Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik
Institut für Simulation und Graphik

Diplomarbeit

Visualisierung baumartiger anatomischer Strukturen mit MPU Implicits

Christian Schumann

Otto-von-Guericke-Universität Magdeburg
Institut für Simulation und Graphik
Fakultät für Informatik

Diplomarbeit

Visualisierung baumartiger anatomischer Strukturen mit MPU Implicits

von:	Christian Schumann
geboren am:	02.12.1979
in:	Weißenfels
Matrikelnummer:	157914

1.Gutachter:	Prof. Dr.-Ing. Bernhard Preim
2.Gutachter:	Prof. Dr. Stefan Schirra

Betreuer:	Prof. Dr.-Ing. Bernhard Preim
	Dipl.-Ing. Steffen Oeltze

Bearbeitungszeitraum:	24.01.2006 - 26.07.2006

Selbstständigkeitserklärung

Hiermit versichere ich, Christian Schumann (Matrikel-Nr. 157914), die vorliegende Arbeit allein und nur unter Verwendung der angegebenen Quellen angefertigt zu haben.

Christian Schumann, Juli 2006

Danksagung

In erster Linie möchte ich mich bei meinen beiden Betreuern Prof. Dr.-Ing. Bernhard Preim und Dipl.-Ing. Steffen Oeltze bedanken, die mir während des gesamten Bearbeitungszeitraums eine hervorragende Betreuung zu kommen ließen. Neben fachlichen Ratschlägen war auch ihre moralische Unterstützung äußerst wertvoll. Mein Dank gilt darüber hinaus allen anderen Mitgliedern der Arbeitsgruppe Visualisierung, die immer ein offenes Ohr für Fragen und Bitten hatten. Besonderer Dank gilt meinen fleißigen Korrekturlesern Katharina Plugge, Dipl.-Ing. Christian Hansen, Dr. Per Willenius und Dipl.-Ing. Alexandra Baer, die mit ihren wertvollen Ratschlägen zum Gelingen dieser Arbeit beitrugen. Nicht zuletzt möchte ich mich bei meinen Eltern bedanken, die mich bei allem, was ich je tat, uneingeschränkt unterstützt haben.

Christian Schumann, Juli 2006

Inhaltsverzeichnis

1 Einleitung **1**

2 Methoden zur Visualisierung von Gefäßbäumen **5**
2.1 Morphologie von anatomischen Gefäßen - Medizinische Hintergründe . . . 5
2.2 Überblick Gefäßvisualisierung . . . 8
2.3 Erzeugung eines Gefäßmodells . . . 10
2.3.1 Segmentierung . . . 10
2.3.2 Skelettierung und Graphenanalyse . . . 12
2.4 Modellbasierte Verfahren zur Oberflächenrekonstruktion . . . 14
2.4.1 Einfache Primitive . . . 14
2.4.2 Simplex Meshes . . . 15
2.4.3 Sub Division Surfaces . . . 17
2.4.4 Convolution Surfaces . . . 20
2.4.5 Freiformflächen . . . 21
2.5 Zusammenfassung . . . 22

3 Direkte Oberflächenrekonstruktion aus Volumendaten **25**
3.1 Explizite Beschreibung von Oberflächenstrukturen . . . 25
3.1.1 Marching Cubes . . . 25
3.1.2 Constrained Elastic Surface Nets . . . 26
3.2 Implizite Beschreibung von Oberflächenstrukturen . . . 28
3.2.1 Modellierung mit impliziten Oberflächen . . . 30
3.2.2 Polygonalisierung impliziter Oberflächen . . . 31
3.2.3 Oberflächenrekonstruktion mit impliziten Oberflächen . . . 35
3.3 MPU Implicits im Detail . . . 40
3.3.1 Blending auf Basis der Partition of Unity-Methode . . . 41
3.3.2 Adaptive Approximation auf Basis eines Octrees . . . 42
3.3.3 Bestimmung der lokalen Approximation . . . 44
3.4 Praktische Aspekte beim Einsatz von MPU Implicits . . . 47
3.4.1 Parameter des Algorithmus . . . 47
3.4.2 Anforderungen an die Eingabedaten . . . 49
3.5 Zusammenfassung . . . 50

4 Entwurf **51**
4.1 Einordnung in die Visualisierungspipeline . . . 51
4.2 Extraktion einer Punktwolke aus dem Segmentierungsergebnis . . . 52
4.2.1 Identifizierung dünner Strukturen . . . 55
4.2.2 Klassifizierung der Voxel . . . 56
4.2.3 Überabtastung der Voxel . . . 58
4.2.4 Reduzierung treppenartiger Artefakte . . . 58

4.2.5 Extraktion der Punkte und Normalenvektoren . . . 63
4.2.6 Überführung in Weltkoordinaten . . . 70
4.3 Ermittlung geeigneter Parameter . . . 71
4.3.1 Parameter für die Generierung der Impliziten Funktion . . . 72
4.3.2 Parameter für die Polygonaliserung . . . 74
4.4 Registrierung . . . 76
4.5 Zusammenfassung . . . 78

5 Implementierung 79
5.1 Entwicklungswerkzeuge . . . 79
5.1.1 Die Programmiersprache C++ . . . 79
5.1.2 Open Inventor . . . 79
5.1.3 Die Softwareplattform MeVisLab . . . 80
5.2 Umsetzung der Visualisierungspipeline . . . 81
5.3 Das Modul MPUPrepare . . . 82
5.3.1 Die Bedienoberfläche . . . 82
5.3.2 Repräsentation des Segmentierungsergebnisses . . . 82
5.3.3 Identifizierung dünner Strukturen . . . 85
5.3.4 Klassifizierung der Voxel . . . 85
5.3.5 Überabtastung der Voxel . . . 86
5.3.6 Reduzierung treppenartiger Artefakte . . . 87
5.3.7 Extraktion der Punkte und Normalenvektoren . . . 88
5.3.8 Ermittlung der Parameter . . . 89
5.3.9 Export der Punktwolke . . . 90
5.4 MPUI-Programm . . . 90
5.5 Das Modul PolyImport . . . 92

6 Ergebnisse und Validierung des Verfahrens 95
6.1 Ergebnisse . . . 95
6.1.1 Beispiele . . . 95
6.1.2 Rekonstruktion dünner Zweige . . . 100
6.1.3 Rekonstruktion von Verzweigungen . . . 101
6.1.4 Ermittelte Parameter . . . 101
6.1.5 Glattheit der Rekonstruktion . . . 102
6.1.6 Komplexitätsbetrachtungen . . . 104
6.1.7 Auflösung des Polygongitters . . . 106
6.1.8 Rekonstruktion anderer anatomischer Strukturen . . . 107
6.2 Validierung . . . 108

7 Zusammenfassung und Ausblick 111

Literaturverzeichnis 119

A Anhang 127
A.1 Definition eines Strukturelements . . . 127

1 Einleitung

Die Visualisierung baumartiger anatomischer Strukturen ist für viele Einsatzgebiete innerhalb der Medizin interessant. Sowohl Diagnostik, Therapieplanung, präoperative Planung von Eingriffen als auch die Simulation von Operationen im Rahmen der medizinischen Ausbildung können von einer informativen Darstellung der betroffenen Gefäßbäume profitieren. Eine solche Visualisierung muss je nach Anwendung in der Lage sein, sowohl die Topologie eines Gefäßbaumes als auch Details, wie etwa krankhafte Veränderungen, darzustellen.

Entsprechende Verfahren lassen sich in zwei Gruppen einteilen:

- Verfahren, die sich eng an den tomographischen Daten orientieren. Die Darstellung erfolgt dabei entweder direkt (*Direct Volume Rendering*) oder auf Basis einer Zwischenrepräsentation (*Surface Rendering*).

- *Modellbasierte Verfahren*, die zuerst eine symbolische Repräsentation des Gefäßbaumes ermitteln, und dann die Bestandteile des Modells auf geometrische Primitive abbilden. Hierbei wird zumeist vereinfachend angenommen, dass nicht-pathologische Gefäße einen kreisrunden Querschnitt besitzen.

Die erste Gruppe gibt die tomographischen Daten sehr genau wieder. Dadurch sind diese Verfahren zum Beispiel auch für die Diagnose von Gefäßkrankheiten geeignet. Leider geht mit dieser Genauigkeit auch oft eine artefaktbehaftete Darstellung einher. So führt zum Beispiel die Anwendung des bekannten *Marching Cubes*-Algorithmus [LC87] zu einer treppenartigen Darstellung, die vor allem bei dünnen Strukturen, wie feinen Ästen, unverhältnismäßig stark ausfällt. Weitere Ursachen für Artefakte sind Bildrauschen und Partialvolumeneffekt. Ein weiterer Nachteil der *Direct Volume Rendering*- und *Surface Rendering*-Verfahren liegt im Fehlen einer strukturellen Repräsentation des Gefäßsystems. Diese Verfahren generieren lediglich Oberflächen; Informationen über die Verzweigungsstruktur der repräsentierten Gefäßsysteme fehlen. Dadurch sind sie für eine umfangreiche Exploration sowie strukturelle Analyse von Gefäßsystemen nicht geeignet.

Weit besser geeignet für die strukturelle Analyse von Gefäßbäumen sind *modellbasierte Verfahren*. Die ihnen zugrunde liegende, graphbasierte Repräsentation des Gefäßbaumes unterstützt zum Beispiel die selektive Darstellung von Teilstrukturen oder das Vermessen von Pfaden und Verzweigungswinkeln. Sie eignen sich somit zum Beispiel für die Kommunikation von Verzweigungsmustern, welche für die Leber-lebend-Spende äußerst wichtig ist. Durch die meist zugrunde liegende Modellannahme kreisrunder Querschnitte sind diese Verfahren jedoch ungenauer und eignen sich daher nicht für die Gefäßdiagnose. Ein weiterer Nachteil resultiert aus der Abbildung von Bestandteilen des Modells auf einfache geometrische Primitive. Viele Verfahren nutzen hier Zylinder ([BGSC85], [GKS+93], [MMD96]) oder Kegelstümpfe ([PTN97], [HPSP01]), was zu Diskontinuitäten in der Schattierung der resultierenden Oberfläche, der Erzeugung von Strukturen im Inneren der Gefäße, sowie zu einer unnatürlichen Abbildung von Verzweigungen führen kann.

Einige wenige modellbasierte Verfahren widmen sich speziell der Vermeidung dieser Darstellungsprobleme durch die Nutzung anderer Geometrierepräsentationen wie *Subdivision Surfaces* [FFKW02], *Convolution Surfaces* [Oel04] oder *Simplex Meshes* [BRB05]. Eine qualitativ hochwertige Darstellung ist so möglich. Die aus der Verwendung kreisförmiger Gefäßquerschnitte resultierende Ungenauigkeit bleibt jedoch bestehen.

Innerhalb dieser Arbeit soll ein Verfahren entwickelt werden, das die Vorteile der beiden Gruppen kombiniert. Ziel ist ein Kompromiss zwischen Genauigkeit und einer möglichst glatten, organischen Darstellung. Das Verfahren soll in der Lage sein, die Oberfläche von Gefäßsystemen auf Basis von Patientendaten zu rekonstruieren.

Die Oberfläche soll dabei ein möglichst organisches Erscheinungsbild aufweisen. Hierunter ist eine Darstellung mit einer kontinuierlichen Schattierung und ohne jegliche Artefakte zu verstehen. Ein treppenartiges Erscheinungsbild oder Strukturen im Inneren der Gefäße sollen vermieden werden. Zu einer organischen Darstellung von Gefäßen zählt zudem die natürliche Darstellung von Verzweigungen und Gefäßenden.

Damit das Verfahren auch zur Diagnose von Gefäßkrankheiten wie Stenosen und Aneurysmen genutzt werden kann, muss es in der Lage sein, beliebige Gefäßquerschnitte darstellen zu können. Dies entspricht der Anforderung, dass sich die Oberfläche möglichst nah am Segmentierungsergebnis orientiert.

Um auch eine Nutzung im Rahmen der angesprochenen explorativen und analytischen Aufgaben zu ermöglichen, sollte die erzeugte Oberfläche mit einem Gefäßmodell kombiniert werden. Die Anforderungen lassen sich durch folgende Punkte zusammenfassen:

Ein geeignetes Verfahren soll

- das Segmentierungsergebnis und somit auch die Gefäßquerschnitte möglichst genau wiedergeben,
- eine glatte, geschlossene, organische wirkende Oberfläche ohne Strukturen im Inneren erzeugen,
- Verzweigungen durch organisch wirkende Übergänge darstellen können,
- Gefäßenden schließen,
- in der Lage sein, auch sehr dünne Strukturen zu rekonstruieren und
- die generierte Oberfläche mit einem symbolischen Gefäßmodell kombinieren.

Für die Darstellung organischer Oberflächen haben sich implizite Oberflächen als geeignet erwiesen. In dieser Arbeit wird deshalb untersucht, inwiefern *Multi-level Partition of Unity Implicits (MPU Implicits)* [OBA+03b], ein relativ junges, aber bereits verbreitetes implizites Verfahren für die Oberflächenrekonstruktion, dazu geeignet ist, baumartige anatomische Strukturen zu rekonstruieren. Dazu muss eine Methode entwickelt werden, die basierend auf dem Segmentierungsergebnis eine Punktwolke erzeugt, da diese die Grundlage für die Rekonstruktion mit MPU Implicits darstellt.

Die Gliederung der Arbeit gestaltet sich folgendermaßen:

- Im anschließenden **Kapitel 2** wird ein Überblick über die dreidimensionale Gefäßvisualisierung gegeben. Hierbei wird die Gültigkeit der Annahme kreisförmiger Gefäßquerschnitte diskutiert. Modellbasierte Gefäßvisualisierungsverfahren werden vorgestellt.

- In **Kapitel 3** werden Verfahren zur direkten Oberflächenrekonstruktion aus Volumendaten vorgestellt. Es werden sowohl explizite als auch implizite Beschreibungen von Oberflächenstrukturen betrachtet. Basierend darauf wird die Nutzung von *Multi-level Partition of Unity Implicits* motiviert. Der Algorithmus wird anschließend ausführlich betrachtet.

- **Kapitel 4** legt den Entwurf einer auf MPU Implicits basierenden Visualisierungsmethode für Gefäßbäume dar. Dabei wird hauptsächlich die Überführung des Segmentierungsergebnisses in eine Punktwolke, die als Input für die Generierung der impliziten Oberfläche dient, beschrieben.

- **Kapitel 5** widmet sich der Umsetzung der in Kapitel 4 beschriebenen Methode. Benutzte Werkzeuge und Programme werden vorgestellt, relevante Algorithmen und Datenstrukturen des Entwurfs werden erläutert.

- In **Kapitel 6** findet eine Auswertung des beschriebenen Verfahrens statt. Dazu werden Ergebnisse des umgesetzten Verfahrens präsentiert und sowohl qualitativ, als auch quantitativ mit den Ergebnissen etablierter Verfahren verglichen.

- **Kapitel 7** fasst die Resultate der vorliegenden Arbeit zusammen. Probleme werden dargelegt und mögliche zukünftige Verbesserungen aufgezeigt.

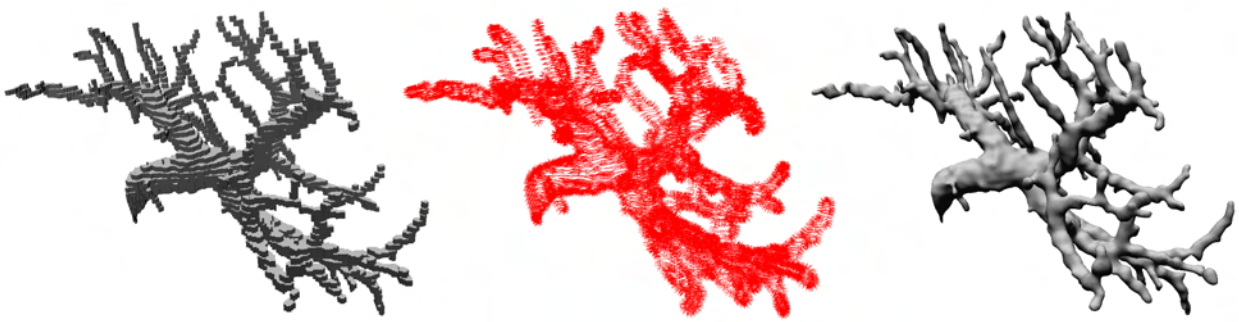


Abbildung 1.1: Visualisierung eines Lebergefäßbaumes mit MPU Implicits: Eine aus dem Segmentierungsergebnis (*links*) erzeugte Punktwolke (*mitte*) dient als Eingabe für die Generierung der impliziten Oberfläche (*rechts*).

2 Methoden zur Visualisierung von Gefäßbäumen

In diesem Kapitel soll ein Überblick über den gegenwärtigen Stand der Gefäßvisualisierung gegeben werden. Dabei wird zuerst die Morphologie von Gefäßen betrachtet. Die daraus gewonnenen Erkenntnisse stellen bei der darauf folgenden Betrachtung etablierter Verfahren ein wichtiges Kriterium für ihre Bewertung dar. Weitere Anforderungen wurden in der Einleitung aufgestellt.

Übersicht: Im folgenden Abschnitt wird die Annahme, dass der Querschnitt von Gefäßen kreisförmig ist, diskutiert. Dabei wird auf relevante medizinische Grundlagen eingegangen. Anschließend daran wird ein Überblick über konventionelle Techniken der Gefäßvisualisierung gegeben. Abschnitt 2.3 zeigt Möglichkeiten zur Rekonstruktion eines Gefäßmodells auf. In Abschnitt 2.4 wird gezeigt, wie ein solches Modell in eine Oberfläche überführt werden kann. Im letzten Abschnitt werden die Erkenntnisse über die vorgestellten Verfahren zusammengefasst und aktuelle Entwicklungen im Bereich der Gefäßvisualisierung aufgezeigt. Es wird gezeigt, warum die Nutzung eines Gefäßmodells trotz der aus der Modellannahme resultierenden Probleme wünschenswert ist.

2.1 Morphologie von anatomischen Gefäßen - Medizinische Hintergründe

Viele der etablierten Verfahren zur Gefäßvisualisierung stützen sich auf die Annahme, dass gesunde Gefäße einen kreisrunden ([BGSC85], [GKS+93], [MMD96], [HPSP01], [FFKW02], [OP05]) oder ellipsoiden [PTN97] Querschnitt haben. Als Referenz wird hierfür oft [MMD96] angeführt. In der entsprechenden Arbeit wird diese Aussage für nicht-pathologische Gefäße aufgestellt, aber nicht begründet.

Die Vorstellung eines kreisrunden Querschnitts ist eine Idealvorstellung, die aus mehreren Gründen unzutreffend sein kann. Zum einen wurde diese Annahme nur für Blutgefäße aufgestellt und fand auch nur bei Blutgefäßen Anwendung. Neuere Verfahren werden jedoch auch zur Darstellung von Bronchialbäumen genutzt (siehe z.B. [OP05]). Dies macht durchaus Sinn, da ihre Struktur der von Blutgefäßen sehr ähnlich ist. Allerdings weicht der Querschnitt vor allem bei der Luftröhre sowie den beiden Hauptbronchien stark von der Kreisförmigkeit ab (Abbildung 2.1).

Aber auch bei Blutgefäßen kann die Annahme, dass der Gefäßquerschnitt kreisrund ist, falsch sein. Die Annahme wurde ursprünglich nur für nicht-pathologische Gefäße aufgestellt. Aber auch in diesem Fall kann sie unzutreffend sein. So kann zum Beispiel eine Verformung eines Gefäßes auftreten, wenn der äußere Druck, *extravasaler Druck* genannt, groß genug ist. Besonders bei Venen stellt dies keine Ausnahme dar. Venen haben eine weitaus dünnere Wand als Arterien, vor allem die Muskelschicht, die *Tunica media*, ist vergleichsweise schwach. Als Teil des Niederdrucksystems ist ihr *intravasaler Druck*, der Druck, den das Blut von innen auf das Gefäß ausübt, auch geringer. Er übersteigt im Durchschnitt normalerweise nicht 20 mm Hg. Die Differenz zwischen intra- und extravasalen Druck wird als *transmuraler Druck* bezeichnet. Fällt der transmurale Druck unter

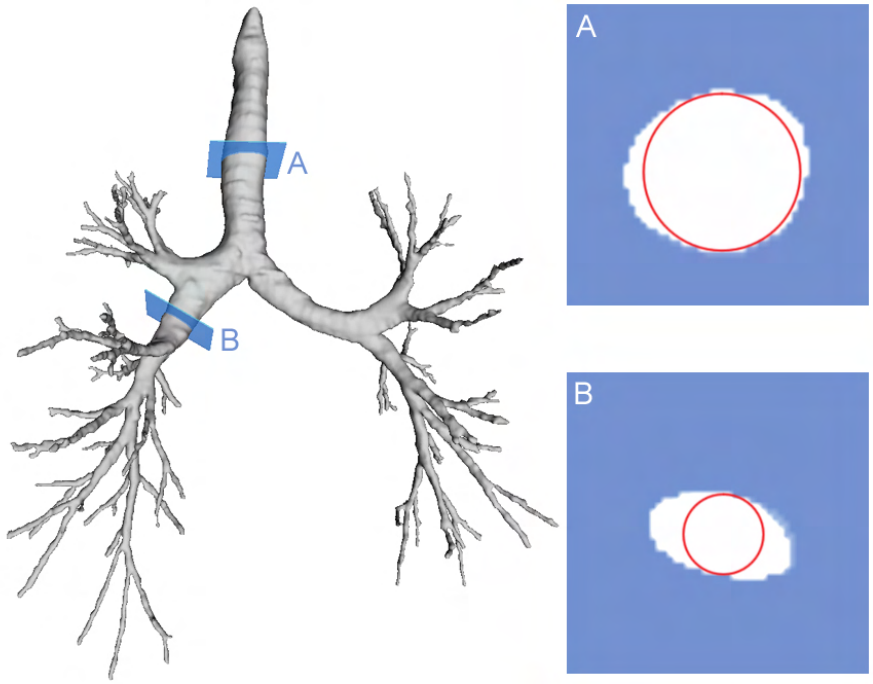


Abbildung 2.1: Morphologie von Bronchialbäumen: Bei einem Schnitt durch die Luftröhre (A) oder eine Hauptbronchie (B) ist zu erkennen, dass der Querschnitt stark von der Kreisförmigkeit abweicht. Zur Verdeutlichung wurden die Querschnitte mit einem Kreis überlagert.

10 mm Hg, kann es zu Verformungen kommen. Der Querschnitt wird hierbei ellipsoid. Geht der transmurale Druck gegen 0, kollabiert die Vene und ihr Querschnitt gleicht einer „8“. Ein kreisförmiger Gefäßquerschnitt kann wiederum durch einen geringen Druckzuwachs erreicht werden [SLT04]. Eine weitere Dehnung des Gefäßes ist dann nur noch bei einer deutlichen Druckerhöhung möglich. Das Dehnungs- und Verformungsverhalten der Venen in Abhängigkeit vom transmuralen Druck sowie die damit verbundene Querschnittsänderung sind in Abbildung 2.2 dargestellt.

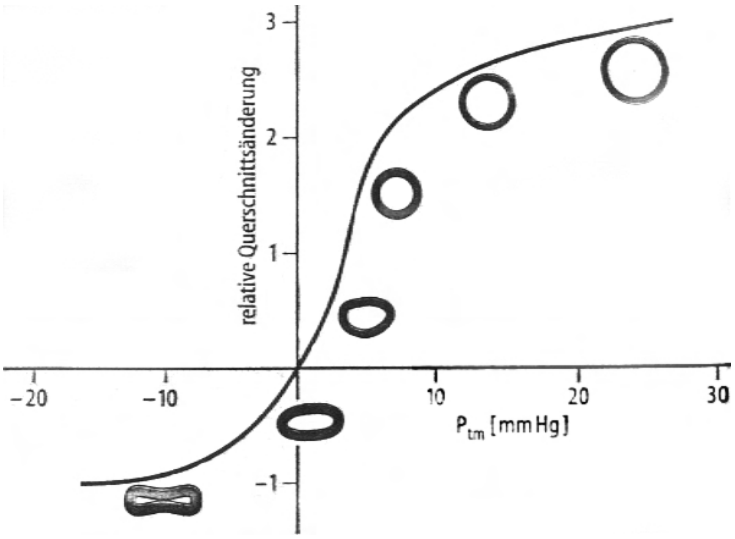


Abbildung 2.2: Änderung des Venenquerschnitts in Abhängigkeit vom transmuralen Druck (P_{tm}). *Quelle:* [SLT04]

Verformungen der Venen hängen also vom transmuralen Druck ab, der vor allem beim stehenden Menschen negativ sein kann. Kollapse der *Vena jugularis interna* (innere Drosselvene) und der *Vena cava superior* (obere Hohlvene) wurden durch angiographische Aufnahmen nachgewiesen [Mün02]. Eine von der Kreisförmigkeit abweichende Morphologie selbst bei nicht-pathologischen Blutgefäßen ist also durchaus nicht unnormal.

Ein weitaus zwingenderer Grund für die möglichst genaue Wiedergabe des Segmentierungsergebnisses ist jedoch die korrekte Wiedergabe von krankhaften Veränderungen im Gefäßsystem. Gerade hier weicht der Gefäßquerschnitt häufig von der Kreisförmigkeit ab. Aneurysmen können seitlich am eigentlichen Verlauf des Gefäßes wachsen (*sakkuläre Aneurysmen*) und sehr unterschiedliche Formen aufweisen, die nicht durch Skelett und Radieninformationen beschrieben werden können. Dies ist in Abbildung 2.3 und Abbildung 2.4 sehr deutlich zu sehen.

