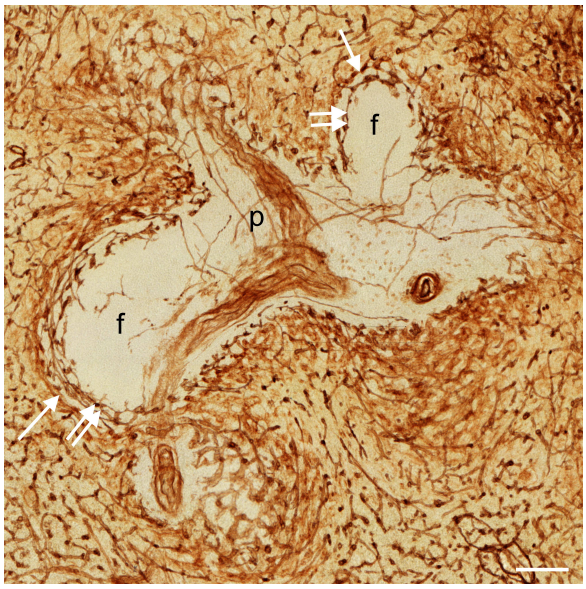
(i) ROI 3 einfach gefärbt, Mesh mit Isowert 119

Abbildung 27: Die Einfach-Färbung der ROI 1 bis 3. In der linken Spalte sind die Regionen von Interesse nochmals aus dem Bild mit der Nummer 1 in der Schnittserie dargestellt. Diese sind hier gegenüber der Abbildung 23 um 90° im Uhrzeigersinn gedreht. Die mittlere Spalte zeigt die Volumendarstellung der ausgerichteten Schnitte 1 bis 24. In der rechten Spalte sieht man die Meshes.

[260] Hier wurden die Parameter 9 (für die Octree-Tiefe) und 0,9 (für die Ratio Modell/Gitter) verwendet.



(a) 3800 × 161 × 3800, Mesh bei Isowert 119



(b) 4000 × 24 × 4000, Volume *(Quelle: [Ste+18].)*

(c) 6000 × 24 × 6000, Volume

Abbildung 28: Die mehreren Varianten der Region von Interesse Nummer 4. In Bild (a) ist das Mesh von ROI 4 dargestellt. Das Bild (b) zeigt die Volumendarstellung, die letztendlich publiziert wurde. Auf diesem erkennt man zwei Follikel, welche mit einem 'f' gekennzeichnet sind. Ebenso ist die PALS zu sehen. Sie ist mit einem 'p' markiert. Zudem sind unter dem Follikel rechts Pinselarteriolen erkennbar. Der weiße Balken unten rechts ist $100\mu m$ breit und dient als Maßstab. Das Bild (c) stellte sich als zu unübersichtlich heraus. Man sieht in der linken unteren Ecke einen Teil der Milzkapsel.[256]

Kanal	R	G	B
Color 1	0,39161506	0,61583800	0,68365290
Color 2	0,58560830	0,58401763	0,56212664
Color 3	0,68600800	0,69213814	0,22436099

Tabelle 7: Die verwendeten Parameter der Colour Deconvolution unter Fiji. Erstellt wurden sie mit 'From ROI'. Zur Reproduktion der Kanäle 1 bis 3 wählt man 'User values'.

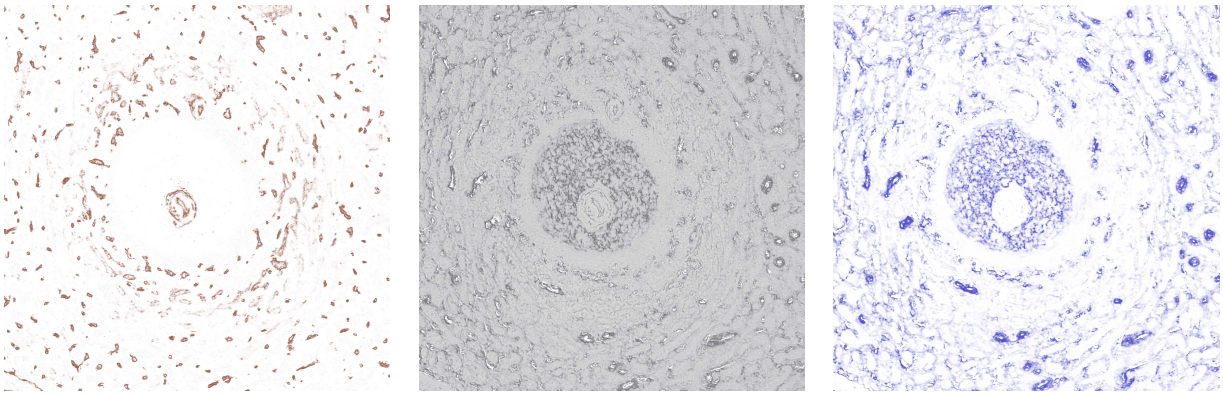
und mittels Taubin Smooth[261] [Tau95] geglättet.[262] [263] Der letzte Schritt bestand in der Entfernung aller isolierten Komponenten, die kleiner als 2% bezüglich der Raumdiagonale waren.[264] [265] [266]

Bei ROI 4 wurde die höchste Auflösung der Scans beibehalten. Damit ergeben sich die Abstände $d_x = d_z = 0,3\mu m \wedge d_y = 1\mu m$ bei einer Dimension von $dim(4000, 161, 4000)$. Auch hier gab es mehrere Versuche der besten Darstellung, welche in Abbildung 28 zu sehen sind. Ein anfänglich erstelltes Mesh (mit der Dimension $dim(3800, 161, 3800)$) wurde nicht genommen, da hier die Trennung von Kapillaren und Sinus nicht deutlich zu sehen war (Abbildung 28 Bild (a)). Stattdessen wurde auf das Bild (b) zurückgegriffen. Hier ist die Trennung von perifollikulären Kapillaren und Sinus mit weißen Pfeilen markiert. Die Dimension für diese Darstellung beträgt $dim(4000, 24, 4000)$. Im Bild (c) sind diese auch erkennbar, aber dennoch sind die Strukturen von Interesse zu klein.

Zweifach-Färbung Bei der Zweifach-Färbung wurde sehr schnell auf die Verfahrensweise der Trennung der Farbkanäle zurückgegriffen. Hier ergab sich die Schwierigkeit, dass die blaue Färbung nicht rein Blau war. Die Schattierung dieser Farbe konnten durchaus Anteile von Rot und/oder Grün beinhalteten. Damit ergeben sich zwei Vorgehensweisen, um die Braun- und die Blau-Färbungen für die Rekonstruktion aufzubereiten. Es soll hier noch angemerkt sein, dass die Strukturen von Interesse die Hülsenkapillaren und FDZs, die Follikulären Dendritischen Zellen, in Anbetracht der Auflösung für Manipulation groß genug waren. Dies heißt, die digitalisierten Schnitte wurden nach der Interpolation weiter verarbeitet und erst dann in Volumendaten umgewandelt.

Für die Rekonstruktion der braunen Strukturen wurden die Bilderinhalte in den CMYK-Farbraum umkodiert und der daraus resultierende K-Kanal genommen. Für die blauen Strukturen wurde eine Color Deconvolution[267] [RJ01; OZS14] unter Zuhilfenahme von Fiji [Sch+12] vorgenommen.[268] Durch die weiter oben beschriebene Schwierigkeit wurde aber zuvor der Hintergrund, ebenfalls mit Fiji, entfernt.[269] [270] Die für die Colour Deconvolution verwendeten Parameter sind der Tabelle 7 zu entnehmen. Das Ergebnis dieser Operation ist in Abbildung 29 auf Seite 77 exemplarisch am ersten Bild des ROI 1 veranschaulicht.

[261] Dies geschah mit den Standard-Parametern: 0,5–(-0,53)–10.
[262] Gemeint ist die Filterfunktion in MeshLab 'Smoothing, Fairing and Deformation→Taubin Smooth'.
[263] Vgl. [Ste+18] Kapitel 4.5.2.
[264] Gemeint ist die Filterfunktion in MeshLab 'Remove Isolated Pieces (wrt Diameter)'.
[265] Vgl. [Ste+18] Kapitel 4.5.2.
[266] Taubin Smooth und die Entfernung kleinerer Komponenten konnten mit MLX-Dateien für den Meshlabserver und C++-Code als Automatisierungshilfe recht zügig umgesetzt werden.
[267] Hierbei handelt es sich um den Menüpunkt 'Image→Color→Colour Deconvolution' im Programm Fiji.
[268] Vgl. [Ste+18] Kapitel 4.5.4.
[269] Hierbei handelt es sich um den Menüpunkt 'Process→Substract Background...'. Verwendet wurden die Default-Einstellungen.
[270] Vgl. [Ste+18] Kapitel 4.5.4.



(a) Color 1 (b) Color 2 (c) Color 3

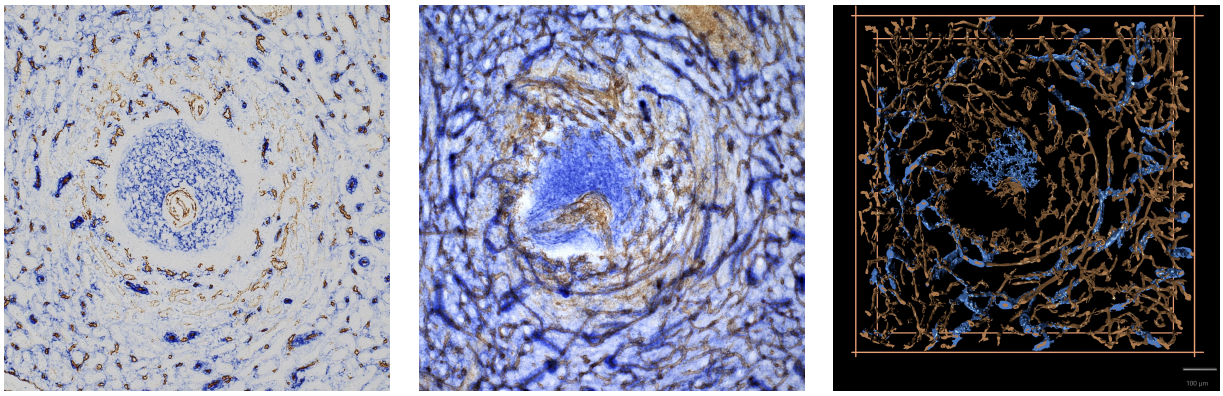
Abbildung 29: Die Kanäle Color 1,2 und 3 sind der ausgeführten Colour Deconvolution mit Hilfe des Tools Fiji.