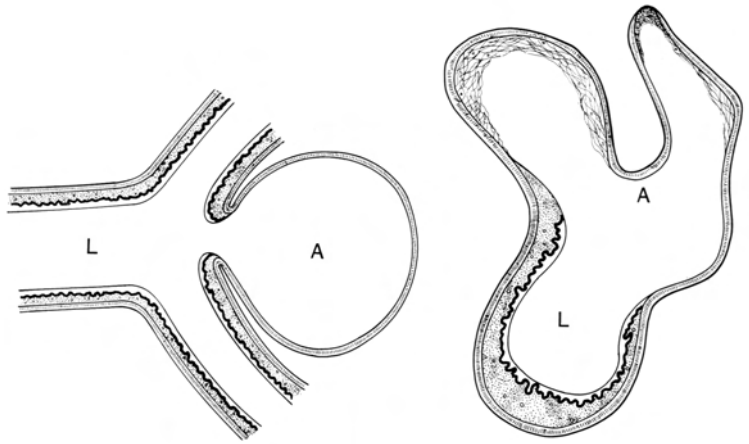


Abbildung 2.3: Illustration der Form *intrakranialer* Aneurysmen (Aneurysmen im Schädel): Schnitt entlang des Gefäßverlaufs durch ein *sakkuläres* Aneurysma (*links*) sowie Querschnitt senkrecht zum Gefäßverlauf bei einem intrakranialen Aneurysma mit kollabierten Beutel. Das ursprüngliche Gefäß ist mit *L* markiert, der Beutel des Aneurysmas mit *A*. *Quelle:* [Osb99]

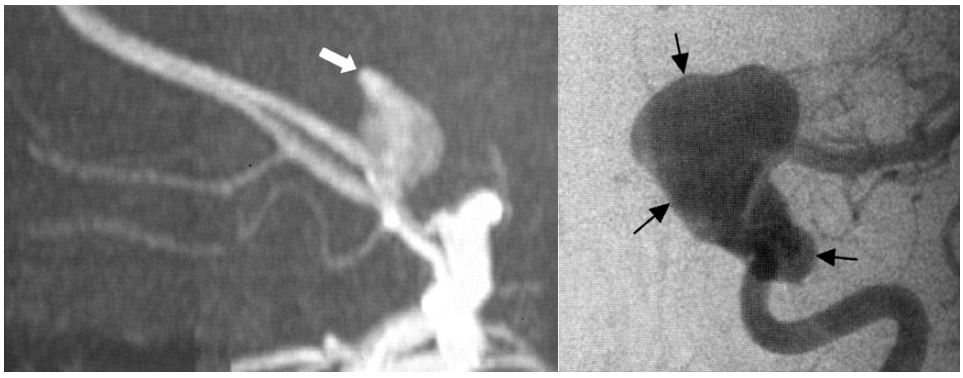


Abbildung 2.4: Morphologie von Aneurysmen: MR-Angiogramm eines intrakranialen Aneurysmas (*links*), Digitale Subtraktionsangiographie eines Aneurysma der *Arteria carotis interna* (*rechts*). Die Beutel sind mit Pfeilen markiert. *Quelle:* [Osb99]

Stenosen sind eine weitere häufige Form von krankhaften Veränderungen. Hierbei handelt es sich um Verengungen der Blutgefäße. Der Grund für eine solche Verengung ist häufig die sogenannte *Plaque*, Ablagerungen in Form von Kalk, Blutfetten, Cholesterin, Blutgerinseln oder Bindegewebe. Da die Ablagerung nicht gleichförmig auf dem gesamten Umfang des Gefäßes stattfindet, ist ein kreisförmiger Querschnitt des *Blutlumens* (das vom Blut eingenommene Volumen) nicht mehr gegeben (siehe Abbildung 2.5). Obwohl sich die Arbeit bisher immer auf Gefäße bezog, hängt es letztendlich von der Aufnahmetechnik und dem verwendeten Segmentierungsverfahren ab, ob das Lumen oder das Gefäß im Segmentierungsergebnis enthalten sind. So können moderne bildgebende Verfahren wie das *Multislice-CT* mit bis zu 64 Detektor-Zeilen Unterschiede zwischen Lumen und Ablagerung wiedergeben. Ist eine Darstellung des Lumens erwünscht, so muss die durch die Ablagerung veränderte Form des Lumens auch berücksichtigt werden.

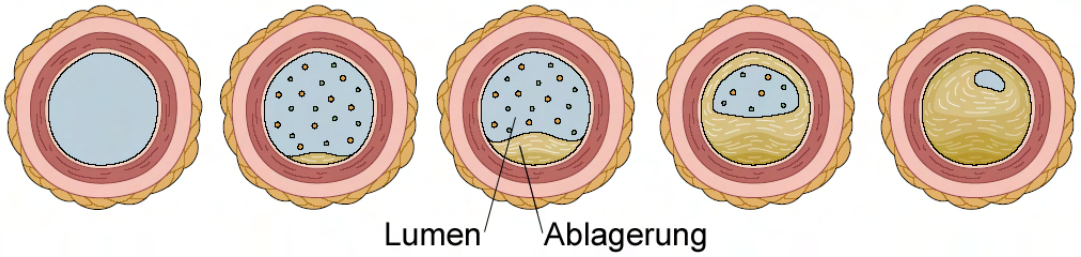


Abbildung 2.5: Veränderung des Lumenquerschnitts durch eine Stenose: Ablagerung von Cholesterin führt zu einer ungleichmäßigen Verengung, die mit der Zeit zunimmt. *Quelle:* [ES04]

Aufgrund der vorgestellten Ursachen für eine von der Kreisförmigkeit abweichende Morphologie ist es wünschenswert, anatomische Gefäße möglichst genau zu visualisieren, zum einem, um die Form gesunder Gefäße wahrheitsgemäß wieder zu geben. Hierzu zählen auch Bronchialbäume, deren Gefäßquerschnitt nicht kreisförmig ist. Aber vor allem für die Diagnose von Gefäßkrankheiten ist eine möglichst genaue Wiedergabe der Form unerlässlich.

2.2 Überblick Gefäßvisualisierung

Die gängigsten Verfahren zur Visualisierung von Gefäßen basieren auf einer zweidimensionalen Bildgebung. Grundlage dieser Verfahren sind zweidimensionale Bildgebungsverfahren wie die *Digitale Subtraktionsangiographie (DSA)* und *2D Ultraschall (US)*. Die Resultate der Bildgebungsverfahren werden direkt befundet, Verdeckungen sowie Rauschen erschweren hierbei jedoch die Diagnose [BFC04]. Dreidimensionale bildgebende Verfahren wie die *Computer Tomographie Angiographie (CTA)* und die *Magnet-Resonanz-Angiographie (MRA)* eliminieren das Problem der Verdeckung. Jede Schicht des resultierenden 3D-Datensatzes stellt nur dar, was in dieser Schicht enthalten ist. Eine schichtweise Darstellung der Daten, so genanntes *Slicing*, ermöglicht durch eine mentale Rekonstruktion die Betrachtung des Gefäßbaumes. Die hierfür notwendige Erfahrung ist jedoch meist Radiologen vorbehalten. Für Chirurgen ist das Verfahren somit eher ungeeignet.

Bei der *Multiplanaren Rekonstruktion (MPR)* kann der Benutzer beliebig ausgerichtete Ebenen durch den 3D-Datensatz legen und ist somit nicht mehr auf die parallelen Schich-

ten angewiesen. Dennoch ist es schwer, auf diese Weise dem Verlauf eines Gefäßes zu folgen. Eine Erweiterung dieses Verfahrens stellt die *Curved Planar Reformation (CPR)* dar. Dabei wird eine gekrümmte Fläche im Raum bestimmt, durch die das betrachtete Gefäß verläuft. Eine Projektion dieser Fläche auf eine Ebene erlaubt die zweidimensionale Betrachtung des Gefäßes. Ein Überblick über verschiedene CPR-Verfahren ist in [KFW+02] gegeben.

All diese Verfahren sind bedingt geeignet, einzelne Gefäße darzustellen, die Betrachtung eines ganzen Gefäßsystems ist mit ihnen jedoch nicht möglich. Für das Erfassen der gesamten Topologie eines Gefäßbaumes werden echte dreidimensionale Darstellungsverfahren benötigt. Um Lagebeziehungen zwischen Teilen des Baumes erfassen zu können, muss man in der Lage sein, den Baum von verschiedenen Seiten zu betrachten. Der Benutzer soll in der Lage sein, sowohl den gesamten Baum, als auch einzelne Teile zu untersuchen.

Eine Möglichkeit, einen tomographischen Datensatz dreidimensional darzustellen, ist das *Direct Volume Rendering (DVR)*. Dabei werden die Werte des Datensatzes mit Hilfe von *Transferfunktionen* auf Farb- und Transparenzwerte abgebildet und durch Projektion auf die Bildebene direkt dargestellt (Abbildung 2.6 links). *Maximum Intensity Projection (MIP)* ist hierfür ein verbreitetes Verfahren. Dabei wird für jeden Pixel entlang des Abtaststrahls das Voxel mit dem höchsten Datenwert ermittelt und dargestellt. Da auf diese Weise jedoch kleine Gefäße oder Gefäße mit zu geringem Kontrast fälschlicherweise als durch größere Gefäße verdeckt dargestellt werden, kommt es zu einer fehlerhaften Wiedergabe der Tiefeninformationen. Als Weiterentwicklung von MIP löst die *Closest Vessel Projection (CVP)* [Zui95] speziell dieses Problemen. Hierbei wird das Voxel dargestellt, das vom Betrachter aus gesehen entlang des Sehstrahls als erstes einen bestimmten Schwellwert überschreitet und ein lokales Maximum darstellt. Auf diese Weise werden kleine Gefäße, die vor größeren Gefäßen liegen, korrekt dargestellt.

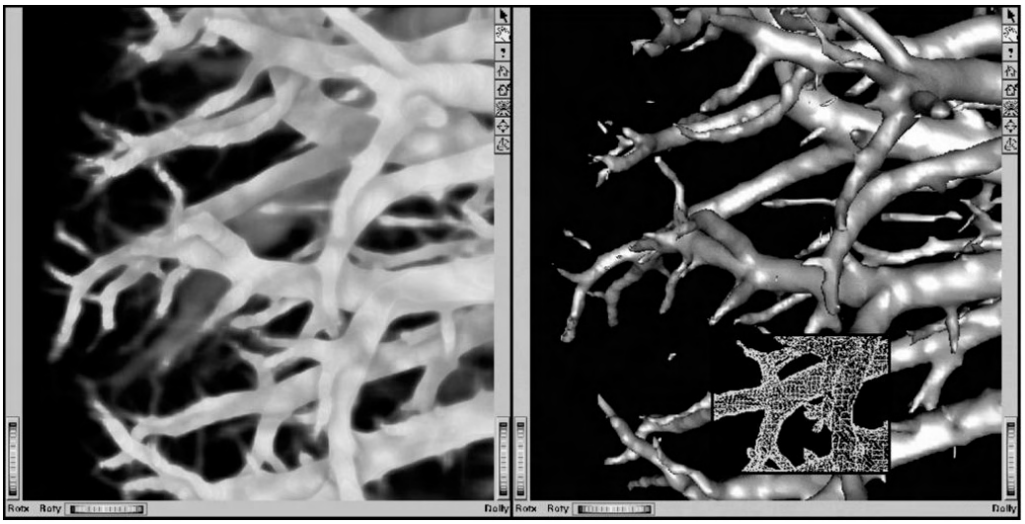


Abbildung 2.6: Visualisierung eines Gefäßbaumes mit *Direct Volume Rendering* (*links*) und *Surface Rendering* (*rechts*). *Quelle:* [HSEP00]

Die Gruppe der *Direct Volume Rendering*-Verfahren ist dennoch nur bedingt geeignet für die Visualisierung von Gefäßen. Vor allem technisch bedingte Probleme der Bildaufnahme wie Bildrauschen und Partialvolumeneffekt schränken die Nutzbarkeit ein. Durch die begrenzte Auflösung, die vor allem in Bezug auf kleine Strukturen wie Gefäße verhält-

nismäßig klein ist, kommt es zu störenden Aliasing-Artefakten. Darüber hinaus kann die Darstellung andere Strukturen beinhalten, die ähnliche Intensitätswerte aufweisen wie die Gefäße.

Weitaus geeigneter für die Darstellung baumartiger Strukturen sind Methoden aus der Kategorie *Surface Rendering*. Hierbei werden die interessierenden anatomischen Strukturen in eine Zwischenrepräsentation überführt. Ausgangspunkt dieser Verfahren können sowohl der originale Volumendatensatz, als auch das Segmentierungsergebnis sein. In den meisten Fällen basiert die Zwischenrepräsentation auf Polygonen (Abbildung 2.6 rechts). Entsprechende Verfahren werden in Kapitel 3 genauer vorgestellt, weil sie für diese Arbeit besonders relevant sind. Diese Form der Repräsentation erlaubt zum Beispiel eine effiziente hardwarebeschleunigte Darstellung.

2.3 Erzeugung eines Gefäßmodells

Grundlage modellbasierter Gefäßvisualisierungsverfahren ist die Rekonstruktion einer symbolischen Beschreibung des Gefäßsystems. Sie enthält den Gefäßverlauf, die Verzweigungsstruktur und Informationen über den lokalen Querschnitt des Gefäßbaumes. Die meisten Verfahren gehen hierbei von einem kreisförmigen Querschnitt aus. Die Gültigkeit dieser Annahme wurde bereits in Abschnitt 2.1 diskutiert. Diese vereinfachende Annahme schränkt die Nutzung in einigen Anwendungsgebieten, wie zum Beispiel der Gefäßdiagnose, ein, da die Darstellung pathologischer Veränderungen eine genaue Wiedergabe des Gefäßquerschnitts erfordert. Steht hingegen eine Wiedergabe der Gefäßtopologie sowie des Durchmesserverlaufs im Vordergrund, sind diese Verfahren bestens geeignet. Sie dienen nicht nur der Darstellung der Blutgefäße, sie geben auch *strukturelle* oder *topologische Informationen* wieder. So können sie zum Beispiel die räumliche Ausdehnung vermitteln und Lagebeziehungen zwischen einzelnen Gefäßen hervorheben [GKS+93]. Die Länge eines Pfades entlang des Gefäßverlaufs sowie Winkel an Verzweigungen können ermittelt werden [MMD96]. Darüber hinaus erlaubt die Analyse der Gefäßstruktur vielfältige Explorationsmöglichkeiten. So kann zum Beispiel die Anzeigekomplexität durch das Ausblenden ausgewählter Zweige oder Teilbäume kontrolliert werden. Die Auswahl der auszublendenden Strukturen kann dabei auf Basis der im Gefäßmodell enthaltenen Daten getroffen werden [HPSP01]. Auch die manuelle Auswahl von Teilstrukturen ist möglich und erlaubt das selektive Einfärben oder Ausblenden [NSE99].

In den folgenden Abschnitten werden Möglichkeiten zur Rekonstruktion einer solchen Gefäßrepräsentation aufgezeigt. Dabei steht das bei MeVis[1] entwickelte Verfahren ([SSPP00], [HPSP01], [SPSP02]) im Vordergrund , da es heute als State-of-the-Art gilt und auch in der in dieser Arbeit vorgestellten Methode Verwendung findet.

2.3.1 Segmentierung

Die *Segmentierung* stellt einen wichtigen Schritt bei der Rekonstruktion eines Gefäßmodells dar. Hierbei ist das Segmentierungsergebnis Ausgangspunkt für die Skelettierung. Darüberhinaus bildet das Segmentierungsergebnis die Grundlage vieler Surface Rendering-Verfahren.

[1] MeVis – Center for Medical Diagnostic Systems and Visualization in Bremen, http://www.mevis.de/, letzter Stand: 06.06.2006

Die Aufgabe der Segmentierung ist die Identifizierung der Voxel, die zu der betrachteten Struktur gehören. Es handelt sich also um eine binäre Klassifizierung des Datensatzes. Sie kann manuell durchgeführt werden, indem der Nutzer in jeder Bildschicht die Kontur des Objektes nachzeichnet. Dieser Vorgang ist jedoch äußerst zeitintensiv. Für Gefäßbäume ist die Anwendung sogar praktisch unmöglich, da eine Vielzahl von Ästen verfolgt werden muss. Deswegen wurde ein Vielzahl halbautomatischer und automatischer Verfahren entwickelt.

Entsprechende etablierte Verfahren sind *schwellwertbasierte Segmentierung, Region Growing, Wasserscheidentransformation* und *Live-Wire Segmentierung*. Ein aktueller umfassender Überblick über diese und weitere Verfahren ist in [Poh04] gegeben.

Ein verbreitetes Segmentierungsverfahren ist *Region Growing*. Ausgehend von einem oder mehreren Saatpunkten wird eine zusammenhängende Region extrahiert. Die Saatpunkte werden dabei vom Benutzer festgelegt. Sie stellen eine initiale Region dar, die nun schrittweise vergrößert wird. Unter Beachtung eines Homogenitätskriteriums werden nach und nach Nachbarvoxel hinzugefügt. Im einfachsten Fall besteht das Kriterium darin, dass der Grauwert der betrachteten Nachbarvoxel in einem bestimmtem Grauwertbereich liegen muss, damit das Voxel zur Region hinzugefügt wird.

In [SPSP02] kommt ein modifiziertes Region Growing-Verfahren zum Einsatz, das speziell für die Erkennung dünner Strukturen angepasst wurde. In einer Vorverarbeitung wird Rauschen durch die Anwendung eines *Median-* oder *Gauss*-Filters reduziert. Schwankungen in der Kontrastmittelverteilung werden mit Hilfe eines *Laplace*-Filters ausgeglichen (*Background Compensation*).

Anschließend wird eine Segmentierung basierend auf Region Growing durchgeführt. Der hierfür benutzte Schwellwert wird automatisch berechnet. Dazu wird eine ganze Serie von Segmentierungen vorgenommen. Die erste Segmentierung benutzt dabei den Grauwert des gewählten Saatvoxels als Schwellwert (θ_{beg}). Bei den darauf folgenden Segmentierungen wird der Schwellwert immer weiter reduziert, bis er den Endwert θ_{end} erreicht hat. Dieser Endwert sollte so gewählt sein, dass kein zu einem Gefäß gehörendes Voxel mehr im Ergebnis enthalten ist. Stellt man die benutzten Schwellwerte und die Anzahl der jeweils segmentierten Voxel in einem Diagramm gegenüber (siehe Abbildung 2.7), so weist die resultierende Kurve einen abfallenden Verlauf auf, deren Anstieg bis zu einem bestimmten Schwellwert θ_I fast konstant ist (Bereich I in Abbildung 2.7), danach rapide abfällt (Bereich II) und nach θ_{II} wieder konstant ist (Bereich III). Bereich I stellt die Segmentierungen mit zu geringem Schwellwert dar. Somit sind zu viele Hintergrundvoxel im Segmentierungsergebnis enthalten. Bereich III enthält die Ergebnisse mit zu hohem Schwellwert. Hier sind zu wenig zum Gefäß gehörende Voxel enthalten. Der optimale Schwellwert θ_{opt} liegt im mittlerem Bereich II. Passt man an die Datenpaare zwei Geraden an, so ergibt ihr Schnittpunkt eine gute Schätzung für den optimalen Schwellwert θ_{opt}. Selle et al. zufolge liefert dieses Verfahren für die meisten Datensätze ein gutes Segmentierungsergebnis [SPSP02]. Der so ermittelte Schwellwert kann jedoch interaktiv angepasst werden, sollte das Resultat nicht befriedigend sein.

Ein Überblick über weitere auf die Segmentierung von Gefäßen spezialisierte Verfahren ist in [KQ04] zu finden. Zusätzlich zu den bereits genannten Verfahren werden darin Ansätze basierend auf *Modellannahmen*, *Tracking*, *künstlicher Intelligenz* und *neuralen Netzen* vorgestellt.

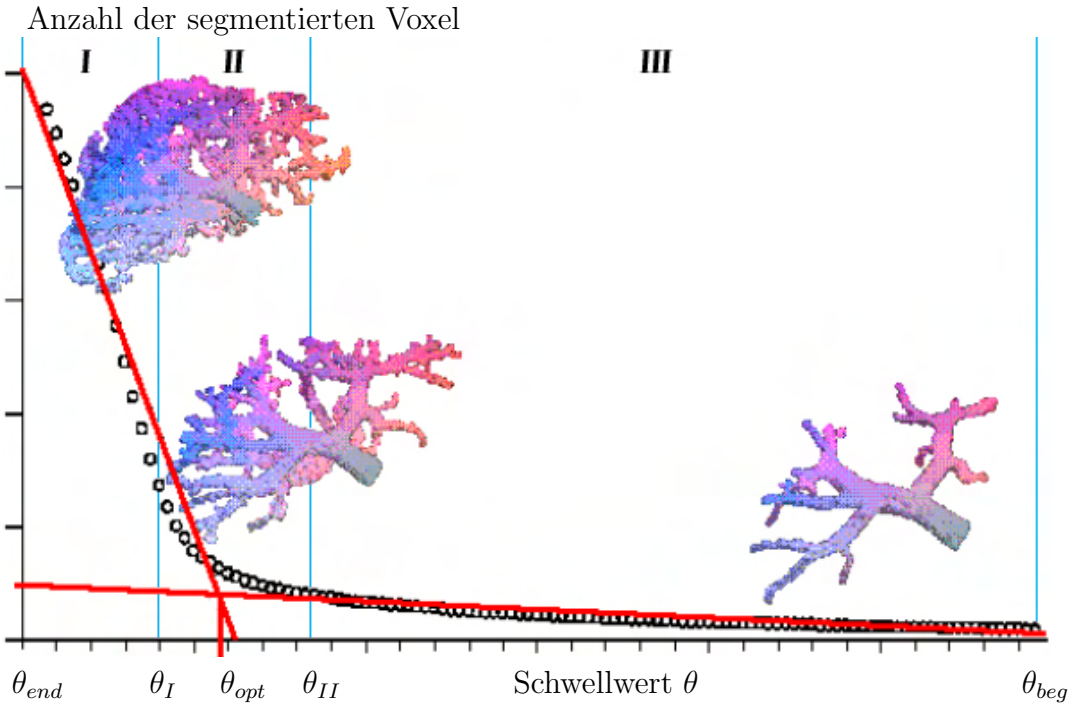


Abbildung 2.7: Bestimmung eines optimalen Schwellwertes θ_{opt}: An die Datenpaare bestehend aus Schwellwert und Anzahl segmentierter Voxel werden zwei Linien angepaßt. Ihr Schnittpunkt markiert θ_{opt}. *Quelle:* [SPSP02]

2.3.2 Skelettierung und Graphenanalyse

Ausgehend vom Segmentierungsergebnis liegt der nächste Schritt zur Rekonstruktion eines Gefäßmodells in der Extraktion eines *Skeletts*. Das Skelett bezeichnet hierbei die Menge der Voxel, die die Mittellinien der Gefäße repräsentieren [Blu67]. Zusätzlich enthält es für jeden Skelettvoxel Radieninformationen, welche den lokalen Gefäßquerschnitt beschreiben. Verschiedene Definitionen des Begriffs Skelett sind in [BFC04] zu finden. Die Extraktion des Skeletts wird als *Skelettierung* oder *Medial Axis Transform (MAT)* bezeichnet. Entsprechende Verfahren basieren auf *topologischem Thinning*, *Distanztransformationen* oder *Voronoi Diagrammen*. Einige setzen eine korrekte Segmentierung des Datensatzes voraus, andere arbeiten direkt auf den Originaldaten. Für die Skelettierung von Gefäßen sind vor allem auf Segmentierungsergebnissen basierende Verfahren von Interesse [BFC04].

Die bei MeVis entwickelte Gefäßvisualisierungspipeline [HPSP01] nutzt zur Skelettierung topologisches Thinning auf Basis des Segmentierungsergebnisses. Der binäre Datensatz wird hierbei schrittweise erodiert bis nur noch die Skelettvoxel übrig bleiben. Dabei wird besondere Sorgfalt auf den Erhalt der Gefäßtopologie verwandt.

Aufgrund der diskreten Natur der Ausgangsdaten sowie dem in den Daten vorhandenen Rauschen kann die Skelettierung zu einem artefaktbehafteten Ergebnis führen. Deswegen ist oft eine Nachbearbeitung des Skeletts erforderlich, um die Artefakte zu reduzieren. Als ein notwendiger Schritt wurde die Entfernung winziger Zweige erkannt [HPSP01]. Dieser Vorgang wird als *Pruning* bezeichnet. In [HPSP01] wird ein Zweig entfernt, wenn er im Verhältnis zum Radius eines in der Hierarchie vorangegangenen Zweiges zu klein ist. Die Grenze für dieses Verhältnis wurde durch Versuche ermittelt. Es ist fraglich, ob dieses Verhältnis bei jedem Datensatz zum gewünschten Ergebnis führt und nur irrelevante

Zweige entfernt. Selle et al. beschreiben ein Verfahren, das „irrelevante“ Zweige auf Basis des Gradienten der Distanztransformation des Segmentierungsergebnisses identifiziert und entfernt (Abbildung 2.8)[SPSP02]. Durch einen Parameter kann das Ergebnis beeinflusst werden. Es hängt also vom Benutzer ab, ob winzige Zweige als wichtige Daten, oder als Rauschen interpretiert werden. Somit kann Pruning in Abhängigkeit der gewählten Parameterbelegung auch die Genauigkeit des Verfahrens verringern.

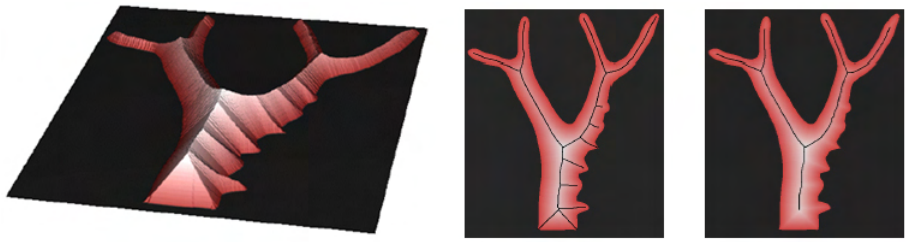


Abbildung 2.8: *Pruning* auf Basis des Gradienten der Distanztransformation: Visualisierung des Gradienten als Höhenfeld (*links*). Die Erkennung oder Vernachlässigung kleiner Zweige kann durch einen Parameter gesteuert werden (*mitte* und *rechts*). *Quelle:* [SPSP02]

In [HPSP01] wird des Weiteren eine Glättung des Skelettes mit einem [121]-*Binomialfilter* vorgeschlagen, da dieses auf Grund der auf das Voxelgitter beschränkten Positionen der Skelettvoxel einen zackigen Verlauf aufweisen kann (Abbildung 2.9).

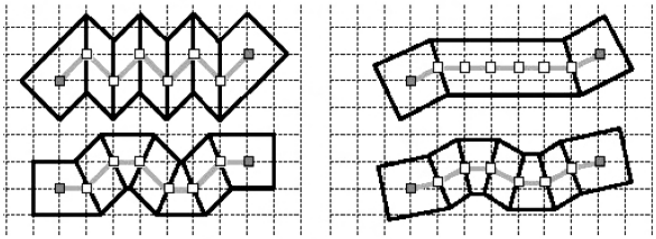
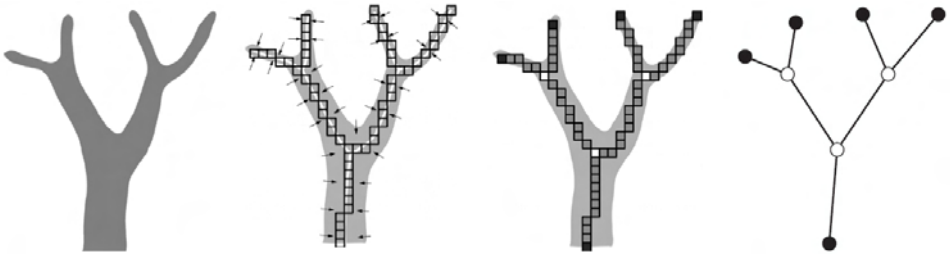


Abbildung 2.9: Glättung eines gezackten Skeletts (*links*) durch die Anwendung eines Binomialfilters (*rechts*). *Quelle:* [HPSP01]

Um eine Verzerrung des Gefäßverlaufs aufgrund von kleinen Seitenästen zu verhindern, wird eine Wichtung der Skelettvoxel auf Basis ihrer Relevanz vorgeschlagen, die bei der Glättung beachtet wird. Das gleiche Verfahren nutzen die Autoren für die Glättung des Durchmesserverlaufs. Ähnlich wie beim Pruning kann bei dieser Glättung nicht sicher gestellt werden, dass nicht auch relevante Informationen entfernt werden. Der zackige Verlauf mag teilweise aus der diskreten Natur der Segmentierung resultieren. Ebenso kann er aber auch der Wahrheit entsprechen. Es ist nicht möglich, hierüber eine Aussage zu treffen. Mit der Glättung kann somit auch eine Reduzierung der Genauigkeit einhergehen.

Wurde das Skelett erzeugt und nachbearbeitet, wird es einer *Graphenanalyse* unterzogen. Dabei wird das Skelett in einen gerichteten azyklischen Graph überführt. Skelettvoxel, die Verzweigungen oder Endpunkte darstellen, werden hierbei als Knoten betrachtet, die Verbindungen zwischen ihnen als Kanten. Für jede Kante werden die Positionen der Skelettvoxel entlang der Kante sowie die korrespondierenden minimalen und maximalen Radien abgespeichert. Diese Repräsentation erlaubt eine weiterführende Analyse des Gefäßbaumes. So können zum Beispiel separate Gefäßbäume im Segmentierungsergebnis erkannt und getrennt werden [HPSP01].

Die hier beschriebenen Schritte zur Rekonstruktion eines Gefäßmodells sind in Abbildung 2.10 zusammengefasst. Sie werden ausführlich in [SSPP00], [HPSP01] und [SPSP02] beschrieben.



(a) (b) (c) (d)

Abbildung 2.10: Umwandlung des Segmentierungsergebnisses (a) in ein Skelett (b). Aus dem nachbearbeiteten Skelett werden Skelettvoxel an Verzweigungen und Endpunkten extrahiert (c) und in einen gerichteten azyklischen Graphen (d) überführt. *Quelle:* [SSPP00]

2.4 Modellbasierte Verfahren zur Oberflächenrekonstruktion

Ein Gefäßmodell enthält mit dem Gefäßverlauf, der Verzweigungsstruktur und einer Beschreibung der lokalen Querschnitte alle Informationen, um eine näherungsweise Oberflächenrepräsentation des beschriebenen Gefäßbaumes zu rekonstruieren. Im Folgenden werden verschiedene Verfahren für die Oberflächenrekonstruktion auf Basis dieser Informationen vorgestellt. Der größte Teil der Verfahren basiert auf der Annahme kreisförmiger Querschnitte. Auf Konstruktion und Form der symbolischen Beschreibung wird hierbei nur eingegangen, wenn diese stark von dem in Abschnitt 2.3.2 vorgestellten Modell abweichen. Es werden vor allem die verwendeten Primitive und Geometrierepräsentationen betrachtet und in Hinsicht auf die in der Einleitung aufgestellten Anforderungen bewertet.

2.4.1 Einfache Primitive

Die Annahme kreisförmiger Querschnitte legt die Darstellung mit Hilfe von Primitiven nahe, die einen kreisförmigen Querschnitt aufweisen. Frühe Gefäßvisualisierungsverfahren nutzen Zylinder zur Repräsentation der Kanten des Gefäßmodells [BGSC85], [GKS+93], [MMD96]. Die Radien der Zylinder entsprechen dabei dem lokalen Radius des Gefäßes. Diese Verfahren sind bereits gut geeignet, die Topologie des Gefäßbaumes sowie lokale Durchmesser darzustellen. Der Verlauf der Durchmesser wird jedoch nicht wiedergegeben. Darüber hinaus sind die Übergänge zwischen benachbarten Zylindern sichtbar (Abbildung 2.11, links). Die Bildung von unerwünschten Strukturen im Inneren der Gefäße an Verzweigungen macht diese Methoden für Anwendungen, die eine Darstellung des Gefäßinneren erfordern, wie zum Beispiel die virtuelle Endoskopie, uninteressant. Auch eine teiltransparente Darstellung kann so nicht qualitativ hochwertig erfolgen.

Durch die Verwendung von Kegelstümpfen ([PTN97], [HPSP01]) können einige dieser Probleme abgeschwächt, jedoch nicht vollkommen behoben werden. Der Durchmesserverlauf wird korrekt dargestellt und die Diskontinuitäten entlang der Oberfläche werden

geringer, verschwinden jedoch nicht. Die Verwendung von Halbkugeln zum Schließen der Gefäßenden erhöht die Darstellungsqualität weiter (Abbildung 2.11, rechts)[HPSP01]. Die Darstellung der Verzweigungen ist jedoch nach wie vor nicht befriedigend (Abb. 2.12, links). Auch werden weiterhin Strukturen im Inneren gebildet (Abbildung 2.12, rechts).

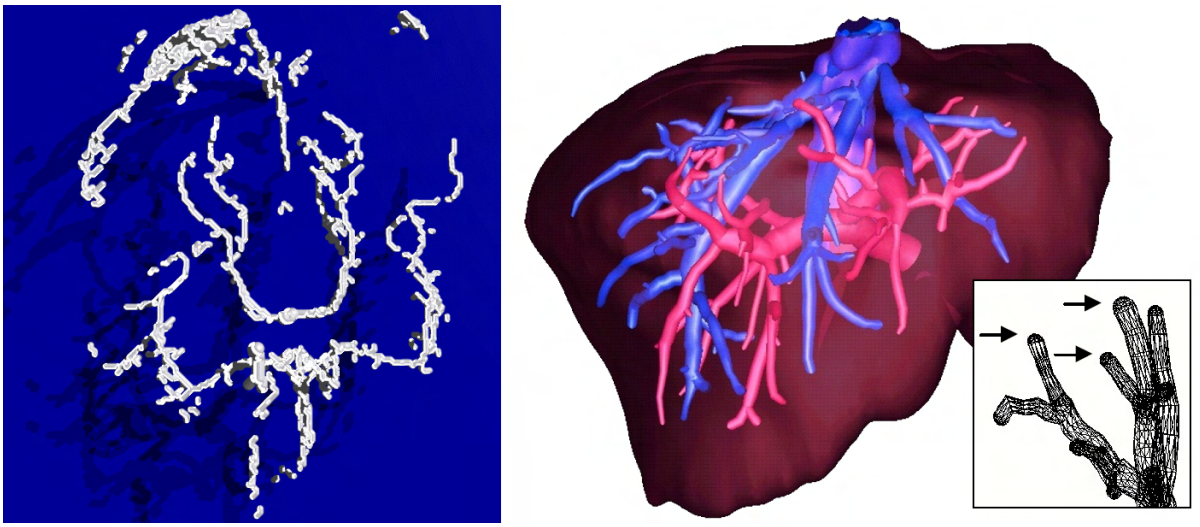


Abbildung 2.11: Visualisierung mit einfachen Primitiven: Die Darstellung mit Zylindern erlaubt die Beurteilung der Gefäßtopologie (*links, Quelle:* [GKS+93]). Die Verwendung von Kegelstümpfen (*rechts, Quelle:* [HPSP01]) verbessert die Darstellung in Hinsicht auf die Wiedergabe des Durchmesserverlaufs und verringert die Diskontinuitäten an Übergängen benachbarter Primitive. Die Gefäßenden werden durch Halbkugeln geschlossen (*rechts, Inlet*).

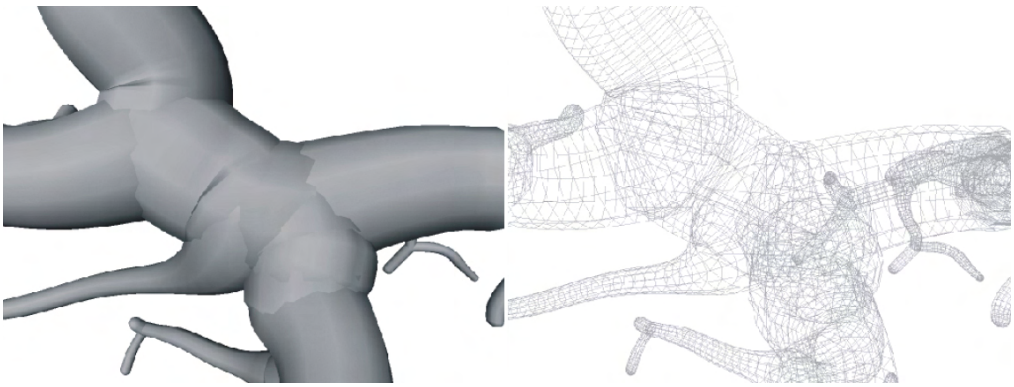


Abbildung 2.12: Visualisierung mit Kegelstümpfen: die Darstellung an Verzweigungen ist noch nicht befriedigend. Es sind deutlich Diskontinuitäten in der Schattierung der Oberfläche sichtbar (*links*). Die Drahtgitterdarstellung zeigt zudem, dass das Modell Strukturen im Inneren enthält (*rechts*). *Quelle:* [Oel04]

2.4.2 Simplex Meshes

Simplex Meshes wurden in [Del99] als eine Form deformierbarer Modelle vorgestellt. Topologisch entspricht ein Simplex Mesh einem Dreiecksnetz. Dabei kann ein Punkt eines

Simplex Meshes auf ein Dreieck eines Dreiecksnetzes abgebildet werden. Da ein Dreieck drei Kanten hat, an denen Nachbardreiecke anliegen können, hat ein Punkt eines Simplex Meshes immer drei Nachbarpunkte. Kanten entsprechen Verbindungen benachbarter Punkte. Jeder Kante eines Dreiecksnetzes entspricht somit eine Kante des Simplex Meshes. Diese *topologische Dualität* ist in Abbildung 2.13 verdeutlicht.

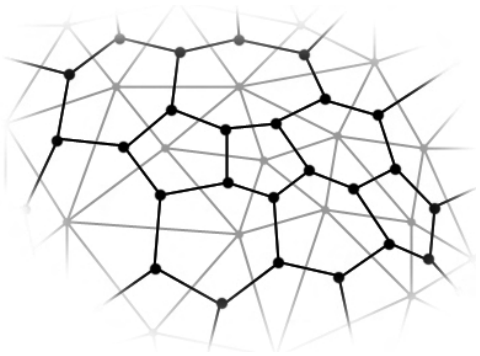


Abbildung 2.13: Topologische Dualität zwischen Dreiecksnetz (*hell*) und Simplex Mesh (*dunkel*): Jedem Dreieck des Dreiecksnetzes entspricht ein Punkt des Simplex Meshes. Die Kanten entsprechen sich gegenseitig.

Die Deformation des Simplex Meshes basiert auf einer internen Kraft F_{int}, die die Erhaltung der Form des Simplex Meshes begünstigt, sowie einer externen Kraft F_{ext}, die es an ein anderes Objekt angleichen soll.

Für die Nutzung im Bereich der Gefäßvisualisierung wurden Simplex Meshes von Bornik et al. vorgeschlagen [BRB05]. Als Ausgangsbasis dient wiederum eine symbolische Repräsentation des Gefäßbaumes. Ein erster Schritt besteht in der Generierung eines initialen Simplex Meshes. Dabei werden ausgehend von einem an der Gefäßbaumwurzel lokalisierten, zylinderförmigen Simplex Mesh aufeinander folgende Querschnitte miteinander verbunden. Die Verbindung der Querschnitte erfolgt dabei so, dass ein Simplex Mesh entsteht. Das heißt, es muss gewährleistet werden, dass alle Punkte drei Nachbarpunkte haben. Zu diesem Zweck müssen entlang der neu erzeugten Kanten zusätzlich Punkte eingefügt werden (Abbildung 2.14). Abgehende Äste werden auf ähnliche Weise mit der vorhandenen Geometrie verbunden.

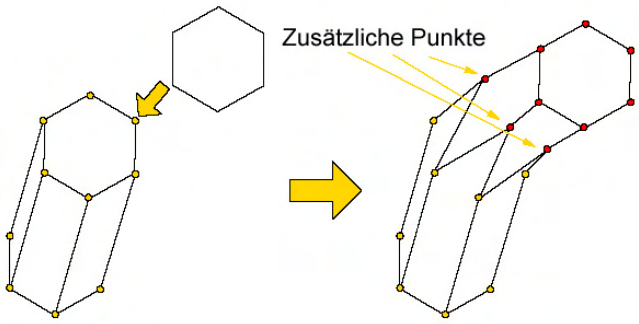


Abbildung 2.14: Verbindung benachbarter Querschnitte bei der Erzeugung des initialen Meshes. Um die Topologie eines Simplex Meshes zu generieren, müssen neue Punkte eingefügt werden (durch Pfeile markiert). *Quelle:* [BRB05]

Anschließend wird die Qualität des Meshes durch Anwendung der zuvor beschriebenen Deformation verbessert. Dabei wird, wenn lediglich das Gefäßmodell als Input zur Verfügung steht, die externe Kraft F_{ext} so gewählt, dass sie auf Samplingpunkte auf den kreisförmigen Querschnitten gerichtet ist. Auf diese Art kann vor allem die visuelle Qualität an Verzweigungen erhöht werden. Ist zusätzlich ein Segmentierungsergebnis vorhanden, kann F_{ext} in Richtung des nächstgelegenen Randvoxels bestimmt werden. Die Autoren geben an, dass das Ergebnis so an das Segmentierungsergebnis angenähert werden kann.

Das Verfahren und speziell der Ansatz zur Annäherung an das Segmentierungsergebnis sind sehr vielversprechend. Das Resultat ist eine geschlossene glatte Oberfläche (Abbildung 2.15), Strukturen im Inneren werden vermieden. Leider wird nicht näher auf die Berechnung von F_{int} und F_{ext} eingegangen. Es werden nur einige wenige Ergebnisse gezeigt. Eine Auswertung der Genauigkeit findet jedoch nicht statt. Die Komplexität der erzeugten Polygonnetze wird als vergleichbar mit dem Resultat des Marching Cubes-Algorithmus eingestuft. Die Berechnungszeit für die gezeigten Beispiele bewegt sich im Bereich weniger Sekunden.

Abbildung 2.15: Visualisierung einer menschlichen Pfortader durch ein Simplex Mesh. *Quelle:* [BRB05]

2.4.3 Sub Division Surfaces

Das Konzept der *Subdivision Surfaces (SDS)* wurde erstmals 1978 von Ed Catmull und Jim Clark vorgestellt [CC78], es dauerte jedoch 20 Jahre, bis diese Geometrierepräsentation breiten Einzug in die Computergrafik hielt. PIXAR[2] nutzten Subdivision Surfaces 1998 erstmals zur Animation eines Characters[3] und machten Subdivision Surfaces durch Erweiterungen wie die Wichtung von Kanten für ein breiteres Anwendungsfeld interessant [DKT98]. Subdivision Surfaces erlauben die Generierung eines glatten Models auf Basis eines groben, polygonalen Initialmodells. Da dies über mehrere Iterationen geschieht, können beliebig fein unterteilte Modelle generiert werden.

Diese Eigenschaften machen sich Felkel et al. zu Nutze, um auf Basis eines Gefäßmodells eine glatte Oberfläche zu rekonstruieren [FFKW02]. Bei der Konstruktion des Basismodells wird dabei ähnlich wie bei der Visualisierung mit Simplex Meshes vorgegangen. Der kreisförmige Querschnitt wird nun aber durch ein Quadrat angenähert, das den Kreis komplett einschließt. Aufeinander folgende Quadrate werden miteinander verbunden. Dabei

[2] http://www.pixar.com/, letzter Stand: 23.05.2006

[3] In dem Kurzfilm „Geri's game“ wurden SDS zur Modellierung und Animation des Characters „Geri“ sowie seiner Kleidung eingesetzt: http://www.pixar.com/shorts/gg/index.html, letzter Stand: 23.05.2006

wird besondere Sorgfalt auf die Ausrichtung der Quadrate verwendet, um eine Verdrehung des Polygonnetzes zu verhindern. Für die so erzeugten Seitenstücke wird ermittelt, ob ein Seitenast abgeht. Ist dies der Fall, so wird der Ast mit der entsprechenden Fläche verbunden. Die Konstruktion einer Trifurkation ist in Abbildung 2.16 verdeutlicht.

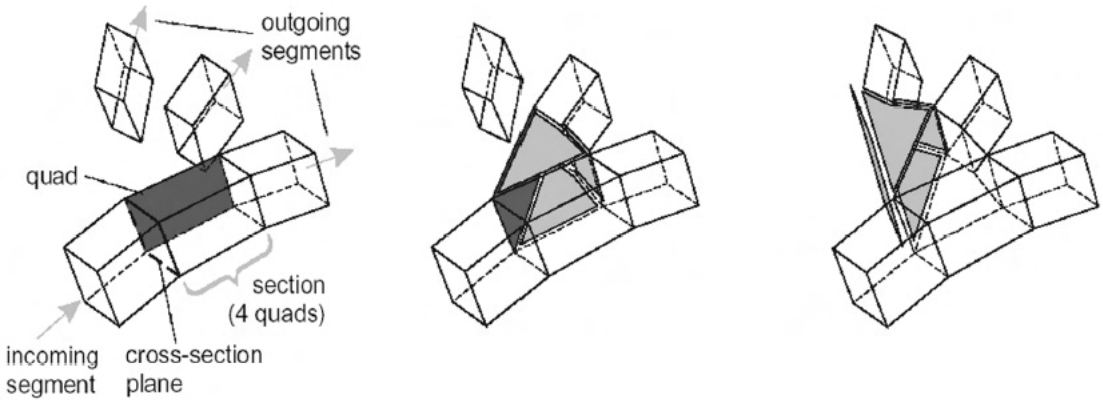


Abbildung 2.16: Erkennung und Rekonstruktion einer Trifurkation: Eine Seitenfläche (*dunkelgrau*) der betrachteten Sektion zeigt in Richtung eines abgehenden Astes (*links*). Die Fläche und der Ast werden verbunden (*mitte*). Ein weiterer Ast wird in der selben Richtung gefunden und mit dem neuen Füllstück verbunden (*rechts*). *Quelle:* [FFKW02]

Alle Seitenäste werden rekursiv verfolgt, um ein Basismodell des gesamten Gefäßbaumes zu rekonstruieren. Ist dies geschehen, wird durch Anwendung des Subdivision Surfaces-Algorithmus eine geglättete Repräsentation des Gefäßbaumes erzeugt (Abbildung 2.17). Die Darstellungsqualität der generierten Oberfläche hängt von der gewählten Iterationsstufe ab. Da ein Viereck je Iteration durch vier Vierecke ersetzt wird, vervierfacht sich die Komplexität des Polygonnetzes bei jeder Stufe. An den Verzweigungen hängt die Darstellungsqualität weiterhin vom Verhältnis zwischen dem Abstand benachbarter Querschnitte und dem lokalen Radius ab. Deswegen wird das Skelett so unterteilt, dass die Abstände dem lokalem Durchmesser entsprechen.

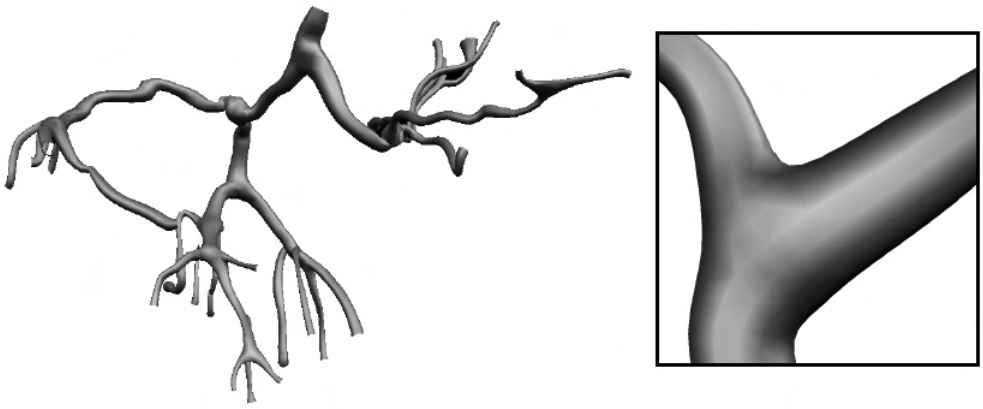


Abbildung 2.17: Ergebnis der Glättung mit Subdivision Surfaces: Visualisierung eines Lebergefäßbaumes (*links*) und vergrößerte Darstellung einer Bifurkation (*rechts*). *Quelle:* [FFKW02]

Innerhalb ihrer Arbeit haben Felkel et al. nicht untersucht, wie genau die erzeugte Oberfläche die Informationen des Gefäßmodells wiedergibt. Da Catmull-Clark-Subdivision Surfaces approximieren und nicht interpolieren, resultiert die Glättung in kreisförmigen Querschnitten, deren Durchmesser kleiner ist als die Kantenlänge der benutzten Quadrate. Zudem stimmen die Zentren der geglätteten Querschnitte nicht immer mit den Zentren der ursprünglichen Querschnitte überein. Der Gefäßverlauf wird somit ebenfalls geglättet und verfälscht. Um dies zu verdeutlichen, wurde ein einfaches Testmodell nach den Vorgaben aus [FFKW02] erstellt. Die Querschnitte sind quadratisch und ihr Abstand entspricht dem lokalen Radius. Das Modell wurde mit der von Felkel et al. verwendeten Bibliothek *Subdivide 2.0*[4] über drei Iterationen geglättet. Ein Vergleich des Ausgangsmodells mit dem geglättetem Model ist in Abbildung 2.18 zu sehen. Es ist deutlich zu erkennen, dass sowohl der Gefäßverlauf als auch die Durchmesser voneinander abweichen.

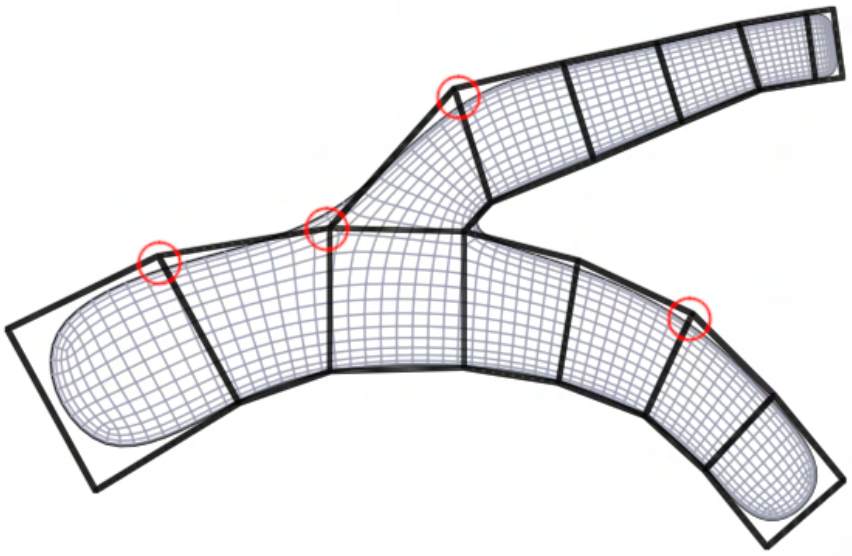


Abbildung 2.18: Glättung eines Testmodells: Der Durchmesser des Resultats (*grau*) ist kleiner als der des Ausgangsmodells (*schwarz*). Darüber hinaus weicht der Gefäßverlauf ab. Besonders starke Abweichungen wurden durch Kreise hervorgehoben.

Die von dem Verfahren benötigte Rechenzeit steigt mit der Anzahl der zu verarbeitenden Segmente, bewegt sich aber im Bereich von Millisekunden. Jedoch wurde nicht die Rechenzeit für die Verfeinerung mit Subdivision Surfaces betrachtet.

Die Methode ist sehr gut geeignet, um eine glatte, organisch wirkende Oberfläche eines Gefäßbaumes zu erzeugen. Verzweigungen werden natürlich dargestellt, im Inneren der Gefäße entstehen keine störenden Flächen. Die Gefäßenden werden in der vorgestellten Methode nicht geschlossen; eine Modifikation dahingehend sollte aber ohne größeren Aufwand möglich sein. Gerade wenn die Genauigkeit keine große Rolle spielt, sondern die Verdeutlichung der Topologie und eine skalierbare, hochqualitative Darstellung im Vordergrund steht, scheint das Verfahren sehr sinnvoll.

[4] http://www.mrl.nyu.edu/ biermann/subdivision/, letzter Stand: 23.05.2006

2.4.4 Convolution Surfaces

Bei *Convolution Surfaces* handelt es sich um eine Form *impliziter Oberflächen* (ein Überblick über implizite Oberflächen wird in Abschnitt 3.2 gegeben), die speziell der skelettbasierten Modellierung dient [BS91]. Convolution Surfaces repräsentieren die Oberfläche komplexer Objekte auf Basis ihrer Skelette. Das Skelett kann hierbei aus Linien, Kurven oder Polygonen bestehen. Durch die Faltung des Skeletts mit einem Filter wird eine implizite Funktion erzeugt, aus der durch Polygonalisierung eine Oberfläche extrahiert werden kann. Die resultierende Oberfläche hat dabei ein glattes, organisches Erscheinungsbild, benachbarte Objekte werden automatisch durch weiche Übergänge miteinander verbunden. Das Verfahren eignet sich somit besonders zur Modellierung organischer Formen.

Diese Eigenschaften machen sich Oeltze et al. zu Nutze, um Gefäßbäume qualitativ hochwertig darzustellen [OP05]. Sie beschreiben ein auf Convolution Surfaces basierendes Verfahren, das vom Gefäßskelett samt Radieninformationen ausgehend eine glatte, organische Oberfläche des beschriebenen Gefäßsystems erzeugt. Da das Skelett nur aus Liniensegmenten besteht, entstehen automatisch Zweige mit kreisförmigen Querschnitten.

In der Arbeit wird besonderer Wert auf eine korrekte Abbildung des Durchmesserverlaufs sowie eine natürliche Darstellung von Verzweigungen gelegt. Die visuelle Qualität der mit dem Verfahren erzeugten Gefäßoberflächen ist außerordentlich gut (Abbildung 2.19). Es gibt keine Diskontinuitäten in der Schattierung, Verzweigungen wirken natürlich und Gefäßenden werden automatisch halbkugelförmig geschlossen.

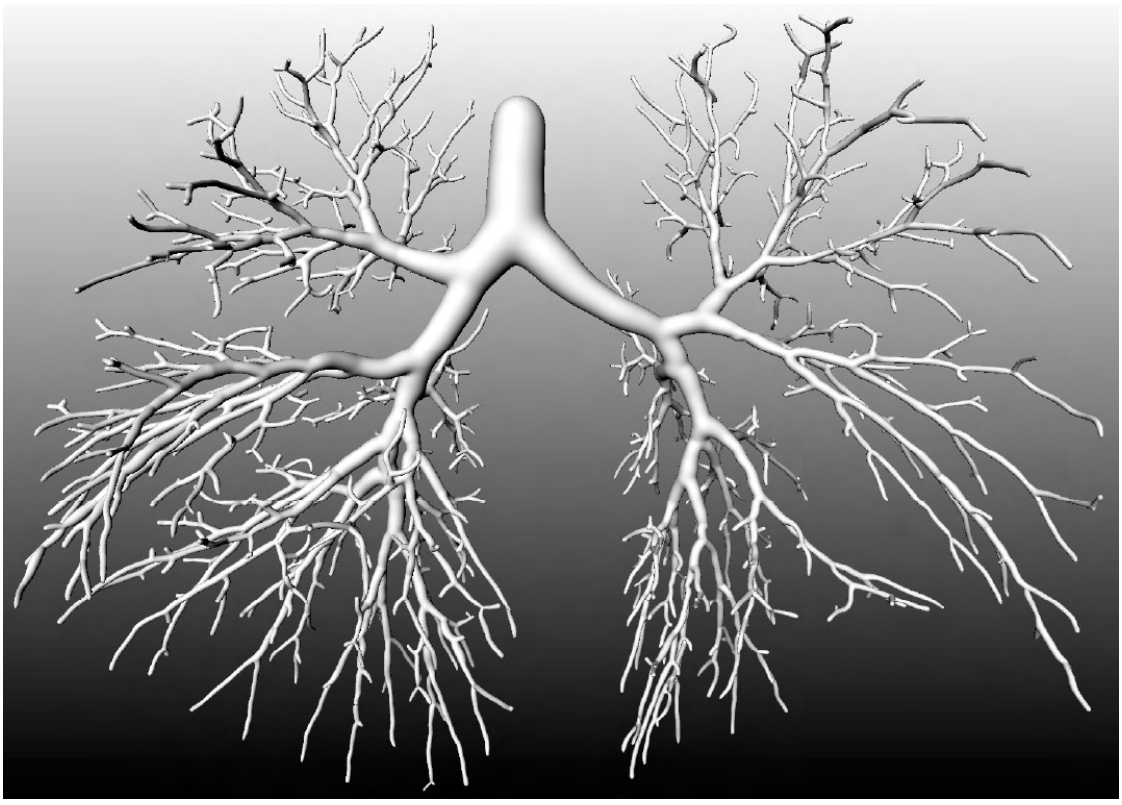


Abbildung 2.19: Visualisierung eines Gefäßbaumes mit Convolution Surfaces. Die Oberfläche weist eine kontinuierliche Schattierung auf, Verzweigungen wirken natürlich. *Quelle:* [OP05]

Die Autoren bewerten ihr Verfahren quantitativ. Das Verfahren gibt die Skelett- und Radieninformationen sehr genau wieder. Geringe Abweichungen sind Resultat der Glättung des Skeletts. Abweichungen zum Segmentierungsergebnis ergeben sich durch die Modellannahme und sind dort am stärksten, wo der Querschnitt am stärksten von der Kreisförmigkeit abweicht [Sch05]. Auch Abweichungen an Verzweigungen sowie den Gefäßenden können beobachtet werden. Ein Vergleich einer Convolution Surface mit dem entsprechendem Segmentierungsergebnis ist in Abbildung 2.20 gegeben.