Jeder dieser Kanäle (K, Color 1, 2 und 3) hatten eine Dimension von $dim(2300, 24, 2300)$ und zudem folgende Abstände $d_x = d_z = 0,416\mu m \wedge d_y = 7\mu m$. Nach der Interpolation mit Lobachev et al. [LSG17] und Farnebäck [Far03] ergaben sich folgende Abstände und Dimensionen: $d_x = d_z = 0,416\mu m \wedge d_y = 1\mu m$ und $dim(2300, 161, 2300)$. Im Anschluss erfolgten mit dem 3D-Slicer (Version 4.6.0) [Fed+12; KPV14] ein Closing[271] [272] [Shi09] mit den Parametern 7–3–7 und eine Weichzeichnung (Gaussian Blur[273]) mit dem Parameter $\sigma = 1$ für die braunen Strukturen (Kanal K). Bei den Blau-Färbungen – gemeint sind die Bilder des Kanals Color 3 – wurde das Closing mit den Parameter 11–4–11 und mit den gleichem Parameter für das Gaussian Blur weich gezeichnet.[274]

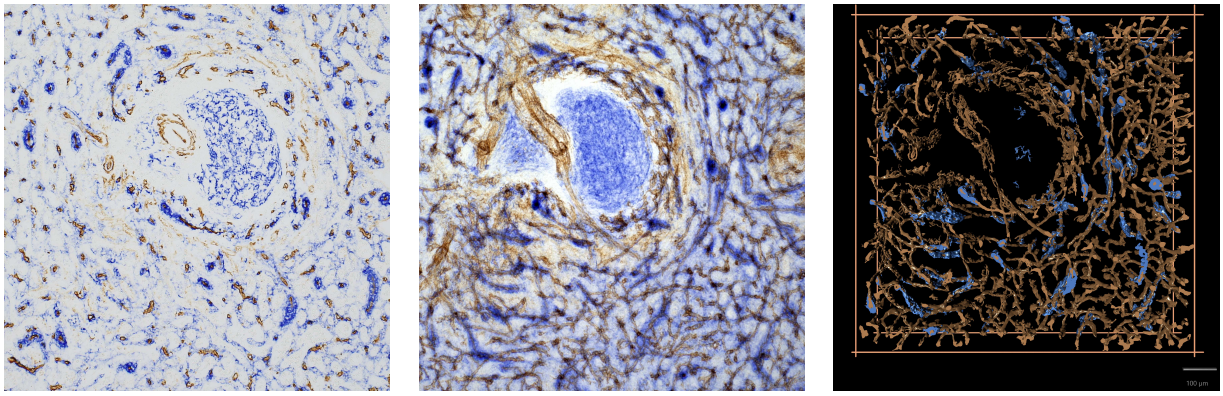
Für die Rekonstruktionen pro Kanal lagen die verwendeten Isowerte in einem Wertbereich von 0 bis 255.[275] Bei allen Modellen wurde der Wert 80 genommen, außer bei ROI 1. Hier wurde beim Kanal Color 1 (braun) auf den Isowert 130 zurückgegriffen.[276] Die Meshes erfuhren auch hier eine Nachbearbeitung. Diese ist mit den Manipulationen bei der Einfach-Färbungen fast identisch. Allerdings wurden bei den blauen Strukturen die isolierten Komponenten entfernt, die bezüglich der Raumdiagonale kleiner als 10% waren. Diese Manipulation war akzeptabel, da die Kapillarhülsen und die FDZ groß genug waren und der Isowert von 80 genügend Strukturen von Interesse erfasste. Zusätzlich wurden beim ROI 1, welches ja auch einen anderen Isowert bei den braunen Strukturen aufweist, nur Komponenten entfernt, die kleiner als 5% bezüglich der Raumdiagonale waren.[277] Die endgültigen Ergebnisse sind in Abbildung 30 veranschaulicht.

Die Darstellungen, die für die Veröffentlichung [Ste+18] verwendet wurden, können unter [LS17] abgerufen werden. Einige sind in dieser Arbeit schon zu sehen (siehe Abbildung 27, 30 und 28). Allerdings befinden sich unter dem Supplementary [LS17] auch Videos, in denen die jeweiligen Meshes um die y-Achse rotieren, um einen besseren dreidimensionalen Eindruck zu vermitteln. Hier verweist die Autorin auf den im Literaturverzeichnis unter [LS17] angegeben Weblink.

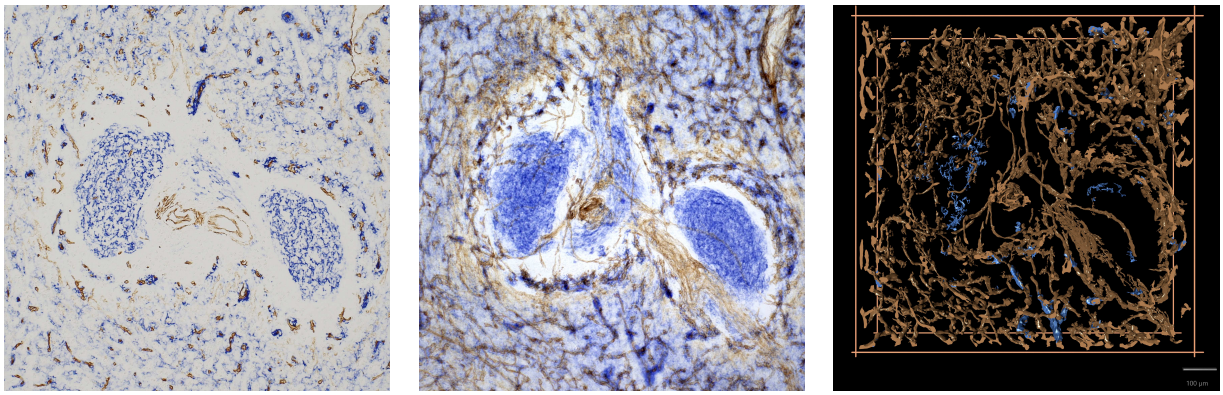
[271] Hierbei handelt es sich um das Modul 'Simple filters→GrayscaleConnectedClosingImageFilter' im 3D-Slicer.
[272] Vgl. [Ste08] Kapitel 3.3 Seite 76ff.
[273] Hierbei handelt es sich um das Modul 'Filtering→Denoising→Gaussian Blur Image Filter' im 3D-Slicer.
[274] Vgl. [Ste+18] 4.5.4.
[275] Vgl. [Ste+18] 4.5.4 letzter Absatz.
[276] Vgl. [Ste+18] 4.5.4.
[277] Abweichungen bezüglich der Prozentangabe können aufgrund von unterschiedlichen Eigenschaften der Gewebestruktur entstehen.



(a) ROI 1 zweifach gefärbt, Schnitt 1 (b) ROI 1 zweifach gefärbt, Volume (c) ROI 1 zweifach gefärbt, Mesh Isowerte 130 (braun) und 80 (blau)



(d) ROI 2 zweifach gefärbt, Schnitt 1 (e) ROI 2 zweifach gefärbt, Volume (f) ROI 2 zweifach gefärbt, Mesh Isowerte 80 (braun) und 80 (blau)



(g) ROI 3 zweifach gefärbt, Schnitt 1 (h) ROI 3 zweifach gefärbt, Volume (i) ROI 3 zweifach gefärbt, Mesh Isowerte 80 (braun) und 80 (blau)

Abbildung 30: Die Zweifach-Färbung der ROI 1 bis 3. Die Zweifach-Färbung wurde erstellt um die Hülsenkapillaren und deren Lage zu veranschaulichen. Die Abweichung des Isowertes bei ROI 3 ist hier, wie auch bei der Braun-Färbung (Einfach-Färbung), mit einem Fixierungsartefakt zu erklären. Die Fixierflüssigkeit wirkt am Rande der Milzschnitte früher und somit länger. Hier hatte der in Kapitel 'Das Fixieren, das Einbetten und das Einblocken' auf Seite 3 erwähnte Faktor Zeit Einfluss auf die Qualität der Schnitte.

4.1.3 Verifikation

Die Erstbegutachtung der Meshes erfolgte anfangs noch mit einer direkten Betrachtung in MeshLab. Waren die richtigen Parameter für die 3D-Modelle gefunden, erfolgte die Verifikation mit einem direkten Volumenrenderer. Dieser liest die ausgerichteten Bilder, aus denen eine RAW erzeugt wurde, und das erstellte 3D-Modell ein. Im Anschluss legt das Programm die Bilder, wie in Abbildung 27 (mittlere Spalte) zu sehen, übereinander und blendet das Mesh ein. Letzteres kann der y-Achse entlang gecroppt werden. Das heißt, es kann auf das passende Layer zum entsprechenden Bild reduziert werden. Das Ziel dieses Programms war es die Verifikation des Meshes gegenüber den Schnittbildern und eine gute Wahrnehmung der Dreidimensionalität zu erreichen. Unter MeshLab wirken die Meshes erst dreidimensional, wenn diese gedreht werden, wobei immer noch die Einblendung der Schnittbilder fehlte. Der Volumenrenderer hatte jedoch den Nachteil, dass die übereinandergelegten Bilder sich in diesem Programm nur bis zu einem gewissen Grad drehen lassen. Der Abstand zwischen den Schnitten ist für ein echtes Volumenrendering hierfür zu groß. Es ist daher nur ein Kippen des Modells möglich, eine 360°-Drehung dagegen nicht. Dies erschwerte oftmals die räumliche Wahrnehmung und somit die Begutachtung der Befunde.

Folglich wurde ein weiteres Studentenprojekt erstellt. Das VR-Tool [Lob+18] wurde mit OpenVR [Git] erstellt. Die Verifikation fand somit mit Hilfe eines Computer statt, an dem eine HTC Vive (inklusive den zwei Kameras und den beiden Controller) und ein Xbox-Controller angeschlossen war. Weiterhin verfügte der PC über eine NVIDIA GTX 1070. Mit dieser konnten die Vorteil des IVR richtig genutzt werden. Die jeweilige Rekonstruktion konnte so von allen Seiten betrachtet werden. Die interessanten Stellen konnten mit einem Vive-Controller markiert werden, mit einem Xbox-Controller wurde das Modell gedreht, gezoomt oder verschoben. Weiterhin war es mit Letzterem möglich die Schnitte einzublenden. Die Arbeitsweise und Funktion ist nochmal im Supplementary [LS17] von [Ste+18] veranschaulicht. Genauer genommen ist damit Video S4 (video_4.mov) aus dem eben genannten Supplementary [LS17] gemeint. Dort befinden sich auch die schon weiter oben erwähnten Videos der Meshes. Die verifizierten Meshes wurden im Anschluss mit Cinema 4D von Maxon gerendert.

4.1.4 Forschungsergebnis

Die vorgestellten Methoden aus Kapitel 3.1 und 3.2 haben die im Kapitel 2.4 dargelegte Problematik der angeschnittenen Gefäße sehr gut lösen können. Die schnellen Algorithmen ermöglichten einen effizienten Arbeitsprozess, der zu 3D-Modellen mit sehr guter Gefäßdarstellung führte.

Die erstellten 3D-Modelle haben gezeigt, dass das perifollikuläre Kapillarnetzwerk mit der roten Pulpa verbunden ist. Dargestellt und beobachtet wurde dies anhand von ROI 1, 2 und 3 der Einfach-Färbung. In der rechten Spalte von Abbildung 27 Seite 74 sind die 3D-Modelle zu sehen, welche die eingefärbten perifollikulären Kapillaren enthalten, die eine Verknüpfung mit der roten Pulpa aufweisen. Im Bild (f) ist im oberen linken Quadraten die rot eingefärbte Verbindung zusehen. In den Bilder (c) und (i) kommen solche Verbindungen häufiger vor. Durch die starke Streuung der Braun-Färbung, die auch die umgebene Sinus beeinflusst, musste ein recht hoher Isowert gewählt werden. Es sollten so möglichst wenig Sinus durch die Rekonstruktion extrahiert werden. Daraus folgt, dass die gezeigten Verbindungen eine Unterzahl darstellen.