Die Rechenzeit beträgt für typische Gefäßbäume, die einige Hundert Segmente enthalten, wenige Sekunden und kann für komplexe Gefäßbäume auf ca. 1 Minute ansteigen (siehe z.B. Abbildung 2.19). Die Auflösung des erzeugten Polygonnetzes kann durch eine parametrisierbare Polygonalisierung beeinflusst werden. Interaktive Frameraten sind somit möglich.

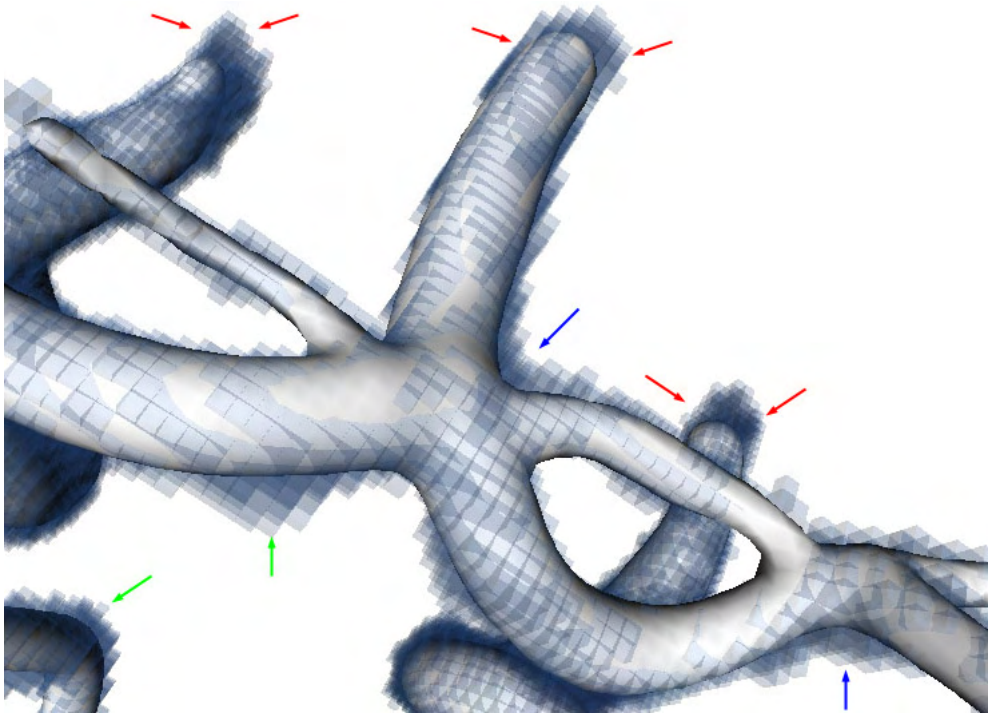


Abbildung 2.20: Überlagerung einer Convolution Surface (*grau*) mit dem entsprechenden Segmentierungsergebnis (*blau*). Abweichungen sind vor allem an Gefäßenden (*rote Pfeile*) und Verzweigungen (*blaue Pfeile*) zu beobachten, aber auch entlang des Gefäßverlaufs (*grüne Pfeile*). *Quelle:* [Sch05]

2.4.5 Freiformflächen

Eine modellbasierte Visualisierungsmethode, die keine kreisförmige Querschnitte voraussetzt, ist in [EDKS94] beschrieben. Die Extraktion des Gefäßmodells resultiert hier in einem durch Spline-Kurven beschriebenen Skelett. Querschnitte werden durch die Voxel beschrieben, die in der Ebene senkrecht zum Gefäßverlauf den Rand des segmentierten Volumens bilden. Somit sind diese Informationen nicht nur zur Beschreibung der Gefäßtopologie geeignet, sondern auch zur Wiedergabe des genauen Querschnitts. Eine korrekte Darstellung von Stenosen und Aneurysmen soll somit möglich sein.

Die Daten des Gefäßmodells sollen durch eine möglichst glatte Oberfläche repräsentiert werden. Dazu werden Freiformflächen an die Daten angepasst. Da die Oberfläche stückweise zusammengesetzt wird, muss die geometrische Kontinuität an Übergängen benachbarter Flächen überwacht werden. In der Arbeit wird nicht näher darauf eingegangen, wie dies zu erreichen sei.

Die Autoren geben an, dass mit der Methode eine exzellente Visualisierung von Gefäßen möglich sei. Der einzige in der Arbeit aufgeführte Nachweis dafür ist in Abbildung 2.21 zu sehen. Sie schränken jedoch die Anwendbarkeit des Verfahrens ein. Die Modellannahme kann trotz beliebiger Querschnitte zu vom Segmentierungsergebnis abweichenden Formen führen. Darüber hinaus sei das Verfahren für sehr kleine Äste kaum anwendbar. Die Rechenkomplexität der Rekonstruktion wird in der Arbeit nicht betrachtet.

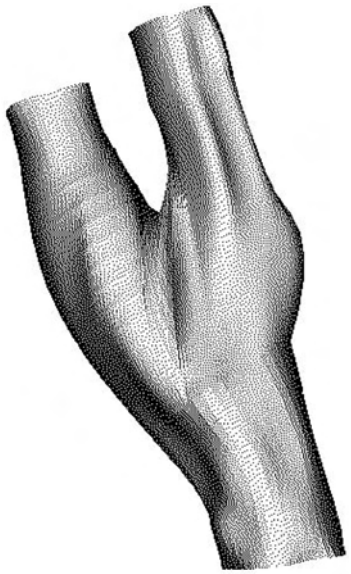


Abbildung 2.21: Beispiel für die Gefäßvisualisierung mit Freiformflächen.
Quelle: [EDKS94]

2.5 Zusammenfassung

Modellbasierte Verfahren sind momentan die State-of-the-Art-Verfahren im Bereich der Gefäßvisualisierung. Vor allem die Methode von Hahn [HPSP01] konnte sich hier etablieren. Die Vorteile der modellbasierten Verfahren liegen in den durch die symbolische Repräsentation geschaffenen analytischen und explorativen Möglichkeiten. Pfadlängen und Verzweigungswinkel können ermittelt werden, das Ausblenden und Einfärben von Teilstrukturen ist möglich. Grundlage hierfür ist ein Gefäßmodell, dass durch Segmentierung, Skelettierung und Graphenanalyse ermittelt wird. Es enthält Informationen über den Gefäßverlauf, Verzweigungen und den lokalen Gefäßquerschnitt.

Die Visualisierung des Modells wird meist durch Geometrierepräsentationen mit kreisförmigen Querschnitten realisiert. Einfache Primitive wie Zylinder und Kegelstümpfe führen hierbei jedoch zu einer wenig organisch wirkenden Darstellung. Vor allem Diskontinuitäten in der Schattierung und eine unnatürliche Darstellung von Verzweigungen, aber

auch die Bildung von Strukturen im Inneren der Gefäße sind hierbei störend. Visualisierungsmethoden auf Basis von Subdivision Surfaces und Convolution Surfaces widmen sich der Vermeidung dieser Probleme und ermöglichen eine qualitativ hochwertige Darstellung. Einige der hier vorgestellten Verfahren sind zum Beispiel in [PB06] genauer beschrieben. Das Buch gibt zudem einen umfassenden Überblick zum Thema Gefäßvisualisierung.

Aktuelle Arbeiten zeigen, dass die Gefäßvisualisierung ein aktives Forschungsgebiet darstellt und modellbasierte Verfahren hierbei nach wie vor von großem Interesse sind. Die Entwicklung einer Segmentierungsmethode speziell für krankhafte Gefäße wird in [LC06] beschrieben. Neue Interaktions- und Explorationstechniken basierend auf einem Gefäßmodell sind Bestandteil der Diplomarbeit von Verena von Hintzenstern [Hin06]. Neben der Selektion, Ausblendung und Einfärbung von Teilbäumen beschreibt die Arbeit Methoden zur Vermessung von Pfaden entlang des Skelettes. Entlang dieser Pfade kann zudem der Radienverlauf ausgewertet werden. *Fokus & Kontext*-Techniken sollen die Navigation in den teilweise sehr komplexen Bäumen erleichtern. In [Han06] werden Techniken zur informativen Darstellung von Gefäßbäumen beschrieben. Texturen werden hierbei genutzt, um auch bei einer monochromen Darstellung, wie sie zum Beispiel bei einer intraoperativen Projektion benötigt wird, einen räumlichen Eindruck zu vermitteln (Abbildung 2.22 rechts). Des Weiteren werden Texturen für eine Multiparameterdarstellung von Gefäßsystemen genutzt. Relevante Informationen wie Verzweigungsgrad eines Gefäßastes, Entfernung zu einem Tumor oder die Zusammengehörigkeit von Gefäßsystemen werden dabei in einer einzigen Darstellung kombiniert wiedergegeben (Abb. 2.22 links). Grundlage für diese Techniken ist wiederum ein Gefäßmodell.

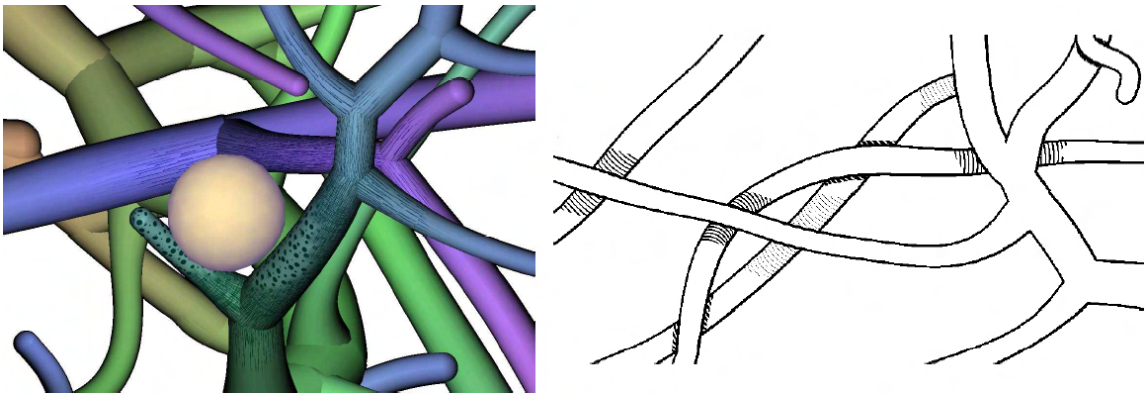


Abbildung 2.22: Verwendung von Textur in der Gefäßvisualisierung: Farben verdeutlichen den Abstand entlang des Gefäßpfades zur Gefäßwurzel. Unterschiedliche Texturen kodieren den Abstand zu einem Tumor (*links*). In monochromen Darstellungen werden Texturen zur Wiedergabe von Schatten genutzt (*rechts*). Die Größe und Deckkraft der Schatten kodiert den Abstand der Gefäße zueinander. *Quelle:* [Han06]

Es wird deutlich, dass die Nutzung eines Gefäßmodells viele Vorteile mit sich bringt. Deswegen wurden in diesem Kapitel entsprechende Verfahren näher betrachtet. Allerdings haben die meisten modellbasierten Verfahren einen großen Nachteil: Sie sind relativ ungenau. Somit sind sie zum Beispiel nicht für die Gefäßdiagnose geeignet.

Eine Ursache für die Ungenauigkeit liegt in der Annahme, dass der Gefäßquerschnitt kreisrund ist. Anatomische Gefäße haben oftmals eine Morphologie, die durch einen von der Kreisförmigkeit abweichenden Querschnitt gekennzeichnet ist. Ursache hierfür können zum einen Verformungen sein, die durch die Druckverhältnisse in den Gefäßen hervorgerufen werden. Zum anderen können pathologische Veränderungen zu einer vollkommen andersartigen Morphologie führen. Bronchialbäume weisen ebenfalls Querschnitte auf, die von der Kreisförmigkeit abweichen (Abschnitt 2.1).

Eine weitere Quelle für Ungenauigkeiten liegt in der Erzeugung und Nachbearbeitung des Skeletts. Hierbei kann der Gefäßverlauf durch Glättung leicht verändert werden. Darüber hinaus entscheidet der Anwender, inwiefern kleine Details wie winzige Äste erhalten bleiben (siehe Abschnitt 2.3.2).

Ein Verfahren, das die Gefäßoberfläche ausschließlich auf Basis des Gefäßverlaufs und der Radieninformationen rekonstruiert, kann das Gefäß somit nur näherungsweise beschreiben. Nur zwei der vorgestellten modellbasierten Verfahren (Simplex Meshes, Freiformflächen) widmen sich der möglichst genauen Wiedergabe der tomographischen Daten. Die Verfahren wurden in den entsprechenden Veröffentlichungen jedoch nur unzureichend beschrieben und nicht in Hinsicht auf die Genauigkeit untersucht.

Um für die Gefäßdiagnose interessant zu sein, muss ein Gefäßvisualisierungsverfahren das repräsentierte Gefäßsystem möglichst genau wiedergeben. Für viele analytische und explorative Aufgaben ist zudem ein symbolisches Modell des Gefäßsystems notwendig. Ein Verfahren, das die Oberfläche möglichst genau auf Basis des Segmentierungsergebnisses rekonstruiert, und diese Oberfläche in Verbindung zu einem Gefäßmodell setzt, würde sich somit für verschiedene Anwendungsgebiete eignen.

3 Direkte Oberflächenrekonstruktion aus Volumendaten

Im letzten Kapitel wurden modellbasierte Ansätze zur Visualisierung von Gefäßsystemen diskutiert. Der größte Nachteil dieser Verfahren ist, dass sie für einige Anwendungen, wie zum Beispiel die Gefäßdiagnose, zu ungenau sind. In Abschnitt 2.1 wurde erläutert, warum eine möglichst genaue Darstellung eines Gefäßsystems wünschenswert sein kann. *Surface Rendering*-Verfahren sind genauer, da sie die Oberfläche ohne jegliche Modellannahme rekonstruieren. Sie erzeugen die Oberfläche direkt auf Basis der Volumendaten oder nutzen das Segmentierungsergebnis. Da die erzeugte Oberfläche Voxel mit dem selben Wert repräsentiert, wird sie auch als *Isooberfläche* bezeichnet. Der repräsentierte Wert heißt *Isowert*.

In diesem Kapitel werden 2 Gruppen von Verfahren zur Extraktion einer Isooberfläche aus Volumendaten betrachtet: Die erste Gruppe nutzt *explizite* Oberflächenbeschreibungen, die zweite *implizite* Repräsentationen. Die Verfahren werden danach beurteilt, wie genau sie das Segmentierungsergebnis wiedergeben und ob sie in der Lage sind, eine glatte organische Oberfläche zu erzeugen. Eine weitere Anforderung stellt die Erhaltung dünner Strukturen dar. Hierunter sind Zweige mit einem Querschnittsdurchmesser von einem Voxel zu verstehen. Des weiteren wird die Komplexität des erzeugten Modells betrachtet.

Übersicht: In Abschnitt 3.1 werden Methoden zur Generierung expliziter Oberflächenmodelle auf Basis von Volumendaten betrachtet. Im darauf folgenden Abschnitt wird eine Einführung zu impliziten Oberflächen gegeben. Es wird gezeigt, wie diese Repräsentation genutzt werden kann, um aus Volumendaten Isooberflächen zu extrahieren. Hierbei wird die Nutzung von MPU Implicits motiviert. Dieser Algorithmus wird in Abschnitt 3.3 genauer vorgestellt. In Abschnitt 3.4 werden praktische Aspekte bei der Nutzung des Verfahrens betrachtet. Anschließend werden die Erkenntnisse dieses Kapitels zusammengefasst.

3.1 Explizite Beschreibung von Oberflächenstrukturen

Explizite Oberflächenbeschreibungen enthalten alle Informationen, die zur direkten Darstellung der Oberfläche nötig sind. Die gebräuchlichste explizite Repräsentation ist die Abbildung der Oberfläche auf eine Menge Polygone. Ein Objekt wird somit durch Dreiecke, Vierecke oder komplexere N-Ecke dargestellt. Auch die Oberflächenbeschreibung mit parametrischen Flächen ist explizit, da die Funktionen, die die parametrische Oberfläche beschreiben, Punkte auf der Oberfläche direkt zurückgeben.

Im Folgenden werden zwei Verfahren zur Erzeugung einer expliziten polygonalen Repräsentation auf Basis von Volumendaten beschrieben: *Marching Cubes* und *Constrained Elastic Surface Nets*.

3.1.1 Marching Cubes

Der *Marching Cubes*-Algorithmus [LC87] ist das bekannteste Verfahren zur Extraktion einer Isooberfläche. Ihm liegt die Betrachtung lokaler *Zellen* zu Grunde. Eine Zelle ist hierbei

ein Würfel, dessen Ecken den Mittelpunkten von acht benachbarten Voxeln (jeweils vier Voxel aus zwei benachbarten Schichten) entsprechen. Für jede Zelle muss nun ermittelt werden, ob sie von der Isooberfläche geschnitten wird. Dazu wird jede Kante der Zelle betrachtet. Durch Vergleich der Datenwerte der beiden Voxel, die zu einer Kante gehören, mit dem Isowert kann ermittelt werden, ob die Kante geschnitten wird. Durch lineare Interpolation entlang der Kante kann nun der genaue Schnittpunkt ermittelt werden. Auf Basis der ermittelten Schnittpunkte wird die Teiloberfläche für die Zelle berechnet. Die Oberflächennormalen für die Beleuchtungsberechnung der Oberfläche werden auf Basis einer linearen Interpolation der Gradienten an den Eckpunkten der Zelle ermittelt.

Das Verfahren repräsentiert die Isooberfläche äußerst genau. Die Anzahl der erzeugten Polygone ist allerdings sehr groß und kann nicht beeinflusst werden. Darüber hinaus ist die Auflösung des Polygonnetzes nicht von der lokalen Krümmung abhängig. Weitere Nachteile, speziell im Bezug auf die Gefäßvisualisierung, werden in [PTN97] hervorgehoben. So kann der Algorithmus kleine Zweige teilweise nicht darstellen. Darüber hinaus erzeugt er eine große Anzahl Polygone. Ein weiterer großer Nachteil des Verfahrens ist das treppenartige Erscheinungsbild des Ergebnisses. Vor allem bei sehr dünnen Gefäßen kann dies unverhältnismäßig stark ausfallen (Abbildung 3.1 links).

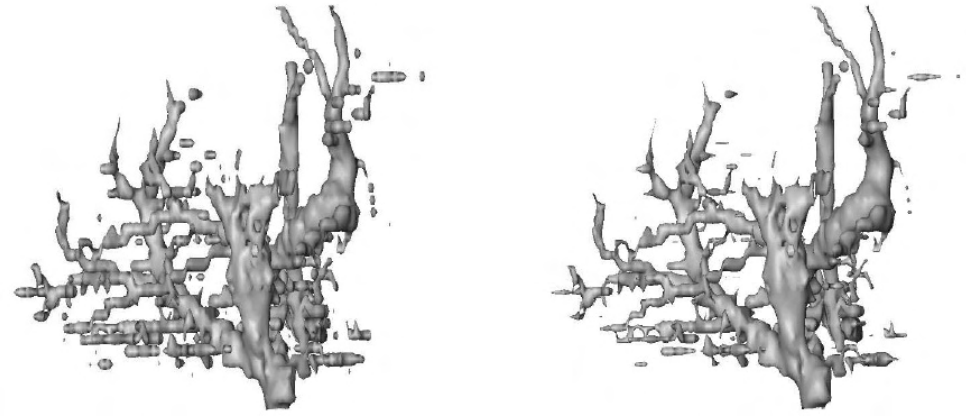


Abbildung 3.1: Das Polygonalisierungsergebnis (*links*) hat ein sehr treppenförmiges Erscheinungsbild. Bereits eine schwache Glättung mit einem *LowPass*-Filter führt zu einem Volumenverlust (*rechts*). *Quelle:* [BHP06]

Eine Glättung ist deswegen ein üblicher Nachbearbeitungsschritt des Marching Cubes-Verfahrens. Bade et al. untersuchen und vergleichen entsprechende Verfahren in [BHP06]. Dabei wird auch die Anwendbarkeit auf polygonale Repräsentationen eines Gefäßbaumes untersucht. Der mit der Glättung einhergehende Volumenverlust ist bereits bei einer schwachen Glättung sehr stark, die resultierende Oberfläche weicht deutlich vom Original ab (Abbildung 3.1 rechts). Vor allem Verzweigungen können dabei stark verfälscht werden. Das Verfahren ist somit kaum geeignet, Gefäßsysteme in hoher Qualität darzustellen.

3.1.2 Constrained Elastic Surface Nets

In [Gib98] wird ein Algorithmus zur Erzeugung einer glatten Oberfläche aus einem segmentierten Volumendatensatz vorgestellt. Als *Surface Net* wird die Vernetzung von Punkten auf der Oberfläche des segmentierten Objekts bezeichnet. Bei der Erzeugung der Punkte

werden genau wie beim Marching Cubes-Verfahren würfelförmige Zellen betrachtet, deren Eckpunkte die Mittelpunkte von acht benachbarten Voxeln repräsentieren. Haben alle acht Eckpunkte den selben Wert, befindet sich die Zelle entweder komplett innerhalb oder komplett außerhalb des segmentierten Volumens. Ist dies nicht der Fall, so schneidet die Zelle die Oberfläche und wird als *Surface Cube* betrachtet. In ihrem Zentrum wird ein Knoten erzeugt. Knoten benachbarter Surface Cubes werden miteinander verbunden, auf diese Weise kann ein Knoten bis zu sechs Nachbarn haben. Das auf diese Weise generierte Surface Net hat aufgrund der „treppenartige“ Anordnung der Knoten keine stetigen Normalen. Um eine glatte Oberfläche mit annähernd stetigen Normalen zu erhalten wird es deswegen elastisch verformt (Abbildung 3.2). Dazu wird für jede Verbindung zwischen benachbarten Knoten die quadrierte Länge der Verbindung als Energiemaß definiert. Für jeden Knoten gilt es, die Summe der Energie der ausgehenden Verbindungen zu minimieren. Um das zu erreichen, wird jeder Knoten in mehreren Iterationen zum Massezentrum seiner Nachbarn verschoben. Damit das Objekt auf diese Weise nicht schrumpft, wird die Verschiebung so begrenzt, dass der Knoten seinen ursprünglichen Surface Cube nicht verlässt.

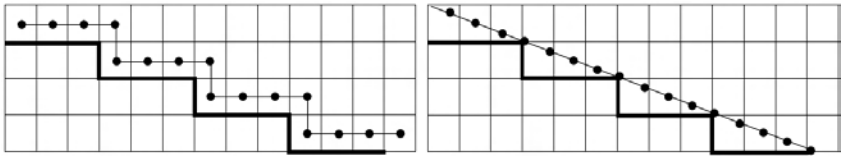


Abbildung 3.2: Das treppenförmige Erscheinungsbild des initialen Surface Nets (*links*) wird über mehrere Iterationen geglättet. Das Resultat ist eine glatte Oberfläche (*rechts*). *Quelle:* [Gib98]

Ist die Positionierung der Knoten abgeschlossen, wird auf Basis des Netzes eine polygonale Oberfläche erzeugt. Dazu wird je ein Dreieck zwischen dem betrachteten Knoten und je zwei Nachbarknoten gespannt.

Der Algorithmus produziert eine glatte Oberfläche, die sehr nah am Segmentierungsergebnis liegt. Scharfe Kanten werden erhalten (Abbildung 3.3 (a) und (b)). Die Autorin behauptet zudem, dass dünne Strukturen erhalten bleiben. Die in der Arbeit enthaltene Illustration kann dies jedoch nicht bestätigen (Abbildung 3.3 (c)).

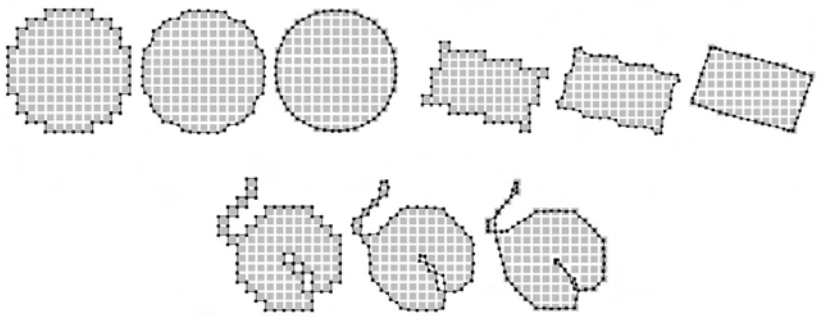


Abbildung 3.3: Glättung verdeutlicht an zweidimensionalen Beispielen: runde (*oben links*) und eckige (*oben rechts*) Objekte werden korrekt wiedergegeben. Sehr dünne Strukturen bleiben zu einem gewissen Grad erhalten (*unten*), verlieren jedoch ihr Volumen. *Quelle:* [Gib98]

Die Struktur bleibt zwar vorhanden, hat jedoch kein Volumen und wird praktisch als Linie dargestellt. Eine Anwendung auf einen entsprechenden dreidimensionalen Datensatz ist nicht enthalten.

Zur Bewertung kann das Verfahren jedoch relativ leicht nachvollzogen werden. Die iterative Verschiebung zum Massezentrum der Nachbarn entspricht der Glättung mit einem *Laplace*-Filter (für eine Beschreibung siehe [BHP06]). Erweitert man diesen Filter um eine Beschränkung der Verschiebung innerhalb eines Voxels und wendet ihn auf Ergebnisse des Marching Cubes-Verfahrens an, erhält man Ergebnisse, die mit denen der Constrained Elastic Surface Nets vergleichbar sein dürften. Angewendet auf eine filigrane Struktur können so die zuvor aus der Illustration gewonnenen Erkenntnisse bestätigt werden (Abb. 3.4).

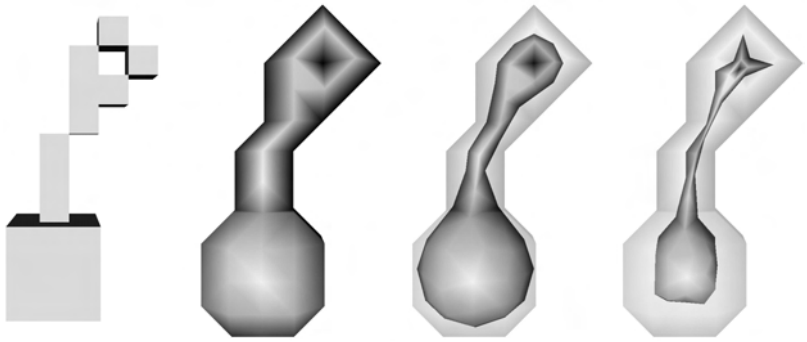


Abbildung 3.4: Glättung eines filigranen dreidimensionalen Testdatensatzes mit einem modifiziertem Laplace-Filter: Segmentierungsergebnis (a), Polygonalisierungsergebnis (b), Glättung mit einer Iteration (c) und mit fünf Iterationen (d).
Mit freundlicher Genehmigung von Ragnar Bade, Otto-von-Guericke-Universität Magdeburg.

Das Ergebnis ist auch nicht überraschend. Die Beschränkung der Punktverschiebung auf den Bereich eines Voxel führt zu einer relativ hohen Genauigkeit. In Bezug auf Strukturen, die selbst nur wenige Voxel dick sind, ist dies jedoch nicht ausreichend. Für eine Anwendung auf baumartige Strukturen müsste das Verfahren also angepasst werden.

Aussagen über die benötigte Rechenzeit sowie die Komplexität des erzeugten Modells wurden in der Veröffentlichung nicht getroffen. Der durchgeführte Test sowie die Anwendung auf andere Datensätze haben jedoch gezeigt, das die Rechenzeit im Bereich weniger Sekunden liegt.

3.2 Implizite Beschreibung von Oberflächenstrukturen

Implizite Oberflächen sind zweidimensionale, geometrische Formen im dreidimensionalen Raum. Ihre Form wird durch eine *implizite Funktion* beschrieben. Im Gegensatz zu einer expliziten Repräsentation liefert die Funktion nicht Punkte auf der Oberfläche. Sie klassifiziert Punkte hinsichtlich deren Lage zu der beschriebenen Oberfläche. Sie beschreibt das Objekt somit *indirekt*. Folgende Funktion beschreibt eine am Ursprung zentrierte Kugel mit dem Radius 1 implizit:

$$f(x, y, z) = x^2 + y^2 + z^2 - 1 \tag{3.1}$$

Für jeden Punkt *P(x,y,z)* gibt die Funktion an, ob der Punkt innerhalb des beschriebenen Objektes, auf der Oberfläche oder außerhalb des Objektes liegt:

$$f(P) = \begin{cases} <0 & : \text{ P liegt im Objekt} \\ =0 & : \text{ P liegt auf der Oberfläche des Objekts} \\ >0 & : \text{ P liegt außerhalb des Objekts} \end{cases} \tag{3.2}$$

Die Menge aller Punkte, deren Funktionswert 0 ist, beschreibt also die Oberfläche des Objektes. 0 ist somit der Isowert. Gerade aus dem Beispiel der Kugel wird deutlich, dass durch eine leichte Modifikation der Formel weitere Oberflächen beschrieben werden können. Die „1“ steht hier direkt für den Radius; würde sie durch eine „2“ ersetzt, würde der Radius dementsprechend steigen. Diese aus der Modifikation des Skalarwertes resultierenden Oberflächen werden als *Offset Surface* bezeichnet. Dies ist auf alle impliziten Oberflächen übertragbar. Nach [OM95] kann eine implizite Oberfläche folgendermaßen verallgemeinert werden:

$$f(x, y, z) = F(x, y, z) - Iso = 0 \tag{3.3}$$

F(x,y,z) wird hierbei als *Skalarfeldfunktion* oder *Feldfunktion* bezeichnet. Eine omplizite Funktion bietet also durch Modifikation des Isowerts die Möglichkeit, verschiedene Oberflächen (Offset Surfaces) zu definieren. Aus der zuvor beschriebenen Punktklassifikation (siehe Formel 3.2) ergibt sich zudem eine Beschreibung des gesamten Volumens des Objektes. Die entsprechende parametrische Funktion zur Beschreibung einer Kugel mit dem Radius 1

$$f(\alpha, \beta) = (\cos\alpha \cos\beta, \sin\alpha, \cos\alpha \sin\beta), \alpha \in [0, \pi], \beta \in [0, 2\pi] \tag{3.4}$$

ist weniger kompakt und definiert lediglich die Oberfläche, nicht aber das eingeschlossene Volumen. Auch eine polygonale Oberfläche beschreibt nur die Oberfläche und benötigt zudem weitaus mehr Speicherplatz. Dennoch sind implizite Oberflächen weitaus weniger verbreitet als explizite. Dies liegt vor allem daran, dass die Darstellung impliziter Oberflächen weitaus komplizierter ist als die ihrer expliziten Entsprechungen [Blo97].

Implizite Oberflächen zeichnen sich jedoch durch andere Vorteile aus. Neben ihrer kompakten Beschreibung und der Eigenschaft, das gesamte Volumen eines Objektes zu beschreiben, liegt ihre Stärke vor allem in der Modellierung weicher, organischer Objekte. Die Überblendung benachbarter Objekte, das sogenannte *Blending* ist integrierter Bestandteil der impliziten Beschreibung. Beispiele hierfür werden in Abschnitt 3.2.1 gegeben. Eine Konstruktion organischer Objekte mit parametrischen Oberflächen erfordert oftmals eine stückweise Approximation durch mehrere Teiloberflächen. Eine geometrische Kontinuität an den Übergängen zu gewährleisten ist hierbei oft problematisch. Ein weiterer Vorteil impliziter Oberflächen, der sich aus der Punktklassifizierung ergibt, liegt in einfachen Kollisionsberechnungen.

Die speziellen Fähigkeiten impliziter Oberflächen haben ihnen in den letzten Jahren vermehrtes Interesse zukommen lassen. Verschiedene Modellierungsmöglichkeiten wurden entwickelt. Ein Überblick darüber wird im folgendem Abschnitt gegeben. Wie bereits erwähnt wurde, ist die Tesselierung impliziter Oberflächen problematisch. Abschnitt 3.2.2 widmet sich entsprechenden Verfahren. In Abschnitt 3.2.3 wird schließlich gezeigt, wie implizite Techniken zur Extraktion einer Isooberfläche genutzt werden können.

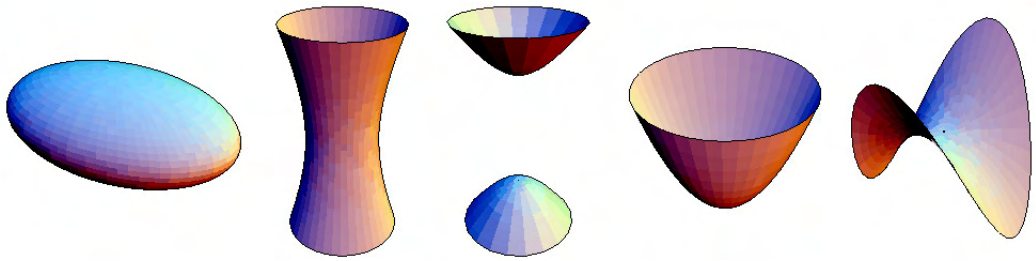
3.2.1 Modellierung mit impliziten Oberflächen

Eine implizite Oberfläche kann auf vielfältige Weise definiert werden. In diesem Abschnitt soll ein Überblick über verschiedene Modellierungstechniken gegeben werden.

Algebraische Oberflächen zählen zu den ersten Konzepten der impliziten Modellierung. Bereits in den 60er Jahren nutzte die Firma MAGI[1] *Quadriken* im Rahmen eines Visualisierungsprogramms [VGF02]. Die implizite Funktion einer Algebraischen Oberfläche ist durch ein Polynom gegeben. Polynome ersten Grades beschreiben Ebenen. Die zuvor erwähnten Quadriken werden durch Polynome zweiten Grades beschrieben:

$$\begin{aligned} f(x,y,z) = ax^2 + bxy + cxz + dx \\ +ey^2 + fyz + gy \\ +hz^2 + iz \\ +j \end{aligned} \tag{3.5}$$

Sind a, e und h jeweils 1, j gleich -1 und alle anderen Koeffizienten 0, so erhält man Formel 3.1. Zur Familie der Quadriken gehört somit auch die Kugel. Weitere beschreibbare Formen sind Ellipsoide, Zylinder, Kegel, parabolische und hyperbolische Oberflächen sowie hyperbolische Paraboloide (Abbildung 3.5).



(a) (b) (c) (d) (e)

Abbildung 3.5: Durch *Quadriken* beschreibbare Objekte: Ellipsoid (a), Hyperboloid (b), zweilagiger Hyperboloid (c), elliptischer Paraboloid (d) und hperbolischer Paraboloid (e). *Quelle:* [Lev96]

Kubische Oberflächen (Polynome dritten Grades) sowie Polynome höheren Grades sind möglich, jedoch weitaus weniger gebräuchlich.

Die **punktbasierte Modellierung** ist wohl die bekannteste Form der Modellierung mit impliziten Oberflächen. In Form von *Metaballs* oder *Blobs* (von *Blobby Objects*) ist dieses Konzept Bestandteil der meisten 3D-Animationspakete. Die Beschreibung von Oberflächen erfolgt hierbei auf Basis von Punkten und mit diesen Punkten assoziierten Skalarfeldern. Diese Form der Modellierung wurde 1982 von Jim Blinn eingeführt [Bli82]. Blinn's *Blobby Objects* dienen der Darstellung von Molekülen. Das Skalarfeld entspricht dem Elektronendichtefeld und wird für ein einzelnes Atom folgendermaßen berechnet:

$$F(p) = be^{-ad^2} \tag{3.6}$$

[1]MAGI (Mathematics Application Group, Inc) wurde 1966 gegründet und zählt zu den frühen Wegbereitern der Computergrafik. Ein Überblick über die Geschichte von MAGI ist unter http://accad.osu.edu/~waynec/history/tree/magi.html zu finden. Letzter Stand: 23.05.2006

d ist der Abstand von p zum Zentrum des Atoms. Diese Exponentielle Funktion hat eine Gauß-Verteilung um d mit der Höhe b und der Standardabweichung a. Durch diese Funktion hat jedes Atom nur einen begrenzten Einflussbereich. Ihre Form und somit auch der Einflussbereich des Atoms kann durch Veränderung der Parameter a und b beeinflusst werden. Die Wechselwirkung der Atome untereinander wird durch Blending der Primitive dargestellt. Das Blending wird hierbei durch eine Summierung der Skalarfelder erzielt:

$$F(p) = \sum_i b_i e^{-a_i d_i^2} \tag{3.7}$$

Eine Modifizierung von a und b beeinflusst hierbei die Form der Überblendung. Desweiteren wird der Grad der Verschmelzung durch die Entfernung der Atome zueinander bestimmt. Ein Beispiel für die Verschmelzung zweier Atome ist in Abbildung 3.6 gegeben.

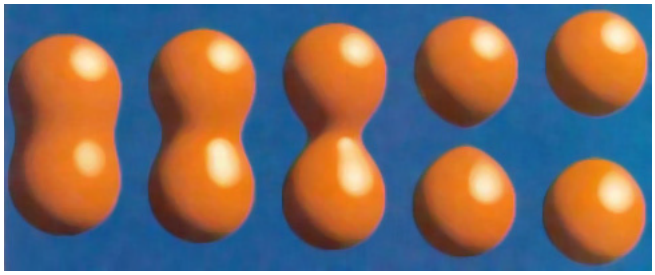


Abbildung 3.6: Verschmelzung zweier Atome als Beispiel für *Blobby Objects*. Der Grad der Verschmelzung ist von den Parametern des Skalarfeldes sowie der Entfernung der Atome voneinander abhängig. *Quelle:* [Bli82]

Metaballs [NHK+85] und *Soft Objects* [WMW86] nutzen den gleichen Ansatz wie *Blobby Objects*, verwenden jedoch andere Feldfunktionen.

Eine Erweiterung der punktbasierten Modellierung stellt die **Modellierung mit Skeletten** dar. Eine Erklärung des Begriffes *Skelett* im Rahmen von Volumendaten wurde in Abschnitt 2.3.2 gegeben. Im Rahmen der impliziten Modellierung kann der Begriff auf beliebige Primitive wie Punkte, Linien, Kurven und Flächen ausgedehnt werden. Wird ein solches Skelett wie bei der punktbasierten Modellierung von einem Skalarfeld umgeben, können sehr komplexe Formen beschrieben werden. Diese Form impliziter Oberflächen wird als *Convolution Surface* bezeichnet [BS91], da das Skalarfeld durch Faltung des Skelets mit einem Filter zu Stande kommt. Eine ausführliche Abhandlung zur Modellierung mit Convolution Surfaces ist in [Oel04] zu finden.

3.2.2 Polygonalisierung impliziter Oberflächen

Wie bereits erwähnt wurde, liegt ein Nachteil impliziter Oberflächen darin, dass sie schwieriger darzustellen sind als explizite Repräsentationen. Das liegt vor allem daran, dass diese Repräsentationen keine Punkte auf der Oberfläche zurückgeben, sondern lediglich einen gegebenen Punkt in Hinsicht auf dessen Lage zu der beschriebenen Oberfläche klassifizieren. Darstellungsverfahren für implizite Funktionen basieren deswegen auf der Auswahl

von Punkten im Raum und der Interpretation der Funktionswerte an diesen Punkten. Zu diesen Verfahren zählen zum Beispiel *Raytracing*, *Section Display*, *Partikelsysteme* und *Contour Tracing*. Nähere Erläuterungen dieser Verfahren sind zum Beispiel in [Blo97], [VGF02] und [Oel04] gegeben. Im Rahmen der vorliegenden Arbeit sind jedoch vor allem polygonale Repräsentationen interessant, da sie eine interaktive Darstellung selbst auf durchschnittlichen Rechner-Systemen erlauben. Darüber hinaus ist somit eine Selektion des Objektes oder die Markierung von Positionen auf dem Objekt möglich. Im Rahmen der bereits diskutierten Exploration und Analyse von Gefäßsystemen spielt dies eine wichtige Rolle.

Im Folgenden werden deshalb verschiedene Möglichkeiten zur Überführung impliziter Oberflächen in explizite Repräsentationen betrachtet. Da das Ergebnis in Form von Polygonen vorliegt, spricht man hierbei von *Polygonalisierung*. Entsprechende Verfahren lassen sich in *Spatial Sampling*, *Surface Tracking* und *Surface Fitting* kategorisieren [AG01].

Spatial Sampling

Grundlage dieser Verfahren ist die Unterteilung des Raumes in *Zellen*. Jene Zellen, die die gesuchte Isooberfläche schneiden, werden ermittelt. Dies geschieht durch einen Vergleich der Funktionswerte an den Eckpunkten der Zellen mit dem Isowert. Eine Zelle schneidet die Isooberfläche, wenn es unter den zu den Eckpunkten der Zelle gehörenden Funktionswerten sowohl größere als auch kleiner Werte als den Isowert gibt. Die Oberfläche wird nun innerhalb einer solchen Zelle polygonalisiert. Dazu werden die Schnittpunkte der Kanten der Zelle mit der Isooberfläche durch Interpolation ermittelt und zu einer Teiloberfläche verbunden. Die Menge all dieser Teiloberflächen ergibt die Gesamtoberfläche.

Die entsprechenden Verfahren unterscheiden sich durch die Form der Zellen. Würfel werden von dem bekannten *Marching Cubes*-Algorithmus benutzt [LC87](siehe auch Abschnitt 3.1.1). Bei der Triangulierung innerhalb einer Zelle kann es hierbei zu Mehrdeutigkeiten kommen, da es für bestimmte Schnittpunktkonfigurationen mehrere Möglichkeiten gibt, wie die Oberfläche den Würfel schneidet. Zudem können mehrere Teile der Oberfläche ein und die selbe Zelle schneiden. Eine Zerlegung der Zelle in 12 Tetraeder kann dieses Problem beheben [Blo88]. Dieser als *Tetrahedral Decomposition* bekannte Vorgang führt jedoch auch zu einer weitaus größeren Anzahl Polygone. Die Polygone können zudem sehr schmal und länglich sein, was zu Problemen bei der Schattierung der Oberfläche führen kann. Zur Reduzierung der Polygonanzahl und der Eliminierung degenerierter Polygone wurde eine Vielzahl an Nachbearbeitungsschritten entwickelt [AG01].

Die Genauigkeit der beschriebenen Verfahren hängt von der Größe der verwendeten Zellen ab. Diese Größe kann zumeist vom Benutzer eingestellt werden. Da diese Größe für alle Zellen gleich ist, erfordert die Erhaltung kleiner Details eine übermäßig feine Polygonalisierung detailarmer Bereiche. Aus Effizienzgründen sind deshalb Verfahren wünschenswert, die die Polygondichte in Abhängigkeit von der Größe lokaler Features steuern. Eine solche adaptive Unterteilung der Zellen wird zum Beispiel in [Blo88] vorgeschlagen.

Surface Tracking

Verfahren dieser Kategorie verfolgen die Oberfläche ausgehend von einem *Saatpunkt* auf der Oberfläche. Einige der Verfahren nutzen Zellen und ähneln somit stark den zuvor

beschriebenen Spatial Sampling-Techniken. Ausgehend von einer Startzelle, die die Oberfläche schneidet, werden Nachbarzellen betrachtet. Schneiden diese ebenfalls die Oberfläche, werden sie auf die selbe Weise behandelt. Somit werden rekursiv alle relevanten Zellen betrachtet. Dieses Ausbreitungsprinzip wurde in [WMW86] beschrieben. Diese Verfahren leiden unter den gleichen Nachteilen wie die Spatial Sampling-Verfahren, da die Oberflächenrekonstruktion innerhalb der Zellen genauso abläuft, und erfordern somit eine ähnliche Nachbearbeitung. Der Vorteil gegenüber den Spatial Sampling-Techniken ist der geringere Rechenaufwand aufgrund der weitaus geringeren Anzahl zu betrachtender Zellen.

Der bekannteste Vertreter dieser Kategorie ist *Bloomenthal's Implicit Polygonizer* [Blo94]. Genau wie beim Marching Cubes-Verfahren kann die Polygonalisierung durch die Angabe der Zellgröße und des Isowerts beeinflusst werden. Dieses Verfahren spielt im weiteren Verlauf der Arbeit eine wichtige Rolle und wird abkürzend *Bloomenthal-Polygonizer* genannt.

Ein alternativer Surface Tracking-Ansatz erzeugt Dreiecke ohne die Nutzung von Zellen direkt auf der Oberfläche. Hierbei wird von einem *Saat-Dreieck* auf der Oberfläche ausgegangen. An den drei Kanten dieses Dreiecks werden neue Dreiecke erzeugt. Ähnlich dem zuvor beschriebenen Ausbreitungsprinzip wird so durch eine rekursive Betrachtung die gesamte Oberfläche abgedeckt. Dieses als *Marching Triangles* [HSIW96] bekannte Verfahren erzeugt möglichst gleich große, gleichseitige Dreiecke. Während so degenerierte Dreiecke vermieden werden, kann es zur Bildung von Lücken in der Oberfläche kommen. In [AG01] wird eine erweiterte Marching Triangles-Methode vorgestellt, die sich dieses Problems annimmt. Darüber hinaus ist dieses Verfahren adaptiv und erzeugt Dreiecke, deren Größe von der lokalen Krümmung der Oberfläche abhängt.

Ein ähnliches Verfahren wird in [AJ04] beschrieben. An einem Startpunkt wird tangential ein Sechseck angelegt. Dessen Punkte werden anschließend auf die Oberfläche projiziert. Die Dreiecke, die diese sieben Punkte miteinander verbinden werden zum Ergebnis hinzugefügt (Abbildung 3.7 links). Die Menge der äußeren Kanten des Sechsecks wird als *Front* bezeichnet. Ziel des Algorithmus ist die Ausbreitung der Front, bis diese wieder auf sich oder eine andere Front trifft. Dazu wird der Punkt der Front ausgewählt, an dem der äußere Winkel zwischen den Kanten der Front („minimal front angle“ in Abbildung 3.7) am kleinsten ist. Die Anzahl der an diesem Punkt neu zu erzeugenden Dreiecke wird in Abhängigkeit von diesem Winkel ermittelt. Die Kantenlänge der neuen Dreiecke wird basierend auf einem Krümmungsmaß für den aktuell betrachteten Punkt bestimmt. Die Dreiecke werden zuerst in einer Tangentialebene angelegt und dann wiederum auf die Oberfläche projiziert. Die neuen Punkte werden der Front hinzugefügt, der Ausgangspunkt entfernt. Diese iterative Ausbreitung der Front, und damit auch des Dreiecksnetzes, ist in Abbildung 3.7 verdeutlicht.

Bei der Ausbreitung der Front muss immer auf Überlappungen geachtet werden. Fronten, die aufeinander treffen werden verschmolzen. Dieser Prozess ist in Abbildung 3.8 dargestellt.

Das Verfahren erzeugt eine adaptive Triangulierung auf Basis eines Krümmungsmaßes. Löcher und degenerierte Dreiecke werden vermieden, Nachbearbeitungsschritte sind somit nicht notwendig. Das Verfahren wurde für verschiedene implizite Repräsentationen, darunter auch MPU Implicits, getestet.

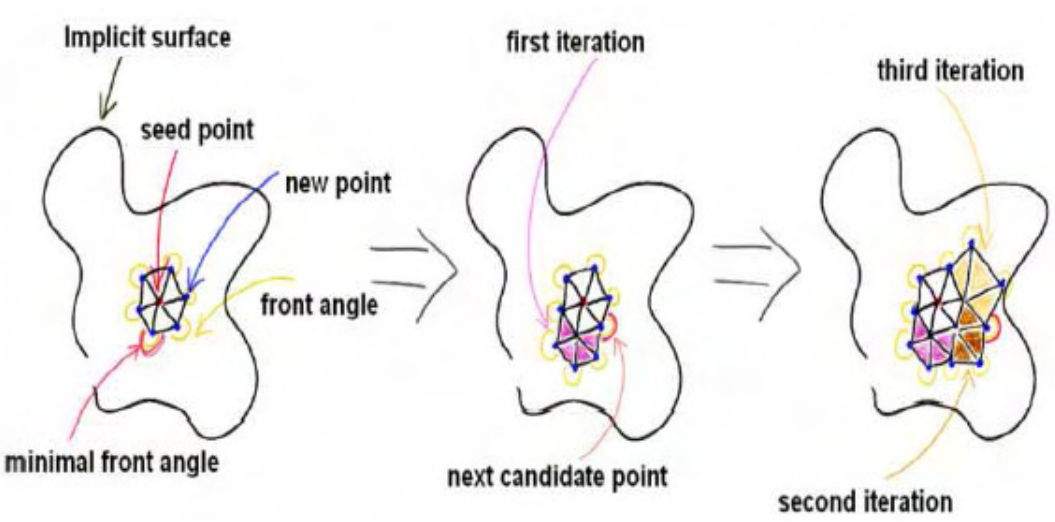


Abbildung 3.7: Ausbreitung der Front: Ausgehend vom Start-Sechseck (*links*) wird der Punkt gewählt, an dem der äußere Winkel zwischen den Kanten der Front am geringsten ist. Neue Dreiecke werden an diesem Punkt erzeugt (*mitte*). Das Ergebnis nach einer weiteren Iteration ist rechts zu sehen. *Quelle:* [AJ04]

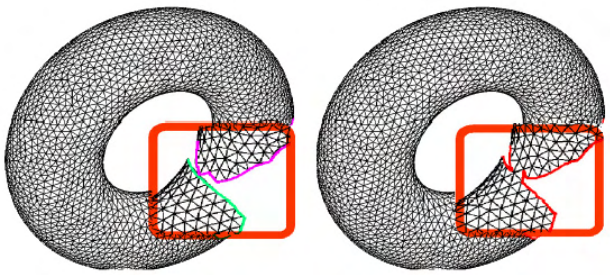


Abbildung 3.8: Verschmelzung zweier aufeinander treffender Fronten. *Quelle:* [AJ04]

Ein Problem, das alle Surface Tracking-Verfahren teilen, ist, dass nur eine zusammenhängende Oberfläche polygonalisiert wird. Eine implizite Funktion kann jedoch auch mehrere, nicht miteinander verbundene Objektteile beschreiben. In diesem Fall muss das Polygonalisierungsverfahren mehrmals für entsprechend gewählte Saatpunkte ausgeführt werden.

Surface Fitting

Surface Fitting-Methoden basieren, wie der Name impliziert, auf der Anpassung einer initialen Oberfläche an die implizit beschriebene Oberfläche. *Primitiv-orientierte* Verfahren betrachten dabei die einzelnen Primitive, aus denen die implizite Oberfläche modelliert wurde. Basierend auf dem Hüllkörper eines jeden Primitivs wird ein einfaches Ausgangsobjekt angelegt. Dessen Punkte werden schrittweise nach innen bewegt, bis sie auf der impliziten Oberfläche liegen [DTC96]. Der Vorteil solch lokaler Verfahren ist, dass sie bei Änderungen eine lokale Neuberechnung erlauben. Probleme ergeben sich jedoch an Schnittstellen mehrerer Primitive. Hier kann es bei der Vereinigung zu Löchern kommen.

Globale Ansätze wie das *Shrink Wrap*-Verfahren [OW93] gehen von einem initialen Ausgangsobjekt aus, das schrittweise an die implizite Oberfläche angenähert wird. Auf diese Weise werden jedoch Löcher im Inneren des Objekts nicht erkannt.

In [SH97] wird ein Verfahren beschrieben, das die *kritischen Punkte*[2] einer impliziten Funktion ermittelt und diese nutzt, um ein Basisnetz auszudehnen und an die implizite Oberfläche anzupassen. Löcher werden somit vermieden. Das Verfahren wird bei einer großen Anzahl von Primitiven allerdings ineffizient, da die Anzahl der kritischen Punkte exponentiell mit der Anzahl der Primitive steigt. Darüberhinaus können bei der skelettbasierten Modellierung Regionen entstehen, in denen die kritischen Punkte nicht bestimmt werden können [AG01].

3.2.3 Oberflächenrekonstruktion mit impliziten Oberflächen

In den letzten beiden Abschnitten wurde dargelegt, wie implizite Oberflächen modelliert und durch Polygonalisierung in ein explizites Modell überführt werden. Im Folgenden wird gezeigt, wie diese Techniken zur Rekonstruktion einer Oberfläche auf Basis von medizinischen Daten genutzt werden können.

Grundlage einiger Oberflächenrekonstruktionsverfahren ist, dass ein Objekt durch Punkte auf dessen Oberfläche beschrieben werden kann. Auf Basis solcher *Punktwolken* können implizite Funktionen ermittelt werden, die die Punkte approximieren oder interpolieren. Durch eine Polygonalisierung dieser impliziten Funktion erhält man ein polygonales Modell der beschriebenen Oberfläche. Die als Input dienenden Punktwolken sind hierbei meist das Ergebnis von 3D Laser Range Scannern.

Die Nutzung eines solchen Verfahrens im Rahmen der medizinischen Visualisierung wurde erstmals in [TBG95] vorgeschlagen. Auf Basis einer Punktwolke wird ein aus Punkten bestehendes Skelett konstruiert. Ausgehend davon wird eine Implizite Funktion ermittelt, die die Oberfläche repräsentiert. Die Arbeit zeigt, dass durch Punktwolken definierte implizite Oberflächen in der Lage sind, die Oberfläche von Organen zu rekonstruieren. Das erzeugte Modell hat dabei ein glattes, organisches Erscheinungsbild (Abbildung 3.9).