Auch war es möglich die Gefäßstrukturen zu begutachten. Auf Seite 75 in Abbildung 28 Bild (b) ist anhand der 'Region von Interesse Nummer 4' eine deutliche Abgrenzung zwischen dem perifollikulären Sinusnetzwerk (ein Pfeil) und dem perifollikulären Kapillarnetzwerk (zwei Pfeile) zu erkennen. Die

Modelle in Bild (a) und (b) aus Abbildung 28 zeigen zudem, dass Kapillaren auch innerhalb der weißen Pulpa (des Follikels) präsent sind.[278]

Zudem gibt es Kapillaren, die von der weißen Pulpa (Follikel) in das perifollikuläre Netzwerk fließen.[279] Dies wurde so von Kusumi et al. [Kus+15] nicht erkannt. Das gezeichnete Kapillarnetzwerk in Abbildung 6 aus Kusumi et al. [Kus+15] soll eher an der Innenseite an der Marginalzone (brauner Bereich) liegen. Die in dieser Dissertation erstellten Modelle legen aber nahe, dass das perifollikuläre Kapillarnetzwerk innerhalb der retikulären Randzellen (MRC) und außerhalb die Marginalzone (MZ) anzusiedeln ist.[280] [281]

Mit der Zweifach-Färbung wurde die Darstellung der Hülsenkapillaren (vgl. Abbildung 30 Seite 78) umgesetzt. Die Ergebnisse der Pipeline förderten zu Tage, dass die Verteilung der Hülsen sowohl in der roten Pulpa als auch in der Follikelumgebung anscheinend gleich ist. Hervorgehoben werden sollte ebenfalls, dass einige der perifollikulären Kapillaren, die Verbindungen zur roten Pulpa haben, auch eine Verbindung mit Kapillaren ausweisen können, die mit Hülsen ausgestattet sind.[282] Die Follikulären Dendritischen Zellen (FDZ) selbst bilden ein großes Netzwerk (vgl. Abbildung 30 und Abbildung 7 aus [Ste+18] Seite 14).

Auch ergab das Betrachten der untersuchten Modelle, dass die Blutzirkulation anscheinend eine offene ist.

4.2 Ausblick

Die hier vorgestellte Pipeline hat bisher nur Serienschnitte von 24 Bildern bearbeitet. Aber sowohl der Rekonstruktions- als auch der Registrierungsschritt können mit einer größeren Bildmenge arbeiten. Ein Vorhaben, das in nächster Zeit wohl geschehen wird, da die Lokalisation der Hülsenkapillaren und deren Darstellung bei der geringen Anzahl an Scans kaum umsetzbar war. Dennoch sind erste Aussagen über deren Verteilung gelungen. Ebenso war eine Klärung über die Dichte und Ausbreitung der Blutgefäßstrukturen möglich.

Beim Einfärben der Modelle hat sich gezeigt, dass die Mediziner oftmals nicht an realistischen Darstellungen (Renderings) interessiert sind,[283] sondern eher zu schematischen Darstellungen neigen. Meistens sind die Grafiken eher als Orientierungshilfe zu verstehen und können bei der Erläuterung des gefundenen Sachverhaltes oder als zusätzliche Hilfe zu weiteren bildgebenden Verfahren dienen (vgl. [Qi+14; Gor+09; PB07] und Abbildung 7 aus [Ste+18]).

Da die medizinischen Volumendaten in dieser Dissertation *ex vivo* begutachtet wurden, waren nachträgliche Meshmanipulationen an den Modellen – wie Simplifizierung, Oberflächenglättung, Löschen von Komponenten, und weitere – vertretbar. Ebenso konnten eine Vorverarbeitung oder Nachbearbeitung der Bilder problemlos stattfinden. Dennoch wurde diskutiert, ob und wie bestimmte Strukturen darstellbar sind oder dargestellt werden dürfen. Hinzukommt, dass die hier vorgestellten Modelle bisher nur von einem einzelnen Mediziner-Team verifiziert wurden. Dies ist für die Strukturen von Interesse, die es zu untersuchen galt, auch hinreichend. Betrachtet man sich jedoch die erzeugten Modelle, so stellt man fest, dass vor allem Blutgefäße enthalten sind, sich an der y-Achse zu

[278] Vgl. [Ste+18] Kapitel 2.2 Seite 6ff.
[279] Vgl. [Ste+18] Kapitel 2.2 bis Kapitel 2.4 Seite 6ff.
[280] Vgl. [Ste+18] Kapitel 3 Seite 11.
[281] Vgl. [Ste+18] Abbildung 7 Seite 14.
[282] Vgl. [Ste+18] Kapitel 2.3 Seite 8f.
[283] Eine Ausnahme bilden 3D-Modelle für die Lehre in der Anatomie.

orientieren scheinen. Dies liegt an der schon beschriebenen Problematik des Anschnittes.[284] Schräg, längs oder tangential angeschnittene Gefäße gehen spätestens bei der Rekonstruktion mit einem bestimmten Isowert verloren. Was dann im rekonstruierten Modell übrig bleibt ist, stellt nur einen Teil dessen dar was vorhanden ist. Dies wirft natürlich die Frage auf, wie die Gefäßstruktur und die Gewebeprobe komplett als 3D-Modell rekonstruiert werden könnte?

Das bisherige Verfahren vermag zwischen einem Sinus und einem quer angeschnittenen Gefäß nicht zu unterschieden. Eine kontextbasierte Segmentierung wäre vielleicht eine Lösung. Doch die Zuordnung der einzelnen Gefäßtypen könnte sich als schwieriger erweisen als beispielsweise bei einer atlas-basierten Segmentierung. Die Gefäßstrukturen und deren Lage sind meistens noch nicht bekannt und sehr klein. Eine Herangehensweise wäre es, die Schnittserien von mehreren Personen manuell segmentieren zu lassen. Diese Segmentierungen könnte man mitteln und im Anschluss ausrichten und rekonstruieren. So würde ein *Gold-Standard*[285] geschaffen werden, mit dem die bisherigen Ausrichtungen aus dieser Dissertation verglichen werden könnten. Eine Möglichkeit, die sich zusammen mit Computerspiel-Entwicklern umsetzen ließe. Als Beispiel, sei hier auf Eyewire [Eye] verwiesen.

Auch wenn die Ausrichtung mit einem Gold-Standard verifiziert werden würden, bleibt immer noch die Schwachstelle der Histotechnik. Der kritischste Arbeitsschritt ist und bleibt das Aufbringen der Schnitte auf die Glasobjektträger. Um starke Verzerrung zu minimieren, wurde bereits überlegt externe Marker an die Gewebeprobe an zubringen. Bei Paraffin ist dies aber nicht leicht. Daher ist eine weitere Überlegung dickere Schnitte und Mikroskope mit besserer z-Auflösung (in dieser Dissertation y-Auflösung) zu verwenden. Gemeint sind hier Konfokalmikroskope und z-Stacks.

Eines der nächsten Ziele wird die Erhöhung der Portabilität sein. Bisher sind Registrierung und Ausrichtung nicht mit den Hybrid-Algorithmus direkt verbunden. Oftmals fanden Registrierung und Rekonstruktion nicht am gleichen Ort (Bayreuth und Marburg) statt. Dies könnte die Abarbeitung in der Pipeline durchaus beschleunigen. Grundsätzlich kann das MC-Modul auf jeder CUDA-fähige Grafikkarte arbeiten, bei der Bedienungsfreundlichkeit muss aber noch nachgebessert werden. Die Ausrichtungsmethode hängt von nicht-freien Bibliotheken ab, was noch eine Klärung bedarf.

Ähnliches gilt für das VR-Tool, welche zur Verifizierung verwendet wurde. Hier wurde die HTC-Vive als HMD[286] gewählt. Diese hat gegenüber der Oculus Rift (ebenfalls ein weit verbreitetes HMD-Modell) den Vorteil einer guten räumlichen Auflösung und ein gutes Tracking. Bei der Oculus Rift ist man bisher an den Arbeitsschreibtisch gebunden. Es wäre zu überlegen, ob man die Verifikation nicht für beide HMD-Modelle anbietet. Denn bis zum Zeitpunkt der Veröffentlichung von Steiniger et al. [Ste+18] war die Anbindung der Oculus-Controller unter OpenVR nicht möglich. Mit Vizard [Wora] ließe sich dies sehr komfortabel umsetzen. Es verwendet Python und kommt so Anwendern ohne Programmiererfahrung entgegen. Zu klären wäre hier, ob der Bedienungskomfort erhalten bliebe, wenn beispielsweise die Anwendung wächst.[287]

In dieser Dissertation wurde ein recht vielversprechendes Verfahren entwickelt, das in weiteren medizinischen Bereichen ein Gewinn sein kann. Die hier vorgestellten Methoden haben sehr gute

[284] Vgl. Kapitel 2.4.

[285] Mit diesem Begriff ist ein anerkanntes Verfahren oder anerkannte medizinische Behandlung gemeint, an der sich ein neues Verfahren oder eine neue Behandlung messen lassen muss. In Bezug auf die Erstellung des 3D-Modells soll hier ein Modell bezeichnet werden, dass der wahrheitsgemäßen Wiedergabe der Milzprobe am nächsten kommt.

[286] HMD steht für Head-Mounted Display.

[287] Mit Vizible [Worb] ließe sogar sehr bequem eine Interaktion zwischen Mediziner und Programmieren an unterschiedlichen Universitäten realisieren.

Ergebnisse geliefert und das Bedürfnis aus Schnittserien 3D-Modelle zur besseren Analyse zu erstellen wird immer vorhanden sein. Beispiele wären hier die Tumorforschung (Beschaffenheit und Aufbau eines Malignoms oder Karzinoms) oder psychologische Konditionierungsstudien bei Mäusen (mit anschließender Begutachtung von Mäusengehirnen). Aber vor allen für immunhistologische Forschungen, die sich mit Strukturen von unter $500\mu m$ befassen, ist in dieser Dissertation ein sehr guter Ansatz vorgestellt worden.

Index

Algorithmusverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Bildquellenangaben

Figure 5 by [Gra18] is in the public domain. It was publicated in 'Henry Gray (1827-1861) in 1918 *Anatomy of the Human Body*' (see [Gra18]). The Illustrator was Henry Vandyke Carter (1831-1897).

Figures 9 (b) and (c), 10, 11, 13, 14 and 15 by [Ulr+14b] are © Eurographics Association 2014; It is reproduced by kind permission of the Eurographics Association.

Figures 17 (a), 18 (a), 19, 20, 21 (a)-(c) and 22 by [Ulr+14a] is published in the Communication papers proceedings of the WSCG 2014 and open access. Therefore it can be downloaded and used for educational and non-commercial purposes without paying any fee.

Figures 23 and 24 by [Lob+16] is licensed under CC BY 4.0.[288]

Figures 27, 28 and 30 by [Ste+18] is licensed under CC BY 4.0.[289]

Figures 9 (a) and 12 are reproduced by kind permissiom of Dr. Oleg Lobachev.

Figures 1, 6, 7 and 25 are reproduced by kind permissiom of Prof Dr. Birte Steiniger.

[288] https://creativecommons.org/licenses/by/4.0/
[289] https://creativecommons.org/licenses/by/4.0/

Literaturverzeichnis

[08] „IEEE Standard for Floating-Point Arithmetic“. In: *IEEE Std 754-2008* (Aug. 2008), S. 1–70. DOI: 10.1109/IEEESTD.2008.4610935.