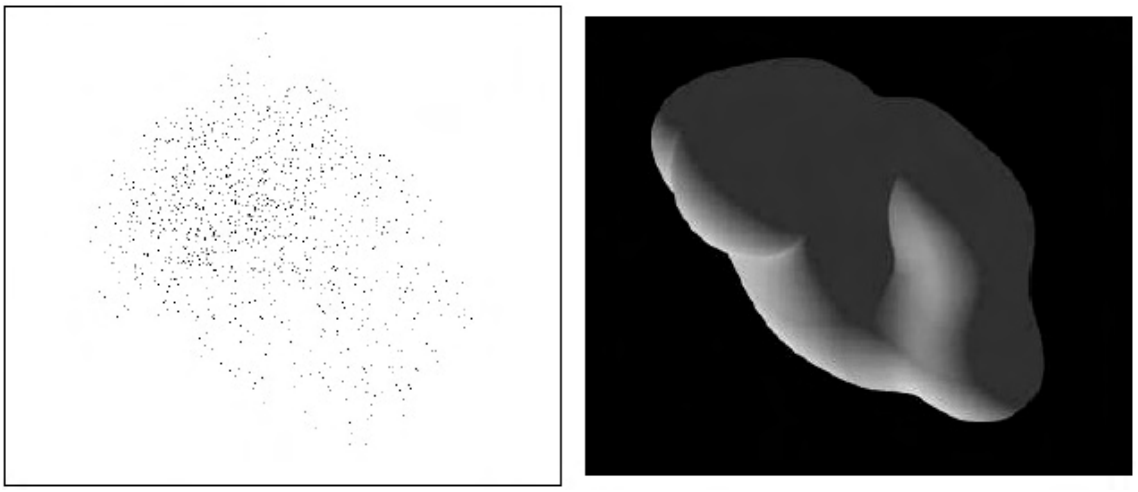


Abbildung 3.9: Rekonstruktion einer Niere mit Impliziten Oberflächen: Der Input liegt in Form einer Punktwolke vor (*links*). Die erzeugte Oberfläche hat ein glattes, organisches Erscheinungsbild (*rechts*). *Quelle:* [TBG95]

[2]Kritische Punkte einer impliziten Funktion sind die Punkte, an denen die Ableitung der Funktion 0 ergibt.

Eine Untersuchung der Genauigkeit findet in der Arbeit allerdings nicht statt. Der Rechenaufwand des Verfahrens lässt sich aus heutiger Sicht schwer beurteilen. Das Verfahren wurde vor elf Jahren auf damals aktueller Hardware getestet. Die Rekonstruktion einfacher Objekte benötigte zwischen 30 Minuten und 2 Stunden. Beachtet man den Anstieg der Rechenleistung in diesem Zeitraum, dürfte das Verfahren heute entsprechende Objekt in wenigen Sekunden oder Minuten rekonstruieren. Innerhalb der Arbeit wird der Ursprung der Punktwolke nicht genauer betrachtet. Es wird zwar erwähnt, dass sie von tomographischen Daten abgeleitet sein kann, aber der entsprechende Prozess ist nicht Teil der Arbeit.

Eine ähnliche Methode wird in [FCA96] vorgestellt. Das aus der Punktwolke extrahierte Skelett liegt hierbei in Form von Spline-Kurven und Bézier-Flächen vor. Die resultierenden impliziten Oberflächen haben ebenfalls ein organisches Erscheinungsbild und eignen sich auch zur Rekonstruktion von verzweigten Objekten (Abbildung 3.10). Weder Genauigkeit noch Berechnungsaufwand werden hier untersucht. Als Ursprung der Punktwolken werden 3D Laser Range Scanner angegeben.

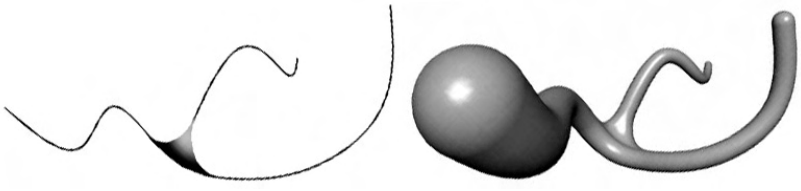


Abbildung 3.10: Rekonstruktion eines verzweigten Objekts mit Impliziten Oberflächen: Das aus der Punktwolke konstruierte Skelett liegt in Form von Spline-Kurven und Bézier-Flächen vor(*links*). Die daraus abgeleitete implizite Funktion erzeugt eine glatte, organische Oberfläche, auch für verzweigte Objekte (*rechts*). *Quelle:* [FCA96]

In [Mas02] wird die Approximation einer Punktwolke durch *Radiale Basis Funktionen (RBF)* genutzt, um die Oberfläche von Organen zu rekonstruieren. Zusätzlich zu den Punkten sind hier Normalenvektoren erforderlich. Die Erzeugung der Punktwolke mit assoziierten Normalen auf Basis eines tomographischen Datensatzes, *Boundary Point Detection* genannt, wird grob skizziert, ist aber nicht Teil der Veröffentlichung. Die Überführung einer Punktwolke in eine Oberflächenrepräsentation am Beispiel einer Leber ist in Abbildung 3.11 dargestellt.

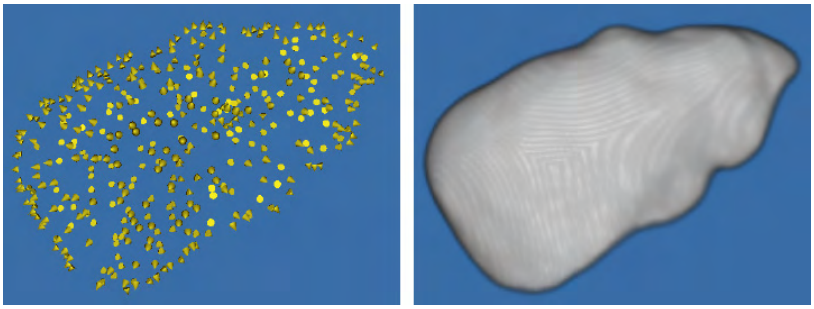


Abbildung 3.11: Rekonstruktion der Oberfläche einer Leber (*rechts*) auf Basis von *Boundary Points*, die aus einem tomographischen Datensatz gewonnen wurden (*links*). *Quelle:* [Mas02]

Die Boundary Points werden in der Abbildung durch Kegel repräsentiert, deren Spitze in Richtung des Normalenvektors zeigt. Das Ergebnis der Rekonstruktion ist eine sehr glatt wirkende Oberfläche (siehe Abbildung 3.11).

Die Genauigkeit des Verfahrens wird im Rahmen der Arbeit untersucht. Sie ist von der Anzahl der benutzten Boundary Points abhängig. Bei einer sehr großen Anzahl ist das Verfahren sehr genau. Eine Betrachtung des Rechenaufwands ist leider nicht Bestandteil der Arbeit, es wird lediglich erwähnt, dass die Rechenzeit mit der Anzahl der verwendeten Punkte steigt.

MPU Implicits, ein der RBF-basierten Rekonstruktion ähnliches Verfahren, wurden in [OBA+03b] vorgestellt (eine genaue Beschreibung des Verfahrens ist in Abschnitt 3.3 gegeben). Auch hier sind für die Rekonstruktion einer Impliziten Oberfläche Punkte und Normalen erforderlich. Die Nutzbarkeit des Verfahrens für die Extraktion einer Oberfläche aus tomographischen Daten wurde in [Bra05] untersucht. Als Input dient das segmentierte Volumen. Zwei verschiedene Methoden werden vorgeschlagen:

- Eine schichtweise Betrachtung der Daten. In jeder Schicht werden Punkte und Normalen auf der Kontur der segmentierten Region erzeugt. Normalen werden durch Glättung des Segmentierungsergebnisses und eine anschließende Gradientenberechnung ermittelt. MPU Implicits werden nun zur Erzeugung eines 2D-Distanzfeldes benutzt. Diese 2D-Distanzfelder werden daraufhin durch lineare Interpolation oder Spline-Interpolation zu einem 3D-Distanzfeld verbunden, das daraufhin mit dem Marching Cubes - Algorithmus polygonalisiert wird.

- Betrachtung des Volumens als Ganzes. Punkte und Normalen werden am Rand des segmentierten Volumens extrahiert und direkt als Input für die Berechnung mit MPU Implicits benutzt.

Die volumenbasierte Methode erzeugt dabei bessere Resultate und wird im auswertenden Teil der Arbeit genauer betrachtet. Die erzeugten Oberflächen sind auch bei topologisch komplexen Objekten qualitativ hochwertig (Abbildung 3.12). Sie zeichnen sich durch eine kontinuierliche Schattierung aus.

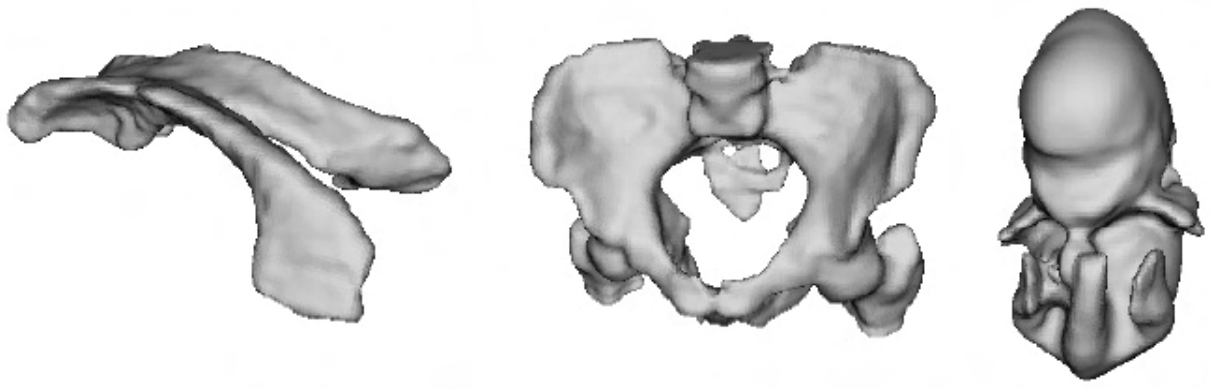


Abbildung 3.12: Ergebnisse der Rekonstruktion mit MPU Implicits: ein Ventrikel (*links*), ein Becken (*mitte*) und ein Embryo (*rechts*). Die erzeugten Oberflächen sind auch bei topologisch komplexen Objekten qualitativ hochwertig. *Quelle:* [Bra05]

Die Genauigkeit des Verfahrens wird als sehr hoch eingestuft. Diese Aussage wird durch einen Vergleich der erzeugten Oberflächen mit Ergebnissen der kommerziellen Software AMIRA[3] untermauert. Es wird hervorgehoben, dass das Verfahren dazu geeignet ist, Rauschen im Segmentierungsergebnis zu beheben. Grund hierfür ist die approximative Natur des Verfahrens. Die durch die MPU Implicits beschriebene Oberfläche verläuft nicht exakt durch die als Input dienenden Punkte, sondern nähert sie so gut wie möglich an. Wie groß die Abweichung ist, kann dabei durch Parameter kontrolliert werden. Dies ist eine große Stärke der Oberflächenrekonstruktion mit MPU Implicits. Die Arbeit enthält auch eine Auswertung der Berechnungskomplexität des vorgestellten Verfahrens. Die Berechnungsdauer wird für verschiedene Datensätze verglichen. Sie schwankt zwischen wenigen Sekunden und einigen Minuten. Dabei wird der weitaus größere Teil der Rechenzeit für die Extraktion der Punkte und Normalen benötigt. Die Berechnungszeit dieses Rechenschritts steigt dabei mit der Anzahl der Punkte. Die Berechnungszeiten für die Erzeugung der MPU Implicits schwanken weniger stark. Sie sind nicht von der Anzahl der Inputpunkte, sondern von der Komplexität des beschriebenen Objekts abhängig [OBA+03b].

Das Verfahren ist somit die vielversprechendste der vorgestellten impliziten Rekonstruktionsmethoden. In Tabelle 3.1 ist ein Vergleich des Verfahrens mit den vorgestellten modellbasierten Gefäßvisualisierungsverfahren sowie den expliziten Verfahren zur Isooberflächenextraktion gegeben.

Verfahren, die der Extraktion einer Isooberfläche aus Volumendatensätzen dienen, sind geeignet, die beschriebene Struktur möglichst genau wiederzugeben. Da dies auch an Verzweigungen gilt, wird für diese Verfahren nicht betrachtet, ob sie die Verzweigungen natürlich wiedergeben. Wenn ein Verfahren die Daten möglichst genau wiedergibt, und dennoch eine organische Oberfläche erzeugt, sollten auch Verzweigungen natürlich dargestellt werden.

Bei dem verbreiteten Marching Cubes-Verfahren geht mit der Genauigkeit jedoch eine stufenartige Darstellung einher. Eine organische Darstellung ist daher nicht gegeben. Somit kann das Verfahren eine wichtige Anforderung nicht erfüllen. Auch eine anschließende Glättung bereitet Probleme, da sie vor allem bei stark verzweigten Strukturen die Morphologie verändert. Constrained Elastic Surface Nets sind als Alternative vielversprechend. Sie erzeugen eine geglättete Repräsentation der Isooberfläche. Dünne Strukturen werden jedoch nur zu einem gewissen Grad erhalten, das Verfahren müsste also für eine Anwendung im Rahmen der Gefäßvisualisierung angepasst werden.

MPU Implicits erlauben die Erzeugung einer organisch wirkenden Isooberfläche mit einstellbarer Genauigkeit. Zudem ergibt sich aus ihrer impliziten Natur die Möglichkeit, durch eine parametrisierbare Polygonalisierung beliebig fein aufgelöste polygonale Oberflächen zu erhalten. Das Verfahren scheint somit geeignet, die Basis für ein Visualisierungsverfahren zu bilden, das die aufgestellten Anforderungen erfüllt. Es bleibt zu untersuchen, inwiefern das Verfahren zur Rekonstruktion verzweigter Objekte mit sehr filigranen Strukturen geeignet ist. Entsprechende Untersuchungen sind nicht in [Bra05] enthalten.

[3] Programm zur Datenanalyse, Visualisierung und Geometrie-Rekonstruktion der Firma Mercury Systems, http://www.tgs.com/index.htm?pro_div/amira_main.htm, letzter Stand: 23.05.2006

Tabelle 3.1: Vergleich der vorgestellten Verfahren in Hinsicht auf Genauigkeit und Darstellungsqualität. Im oberen Bereich befinden sich die modellbasierten Verfahren. In den letzten 3 Zeilen werden Verfahren zur direkten Oberflächenrekonstruktion betrachtet.
Die erreichte **Genauigkeit** wird danach beurteilt, ob sie zur Darstellung der Topologie (T), des Durchmesserverlaufs (D), der genauen Gefäßquerschnitte (Q) und dünner Strukturen (S) ausreichend ist. Eine **organischen Darstellung** ist gegeben, wenn eine glatte Oberfläche mit einer kontinuierlichen Schattierung erzeugt wird. Werden **Verzweigungen** durch organisch wirkende Übergänge natürlich dargestellt? Erzeugt das Verfahren unerwünschte **Strukturen im Inneren**? Werden **Gefäßenden** geschlossen? *variabel* kennzeichnet die Fähigkeit, ein Kriterium zu einem beliebigen Grad zu erfüllen. * kennzeichnet Aussagen, die in der Veröffentlichung aufgestellt, aber nicht bewiesen wurden. ** kennzeichnet eigene, spekulative Aussagen. Zu Feldern, die mit *?* markiert sind, kann keine Aussage getroffen werden.

Verfahren	Genauigkeit	organische Darstellung	Verzweigungen	Strukturen im Inneren	Gefäßenden
Zylinder [GKS+93]	T, S	nein	nein	ja	nein**
Kegelstümpfe [HPSP01]	T, D, S	nein	nein	ja	ja
Subdivision Surfaces [FFKW02]	T, S**	ja, variabel	ja	nein	nein
Convolution Surfaces [Oel04]	T, D, S	ja, variabel	ja	nein	ja
Freiformflächen [EDKS94]	T, D, Q*	ja*	ja*	nein**	?
Simplex Meshes [BRB05]	T, D, Q*, S**	ja	ja	nein	ja
Marching Cubes [LC87]	T, D, Q, S	nein	/	nein	ja
Constrained Elastic Surface Nets [Gib98]	T, D, Q	ja, variabel	/	ja**	ja**
MPU Implicits [Bra05]	T, D, Q, S**, variabel	ja, variabel	/	nein	ja

3.3 MPU Implicits im Detail

Dieser Abschnitt widmet sich der Beschreibung von *Multi-level Partition of Unity Implicits (MPU Implicits)*. Dieses von Ohtake et al. [OBA+03b] vorgestellte Verfahren dient der Rekonstruktion einer Oberfläche auf Basis einer *unstrukturierten Punktwolke*. Hierbei handelt es sich um eine Menge von Punkten ohne jegliche strukturelle Information. Das heißt, es sind keine direkten Anhaltspunkte enthalten, welche Punkte topologisch benachbart sind.

Punktwolken können zum Beispiel das Ergebnis von Laser-Scannern, mechanischen Abtastungen oder Computer Vision-Verfahren sein. Anwendungsgebiete sind beispielsweise die Analyse von Bauteilen oder archäologischen Fundstücken, die Rekonstruktion von Landschaftsmodellen oder die Digitalisierung von Miniaturen im Bereich der Unterhaltungsindustrie.

MPU Implicits zählen momentan zu den State-of-the-Art-Verfahren zur Punktwolken-basierten Rekonstruktion von Oberflächen. Das Verfahren gehört zu den schnellsten Rekonstruktionsmethoden [JR06]. Auch sehr große Punktwolken können effizient verarbeitet werden, da die Rechenkomplexität des Verfahrens nicht von der Anzahl der Punkte, sondern von der topologischen Komplexität des beschriebenen Objektes abhängt. Da das Verfahren approximiert und nicht interpoliert, kann es in den Ausgangsdaten vorhandenes Rauschen unterdrücken. Die Genauigkeit der Approximation kann dabei über Parameter gesteuert werden. Die Punkte der Ausgangsdaten können unregelmäßig verteilt sein; das Verfahren ist in der Lage, auch sehr große Lücken in den Daten zu schließen. Im Gegensatz zu vielen ähnlichen Verfahren werden auch scharfe Kanten und Ecken korrekt dargestellt.

Der Algorithmus zur Erzeugung von MPU Implicits erwartet als Eingabe die zuvor beschriebene Punktmenge. Zusätzlich werden jedoch die Normalenvektoren zu den Punkten benötigt. Dies stellt jedoch kein Problem dar, da die Normalenvektoren über *Least-Squares Fitting* berechnet werden können. Hierbei wird an jeden Punkt eine Ebene so angepasst, dass die Ebene die Punkte in der näheren Umgebung möglichst gut approximiert. Der Normalenvektor dieser Ebene dient dann als Normalenvektor für den betrachteten Punkt.

Die Nutzung von Normalenvektoren kann auch als Vorteil gesehen werden, da die Normalen die lokale Ausrichtung der Oberfläche beschreiben. Diese Informationen werden von dem Algorithmus ausgiebig genutzt (siehe Abschnitt 3.3.3). Wenn bei der Erzeugung der Punktwolke also auch gleichzeitig Normalen berechnet werden können, ist die Tatsache, dass der Algorithmus Normalen benötigt, kein Nachteil.
Der Input $\mathcal{I}$ ist also wie folgt gegeben:

$$\mathcal{I} = (\mathcal{P}, \mathcal{N}) \tag{3.8}$$

$$\begin{aligned} \mathcal{P} &= \{p_1, ..., p_N\} \\ \mathcal{N} &= \{n_1, ..., n_N\} \end{aligned}$$

N ist hierbei die Anzahl der im R^3 definierten Punkte und Normalen. Das Ergebnis des Verfahrens ist eine implizite Funktion $f(x)$, deren Nullstellenmenge die Oberfläche repräsentiert.

Der Algorithmus läßt sich durch folgende Punkte umreißen:

- Die Berechnung einer globalen Approximation wird auf Basis des *Partition of Unity*-Ansatzes realisiert. Hierbei wird die globale Approximation durch die Zusammenfassung lokaler Approximationen ermittelt.
- Grundlage der lokalen Betrachtung ist die Unterteilung des von der Punktwolke eingenommenen Raumes auf Basis eines Octrees.
- Für die lokale Beschreibung wird eine algebraische Fläche an die lokal vorhandenen Punkte angepasst.
- Die Genauigkeit der lokalen Approximation steuert, ob die betrachtete Octree-Zelle weiter unterteilt wird. Für die Bewertung wird ein vom Benutzter definiertes Fehlermaß herangezogen.

Durch die lokale Approximation auf Basis von algebraischen Flächen unterscheidet sich das Verfahren von den zahlreichen Methoden, die auf *Radial Basis Functions (RBFs)* basieren (siehe zum Beispiel [CBC+01]). Diese wichten direkt den Einfluss der Punkte der Punktwolke um zu einer impliziten Repräsentation zu gelangen und sind somit globaler Natur. Da so jeder Punkt bei jeder Evaluierung der Funktion betrachtet werden muss, sind diese Verfahren vergleichsweise ineffizient. Durch die Nutzung von zwei Gruppen von Funktionen, nämlich der Wichtungsfunktionen sowie der lokalen Approximationen, erreichen MPU Implicits hingegen eine sehr hohe Effizienz.

Im Folgenden wird die Oberflächenrekonstruktion auf Basis von MPU Implicits genauer betrachtet.

3.3.1 Blending auf Basis der Partition of Unity-Methode

Die *Partition of Unity*-Methode von Franke und Nielson [FN80] dient der Approximation großer Mengen verstreuter Daten. Die grundlegende Idee ist die Aufteilung des Definitionsbereiches Ω in n kleinere Teilbereiche, in denen eine Approximation leichter erfolgen kann. Nachdem die lokalen Approximationen ermittelt wurden, werden sie zu einer globalen Approximation integriert. Dazu wird jedem Teilbereich eine Partition of Unity-Funktion φ zugewiesen. Die Summe der Funktionswerte all dieser Funktionen muss für jedes x 1 ergeben:

$$\sum_{i=1}^{n} \varphi_i(x) = 1 \qquad \forall\, x \in \Omega \tag{3.9}$$

Die Approximation der Funktion $f(x)$ ergibt sich als Summe aller n lokaler Approximationen $Q_i(x)$ gewichtet mit der zugehörigen Partition of Unity-Funktion $\varphi_i(x)$:

$$f(x) \approx \sum_{i=1}^{n} \varphi_i(x) Q_i(x) \tag{3.10}$$

Übertragen auf implizite Oberflächen stellen die $Q_i(x)$ Primitive dar. Das *Blending* wird durch die Partition of Unity-Funktionen realisiert. Diese werden durch Normalisierung lokaler Wichtungsfunktionen $w(x)$ gewonnen:

$$\varphi_i(x) = \frac{w_i(x)}{\sum_{j=1}^{n} w_j(x)} \tag{3.11}$$

Die Summe aller φ ist somit an jedem Punkt x gleich 1. Als Wichtungsfunktion wird nun eine Funktion benötigt, deren Wert im Zentrum der Teilbereiche groß ist und zum Randbereich hin auf 0 abfällt, wobei mindestens C1-Stetigkeit gefordert wird. Ohtake et al. schlagen hierfür die Nutzung einer quadratischen B-Spline $b(t)$ vor (Abbildung 3.13 links). Der Parameter t wird auf Basis des Abstands von x zum Zentrum c des Teilbereichs und dem Radius R des Teilbereiches berechnet:

$$w_i(x) = b(t) = b\left(\frac{3 \mid x - c_i \mid}{2R_i}\right) \tag{3.12}$$

Da die Wichtungsfunktion somit vom Abstand eines Punktes zum Mittelpunkt eines Teilbereiches abhängt, ergibt sich ein kugelförmiger Einflussbereich. Die Wichtungsfunktionen werden deshalb im Folgenden auch *radiale Wichtungsfunktionen* genannt. Teilbereiche sind somit ebenfalls als kugelförmig aufzufassen, da ihr Einfluss auf die globale Approximation außerhalb der Kugel gleich 0 ist. Die Überblendung solcher Teilbereiche und der damit assoziierten lokalen Approximationen ist in Abbildung 3.13 (rechts) illustriert.

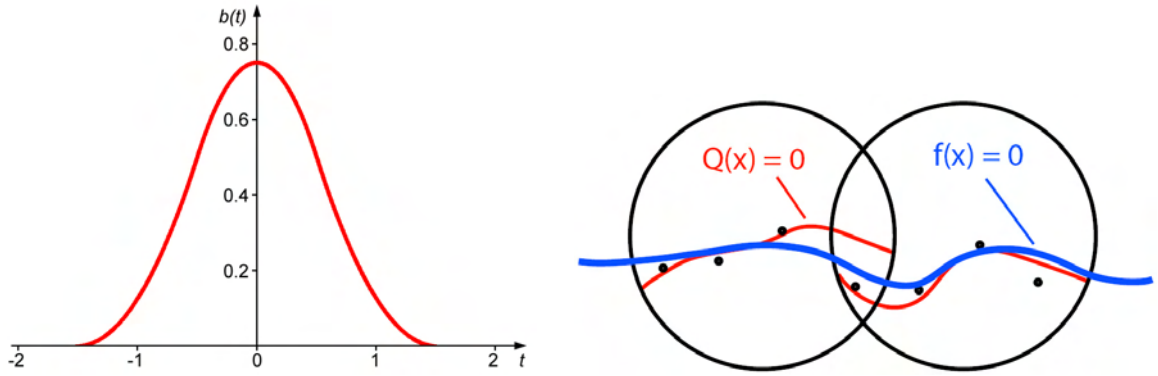


Abbildung 3.13: Überblendung lokaler Approximationen: Der Wichtungsfunktion liegt eine quadratische B-Spline-Funktion zu Grunde (*links*). rechts: Lokale Approximationen $Q_i(x)$ (*rot*) werden auf Basis der in den Teilbereichen (*schwarze Kreise*) vorhandenen Punkte (*schwarz*) ermittelt. Eine Überblendung auf Basis radialer Wichtungsfunktionen führt zu einer globalen Approximation (*blau*). *Quelle:* [Bra05]

3.3.2 Adaptive Approximation auf Basis eines Octrees

Grundlage des Verfahrens ist eine adaptive Unterteilung des Raumes durch einen Octree. Die Punktwolke wird zu Beginn so skaliert, dass ihre *Axis Aligned Bounding Box (AABB)* einem Würfel mit einer Diagonale der Länge 1 entspricht. Somit sind auch alle aus einer Unterteilung resultierenden Zellen würfelförmig. Dies ist notwendig, da die kugelförmigen Teilbereiche auf Basis der Zellen ermittelt werden sollen.

Die AABB ist Ausgangspunkt der Octree-basierten Unterteilung. Jede Zelle, die aus einem Unterteilungsschritt resultiert, wird in acht weitere Zellen unterteilt, wenn die lokale Approximation für diese Zelle zu stark von den mit dieser Zelle assoziierten Punkten abweicht.

Für jede Zelle können drei Schritte identifiziert werden:

1. Ermittlung des zu einer Zelle gehörenden Teilbereiches und der enthaltenen Punkte $\mathcal{P}'$
2. Ermittlung der lokalen Approximation $Q(x)$ der Oberfläche auf Basis von $\mathcal{P}'$
3. Vergleich der lokalen Approximation $Q(x)$ mit $\mathcal{P}'$. Ist die Approximation zu ungenau, wird die Zelle weiter unterteilt.

1. Die Ermittlung des kugelförmigen Teilbereiches auf Basis einer Octree-Zelle entspricht der Ermittlung eines geeigneten Radius R. Der Mittelpunkt des Teilbereichs liegt im Mittelpunkt der Zelle und wird im Folgenden mit c bezeichnet. R wird in Abhängigkeit von der Diagonale d der Zelle bestimmt.

$$R = \alpha d \tag{3.13}$$

α ist hierbei ein vom Benutzer einzugebender Parameter, die Autoren empfehlen einen Wert von 0.75. Wurde so der initiale kugelförmige Bereich bestimmt, werden die in ihm enthaltenen Punkte bestimmt. Um diese Suchoperation effizient zu gestalten, wird ein *kd-Tree* [Ben75] genutzt. Die Komplexität für die Suche der k nächstgelegenen Punkte beträgt somit maximal $O(\sqrt{N} + k)$ [Ben79].

Liegt die Anzahl der Punkte unter dem vom Benutzer spezifizierbaren Wert N_{min}, so wird der Radius R iterativ um λR erhöht, bis die Anzahl der Punkte hoch genug ist. λ ist wiederum vom Benutzer zu spezifizieren (Ohtake et al. schlagen 0.1 vor).

2. Wurde ein Teilbereich ermittelt, der genügend Punkte enthält, wird die lokale Approximation $Q(x)$ für diese Punkte $\mathcal{P}'$ bestimmt (siehe Abschnitt 3.3.3).

3. Anschließend wird auf Basis der Taubin-Entfernung [Tau91] bewertet, ob $Q(x)$ die Punkte aus $\mathcal{P}'$ genau genug approximiert. Die Taubin-Entfernung normalisiert den Funktionswert einer impliziten Funktion $F(x)$ an der Stelle x auf Basis des Wachstums der Funktion an dieser Stelle:

$$dist_x = \mid F(x) \mid / \mid \nabla F(x) \mid \tag{3.14}$$

Auf diese Weise wird ein variierendes Wachstum des Skalarfeldes ausgeglichen. Die Taubin-Entfernung ist somit eine Approximation der euklidischen Entfernung. Um nun die Genauigkeit von $Q(x)$ bewerten zu können, wird die maximale Entfernung, die ein Punkt aus $\mathcal{P}'$ zu ihr hat, ermittelt:

$$\epsilon = \max_{|p_i - c| < R} \mid Q(p_i) \mid / \mid \nabla Q(p_i) \mid \qquad \forall\, p_i \in \mathcal{P}' \tag{3.15}$$

Überschreitet ϵ den vom Benutzer angegebenen Wert ϵ_0, so ist die Approximation zu ungenau, und die Zelle wird weiter unterteilt. Durch Versuche konnten die Ohtake et al. 10^{-4} als geeigneten Wert für ϵ_0 ermitteln.

Die maximale Unterteilungstiefe des Octree kann durch den Parameter *max_level* eingeschränkt werden. Ohtake et al. gehen in ihrer Veröffentlichung zwar nicht auf diesen Parameter ein, er ist jedoch Bestandteil der von ihnen entwickelten Software [OBA+03a]. In dem Programm ist der Parameter mit dem Wert 20 belegt.

Tabelle 3.2 gibt einen Überblick über die Parameter, die die Unterteilung des Octree steuern. Auswirkungen dieser Parameter werden in Abschnitt 3.4.1 genauer diskutiert.

Tabelle 3.2: Übersicht über die Parameter zur Kontrolle der Octree-Unterteilung. Die empfohlenen Werte basieren auf Aussagen in [OBA+03b].

Prameter	Bedeutung	empfohlener Wert
α	steuert die Startgröße des kugelförmigen Bereiches in Abhängigkeit von der Diagonale der Octree-Zelle	0.75
λ	Wachstumsfaktor - steuert Wachstum des kugelförmigen Bereichs	0.1
N_{min}	Mindestanzahl der für eine lokale Approximation benötigten Punkte	15
ϵ_0	Maximal zulässiger Fehler v	10^{-4}
max_level	Maximale Unterteilungstiefe des Octree	20

3.3.3 Bestimmung der lokalen Approximation

Wurden ein Teilbereich und die zugehörige Punktmenge $\mathcal{P}'$ ermittelt, gilt es, die lokale Approximation $Q(x)$ zu berechnen. Im Folgenden sei $\mathcal{N}'$ die Menge der zu $\mathcal{P}'$ gehörenden Normalen. n ist das arithmetische Mittel dieser Normalen. θ stellt den größten Winkel zwischen n und den in $\mathcal{N}'$ enthaltenen Normalen dar. Auf Basis von $\mathcal{P}'$ und θ wird eine von drei verschiedenen Approximationsmethoden gewählt:

- Anpassen einer Quadrik
- Anpassen eines bivariaten quadratischen Polynoms
- Approximation von Kanten und Ecken

Anpassen einer Quadrik

Eine Quadrik wird genutzt, wenn die Anzahl der enthaltenen Punkte groß ist und die Normalen stark voneinander abweichen:

$$\begin{aligned} |\mathcal{P}'| &> 2N_{min} \\ \theta &\geq 90^\circ \end{aligned}$$

Eine Quadrik wird somit zur Approximation größerer Flächen oder gar mehrerer Flächen in einem Teilbereich genutzt. Ohtake et al. nutzen eine alternative Darstellung der Quadriken:

$$Q(x) = x^T Ax + b^T x + c \tag{3.16}$$

Mit

$$A = \begin{bmatrix} a & b & c \\ 0 & e & f \\ 0 & 0 & h \end{bmatrix} \quad , \quad b = \begin{bmatrix} d \\ g \\ i \end{bmatrix} \quad \text{und} \quad c = j$$

ist diese Formel äquivalent zu Formel 3.5. Die Anpassung einer Quadrik an die Punkte $\mathcal{P}'$ entspricht also der Ermittlung von A, b und c. Hierfür werden neun Hilfspunkte $\{q_i\}$

eingeführt: die acht Eckpunkte der Octree-Zelle sowie deren Zentrum. Mit Hilfe eines kd-Trees werden für jeden Hilfspunkt die sechs nächstgelegenen Punkte $p^{(1)}$,... $p^{(6)}$ aus $\mathcal{P}'$ sowie die zugehörigen Normalen $n^{(1)}$,... $n^{(6)}$ ermittelt. Basierend auf dem Skalarprodukt zwischen $(q - p^{(i)})$ und $n^{(i)}$ wird für jeden Hilfspunkt die Richtung der Normalen dieser sechs Punkte berechnet. Zeigen die sechs Normalenvektoren bezüglich der Position des Hilfspunktes nicht in die selbe Richtung, so wird der Hilfspunkt gelöscht (siehe Abbildung 3.14 links).

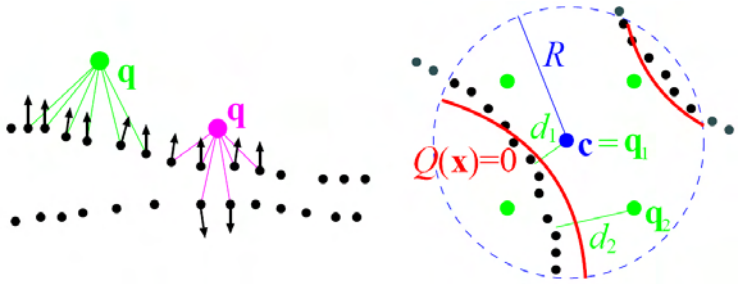


Abbildung 3.14: Nutzung von Hilfspunkten zur Anpassung einer Quadrik: Der magentafarbene Hilfspunkt wird nicht genutzt, weil die Normalen der sechs nächstgelegenen Punkte nicht in die selbe Richtung zeigen (*links*). Rechts: Für alle verbliebenen Hilfspunkte wird ein Entfernungsmaß d berechnet. *Quelle:* [OBA+03b]

Wenn kein Hilfspunkt übrig bleibt, wird die Zelle weiter unterteilt. Anderenfalls wird für jeden verbliebenen Hilfspunkt ein Entfernungsmaß d zu den sechs nächstgelegenen Punkten berechnet. Grundlage hierfür ist wiederum das zuvor beschriebene Skalarprodukt:

$$d = \frac{1}{6} \sum_{i=1}^{6} n^{(i)} \cdot (q - p^{(i)}) \tag{3.17}$$

Die Ermittlung der Unbekannten A, b und c kann nun durch eine Minimierung der folgenden Summe erreicht werden:

$$\frac{1}{\sum w(p_i)} \sum_{p_i \in \mathcal{P}'} w(p_i) Q(p_i)^2 + \frac{1}{m} \sum_{i=1}^{m} (Q(q_i) - d_i)^2 \tag{3.18}$$

Dies entspricht der *Least Squares Methode* (siehe z.B. [PFTV92]) für zwei Mengen von Punkten. Der erste Summand entspricht der Summe aller quadrierten Differenzen zwischen dem approximierten und dem „echten“ Funktionswert für die Punkte aus $\mathcal{P}'$. Der „echte“ Funktionswert ist hierbei 0, da sich die Punkte auf der Isooberfläche befinden. Der zweite Summand enthält die entsprechende Summe der quadrierten Differenzen für die m verbliebenen Hilfspunkte. Der „echte“ Funktionswert ist hierbei durch das Distanzmaß d gegeben (Abbildung 3.14 rechts).

Anpassen eines bivariaten quadratischen Polynoms

Ein bivariates quadratisches Polynom wird genutzt, wenn die Anzahl der enthaltenen Punkte groß ist und die Normalen nur geringfügig voneinander abweichen:

$$\begin{aligned} |\mathcal{P}'| &\geq 2N_{min} \\ \theta &< 90° \end{aligned}$$

Es wird angenommen, dass die Punkte aus $\mathcal{P}'$ eine relativ glatte Fläche bilden. Diese Fläche wird durch ein bivariates quadratisches Polynom angenähert. Diese spezielle Form eines Polynoms gibt für eine zweidimensionale Koordinate (u, v) eine Höhe zurück:

$$w = Au^2 + Buv + Cv^2 + Du + Ev + F \tag{3.19}$$

Durch Subtraktion der rechten Seite erhält man auf einer Seite 0 und kann $Q(x)$ einsetzen:

$$Q(x) = w - (Au^2 + Buv + Cv^2 + Du + Ev + F) \tag{3.20}$$

Es gilt nun, dieses Höhenfeld an $\mathcal{P}'$ anzupassen. Dazu werden u, v und w als Koordinaten der Punkte aus $\mathcal{P}'$ in einem lokalen Koordinatensystem (u, v, w) aufgefasst. Dessen Ursprung liegt in c. w ist parallel zu der Durchschnittsnormale n, die Ebene (u, v) wird orthogonal zu w angelegt (siehe Abbildung 3.15 links). Die Unbekannten A, B, C, D, E und F werden wiederum durch Least-Squares Fitting ermittelt.

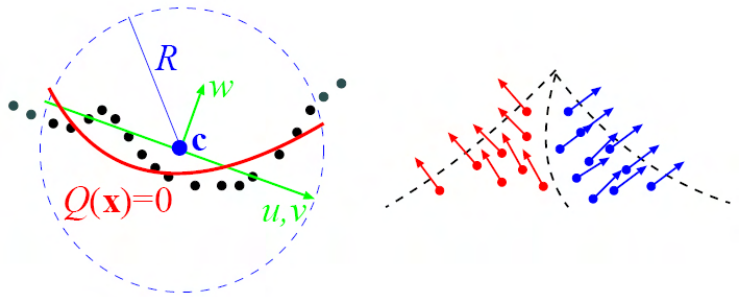


Abbildung 3.15: Links: Ein lokales Koordinatensystem (*grün*) wird zur Anpassung eines zweidimensionalen quadratischen Polynoms genutzt. Rechts: Zur Rekonstruktion von Kanten und Ecken werden Punkte auf Basis der Richtung ihrer Normalen in Teilmengen unterteilt. *Quelle:* [OBA+03b]

Approximation von Kanten und Ecken

Das Vorhandensein einer Kante oder Ecke wird angenommen, wenn in $\mathcal{P}'$ nur wenig Punkte enthalten sind:

$$|\mathcal{P}'| \quad \leq \quad 2N_{min}$$

Die zu $\mathcal{P}'$ gehörende Oberfläche wird durch die Kombination mehrerer Teilflächen repräsentiert. In einem ersten Schritt wird $\mathcal{P}'$ in Teilmengen zerlegt, die jeweils eine der an die Kante oder Ecke angrenzenden Flächen repräsentieren. Diese Zerlegung basiert auf dem in [KBSS01] vorgestellten Verfahren zur Erkennung von Kanten und Ecken. Die Winkel zwischen den in $\mathcal{N}'$ enthaltenen Normalenvektoren werden betrachtet. Basierend darauf werden die Punkte so in Teilmengen sortiert, dass die Normalen innerhalb einer Teilmenge so wenig wie möglich voneinander abweichen (Abbildung 3.15 rechts). Auf diese Weise kann der Algorithmus bis zu vier Teilmengen identifizieren und ist somit in der Lage, Ecken zu rekonstruieren, an die bis zu vier Flächen grenzen.

Anschließend wird an jede der Teilmengen eine quadratische Funktion angepasst. Deren Kombination zu einer Oberfläche, die $\mathcal{P}'$ approximiert, wird auf Basis von Bool'schen Operationen durchgeführt.

3.4 Praktische Aspekte beim Einsatz von MPU Implicits

Die Qualität der Rekonstruktion mit MPU Implicits hängt sowohl von Parametern des Algorithmus (siehe Tabelle 3.2), als auch von den Eingabedaten ab. Um MPU Implicits erfolgreich nutzen zu können, ist es deswegen notwendig, sowohl die Auswirkungen der Parameter als auch Anforderungen an die Eingabedaten zu kennen.

Im letztem Abschnitt wurden bereits einige verschiedene Parameter erwähnt, die die Erzeugung der impliziten Oberfläche beeinflussen können. Im Folgenden werden die Parameter des Algorithmus sowie deren Auswirkungen zusammengefasst. Daraufhin werden Anforderungen an den Eingabedatensatz hergeleitet. Die benutzten Abbildungen wurden mit Hilfe der von Ohtake et al. bereitgestellten MPU Implicits Software [OBA+03a] erzeugt.

3.4.1 Parameter des Algorithmus

Der Parameter α dient der Festlegung der Startgröße des kugelförmigen Teilbereichs in Abhängigkeit von der Größe der Octree-Zelle. Die Standardbelegung ist 0.75. Größere Werte führen zu glatteren Approximationen, da mehr Punkte für die lokale Approximation benutzt werden. Größere Werte führen jedoch auch zu längeren Rechenzeiten, weil sowohl die Suche nach den Punkten, als auch das anschließende Anpassen der lokalen Approximationen aufwändiger wird. Die Auswirkung von α auf die Glattheit der Rekonstruktion ist in Abbildung 3.16 dargestellt.

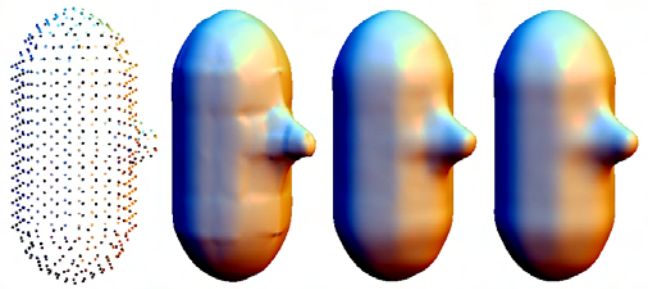
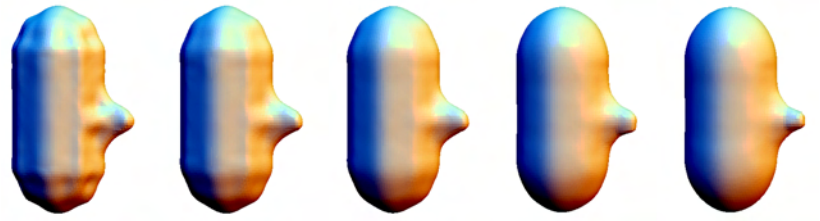


Abbildung 3.16: Auswirkung von α auf die Glattheit der Rekonstruktion: die Ausgangsdaten (*links*), Rekonstruktion mit $\alpha = 0.5$ (*mitte links*), $\alpha = 0.75$ (*mitte rechts*) und $\alpha = 1$ (*rechts*).

λ steuert das Wachstum des kugelförmigen Teilbereichs, wenn im Teilbereich zu wenig Punkte vorhanden sind. Ohtake et al. schlagen einen Wert von 0.1 vor. Das bedeutet, dass der Radius des Teilbereichs um 10% seiner ursprünglichen Größe wächst. Größere Werte wirken sich ähnlich aus wie größere Werte von α. Wenn der Teilbereich stärker wächst, sind nach einem Wachstumsschritt mehr Punkte enthalten, was wiederum zu einer glatteren Approximation aber auch zu höheren Rechenzeit führen kann.

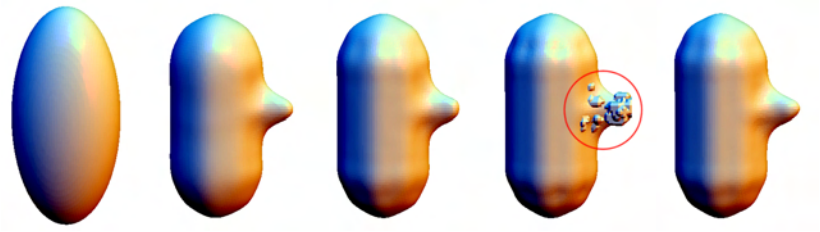
Die minimale, für eine lokale Rekonstruktion notwendige Punktzahl N_{min} beeinflusst ebenfalls die Glattheit. Ein größeres N_{min} führt dazu, dass für die lokalen Rekonstruktionen mehr Punkte benutzt werden. Somit sind auch mehr Details enthalten, die ein einzelnes quadratisches Polynom aber weniger genau darstellen kann als die Vereinigung kleinerer Flächen. Ein größeres N_{min} begünstigt also eine geglättete Darstellung (Abbildung 3.17).



N_{min}=15 N_{min}=30 N_{min}=45 N_{min}=75 N_{min}=100

Abbildung 3.17: Auswirkung von N_{min}: größere Werte begünstigen eine geglättete Darstellung, während kleinere Werte zu einer genaueren Approximation führen.

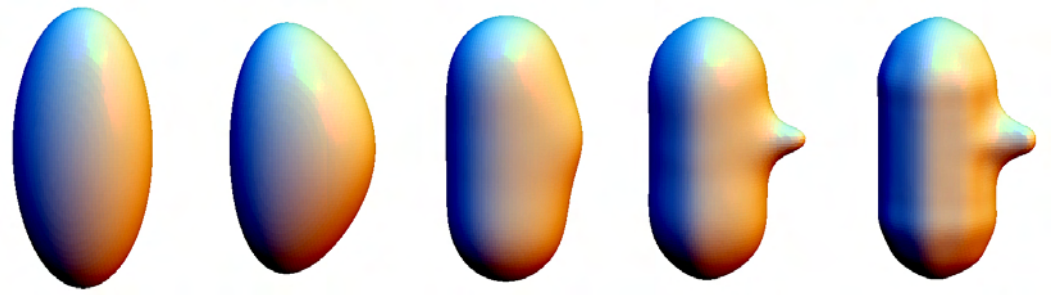
Die Rekursionstiefe der Octree-Aufteilung kann durch die Parameter ϵ_0 und *max_level* kontrolliert werden. Ist der Fehler ϵ der lokalen Approximation (siehe Formel 3.15) größer als ϵ_0, wird die Zelle erneut unterteilt. Der Wert bezieht sich auf die Diagonale der AABB der Punktwolke. Der von den Autoren vorgeschlagene Wert 10^{-4} entspricht somit 0.01% der Diagonale. Theoretisch kann die erzeugte Oberfläche somit beliebig genau an die Eingabedaten angepasst werden. Größere Werte führen zu einer groben und sehr glatt wirkenden Rekonstruktion. Kleinere Werte erzwingen eine genauere Rekonstruktion, die mehr Details aufweist. Unterhalb einer bestimmten Größe von ϵ_0 kann es so jedoch auch zur Entstehung von Artefakten kommen (siehe Abbildung 3.18).



ϵ_0=0.07 ϵ_0=0.05 ϵ_0=0.02 ϵ_0=0.01 ϵ_0=0.01

Abbildung 3.18: Mit dem maximalen Fehler ϵ_0 kann die Genauigkeit der Approximation gesteuert werden. Kleinere Werte führen zu genaueren Ergebnissen. Zu kleine Werte können jedoch zu Artefakten führen (*durch Kreis hervorgehoben*). Dies kann durch Einschränkung von *max_level* verhindert werden. Die vier ersten Abbildungen wurden mit *max_level*=20 generiert. Bei der Abbildung rechts konnten die Artefakte durch Reduzierung von *max_level* auf den Wert 5 unterbunden werden.

Ohtake et al. gehen in ihrer Arbeit nicht auf diese Artefakte ein. Die Entstehung der Artefakte bei einem zu geringen Wert für ϵ_0 wurde aber auch in [Bra05] beobachtet. Es scheint, dass sie auftreten, wenn eine Zelle zu klein wird und zu wenig Punkte enthält. In diesem Fall kann eine Quadrik nicht zuverlässig an die Punkte angepasst werden. Durch Einschränkung der maximalen Rekursionstufe durch den Parameter *max_level* kann die Entstehung der Artefakte verhindert werden (Abbildung 3.18 rechts), da die Zellen so eine bestimmte Größe nicht unterschreiten und genügend Punkte enthalten. Zu geringe Werte für *max_level* führen jedoch zu einer ungenauen, aber sehr organisch wirkenden Rekonstruktion (Abb. 3.19), da die Quadrik so eine größere Menge Punkte approximieren muss. In diesem Fall kann nicht mehr gewährleistet werden, dass der lokale Fehler kleiner als ϵ_0 ist.



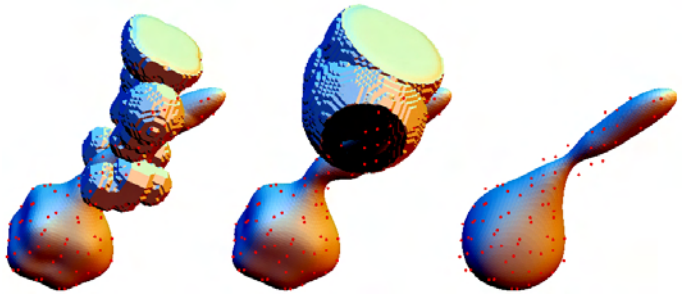
max_level=0 *max_level*=1 *max_level*=2 *max_level*=3 *max_level*=4

Abbildung 3.19: Auswirkung von *max_level* auf die Genauigkeit der Rekonstruktion: Kleine Werte führen zu einer sehr organischen, aber ungenauen Rekonstruktion. Mit größeren Werten steigt die Genauigkeit. ϵ_0 ist bei allen Versuchen 0.03.

Beide Parameter haben entscheidenden Einfluss auf Genauigkeit und Darstellungsqualität der Rekonstruktion. Die Modifizierung nur eines der beiden Parameter kann nicht immer zu den gewünschten Ergebnissen führen. Vor allem die auftretenden Artefakte bei einem zu geringen ϵ_0 sind hierbei problematisch. Deswegen ist es notwendig, ϵ_0 und *max_level* aufeinander abzustimmen.

3.4.2 Anforderungen an die Eingabedaten

Es wurde bereits erwähnt, dass das Verfahren in der Lage ist, Daten zu verarbeiten, die starkes Rauschen enthalten oder gar große Lücken aufweisen. Große Datenmengen bereiten dem Verfahren ebenso wenig Probleme. Ohtake et al. haben Datensätze mit bis zu 4 Millionen Punkten erfolgreich rekonstruiert. Bei einigen Datensätzen kann es jedoch zur Bildung von Artefakten kommen, die nur vermieden werden können, wenn die Unterteilung des Octrees extrem gering ist. Ursache hierfür ist eine zu geringe Anzahl von Punkten. Vor allem feine Details, die durch Punkte mit stark voneinander abweichenden Normalen definiert werden, können zur Bildung solcher Artefakte führen. In Abbildung 3.20 wird die Bildung solcher Artefakte an einem einfachen Testdatensatz gezeigt.



max_level=5 *max_level*=3 *max_level*=2

Abbildung 3.20: Artefakte entstehen durch eine zu geringe Anzahl Punkte. Sie können nur durch eine sehr geringe Octree-Aufteilung unterbunden werden.

Das dickere Ende des Objekts kann korrekt rekonstruiert werden. Die dünne Struktur, die sich daran anschließt, wird jedoch nur artefaktfrei rekonstruiert, wenn die maximale Unterteilung auf 2 beschränkt ist. Dies führt jedoch zu einer Approximation, die für alle Objektteile sehr ungenau ist. Deswegen ist es erforderlich, dass die Punktwolke genügend Punkte zur Beschreibung feiner Details enthält. Große glatte Flächen können hingegen durch wenige Punkte beschrieben werden.

3.5 Zusammenfassung

In diesem Kapitel wurden verschiedene explizite und implizite Verfahren zur Extraktion von Isooberflächen vorgestellt. Dabei konnten Multi-level Partition of Unity Implicits als geeignetes Verfahren identifiziert werden. Diese spezielle Form impliziter Oberflächen dient der Rekonstruktion von Objekten auf Basis von Punktwolken. Das Verfahren ist in der Lage, organisch wirkende Oberflächen zu erzeugen. Es hat sich in den letzten Jahren bewehrt und wird als State-of-the-Art-Technik betrachtet. Neben den bereits angesprochenen Anwendungsfeldern werden MPU Implicits auch in anderen Bereichen genutzt. Ohtake et al. untersuchen ihre Nutzbarkeit für die Detektion von *Ridge-Valley-Linien*[4] auf Objekten im Rahmen von *Non Photorealistic Rendering* [OBS04] (Abbildung 3.21 rechts). In [AJ05] dienen sie als Grundlage des intuitiven Freiformflächen-Modellierprogramms *FreeFormSKETCH* (Abbildung 3.21 links). Um ein Arbeiten in Echtzeit zu ermöglichen, wurden die MPU Implicits hierfür um eine lokale Neu-Polygonalisierung erweitert. Guo et al. nutzen MPU Implicits für physikalische Deformationen von Objekten in Echtzeit [GQ05].

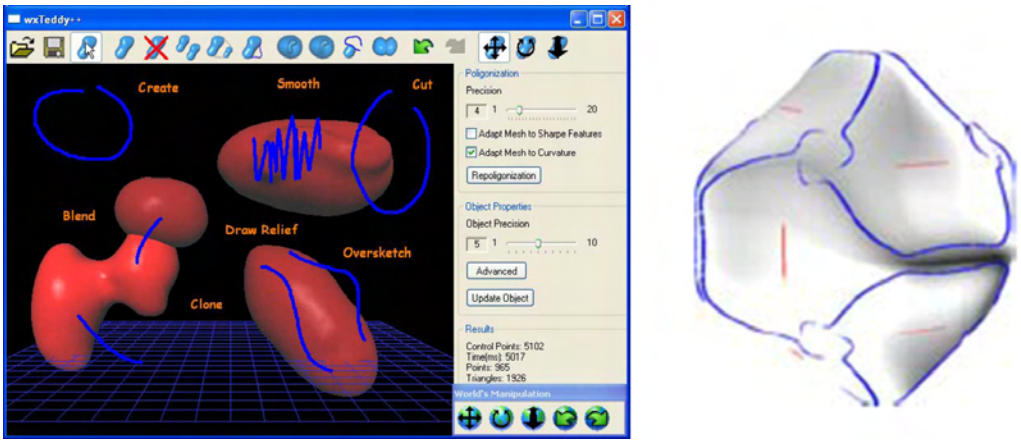


Abbildung 3.21: Anwendung von MPU Implicits: Oberflächenrepräsentation in dem kalligraphischen Modellierungstool *FreeFormSKETCH* (*links*), Detektion von Ridge-Valley-lines (*rechts*). *Quelle:* [AJ05], [OBS04]

Eine Anwendung im Rahmen der medizinischen Visualisierung wurde bereits in Abschnitt 3.2.3 vorgestellt. Braude nutzt MPU Implicits zur Rekonstruktion von Organen auf Basis von tomographischen Bilddaten [Bra05]. In dieser Arbeit soll eine ähnliche Methode zur Rekonstruktion von Blutgefäßen auf Basis von MPU Implicits entwickelt werden. Dabei gilt es, auf die praktischen Aspekte, die in Abschnitt 3.4 aufgezeigt wurden, einzugehen. Das Verfahren kann Artefakte auf der Oberfläche erzeugen, wenn für eine lokale Approximation zu wenige Punkte vorhanden sind. Deswegen sollte die Unterteilungstiefe des Octree mit Hilfe der beiden Parameter ϵ_0 und *max_level* genau überwacht werden. Darüber hinaus sollte die als Eingabe dienende Punktwolke feine Details und dünne Strukturen durch eine ausreichende Anzahl von Punkten beschreiben.