[85] „IEEE Standard for Binary Floating-Point Arithmetic“. In: *ANSI/IEEE Std 754-1985* (1985). DOI: 10.1109/IEEESTD.1985.82928.

[ACE05] Dominique Attali, David Cohen-Steiner und Herbert Edelsbrunner. „Extraction and Simplification of Iso-surfaces in Tandem“. In: *Proceedings of the Third Eurographics Symposium on Geometry Processing*. SGP '05. Vienna, Austria: Eurographics Association, 2005, S. 139–148. ISBN: 3-905673-24-X. URL: http://dl.acm.org/citation.cfm?id=1281920.1281943.

[ARH05] M. Auer, P. Regitnig und G. A. Holzapfel. „An automatic nonrigid registration for stained histological sections“. In: *IEEE Transactions on Image Processing* 14.4 (2005), S. 475–486. ISSN: 1057-7149. DOI: 10.1109/TIP.2005.843756.

[Bay+08] Herbert Bay, Andreas Ess, Tinne Tuytelaars und Luc Van Gool. „Speeded-up robust features (SURF)“. In: *Computer vision and image understanding* 110.3 (Juni 2008). Erstmals vorgestellt 2006. http://www.vision.ee.ethz.ch/~surf/index.html, S. 346–359. DOI: 10.1016/j.cviu.2007.09.014. URL: http://www.vision.ee.ethz.ch/~surf/index.html.

[BB10] U. Bağci und Li Bai. „Automatic Best Reference Slice Selection for Smooth Volume Reconstruction of a Mouse Brain From Histological Images“. In: *IEEE Transactions on Medical Imaging* 29.9 (2010), S. 1688–1696. ISSN: 0278-0062. DOI: 10.1109/TMI.2010.2050594.

[BCU12] U. Bağci, Xinjian Chen und J. K. Udupa. „Hierarchical Scale-Based Multiobject Recognition of 3-D Anatomical Structures“. In: *IEEE Transactions on Medical Imaging* 31.3 (März 2012), S. 777–789. ISSN: 0278-0062. DOI: 10.1109/TMI.2011.2180920.

[BD08] Gary Bradski und A. Daebler. „Learning OpenCV. Computer vision with OpenCV library“. In: (Jan. 2008), S. 222–264.

[BK08] Gary Bradski und Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. Hrsg. von Mike Loukides und Rachel Monaghan. 1005 Gravenstein Highway North, Sebastoplo, CA 95472: O'Reilly Media Inc., Sep. 2008. ISBN: 978-0-596-51613-0.

[BK89] Ruzena Bajcsy und Stane Kovačič. „Multiresolution elastic matching“. In: *Computer Vision, Graphics, and Image Processing* 46.1 (Apr. 1989), S. 1–21. ISSN: 0734-189X. DOI: 10.1016/S0734-189X(89)80014-3.

[BL79] S. Beucher und C. Lantuejoul. „Use of Watersheds in Contour Detection“. In: *International Workshop on Image Processing: Real-time Edge and Motion Detection/Estimation, Rennes, France*. Sep. 1979.

[BM92] Paul J. Besl und Neil D. McKay. „A Method for Registration of 3-D Shapes“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), S. 239–256. ISSN: 0162-8828. DOI: 10.1109/34.121791.

[Boo02] Carl de Boor. „Spline Basics“. In: *Handbook of Computer Aided Geometric Design*. Hrsg. von Gerald Farin, Josef Hoschek und Myung-Soo Kim. Amsterdam: North-Holland, 2002. Kap. 6, S. 141–163. ISBN: 978-0-444-51104-1. DOI: https://doi.org/10.1016/B978-044451104-1/50007-1. URL: http://www.sciencedirect.com/science/article/pii/B9780444511041500071.

[Boo89] Fred L. Bookstein. „Principal Warps: Thin-Plate Splines and the Decompositions of Deformations“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11.6 (Juni 1989). doi: 10.1109/34.24792. url: http://user.engineering.uiowa.edu/~aip/papers/bookstein-89.pdf (besucht am 21. 01. 2018).

[Bou94] Paul Bourke. *Polygonising a scalar field*. Mai 1994. url: http://paulbourke.net/geometry/polygonise/ (besucht am 09. 12. 2018).

[Bro92] Lisa Gottesfeld Brown. „A Survey of Image Registration Techniques“. In: *ACM Computing Surveys (CSUR)* 24.4 (Dez. 1992), S. 325–376. issn: 0360-0300. doi: 10.1145/146370.146374. url: http://www.geo.uzh.ch/microsite/rsl-documents/research/SARlab/GMTILiterature/PDF/Bro92.pdf.

[Can86] John Canny. „A Computational Approach to Edge Detection“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (Nov. 1986), S. 679–698. doi: 10.1109/tpami.1986.4767851.

[CBP11] Amalia Cifor, Li Bai und Alain Pitiot. „Smoothness-guided 3-D reconstruction of 2-D histological images“. In: *NeuroImage* 56.1 (Mai 2011), S. 197–211. issn: 1053-8119. doi: 10.1016/j.neuroimage.2011.01.060.

[CH09] W. Cheung und G. Hamarneh. „n -SIFT: n -Dimensional Scale Invariant Feature Transform“. In: *IEEE Transactions on Image Processing* 18.9 (Sep. 2009), S. 2012–2021. issn: 1057-7149. doi: 10.1109/TIP.2009.2024578.

[Cha+11] Jonathan Chappelow, B. Nicolas Bloch, Neil Rofsky, Elizabeth Genega, Robert Lenkinski, William DeWolf und Anant Madabhushi. „Elastic registration of multimodal prostate MRI and histology via multiattribute combined mutual information“. In: *Medical Physics* 38.4 (Apr. 2011), S. 2005–2018. doi: 10.1118/1.3560879.

[Che95] Evgeni V. Chernyaev. *Marching cubes 33: construction of topologically correct isosurfaces*. Techn. Ber. CERN-CN-95-17. Geneva: CERN, Nov. 1995. url: http://cds.cern.ch/record/292771,%20http://cds.cern.ch/record/292771/files/cn-95-017.pdf http://cds.cern.ch/record/292771/files/cn-95-017.pdf.

[Cig+08] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli und Guido Ranzuglia. „MeshLab: an Open-Source Mesh Processing Tool“. In: *Eurographics Italian Chapter Conference*. Hrsg. von Vittorio Scarano, Rosario De Chiara und Ugo Erra. The Eurographics Association, 2008. isbn: 978-3-905673-68-5. doi: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.

[CR03] Haili Chui und Anand Rangarajan. „A new point matching algorithm for non-rigid registration“. In: *Computer Vision and Image Understanding* 89.2–3 (2003), S. 114–141. issn: 1077-3142. doi: 10.1016/s1077-3142(03)00009-2.

[Cuc+09] Gianmauro Cuccuru, Enrico Gobbetti, Fabio Marton, Renato Pajarola und Ruggero Pintus. „Fast Low-memory Streaming MLS Reconstruction of Point-sampled Surfaces“. In: *Proceedings of Graphics Interface 2009*. GI '09. Kelowna, British Columbia, Canada: Canadian Information Processing Society, 2009, S. 15–22. isbn: 978-1-56881-470-4. url: http://dl.acm.org/citation.cfm?id=1555880.1555893.

[DM98] Leonardo Dagum und Ramesh Menon. „OpenMP: an industry standard API for shared-memory programming“. In: *Computational Science & Engineering, IEEE* 5.1 (1998), S. 46–55. doi: 10.1109/99.660313.

[Dre09] M. Drewes. *Histologie: Spezielle Histologie*. 3. Aufl. Histologie: Spezielle Histologie / [Maximilian Drewes ; Ulrike Bommes-Ebert] 2. Medi-Learn, 2009. isbn: 978-3-938802-52-6.

[DT07] Christopher DeCoro und Natalya Tatarchuk. „Real-time Mesh Simplification Using the GPU“. In: *Symposium on Interactive 3D Graphics (I3D)*. Bd. 2007. Apr. 2007, S. 6. doi: 10.1145/1230100.1230128.

[Dup+10] Guilhem Dupuy, Bruno Jobard, Sebastien Guillon, Noomane Keskes und Dimitri Komatitsch. „Parallel extraction and simplification of large isosurfaces using an extended tandem algorithm“. In: *Computer-Aided Design* 42.2 (2010). ACM Symposium on Solid and Physical Modeling and Applications, S. 129–138. issn: 0010-4485. doi: 10.1016/j.cad.2009.04.016. url: http://www.sciencedirect.com/science/article/pii/S0010448509001493.

[Dyk+08] Christopher Dyken, Gernot Ziegler, Christian Theobalt und Hans-Peter Seidel. „High-speed Marching Cubes using HistoPyramids“. In: *Computer Graphics Forum* 27.8 (2008), S. 2028–2039. issn: 1467-8659. doi: 10.1111/j.1467-8659.2008.01182.x.

[EKE01] Klaus Engel, Martin Kraus und Thomas Ertl. *High-quality pre-integrated volume rendering using hardware-accelerated pixel shading*. 2001. doi: 10.1145/383507.383515.

[Elo] Jarno Elonen. *Thin Plate Spline editor - an example program in C++*. url: https://elonen.iki.fi/code/tpsdemo/ (besucht am 09. 12. 2018).

[Eye] Eyewire. *Eyewire*. url: https://eyewire.org/explore (besucht am 09. 12. 2018).

[Far03] Gunnar Farnebäck. „Two-Frame Motion Estimation Based on Polynomial Expansion“. In: *Image Analysis: 13th Scandinavian Conference, SCIA 2003 Halmstad, Sweden, June 29 – July 2, 2003 Proceedings*. Hrsg. von Josef Bigun und Tomas Gustavsson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, S. 363–370. isbn: 978-3-540-45103-7. doi: 10.1007/3-540-45103-x_50. url: https://doi.org/10.1007/3-540-45103-X_50.

[FB81] Martin A. Fischler und Robert C. Bolles. „Random Sample Consensus: A Paradigm for Model Fitting with Applications To Image Analysis and Automated Cartography“. In: *Communications of the ACM* 24.6 (Juni 1981), S. 381–395. doi: 10.1145/358669.358692.

[Fed+12] Andriy Fedorov, Reinhard Beichel, Jayashree Kalpathy-Cramer, Julien Finet, Jean-Cristophe Fillion-Robin, Sonia Pujol, Christian Bauer, Dominique Jennings, Fiona Fennessy, Milan Sonka, John Buatti, Stephen Aylward, James Miller, Steve Pieper und Ron Kikinis. „3D Slicer as an Image Computing Platform for the Quantitative Imaging Network“. In: *Magnetic Resonance Imaging* 30.9 (Nov. 2012), S. 1323–1341. doi: 10.1016/j.mri.2012.05.001.

[Flea] DocCheck Flexikon. *Mastzelle*. url: http://flexikon.doccheck.com/de/Mastzelle (besucht am 09. 12. 2018).

[Fleb] DocCheck Flexikon. *Milz*. url: http://flexikon.doccheck.com/de/Milz (besucht am 09. 12. 2018).

[FMP77] Henry Fuchs, Zvi M. Kedem und Samuel P. Uselton. „Optimal Surface Reconstruction from Planar Contours.“ In: 20 (Aug. 1977), S. 693–702. doi: 10.1145/359842.359846.

[FS11] D. Fong und M. Saunders. „LSMR: An Iterative Algorithm for Sparse Least-Squares Problems“. In: *SIAM J. Sci. Comput.* 33.5 (2011), S. 2950–2971. doi: 10.1137/10079687X. url: https://web.stanford.edu/group/SOL/software/lsmr/LSMR-SISC-2011.pdf.

[Gar] Michael Garland. „Quadric-Based Polygonal Surface Simplification“. (besucht am: 11.06.2017). Diss. url: http://mgarland.org/research/thesis.html,http://reports-archive.adm.cs.cmu.edu/anon/1999/CMU-CS-99-105.pdf.

[Gar04] Michael Garland. *QSlim 2.1*. Juli 2004. url: http://mgarland.org/software/qslim.html (besucht am 09. 12. 2018).

[GDG11] Nico Grund, Evgenij Derzapf und Michael Guthe. „Instant Level-of-Detail“. In: *Proceedings of the Vision, Modeling, and Visualization Workshop 2011, Berlin, Germany, 4-6 October, 2011*. 2011, S. 293–299. doi: 10.2312/PE/VMV/VMV11/293-299. url: http://dx.doi.org/10.2312/PE/VMV/VMV11/293-299.

[GH97] Michael Garland und Paul S. Heckbert. „Surface simplification using quadric error metrics“. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. 1997, S. 209–216. doi: 10.1145/258734.258849.

[GH98] Michael Garland und Paul S. Heckbert. „Simplifying surfaces with color and texture using quadric error metrics“. In: *Proceedings of the Conference on Visualization '98*. 1998, S. 263–269. doi: 10.1109/visual.1998.745312.

[Gij+05] J. M. Gijtenbeek, P. Wesseling, C. Maass, L. Burgers und J. A. van der Laak. „Three-dimensional reconstruction of tumor microvasculature: simultaneous visualization of multiple components in paraffin-embedded tissue“. In: *Angiogenesis* 8.4 (2005), S. 297–305. doi: 10.1007/s10456-005-9019-4.

[Gil+06] H. J. Gilhuis, J. A. van der Laak, J. Pomp, A. C. Kappelle, J. M. Gijtenbeek und P. Wesseling. „Three-dimensional (3D) reconstruction and quantitative analysis of the microvasculature in medulloblastoma and ependymoma subtypes“. In: *Angiogenesis* 9.4 (Dez. 2006), S. 201–208. doi: 10.1007/s10456-006-9054-9.

[Git] GitHub. *ValveSoftware/openvr: OpenVR SDK*. url: https://github.com/ValveSoftware/openvr (besucht am 09. 12. 2018).

[Gor+09] Subrahmanyam Gorthi, Valérie Duay, Nawal Houhou, Meritxell Bach Cuadra, Ulrike Schick, Minerva Becker, Abdelkarim S. Allal und Jean-Philippe Thiran. „Segmentation of Head and Neck Lymph Node Regions for Radiotherapy Planning Using Active Contour-Based Atlas Registration“. In: *IEEE Journal of Selected Topics in Signal Processing* 3.1 (Feb. 2009), S. 135–147. doi: 10.1109/jstsp.2008.2011104.

[Gos05] A Ardeshir Goshtasby. *2-D and 3-D image registration: for medical, remote sensing, and industrial applications*. Wiley, 2005.

[Gra18] Henry Gray. *Illustrations. Fig. 1217. Gray, Henry. 1918. Anatomy of the Human Body*. Illustrator Henry Vandyke Carter. 1918. url: http://www.bartleby.com/107/illus1217.html (besucht am 09. 12. 2018).

[Gru13] Nico Grund. „Rekonstruktion und Verarbeitung von Objekten und Szenen aus Kamerabildern“. Diss. Phillips-Universität Marburg, 2013.

[GTN03] S. Gefen, O. Tretiak und J. Nissanov. „Elastic 3-D alignment of rat brain histological images“. In: *IEEE Transactions on Medical Imaging* 22.11 (2003), S. 1480–1489. issn: 0278-0062. doi: 10.1109/TMI.2003.819280.

[Gue+01] E. Guest, E. Berry, R. A. Baldock, M. Fidrich und M. A. Smith. „Robust point correspondence applied to two- and three-dimensional image registration“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23.2 (2001), S. 165–179. issn: 0162-8828. doi: 10.1109/34.908967.

[Ham50] Richard W. Hamming. „Error Detecting and Error Correcting Codes“. In: *The Bell System Technical Journal* 29.2 (Apr. 1950), S. 147–160. issn: 0005-8580. doi: 10.1002/j.1538-7305.1950.tb00463.x. url: http://sb.fluomedia.org/hamming/.

[Han09] Heinz Handels. *Medizinische Bildverarbeitung*. Teubner B.G. GmbH, 11. März 2009. isbn: 3835100777. url: http://www.ebook.de/de/product/5773071/heinz_handels_medizinische_bildverarbeitung.html.

[HDF12] Jared Heinly, Enrique Dunn und Jan-Michael Frahm. „Comparative Evaluation of Binary Features“. In: *Computer Vision – ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part II*. Hrsg. von Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato und Cordelia Schmid. Bd. 7573. Berlin, Heidelberg: Springer Berlin Heidelberg, Okt. 2012, S. 759–773. isbn: 978-3-642-33709-3. doi: 10.1007/978-3-642-33709-3_54. url: http://cs.unc.edu/~jheinly/binary_descriptors.html.

[HGB93] W. Heiden, T. Goetze und J. Brickmann. „Fast generation of molecular surfaces from 3D data fields with an enhanced 'marching cubes' algorithm“. In: *Journal of Computational Chemistry* 14.2 (1993), S. 246–250. doi: 10.1002/jcc.540140212.

[HS88] Chris Harris und Mike Stephens. „A combined corner and edge detector.“ In: *Alvey vision conference*. Bd. 15. 50. Manchester, UK. 1988, S. 10–5244. doi: 10.5244/c.2.23.

[IBM] IBM. *IBM Spectrum LSF Suites - Overview - United States*. url: https://www.ibm.com/us-en/marketplace/hpc-workload-management (besucht am 09. 12. 2018).

[Inc] Abcam Inc. *CD Antigens*. url: http://docs.abcam.com/pdf/immunology/cdantigen_poster.pdf (besucht am 09. 12. 2018).

[Jav] Java Platform SE 7. *BufferedImage*. url: https://docs.oracle.com/javase/7/docs/api/java/awt/image/BufferedImage.html (besucht am 09. 12. 2018).

[JC06] Gunnar Johansson und Hamish Carr. „Accelerating marching cubes with graphics hardware“. In: *Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative research*. CASCON '06. Toronto, Ontario, Canada: ACM, 2006. doi: 10.1145/1188966.1189018.

[JS01] Mark Jenkinson und Stephen Smith. „A global optimisation method for robust affine registration of brain images“. In: *Medical Image Analysis* 5.2 (2001), S. 143–156. issn: 1361-8415. doi: http://dx.doi.org/10.1016/S1361-8415(01)00036-6.

[Ju+06] Tao Ju, Joe Warren, James Carson, Musodiq Bello, Ioannis Kakadiaris, Wah Chiu, Christina Thaller und Gregor Eichele. „3D volume reconstruction of a mouse brain from histological sections using warp filtering“. In: *Journal of Neuroscience Methods* 156.1–2 (Sep. 2006), S. 84–100. issn: 0165-0270. doi: http://dx.doi.org/10.1016/j.jneumeth.2006.02.020. url: http://www.sciencedirect.com/science/article/pii/S0165027006001282.

[Ju04] Tao Ju. „Robust Repair of Polygonal Models“. In: *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2004* 23.3 (Aug. 2004), S. 888–895. issn: 0730-0301. doi: 10.1145/1186562.1015815. url: http://doi.acm.org/10.1145/1015706.1015815.

[KB17] Adrian Kaehler und Gary Bradski. *Learning OpenCV 3*. Hrsg. von Dawn Schanafelt, Kristen Brown, Rachel Monaghan und James Fraleigh. 3. Aufl. 1005 Gravenstein Highway North, Sebastoplo, CA 95472: O'Reilly Media Inc., Dez. 2017. isbn: 978-1-491-93799-0.

[KH10] D. B. Kirk und W. W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Applications of GPU Computing Series. Elsevier Science, 2010. isbn: 978-0-12-381472-2. url: https://books.google.de/books?id=qW1mncii_6EC.

[Kha+14] Adnan Khan, Nasir Rajpoot, Darren Treanor und Derek Magee. „A Non-Linear Mapping Approach to Stain Normalisation in Digital Histopathology Images using Image-Specific Colour Deconvolution“. In: *IEEE Transactions on Biomedical Engineering* 61 (Juni 2014). doi: 10.1109/tbme.2014.2303294.

[Khr18] Khronos OpenCL Working Group. *The OpenCL™ Specification*. Mai 2018. URL: https://www.khronos.org/registry/OpenCL/specs/2.2/pdf/OpenCL_API.pdf (besucht am 09. 12. 2018).

[Kim+97] Boklye Kim, Jennifer L. Boes, Kirk A. Frey und Charles R. Meyer. „Mutual Information for Automated Unwarping of Rat Brain Autoradiographs“. In: *NeuroImage* 5.1 (1997), S. 31–40. ISSN: 1053-8119. DOI: 10.1006/nimg.1996.0251.

[Kob+01] Leif P. Kobbelt, Mario Botsch, Ulrich Schwanecke und Hans-Peter Seidel. „Feature sensitive surface extraction from volume data“. In: *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 2001, S. 57–66. ISBN: 1-58113-374-X. DOI: http://doi.acm.org/10.1145/383259.383265.

[Kor+09] Werner Korb, Andreas Boehm, Norman Geissler, Anke Hoffmeier, Michale Stephan, Christine Ulrich und Gero Strauss. „Automation und Mechatronik im OP“. In: *DESIGN&ELEKTRONIK Entwicklerforum Embedded goes medical*. Leipzig: HTWK Leipzig, Sep. 2009. ISBN: 978-3-7723-5519-6.

[KPV14] Ron Kikinis, Steve D. Pieper und Kirby G. Vosburgh. „3D Slicer: A Platform for Subject-Specific Image Analysis, Visualization, and Clinical Support“. In: *Intraoperative Imaging and Image-Guided Therapy*. Hrsg. von Ferenc A. Jolesz. New York, NY: Springer New York, 14. Jan. 2014, S. 277–289. ISBN: 978-1-4614-7657-3. DOI: 10.1007/978-1-4614-7657-3_19. URL: http://www.ebook.de/de/product/22281670/intraoperative_imaging_and_image_guided_therapy.html.

[Kri+85] J. H. van Krieken, J. Te Velde, J. Hermans und K. Welvaart. „The splenic red pulp; a histomorphometrical study in splenectomy specimens embedded in methylmethacrylate“. In: *Histopathology* 9.4 (1985), S. 401–416. DOI: 10.1111/j.1365-2559.1985.tb02824.x.

[Kus+15] Satoshi Kusumi, Daisuke Koga, Tatsuo Kanda und Tatsuo Ushiki. „Three-dimensional reconstruction of serial sections for analysis of the microvasculature of the white pulp and the marginal zone in the human spleen“. In: *Biomedical Research* 36.3 (2015), S. 195–203. DOI: 10.2220/biomedres.36.195.

[Lan06] Gudrun Lang. *Histotechnik: Praxislehrbuch für die Biomedizinische Analytik*. Springer, 2006. ISBN: 9783211331415. URL: https://books.google.de/books?id=FulU0tG7kaQC.

[LC87] William E. Lorensen und Harvey E. Cline. „Marching cubes: A high resolution 3D surface construction algorithm“. In: *ACM SIGGRAPH Computer Graphics* 21.4 (1987), S. 163–169. DOI: 10.1145/37401.37422.

[LCS11] Stefan Leutenegger, Margarita Chli und Roland Y. Siegwart. „BRISK: Binary robust invariant scalable keypoints“. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, S. 2548–2555. DOI: 10.1109/iccv.2011.6126542. URL: http://margaritachli.com/papers/ICCV2011paper.pdf.

[LF98] W. Linss und J. Fanghänel. *Histologie: Zytologie, allgemeine Histologie, mikroskopische Anatomie*. De-Gruyter-Lehrbuch. Berlin, New York: de Gruyter, 1998. ISBN: 3-11-014032-2. URL: https://books.google.de/books?id=S1HRxeGOQfMC.

[Lib] Open Source Computer Vision Library. *OpenCV library*. URL: http://opencv.org (besucht am 09. 12. 2018).

[Lin00] Peter Lindstrom. „Out-of-core simplification of large polygonal models“. In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. 2000, S. 259–262. DOI: 10.1145/344779.344912.

[Lin15] Tony Lindeberg. „Image matching using generalized scale-space interest points“. In: *Journal of Mathematical Imaging and Vision* 52.1 (2015), S. 3–36.

[Liv99] Y. Livnat. *NOISE, WISE and SAGE: Algorithms for Rapid Isosurface Extraction*. SCI Institute Technical Report UUSCI-1999-001. University of Utah, Dez. 1999.

[Lob+16] Oleg Lobachev, Christine Ulrich, Birte S. Steiniger, Verena Wilhelmi, Vitus Stachniss und Michael Guthe. *Supplementary Text Material for 'Feature-Based Multi-Resolution Registration of Immunostained Serial Sections'*. https://creativecommons.org/licenses/by/4.0/. Aug. 2016. URL: https://www.sciencedirect.com/science/article/pii/S136184151630127X.

[Lob+17] Oleg Lobachev, Christine Ulrich, Birte S. Steiniger, Verena Wilhelmi, Vitus Stachniss und Michael Guthe. „Feature-based multi-resolution registration of immunostained serial sections“. In: *Medical Image Analysis* 35 (Jan. 2017), S. 288–302. ISSN: 1361-8415. DOI: http://dx.doi.org/10.1016/j.media.2016.07.010. URL: http://www.sciencedirect.com/science/article/pii/S136184151630127X.

[Lob+18] Oleg Lobachev, Moritz Berthold, Birte S. Steiniger und Michael Guthe. „Inspection of histological 3D reconstructions in virtual reality“. Eingereicht bei TVCG. 2018.

[Lom] Herve Lombaert. *Manual Registration with Thin Plates*. URL: http://profs.etsmtl.ca/hlombaert/thinplates/ (besucht am 09. 12. 2018).

[Low04] David G. Lowe. „Distinctive image features from scale-invariant keypoints“. In: *International journal of computer vision* 60.2 (Nov. 2004), S. 91–110. DOI: 10.1023/b:visi.0000029664.99615.94. URL: http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf.

[Low99] David G. Lowe. „Object recognition from local scale-invariant features“. In: *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. Bd. 2. IEEE. 1999, S. 1150–1157. DOI: 10.1109/iccv.1999.790410.

[LP12] R. Lüllmann-Rauch und F. Paulsen. *Taschenlehrbuch Histologie*. 4. Aufl. DeL. Rüdigerstraße 14, D-70469 Stuttgart: George Thieme Verlag KG, Juli 2012. ISBN: 978-3-13-129244-5. DOI: 10.1055/b-002-11390. URL: https://books.google.de/books?id=s8eCBtGUzxcC.

[LS17] Oleg Lobachev und Birte S. Steiniger. *Capillary networks and follicular marginal zones in the human spleen. Three-dimensional models based on immunostained serial sections - Supplementary videos*. https://creativecommons.org/licenses/by-sa/4.0/. Juli 2017. URL: https://doi.org/10.5281/zenodo.1039241 (besucht am 09. 12. 2018).

[LSG17] Oleg Lobachev, Birte S. Steiniger und Michael Guthe. *Compensating Anisotropy in Histological Serial Sections with Optical Flow-Based Interpolation*. Mai 2017. DOI: 10.1145/3154353.3154366.

[LT97] Kok-Lim Low und Tiow-Seng Tan. „Model Simplification Using Vertex-clustering“. In: *Proceedings of the 1997 Symposium on Interactive 3D Graphics*. I3D '97. Providence, Rhode Island, USA: ACM, 1997, 75–ff. ISBN: 0-89791-884-3. DOI: 10.1145/253284.253310. URL: http://doi.acm.org/10.1145/253284.253310.

[Lue01] David P. Luebke. „A Developer's Survey of Polygonal Simplification Algorithms“. In: *IEEE Computer Graphics and Applications* 21.3 (Mai 2001), S. 24–35. ISSN: 0272-1716. DOI: 10.1109/38.920624. URL: http://dx.doi.org/10.1109/38.920624.

[Ma+08] Bin Ma, Zhuang Lin, Simon Winkelbach, Werner Lindenmaier und Kurt Dittmar. „Automatic registration of serial sections of mouse lymph node by using ImageReg“. In: *Micron* 39.4 (2008), S. 387–396. ISSN: 0968-4328. DOI: 10.1016/j.micron.2007.03.005.

[Mai+10] Elmar Mair, Gregory D. Hager, Darius Burschka, Michael Suppa und Gerhard Hirzinger. „Adaptive and Generic Corner Detection Based on the Accelerated Segment Test“. In: *Computer Vision - ECCV 2010*. Bd. 6312. LNCS. Poster presentation, (Venue impact ratings 29th in CS by citeseer). Springer Berlin Heidelberg, Sep. 2010, S. 183–196. doi: 10.1007/978-3-642-15552-9_14.

[Mal+04] Grégoire Malandain, Éric Bardinet, Koen Nelissen und Wim Vanduffel. „Fusion of autoradiographs with an MR volume using 2-D and 3-D linear transformations“. In: *NeuroImage* 23.1 (Apr. 2004), S. 111–127. issn: 1053-8119. doi: http://dx.doi.org/10.1016/j.neuroimage.2004.04.038.

[Mer] Merck KGaA. *CBL171 | Anti-Actin Antibody, smooth muscle, clone ASM-1*. url: http://www.merckmillipore.com/DE/de/product/Anti-Actin-Antibody-smooth-muscle-clone-ASM-1,MM_NF-CBL171 (besucht am 09. 12. 2018).

[Mes15] Message Passing. *MPI: A Message-Passing Interface Standard*. Juni 2015. url: https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf (besucht am 09. 12. 2018).

[MS91] Heinrich Mueller und Michael Stark. *Adaptive Generation of Surfaces in Volume Data*. Techn. Ber. 1991. url: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.46.2098.

[MSS94] Claudio Montani, Riccardo Scateni und Roberto Scopigno. „A modified look-up table for implicit disambiguation of Marching Cubes“. In: *The Visual Computer* 10.6 (Dez. 1994), S. 353–355. doi: 10.1007/bf01900830.

[Nik+03] Christophoros Nikou, Fabrice Heitz, Astrid Nehlig, Izzie Jacques Namer und Jean-Paul Armspach. „A robust statistics-based global energy function for the alignment of serially acquired autoradiographic sections“. In: *Journal of Neuroscience Methods* 124.1 (2003), S. 93–102. issn: 0165-0270. doi: 10.1016/S0165-0270(02)00369-2.

[NS00] Sabine Noll und Susanne Schaub-Kuhnen. *Praxis der Immunhistochemie*. Hrsg. von Heinz Höfler und Klaus-Michel Müller. Bd. 1. 978-3-437-45526-1. München ; Jena: Urban & Fischer, 2000. isbn: 3-437-45526-5. url: https://books.google.de/books?id=IzEIywAACAAJ.

[Nvia] Nvidia. *CUDA Zone | NVIDIA Developer*. url: https://developer.nvidia.com/cuda-zone (besucht am 09. 12. 2018).

[Nvib] Nvidia. *NVIDIA CUDA SDK: Physically-Based Simulation - Marching Cubes*. url: http://www.nvidia.com/content/cudazone/cuda_sdk/Physically-Based_Simulation.html#marchingCubes (besucht am 09. 12. 2018).

[Nvic] Nvidia. *Parallele Berechnungen mit CUDA | Was ist CUDA?/NVIDIA*. url: http://www.nvidia.de/object/cuda-parallel-computing-de.html (besucht am 09. 12. 2018).

[Nvi18] Nvidia. *CUDA C Programming Guide*. Version 9.1. März 2018. url: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.htm (besucht am 09. 12. 2018).

[NY06] Timothy S. Newman und Hong Yi. *A survey of the marching cubes algorithm*. 2006.

[Ope15] OpenMP Architecture Review Board. *OpenMP Application Program Interface Version 4.5*. Nov. 2015. url: http://www.openmp.org/mp-documents/spec30.pdf#urldate# (besucht am 09. 12. 2018).

[Our+01] S. Ourselin, A. Roche, G. Subsol, X. Pennec und N. Ayache. „Reconstructing a 3D structure from serial histological sections“. In: *Image and Vision Computing* 19 (1–2 Juni 2001), S. 25–31. doi: 10.1016/S0262-8856(00)00052-4.

[OZS14] Devrim Onder, Selen Zengin und Sulen Sarioglu. „A Review on Color Normalization and Color Deconvolution Methods in Histopathology“. In: *Applied Immunohistochemistry & Molecular Morphology* 22.10 (Juni 2014), S. 713–719. ISSN: 1541-2016. DOI: 10.1097/pai.0000000000000003.

[PB07] Bernhard Preim und Dirk Bartz. *Visualization in Medicine: Theory, Algorithms, and Applications*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007. ISBN: 0123705967, 9780080549057.

[PH97] Jovan Popovic und Hugues Hoppe. „Progressive Simplicial Complexes“. In: *Proceedings of SIGGRAPH 97*. Aug. 1997, S. 217–224.

[PLD05] H. Peng, Fulmi Long und C. Ding. „Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.8 (Aug. 2005), S. 1226–1238. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2005.159.

[Pom+13] François Pomerleau, Francis Colas, Roland Siegwart und Stéphane Magnenat. „Comparing ICP Variants on Real-World Data Sets“. In: *Autonomous Robots* 34.3 (Feb. 2013), S. 133–148. DOI: 10.1007/s10514-013-9327-2.

[Pre+07] William H. Press, Saul A. Teukolsky, William T. Vetterling und Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. 3. New York, NY, USA: Cambridge University Press, 2007. ISBN: 9780521880688.

[Qi+14] Shouliang Qi, Han Triest, Yong Yue, Mingjie Xu und Yan Kang. „Automatic pulmonary fissure detection and lobe segmentation in CT chest images“. In: *BioMedical Engineering OnLine* 13.1 (Mai 2014), S. 59. DOI: 10.1186/1475-925X-13-59.

[RB93] Jarek Rossignac und Paul Borrel. „Multi-resolution 3D approximations for rendering complex scenes“. In: *Modeling in Computer Graphics*. Hrsg. von Bianca Falcidieno und Tosiyasu L. Kunii. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, S. 455–465. ISBN: 978-3-642-78114-8.

[RD06] Edward Rosten und Tom Drummond. „Machine Learning for High-Speed Corner Detection“. In: *Computer Vision–ECCV 2006* (2006), S. 430–443. DOI: 10.1007/11744023_34.

[Rec+04] Frank Reck, Carsten Dachsbacher, Roberto Grosso, Günther Greiner und Marc Stamminger. „Realtime Isosurface Extraction with Graphics Hardware“. In: *Eurographics 2004, Short Presentations and Interactive Demos*. 2004, S. 33–36.

[Rei+01] Erik Reinhard, Michael Ashikhmin, Bruce Gooch und Peter Shirley. „Color Transfer Between Images“. In: *IEEE Computer Graphics and Applications* 21.5 (Sep. 2001), S. 34–41. ISSN: 0272-1716. DOI: 10.1109/38.946629. URL: http://www.cs.tau.ac.il/~turkel/imagepapers/ColorTransfer.pdf.

[RJ01] Arnout C. Ruifrok und Dennis Johnston. „Quantification of histochemical staining by color deconvolution.“ In: *Analytical and Quantitative Cytology and Histology* 23.4 (Aug. 2001), S. 291–299. URL: https://www.ncbi.nlm.nih.gov/pubmed/11531144#.

[Roc+01] A. Roche, X. Pennec, G. Malandain und N. Ayache. „Rigid registration of 3-D ultrasound with MR images: a new approach combining intensity and gradient information“. In: *IEEE Transactions on Medical Imaging* 20.10 (2001), S. 1038–1049. ISSN: 0278-0062. DOI: 10.1109/42.959301.

[Roe] Stefan Roettger. *Ohm Computergrafik/OpenGL Indexed Face Sets*. URL: http://schorsch.efi.fh-nuernberg.de/roettger/index.php/Computergrafik/GLIndexedFaceSets (besucht am 09. 12. 2018).

[Roe12] Stefan Roettger. *The Volume Library*. 2012. URL: http://schorsch.efi.fh-nuernberg.de/data/volume/ (besucht am 09. 12. 2018).

[Rue+99] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach und D. J. Hawkes. „Nonrigid registration using free-form deformations: application to breast MR images“. In: *IEEE Transactions on Medical Imaging* 18.8 (1999), S. 712–721. ISSN: 0278-0062. DOI: 10.1109/42.796284.

[Rul] Aaron Rulseh. *Nano-CT Resolves Fine, Structural Detail of Bones | Medgadget*. URL: https://www.medgadget.com/2010/09/nanoct_resolves_fine_structural_detail_of_bones.html (besucht am 09. 12. 2018).

[Saa+12] Stephan Saalfeld, Richard Fetter, Albert Cardona und Pavel Tomancak. „Elastic volume reconstruction from series of ultra-thin microscopy sections“. In: *Nature Methods* 9.7 (Juli 2012), S. 717–720. ISSN: 15487091. DOI: 10.1038/nmeth.2072.

[Sal06] David Salomon. *Curves and Surfaces for Computer Graphics*. 0-387-28452-4 (e-book). Berlin, Germany / Heidelberg, Germany / London, UK / etc.: Springer-Verlag, 2006, S. xvi + 460. ISBN: 0-387-24196-5. DOI: 10.1007/0-387-28452-4.

[SBS11] Birte S. Steiniger, Michael Bette und Hans Schwarzbach. „The open microcirculation in human spleens: a three-dimensional approach“. In: *Journal of Histochemistry & Cytochemistry* 59.6 (Apr. 2011), S. 639–648. DOI: 10.1369/0022155411408315.

[Sch] Ben Schwan. *Nano-MRT-Verfahren erlaubt Blick auf kleinste Strukturen | heise online*. URL: http://heise.de/-1803312 (besucht am 09. 12. 2018).

[Sch+01] Julia A. Schnabel, Daniel Rueckert, Marcel Quist, Jane M. Blackall, Andy D. Castellano-Smith et al. „A Generic Framework for Non-rigid Registration Based on Non-uniform Multi-level Free-Form Deformations“. In: *Medical Image Computing and Computer-Assisted Intervention*. LNCS 2208. Springer, 2001, S. 573–581. ISBN: 978-3-540-42697-4. DOI: 10.1007/3-540-45468-3_69.

[Sch+09] Michael Schünke, Udo Schumacher, Markus Voll, Karl H. Wesker und Erik Schulte. *Prometheus LernAtlas - Innere Organe: Histologischer Aufbau der Milz*. 2009. URL: https://eref.thieme.de/cockpits/clAna0001/0/coAna00038/4-6982 (besucht am 09. 12. 2018).

[Sch+12] Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis Rueden, Stephan Saalfeld, Benjamin Schmid, Jean-Yves Tinevez, Daniel James White, Volker Hartenstein, Kevin Eliceiri, Pavel Tomancak und Albert Cardona. „Fiji: an open-source platform for biological-image analysis“. In: *Nature methods* 9.7 (Juni 2012), S. 676–682. ISSN: 1548-7091. DOI: 10.1038/nmeth.2019. URL: http://europepmc.org/articles/PMC3855844.

[Sch13] Cameron Schaeffer. „A Comparison of Keypoint Descriptors in the Context of Pedestrian Detection: FREAK vs. SURF vs. BRISK“. In: (2013), S. 5. URL: http://cs229.stanford.edu/proj2012/Schaeffer-ComparisonOfKeypointDescriptorsInTheContextOfPedestrianDetection.pdf.

[SG01] Eric Shaffer und Michael Garland. „Efficient Adaptive Simplification of Massive Meshes“. In: *Proceedings of the Conference on Visualization '01*. VIS '01. San Diego, California: IEEE Computer Society, 2001, S. 127–134. ISBN: 0-7803-7200-X. URL: http://dl.acm.org/citation.cfm?id=601671.601690.

[She+96] Han-Wei Shen, Charles D. Hansen, Yarden Livnat und Christopher R. Johnson. „Isosurfacing in span space with utmost efficiency (ISSUE)“. In: *VIS '96: Proceedings of the 7th Conference on Visualization '96*. 28-29 October. San Francisco, California, United States: IEEE Computer Society Press, Okt. 1996, S. 287–294. ISBN: 0-89791-864-9. DOI: 10.1109/VISUAL.1996.568121. URL: https://www.amazon.com/7th-IEEE-Visualization-Conference-1996/dp/0897918649?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0897918649.

[Shi09] F. Y. Shih. *Image Processing and Mathematical Morphology: Fundamentals and Applications*. CRC Press, 2009. ISBN: 9781420089448. DOI: 10.1201/9781420089448. URL: https://books.google.de/books?id=DVpqN%5C_5BYEAC.

[SK10] Jason Sanders und Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. 1. Aufl. 0131387685. Addison-Wesley Professional, 2010. ISBN: 978-0-13-138768-3.

[SK90] Don Speray und Steve Kennon. „Volume probes: interactive data exploration on arbitrary grids“. In: *ACM SIGGRAPH Computer Graphics* 24.5 (1990), S. 5–12. ISSN: 0097-8930. DOI: http://doi.acm.org/10.1145/99308.99310.

[SML88] Alfred Schmitt, Heinrich Müller und Wolfgang Leister. „Ray Tracing Algorithms — Theory and Practice“. In: *Theoretical Foundations of Computer Graphics and CAD*. Hrsg. von Rae A. Earnshaw. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, S. 997–1030. ISBN: 978-3-642-83539-1. DOI: 10.1007/978-3-642-83539-1_42.

[Son+13] Y. Song, D. Treanor, A. J. Bulpitt und D. R. Magee. „3D reconstruction of multiple stained histology images“. In: *Journal of Pathology Informatics* 4.2 (2013), S. 7. DOI: 10.4103/2153-3539.109864.

[Sou] Sourceforge.net. *vvv download*. URL: https://sourceforge.net/projects/volren (besucht am 09. 12. 2018).

[SRB03] Birte S. Steiniger, Lars Rüttinger und Peter J. Barth. „The Three-dimensional Structure of Human Splenic White Pulp Compartments“. In: *Journal of Histochemistry & Cytochemistry* 51.5 (Mai 2003), S. 655–663. DOI: 10.1177/002215540305100511.

[Sta08] Susan Standring. *Gray's Anatomy, The Anatomical Basis of Clinical Practice, Expert Consult - Online and Print, 40th Edition*. 40. Madrid, Spain: Elsevier, Sep. 2008. ISBN: 978-0-8089-2371-8. URL: https://www.amazon.com/Anatomy-Anatomical-Clinical-Practice-Consult/dp/0808923714?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0808923714.

[Ste+18] Birte S. Steiniger, Christine Ulrich, Moritz Berthold, Oleg Lobachev und Michael Guthe. „Capillary networks and follicular marginal zones in human spleens. Three-dimensional models based on immunostained serial sections.“ In: *PLOS ONE* 13.2 (Feb. 2018). Hrsg. von Wasif N. Khan. https://creativecommons.org/licenses/by/4.0/, S. 1–21. ISSN: 1932-6203. DOI: https://doi.org/10.1371/journal.pone.0191019.

[Ste08] Johannes Steinmüller. *Bildanalyse*. Springer-Verlag GmbH, 11. Aug. 2008. ISBN: 978-3-540-79742-5. DOI: 10.1007/978-3-540-79743-2. URL: http://www.ebook.de/de/product/7441792/johannes_steinmueller_bildanalyse.html.

[Ste15] Birte S. Steiniger. „Human spleen microanatomy: why mice do not suffice“. In: *Immunology* 145.3 (2015), S. 334–346. ISSN: 1365-2567. DOI: 10.1111/imm.12469. URL: http://dx.doi.org/10.1111/imm.12469.

[Stü] Reto Stüdeli. *CT: Hounsfield-Einheiten*. URL: http://www.stuedeli.net/reto/medizin/kdb/content/radio/CT_HE.html (besucht am 09. 12. 2018).

[SW03] Scott Schaefer und Joe Warren. „Adaptive Vertex Clustering Using Octrees“. In: *SIAM Geometric Design and Computing*. 2003.

[SW04] Scott Schaefer und Joe Warren. „Dual Marching Cubes: Primal Contouring of Dual Grids“. In: *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*. 6-8 October. Washington, DC, USA: IEEE Computer Society, Okt. 2004, S. 70–76. ISBN: 0-7695-2234-3. DOI: 10.1109/pccga.2004.1348336.

[SW06] Anand Lal Shimpi und Derek Wilson. „NVIDIA's GeForce 8800 (G80): GPUs Re-architected for DirectX 10“. In: *AnandTech* (2006). URL: https://www.anandtech.com/show/2116/8.

[SW12] Uwe Schneider und Dieter Werner. *Taschenbuch der Informatik*. Hrsg. von Uwe Schneider. Bd. 7. Fachbuchverlag, März 2012. ISBN: 978-3-446-42638-2.

[Tau95] Gabriel Taubin. „A Signal Processing Approach to Fair Surface Design“. In: *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '95. New York, NY, USA: ACM, 1995, S. 351–358. ISBN: 0-89791-701-4. DOI: 10.1145/218380.218473. URL: http://doi.acm.org/10.1145/218380.218473.

[Thi98] J.-P. Thirion. „Image matching as a diffusion process: an analogy with Maxwell's demons“. In: *Med. Image Anal.* 2.3 (1998), S. 243–260. ISSN: 1361-8415. DOI: 10.1016/S1361-8415(98)80022-4.

[TK11] A. Tanács und Z. Kato. „Fast linear registration of 3D objects segmented from medical images“. In: *Conf. Biomedical Engineering and Informatics*. BMEI '11. 2011, S. 294–298. DOI: 10.1109/BMEI.2011.6098290.

[TPG99] G. M. Treece, R. W. Prager und A. H. Gee. „Regularised marching tetrahedra: improved iso-surface extraction“. In: *Computers and Graphics* 23.4 (1999), S. 583–598. DOI: 10.1016/s0097-8493(99)00076-x.

[TSD07] Natalya Tatarchuk, Jeremy Shopf und Christopher DeCoro. „Real-Time Isosurface Extraction Using the GPU Programmable Geometry Pipeline“. In: *ACM SIGGRAPH 2007 courses*. SIGGRAPH '07. San Diego, California: ACM, 2007, S. 122–137.

[Ulr+14a] Christine Ulrich, Nico Grund, Evgenij Derzapf, Oleg Lobachev und Michael Guthe. „Parallel iso-surface extraction and simplification“. In: *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*. Hrsg. von Václav Skala. University of West Bohemia, Plzeň, Czech Republic, Juni 2014, S. 361–368. ISBN: 978-80-86943-71-8. URL: http://wscg.zcu.cz/WSCG2014/!!_2014-WSCG-Communication.pdf.

[Ulr+14b] Christine Ulrich, Oleg Lobachev, Birte S. Steiniger und Michael Guthe. „Imaging the Vascular Network of the Human Spleen from Immunostained Serial Sections“. In: *Eurographics Workshop on Visual Computing for Biology and Medicine, VCBM 2014, Vienna, Austria, 2014. Proceedings*. Hrsg. von Ivan Viola, Katja Buehler und Timo Ropinski. The Eurographics Association, 2014, S. 69–78. ISBN: 978-3-905674-62-0. DOI: 10.2312/vcbm.20141185.

[Ulr08] Christine Ulrich. „3D-Rekonstruktion von 2D-Daten“. Diplomarbeit. Philipps-Universität Marburg, Juli 2008.

[Vlf] Vlfeat.org. *SIFT detector and descriptor*. URL: http://www.vlfeat.org/overview/sift.html (besucht am 09. 12. 2018).

[Wan+13] Tao Wan, B. Nicolas Bloch, Shabbar Danish und Anant Madabhushi. „A novel point-based nonrigid image registration scheme based on learning optimal landmark configurations“. In: *Medical Imaging 2013: Image Processing*. Proc. SPIE 8669. 2013, S. 866934. DOI: 10.1117/12.2007153.

[Wel+96] William M. Wells III, Paul Viola, Hideki Atsumi, Shin Nakajima und Ron Kikinis. „Multi-modal volume registration by maximization of mutual information“. In: *Medical Image Analysis* 1.1 (1996), S. 35–51. ISSN: 1361-8415. DOI: 10.1016/S1361-8415(01)80004-9.

[Wel06] Ulrich Welsch. *Sobotta Lehrbuch Histologie: Zytologie, Histologie, Mikroskopische Anatomie*. 2. Aufl. Elsevier, Urban&FischerVerlag, 2006. ISBN: 978-3-437-44430-2. URL: https://books.google.de/books?id=nioHTjTxYVEC.

[Wen12] Gunther Wennemuth. *Taschenatlas Histologie (German Edition)*. München: Urban & Fischer/Elsevier, 2012. ISBN: 978-3-437-41977-5. URL: https://www.amazon.com/Taschenatlas-Histologie-German-Gunther-Wennemuth-ebook/dp/B008AV4YVC?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B008AV4YVC.

[Wes91] L. A. Westover. *Splatting: A Parallel, Feed-forward Volume Rendering Algorithm*. University of North Carolina at Chapel Hill, 1991. URL: https://books.google.de/books?id=ywJEAQAAIAAJ.

[Wika] Wikipedia. *Autoradiographie*. URL: https://de.wikipedia.org/wiki/Autoradiographie (besucht am 09. 12. 2018).

[Wikb] Wikipedia. *Bilineare Filterung*. URL: https://de.wikipedia.org/wiki/Bilineare_Filterung (besucht am 09. 12. 2018).

[Wikc] Wikipedia. *Block-matching algorithm (en)*. URL: https://en.wikipedia.org/wiki/Block-matching_algorithm (besucht am 09. 12. 2018).

[Wikd] Wikipedia. *Gradient (Mathematik)*. URL: https://de.wikipedia.org/wiki/Gradient_(Mathematik) (besucht am 09. 12. 2018).

[Wike] Wikipedia. *Hamming-Abstand*. URL: https://de.wikipedia.org/wiki/Hamming-Abstand (besucht am 09. 12. 2018).

[Wikf] Wikipedia. *Histologie*. URL: https://de.wikipedia.org/wiki/Histologie (besucht am 09. 12. 2018).

[Wikg] Wikipedia. *Hookesches Gesetz*. URL: https://de.wikipedia.org/wiki/Hookesches_Gesetz (besucht am 09. 12. 2018).

[Wikh] Wikipedia. *Mip Mapping*. URL: https://de.wikipedia.org/wiki/Mip_Mapping (besucht am 09. 12. 2018).

[Wiki] Wikipedia. *Platform LSF*. URL: https://en.wikipedia.org/wiki/Platform_LSF (besucht am 09. 12. 2018).

[Wikj] Wikipedia. *Stitching*. URL: https://de.wikipedia.org/wiki/Stitching (besucht am 09. 12. 2018).

[Wikk] Wikipedia. *Transinformation*. URL: https://de.wikipedia.org/wiki/Transinformation (besucht am 09. 12. 2018).

[Wikl] Wikipedia. *Voxel*. URL: https://de.wikipedia.org/wiki/Voxel (besucht am 09. 12. 2018).

[Wir+05] Stefan Wirtz, Nils Papenberg, Bernd Fischer und Oliver Schmitt. „Robust and staining-invariant elastic registration of a series of images from histologic slices“. In: *Proc. SPIE*. Proc. SPIE 5747. 2005, S. 1256–1262. DOI: 10.1117/12.595246.

[Wiß] Christian Wissler. *Dreidimensionale Bilder vom Netzwerk kleinster Blutgefäße: ein neues hochauflösendes Verfahren aus Bayreuth*. (Pressemitteilung Nr. 038/2017 vom 19. April 2017). URL: https://www.uni-bayreuth.de/de/universitaet/presse/pressemitteilungen/2017/038-blutgefaesse/index.html (besucht am 09. 12. 2018).

[Wora] WorldViz. *Vizard*. URL: https://www.worldviz.com/vizard (besucht am 09. 12. 2018).

[Worb] WorldViz. *Vizible*. URL: https://www.worldviz.com/vizible (besucht am 09. 12. 2018).

[WV90] Jane Wilhelms und Allen Van Gelder. „Octrees for faster isosurface generation“. In: *ACM SIGGRAPH Computer Graphics* 24.5 (Nov. 1990), S. 57–62.

[XF04] Z. Xie und G. E. Farin. „Image registration using hierarchical B-splines“. In: *IEEE Transactions on Visualization and Computer Graphics* 10.1 (2004), S. 85–94. ISSN: 1077-2626. DOI: 10.1109/TVCG.2004.1260760.

[Zep04] Klaus Zeppenfeld. *Lehrbuch der Grafikprogrammierung: Grundlagen, Programmierung, Anwendung*. 1. Aufl. Spektrum Akademischer Verlag, 2004. ISBN: 3-8274-1028-2.

[ZF03] Barbara Zitová und Jan Flusser. „Image registration methods: a survey“. In: *Image & Vision Comput.* 21.11 (2003), S. 977–1000. ISSN: 0262-8856. DOI: 10.1016/s0262-8856(03)00137-9.

[Zie+06] Gernot Ziegler, Art Tevs, Christian Theobalt und Hans-Peter Seidel. *GPU point list generation through histogram pyramids*. Research Report MPI-I-2006-4-002. Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany: Max-Planck-Institut für Informatik, Juni 2006. URL: http://hdl.handle.net/11858/00-001M-0000-0014-680E-9.

Eigene Publikationen

In diese Dissertation sind folgende Publikationen eingeflossen:

[Lob+17] Oleg Lobachev, Christine Ulrich, Birte S. Steiniger, Verena Wilhelmi, Vitus Stachniss und Michael Guthe. „Feature-based multi-resolution registration of immunostained serial sections“. In: *Medical Image Analysis* 35 (Jan. 2017), S. 288–302. ISSN: 1361-8415. DOI: http://dx.doi.org/10.1016/j.media.2016.07.010. URL: http://www.sciencedirect.com/science/article/pii/S136184151630127X.

[Ste+18] Birte S. Steiniger, Christine Ulrich, Moritz Berthold, Oleg Lobachev und Michael Guthe. „Capillary networks and follicular marginal zones in human spleens. Three-dimensional models based on immunostained serial sections.“ In: *PLOS ONE* 13.2 (Feb. 2018). Hrsg. von Wasif N. Khan. https://creativecommons.org/licenses/by/4.0/, S. 1–21. ISSN: 1932-6203. DOI: https://doi.org/10.1371/journal.pone.0191019.

[Ulr+14a] Christine Ulrich, Nico Grund, Evgenij Derzapf, Oleg Lobachev und Michael Guthe. „Parallel iso-surface extraction and simplification“. In: *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*. Hrsg. von Václav Skala. University of West Bohemia, Plzeň, Czech Republic, Juni 2014, S. 361–368. ISBN: 978-80-86943-71-8. URL: http://wscg.zcu.cz/WSCG2014/!!_2014-WSCG-Communication.pdf.

[Ulr+14b] Christine Ulrich, Oleg Lobachev, Birte S. Steiniger und Michael Guthe. „Imaging the Vascular Network of the Human Spleen from Immunostained Serial Sections“. In: *Eurographics Workshop on Visual Computing for Biology and Medicine, VCBM 2014, Vienna, Austria, 2014. Proceedings*. Hrsg. von Ivan Viola, Katja Buehler und Timo Ropinski. The Eurographics Association, 2014, S. 69–78. ISBN: 978-3-905674-62-0. DOI: 10.2312/vcbm.20141185.

Sonstige eigene Arbeiten

[Kor+09] Werner Korb, Andreas Boehm, Norman Geissler, Anke Hoffmeier, Michale Stephan, Christine Ulrich und Gero Strauss. „Automation und Mechatronik im OP“. In: *DESIGN&ELEKTRONIK Entwicklerforum Embedded goes medical*. Leipzig: HTWK Leipzig, Sep. 2009. ISBN: 978-3-7723-5519-6.

[Ulr08] Christine Ulrich. „3D-Rekonstruktion von 2D-Daten“. Diplomarbeit. Philipps-Universität Marburg, Juli 2008.

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die von mir angegebenen Quellen und Hilfsmittel verwendet habe.

Weiterhin erkläre ich, dass ich die Hilfe von gewerblichen Promotionsberatern bzw. -vermittlern oder ähnlichen Dienstleistern weder bisher in Anspruch genommen habe, noch künftig in Anspruch nehmen werde.

Zusätzlich erkläre ich hiermit, dass ich keinerlei frühere Promotionsversuche unternommen habe.

Bayreuth, den

Christine Ulrich