[4] Ridge-Valley-Linien zeigen Stellen mit besonders starker Oberflächenkrümmung

4 Entwurf

In Kapitel 2 wurde die möglichst genaue Wiedergabe der Gefäßmorphologie und eine organische Darstellung motiviert. Modellbasierte Verfahren scheinen dafür nicht geeignet. Das ihnen zugrunde liegende Gefäßmodell bildet jedoch die Grundlage für viele explorative und analytische Techniken. Als möglicherweise geeignetes Verfahren für eine genaue und organisch wirkende Darstellung wurden in Kapitel 3 MPU Implicits identifiziert.

In diesem Kapitel wird der Entwurf einer Gefäßvisualisierungsmethode beschrieben, die die Gefäßoberfläche möglichst genau mit Hilfe von MPU Implicits auf Basis des Segmentierungsergebnisses rekonstruiert.

Übersicht: Im ersten Abschnitt wird ein Überblick über die vorgeschlagene Gefäßvisualisierungspipeline gegeben. Abschnitt 4.2 dient der Beschreibung des Verfahrens zur Punktwolkenextraktion. In Abschnitt 4.3 wird eine sinnvolle Parameterbelegung für die Erzeugung und Polygonalisierung der MPU Implicits hergeleitet. Im darauf folgenden Abschnitt wird eine Möglichkeit aufgezeigt, wie die erzeugte Oberfläche auf ein Gefäßmodell abgebildet werden kann. Das Kapitel wird mit einer Zusammenfassung abgeschlossen.

4.1 Einordnung in die Visualisierungspipeline

Die in dieser Arbeit entwickelten Methoden stellen nur einen Teil einer Visualisierungspipeline dar. Die Generierung und Polygonalisierung der MPU Implicits wird als gegeben vorausgesetzt. Ausgangspunkt der Pipeline ist das Segmentierungsergebnis.

Die Pipeline lässt sich in folgende Teilprozesse aufteilen:

- **Punktwolkenextraktion:** Die Generierung einer Punktwolke auf Basis des Segmentierungsergebnisses ist ein entscheidender Beitrag dieser Arbeit und wird ausführlich in Abschnitt 4.2 vorgestellt.

- **Parameterbestimmung:** Eine effiziente Nutzung der MPU Implicits setzt eine sinnvolle Belegung der Parameter des Algorithmus voraus (siehe Abschnitt 3.4.1). Eine automatische Ermittlung geeigneter Parameter auf Basis der Eigenschaften des Segmentierungsergebnisses und der Punktwolke wird in Abschnitt 4.3 beschrieben.

- **MPU Implicits-Generierung:** Das Verfahren wurde in Kapitel 3.3 vorgestellt. In der Pipeline wird eine vorhandene Implementierung genutzt. Als Eingabe erhält der Algorithmus die zuvor generierte Punktwolke sowie die ermittelten Parameter.

- **Polygonalisierung:** Eine Polygonalisierung auf Basis von [Blo94] ist ebenfalls bereits vorhanden. Ein Vorschlag für die Größe der Zellen sowie den verwendeten Isowert kann auf Basis der Volumen- und Punktdaten erfolgen (siehe Abschnitt 4.3.2). Das Polygonalisierungsergebnis kann mit etablierten Verfahren dargestellt werden.

Die gesamte Visualisierungspipeline ist in Abbildung 4.1 dargestellt. In den folgenden Abschnitten werden die Punktwolkenextraktion sowie die Parameterbestimmung beschrieben.

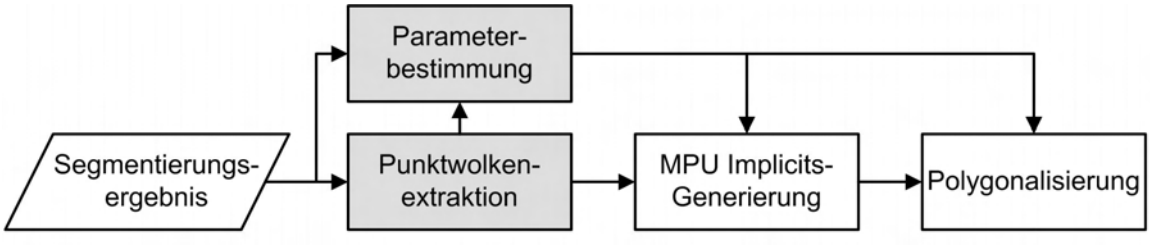


Abbildung 4.1: Überblick über die Gefäßvisualisierungspipeline: Die grau unterlegten Prozesse werden in dieser Arbeit betrachtet. Die weiß unterlegten Prozesse werden als existent vorausgesetzt.

4.2 Extraktion einer Punktwolke aus dem Segmentierungsergebnis

In diesem Abschnitt wird erläutert, wie aus dem Segmentierungsergebnis eine Punktwolke extrahiert wird. Das Segmentierungsergebnis $\mathcal{S}$ liegt hierbei in Form eines dreidimensionalen binären Datensatzes vor. Voxel, die zu dem beschriebenen Objekt gehören, werden als *Objektvoxel* bezeichnet und haben den Wert 1. Alle anderen Voxel sind *Hintergrundvoxel* und haben den Wert 0. Das Ergebnis des hier beschriebenen Verfahrens ist eine Menge $\mathcal{P}$ von N Punkten sowie eine Menge $\mathcal{N}$ von N zugehörigen Normalenvektoren:

$$\begin{aligned} \mathcal{P} &= \{p_1, ..., p_N\} \\ \mathcal{N} &= \{n_1, ..., n_N\} \end{aligned}$$

Anforderungen an das Verfahren

Das Verfahren zur Extraktion der Punkte muss folgende Kriterien erfüllen:

- Die Gefäßoberfläche muss überall durch genügend Punkte beschrieben werden, damit die Oberflächenrekonstruktion mit MPU Implicits für die gesamte Oberfläche artefaktfrei erfolgen kann.
- Die Anordnung der Punkte soll eine glatte und genaue Rekonstruktion der Oberfläche ermöglichen.
- Die Punkte und Normalen der Punktwolke müssen in Weltkoordinaten erzeugt werden.
- Das Verfahren muss schnell sein, damit ein Einsatz im klinischen Alltag sinnvoll wird.

Bei dem Vorgang muss besondere Sorgfalt darauf verwendet werden, dass jedes Merkmal der Oberfläche durch genügend Punkte beschrieben wird, da es sonst zu der Entstehung der in Abschnitt 3.4.2 beschriebenen Artefakte kommen kann. Als Grundlage für die

Positionierung der Punkte wird das Voxelgitter genutzt. Somit ergibt sich für Strukturen, die durch wenig Voxel beschrieben werden, das Problem, dass sie durch zu wenig Punkte repräsentiert werden und es zur Bildung von Artefakten kommen kann (siehe Abbildung 4.2). Die Bildung solcher Artefakte wurde bei Strukturen beobachtet, deren Querschnitt kleiner als drei mal drei Voxel ist. Solche Strukturen werden im Folgenden als *dünne Strukturen* bezeichnet. Eine artefaktfreie Beschreibung dünner Strukturen kann erfolgen, indem in Abhängigkeit von dem Voxelgitter mehr Punkte platziert werden. Damit so die Anzahl der generierten Punkte, und somit auch die Rechenzeit, nicht übermäßig steigt, ist ein adaptiver Ansatz erforderlich. Der Ansatz zur Generierung einer größeren Anzahl von Punkten kommt daher nur zum Einsatz, wo dünne Strukturen beschrieben werden.

Desweiteren soll die Punktwolke zu der Rekonstruktion einer möglichst glatten Oberfläche führen. Bei der Platzierung der Punkte muss deswegen darauf geachtet werden, dass die Punkte nicht direkt auf den Flächen der Voxel platziert werden. Die Punkte sollten eine Oberfläche beschreiben, die zwar durch die Treppenstufen verläuft, sich aber nicht an die Flächen der Voxel anpasst. Um eine genaue Rekonstruktion der Oberfläche zu ermöglichen, dürfen die Punkte dabei nicht zu weit von der Oberfläche entfernt werden.

Eine weitere Anforderung liegt darin, dass die Punkte und Normalen in Weltkoordinaten erzeugt werden. Hierfür ist eine Transformation notwendig, da die Voxel, auf deren Basis die Punkte platziert werden, in Voxelkoordinaten gegeben sind.

Die letzte Anforderung besteht darin, dass das Verfahren die Punktwolke möglichst schnell erzeugt. Dies ist wichtig, wenn das Verfahren im klinischen Alltag eingesetzt werden soll.

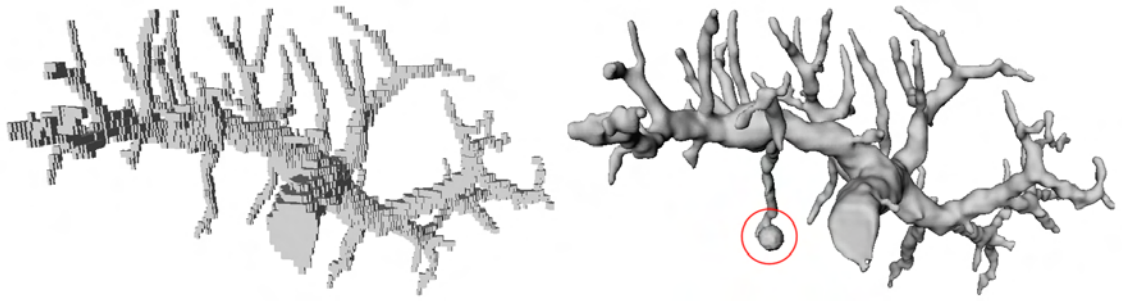


Abbildung 4.2: Werden dünne Strukturen nicht durch genügend Punkte beschrieben, kann es zur Bildung von Artefakten kommen. Ein solches Artefakt wird durch den Kreis hervorgehoben.

Anforderungen an das Segmentierungsergebnis

Ausgangspunkt des Verfahrens ist das Segmentierungsergebnis $\mathcal{S}$. Um eine Anwendung des vorgestellten Verfahrens zu ermöglichen, muss $\mathcal{S}$ die folgenden Anforderungen erfüllen:

- Der beschriebene Gefäßbaum darf sich nicht bis zum Rand des Datensatzes erstrecken, da einige der Verarbeitungsschritte Hintergrundvoxel am Rand des beschriebenen Objektes betrachten.
- Der Datensatz sollte nur ein zusammenhängendes Objekt enthalten, da der verwendete Bloomenthal-Polygonizer nur eine zusammenhängende Struktur polygonalisieren kann.

Diese beiden Anforderungen stellen keine großen Beschränkungen dar. Die erste Anforderung kann erfüllt werden, indem $\mathcal{S}$ vor Anwendung des Verfahrens in jeder Dimension in beide Richtungen um jeweils ein Voxel vergrößert wird.

Eine Lösung für die Erfüllung der zweiten Anforderung liegt in der Aufteilung des Segmentierungsergebnisses in mehrere Datensätze. Durch *Connected Component Labeling* (siehe zum Beispiel [DZ00]) können einzelne, unabhängige Strukturen im Segmentierungsergebnis erkannt und markiert werden. Anschließend kann die Menge aller Voxel mit dem selben Wert als Segmentierungsergebnis einer Teilstruktur aufgefasst werden. Nach Anwendung der hier beschriebenen Visualisierungspipeline auf die einzelnen Teilstrukturen können die Polygonalisierungsergebnisse wieder verbunden werden, um ein 3D-Modell zu erhalten, das das ursprüngliche Segmentierungsergebnis repräsentiert.

Überblick über die Punktextraktion

Der Prozess zur Generierung einer Punktwolke auf Basis des Segmentierungsergebnisses gliedert sich in folgende Schritte:

- **Identifizierung dünner Strukturen:** Voxel, die zu dünnen Strukturen gehören, werden detektiert.
- **Klassifizierung aller Voxel:** Hierbei werden alle Voxel mit Markierungen versehen. Für jedes Voxel wird so die Zugehörigkeit zum Objekt, zum Hintergrund, zu dünnen Strukturen und zum Rand des Objektes gespeichert. Diese Klassifizierung ist Grundlage der nachfolgenden Schritte.
- **Überabtastung der Voxel:** Jedes Voxel wird durch acht Subvoxel repräsentiert.
- **Reduzierung treppenartiger Artefakte** im Subvoxel-Datensatz an dünnen Strukturen. Hierfür werden Subvoxel hinzugefügt.
- **Extraktion der Punkte und Normalen:** Grundlage hierfür sind das Segmentierungsergebnis sowie die in den vorherigen Schritten erzeugten Datensätze.
- **Überführung in Weltkoordinaten:** Die zuvor berechneten Punkte und Normalen werden in das Weltkoordinatensystem transformiert.

In Abbildung 4.3 ist dargestellt, in welcher Relation diese Teilprozesse zueinander stehen. Die Identifizierung dünner Strukturen sowie die Überabtastung benötigen das Segmentierungsergebnis als Eingabe. Das Segmentierungsergebnis bildet zusammen mit den identifizierten dünnen Strukturen die Grundlage für die Klassifizierung des Segmentierungsergebnisses. In der Klassifizierung wird für jedes Voxel gespeichert, ob es sich um ein Hintergrund- oder Objektvoxel handelt, und ob das Voxel zu einer dünnen Struktur gehört oder benachbart ist. Der klassifizierte Datensatz wird genutzt, um die treppenartigen Artefakte in dem zuvor überabgetasteten Segmentierungsergebnis zu reduzieren. Voraussetzung für die Extraktion der Punkte und Normalen sind das Segmentierungsergebnis, der klassifizierte Datensatz und das überabgetastete Segmentierungsergebnis.
Die einzelnen Teilprozesse werden im Folgenden näher erläutert.

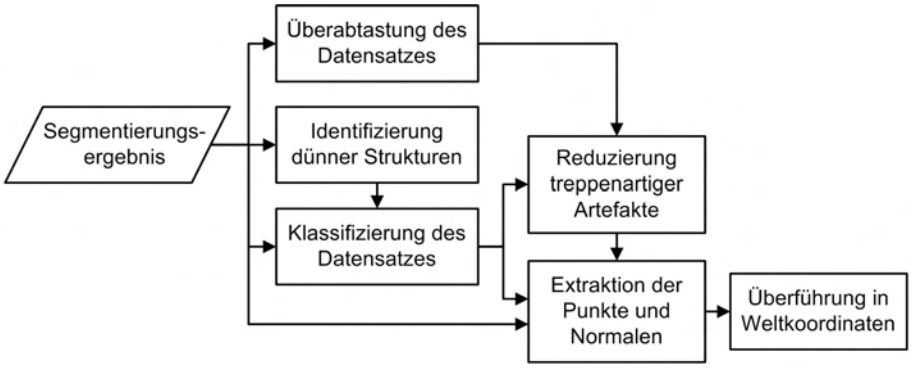


Abbildung 4.3: Überblick über das Verfahren zur Punktextraktion: Vom Segmentierungsergebnis werden weitere Bilddatensätze abgeleitet, die die Grundlage für die Punkt- und Normalenextraktion bilden.

4.2.1 Identifizierung dünner Strukturen

Wie später beschrieben wird (Abschnitt 4.2.5), können Punkte auf Basis eines feiner aufgelösten Voxelgitters platziert werden. Da dies bei Anwendung auf den gesamten Gefäßbaum zu einer sehr großen Menge Punkte und einer hohen Berechnungsdauer führen würde, ist ein adaptiver Ansatz wünschenswert. Hierbei wird das höher aufgelöste Voxelgitter nur zur Beschreibung dünner Strukturen genutzt. Die Punkte zur Beschreibung des restlichen Teils der Oberfläche werden auf Basis des ursprünglichen Voxelgitters erzeugt.

Grundlage dieser adaptiven Betrachtung ist die Identifizierung der Voxel, die zu einer dünnen Struktur gehören. Hierfür sind *morphologische Operatoren* geeignet. Morphologische Operatoren dienen der strukturellen Analyse von zwei- und dreidimensionalen Bilddatensätzen. Dabei wird die Anzahl der Objektvoxel mit Hilfe eines Strukturelements $\mathcal{K}$ vergrößert oder verringert. Die Form des Strukturelements bestimmt dabei, welche Objektvoxel hinzugefügt oder gelöscht werden.

Die beiden grundlegenden morphologischen Operatoren sind *Erosion* und *Dilatation*. Bei der Erosion werden nur jene Objektvoxel beibehalten, deren Nachbarschaft das Strukturelement komplett aufnehmen kann. Die Dilatation führt zur Auffüllung der Nachbarvoxel eines jeden Objektvoxels auf Basis des Strukturelements. Eine Erosion gefolgt von einer Dilatation wird als *Opening* bezeichnet. Dieser Operator ist dazu in der Lage, Strukturen zu entfernen, die das Strukturelement nicht komplett aufnehmen können.

Im Folgenden sei $\mathcal{O}$ die Menge der Objektvoxel aus $\mathcal{S}$. Ein Opening auf Basis des Strukturelements $\mathcal{K}$ angewendet auf $\mathcal{O}$ wird folgendermaßen ausgedrückt:

$$\mathcal{O}' = \mathcal{O} \circ \mathcal{K} = (\mathcal{O} \ominus \mathcal{K}) \oplus \mathcal{K} \tag{4.1}$$

$\ominus$ und $\oplus$ kennzeichnen hierbei Erosion und Dilatation. $\mathcal{O}'$ enthält nur noch jene Objektteile, die das Strukturelement komplett aufnehmen können. Durch Bildung der Differenz zwischen dem originalen sowie dem durch das Opening gewonnenen Datensatz kann die Menge der Voxel, die zu einer dünnen Struktur gehören, ermittelt werden:

$$\mathcal{O}'' = \mathcal{O} - \mathcal{O}' \tag{4.2}$$

Für die Anwendung des Verfahrens zur Erkennung dünner Zweige wird ein $3 \times 3 \times 3$ -Strukturelement genutzt. Ein beispielhaftes Ergebnis der Methode ist in Abbildung 4.4 zu sehen.

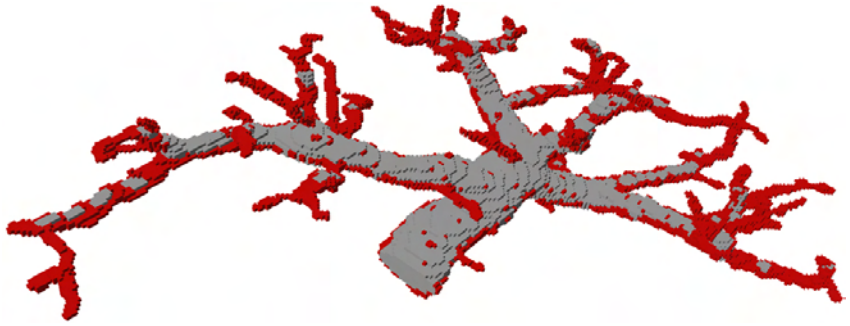


Abbildung 4.4: *Opening* angewendet auf einen dreidimensionalen Datensatz: Die hervorgehobenen Voxel wurden als zu einer dünnen Struktur gehörend erkannt.

4.2.2 Klassifizierung der Voxel

Die Punkte der Punktwolke sollen auf Basis des Voxelgitters positioniert werden. Dazu müssen die Voxel identifiziert werden, innerhalb deren Volumen die Punkte platziert werden sollen. [Bra05] nutzt hierfür die *Randvoxel* des segmentierten Volumens. Randvoxel sind Objektvoxel, die zu Hintergrundvoxeln benachbart sind. Diese Platzierung ist jedoch für dünne Strukturen problematisch. Vor allem Strukturen, die nur ein Voxel dick sind, könnten so nicht beschrieben werden. Die erzeugten Punkte würden nur eine Linie beschreiben.

Statt dessen wird die Platzierung der Punkte in den *äußeren Randvoxeln* vorgeschlagen. Äußere Randvoxel sind Hintergrundvoxel, die direkt an das beschriebene Objekt grenzen. Der Vorteil dieser Verfahrensweise liegt darin, dass selbst eine Struktur, die nur ein Voxel dick ist, komplett von diesen äußeren Randvoxeln eingehüllt wird. Punkte, die auf Basis dieser Voxel platziert werden, umgeben somit diese dünne Struktur.

Für die nachfolgenden Schritte wird ein neuer Voxeldatensatz konstruiert, der für jedes Voxel definiert, ob es sich um ein äußeres Randvoxel, ein Hintergrund- oder Objektvoxel handelt. Zudem wird die Zugehörigkeit oder Nähe zu dünnen Strukturen in den Datenwerten der Voxel kodiert. Diese Informationen werden in dem Datensatz festgehalten, da sie für die nachfolgenden Schritte häufig benötigt werden.
Die Menge aller Voxel wird somit in fünf Gruppen unterteilt:

- **Datenwert 0:** Hintergrundvoxel, die kein Objektvoxel als Nachbar haben
- **Datenwert 1:** Objektvoxel, die zu keiner dünnen Struktur gehören
- **Datenwert 2:** Objektvoxel, die zu einer dünnen Struktur gehören
- **Datenwert 3:** Hintergrundvoxel, die mindestens ein Objektvoxel als Nachbar haben, jedoch kein Objektvoxel einer dünnen Struktur
- **Datenwert 4:** Hintergrundvoxel, die mindestens ein Objektvoxel einer dünnen Struktur als Nachbar haben

Voxel mit dem Datenwert 3 oder 4 sind somit äußere Randvoxel. Bei der Untersuchung, ob ein Hintergrundvoxel zu einem Objektvoxel benachbart ist, wird die *3D-6-Nachbarschaft* betrachtet. Diese Nachbarschaft beinhaltet jene sechs Voxel, die eine Fläche mit dem betrachteten Voxel gemeinsam haben (Abbildung 4.5 mitte links).

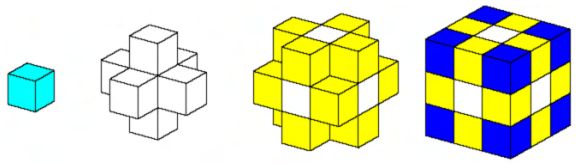


Abbildung 4.5: Nachbarschaft eines Voxels: das Voxel (*links*), seine 3D-6-Nachbarschaft (*mitte links*), seine 3D-18-Nachbarschaft (*mitte rechts*) und seine 3D-26-Nachbarschaft (*rechts*). *Quelle:* [Str05]

Für ein Voxel an der Position (x, y, z) ist die Menge der in der 3D-6-Nachbarschaft enthaltenen Voxel folgendermaßen definiert:

$$\mathcal{N}_6(x,y,z) = \left\{ \begin{bmatrix} x-1 \\ y \\ z \end{bmatrix}, \begin{bmatrix} x+1 \\ y \\ z \end{bmatrix}, \begin{bmatrix} x \\ y-1 \\ z \end{bmatrix}, \begin{bmatrix} x \\ y+1 \\ z \end{bmatrix}, \begin{bmatrix} x \\ y \\ z-1 \end{bmatrix}, \begin{bmatrix} x \\ y \\ z+1 \end{bmatrix} \right\} \tag{4.3}$$

Basierend auf den in Abschnitt 4.2.1 definierten Mengen $\mathcal{O}$ (Objektvoxel), $\mathcal{O}'$ (Objektvoxel die nicht zu dünnen Strukturen gehören) und $\mathcal{O}''$ (Objektvoxel die zu dünnen Strukturen gehören) kann nun ein klassifizierter Datensatz $\mathcal{S}^*$ erzeugt werden. Seine Dimensionen stimmen mit denen von $\mathcal{S}$ überein. Die Dimensionen von $\mathcal{S}$ werden im Folgenden mit dim_x, dim_y und dim_z bezeichnet. $\mathcal{S}^*$ ist folgendermaßen definiert:

$$\mathcal{S}^*(x,y,z) = \begin{cases} 1 & \text{, wenn } (x,y,z) \in \mathcal{O}' \\ 2 & \text{, wenn } (x,y,z) \in \mathcal{O}'' \\ 3 & \text{, wenn } (x,y,z) \notin \mathcal{O} \text{ und } \mathcal{N}_6(x,y,z) \cap \mathcal{O}' \neq \emptyset \\ & \quad \text{und } \mathcal{N}_6(x,y,z) \cap \mathcal{O}'' = \emptyset \\ 4 & \text{, wenn } (x,y,z) \notin \mathcal{O} \text{ und } \mathcal{N}_6(x,y,z) \cap \mathcal{O}'' \neq \emptyset \\ 0 & \text{, sonst} \end{cases}$$
$$\forall\ x = 0, ..., dim_x - 1,$$
$$y = 0, ..., dim_y - 1,$$
$$z = 0, ..., dim_z - 1 \tag{4.4}$$

$\mathcal{S}^*$ enthält Informationen darüber, wo sich Objektvoxel befinden, und welche von ihnen zu einer dünnen Struktur gehören. Für alle Hintergrundvoxel ist zudem bekannt, ob sie direkt zu dem beschriebenen Objekt benachbart sind oder an dünne Strukturen grenzen. Diese Klassifizierung bildet die Voraussetzung für die folgenden Schritte. In Abbildung 4.6 ist die Klassifizierung für den zweidimensionalen Fall illustriert.

0	0	0	0	4	0	0	0
0	0	0	4	2	4	0	0
0	0	0	4	2	4	0	0
0	0	4	2	2	4	0	0
0	0	3	1	1	1	3	0
0	3	1	1	1	1	1	3

Abbildung 4.6: Klassifizierung am Beispiel eines zweidimensionalen Datensatzes: Jeder Pixel wird einer der fünf Gruppen zugeordnet.

4.2.3 Überabtastung der Voxel

Wie bereits erwähnt wurde, stellt das Voxelgitter die Grundlage der Punktplatzierung dar. Die daraus resultierende regelmäßige Positionierung würde jedoch dazu führen, dass dünne Strukturen mit verhältnismäßig wenig Punkten beschrieben werden. Um dies zu verhindern, wird ein höher aufgelöstes Gitter verwendet. Jedes Voxel wird hierbei in gleich große *Subvoxel* aufgeteilt. Es wird eine Aufteilung eines Voxels in acht Subvoxel gewählt. Diese Unterteilung ist ausreichend, damit auch Zweige, deren Querschnitt nur ein Voxel beträgt, artefaktfrei rekonstruiert werden. Eine Aufteilung in mehr Subvoxel würde einen weitaus größeren Datensatz erfordern und zu einer viel größeren Anzahl von Punkten führen. Die Berechnungszeiten für die nachfolgenden Schritte würden stark ansteigen.

Für die Überabtastung wird ein binärer Datensatz $\bar{\mathcal{S}}$ erzeugt. Seine Abmessungen sind in allen drei Dimensionen doppelt so groß wie der originale Datensatz $\mathcal{S}$. Somit entsprechen jedem Voxel aus $\mathcal{S}$ acht Subvoxel in $\bar{\mathcal{S}}$. Nun werden alle Voxel von $\mathcal{S}$ auf $\bar{\mathcal{S}}$ übertragen. Dabei erhalten je acht Subvoxel aus $\bar{\mathcal{S}}$ den Datenwert des selben Voxels in $\mathcal{S}$:

$$\bar{\mathcal{S}}(x,y,z) = \mathcal{S}(\lfloor x/2 \rfloor, \lfloor y/2 \rfloor, \lfloor z/2 \rfloor) \qquad \begin{aligned} ,\forall\ x &= 0,...,2 * dim_x - 1 \\ y &= 0,...,2 * dim_y - 1 \\ z &= 0,...,2 * dim_z - 1 \end{aligned} \tag{4.5}$$

Das Resultat dieser Operation ist in Abbildung 4.7 für einen zweidimensionalen Datensatz illustriert.

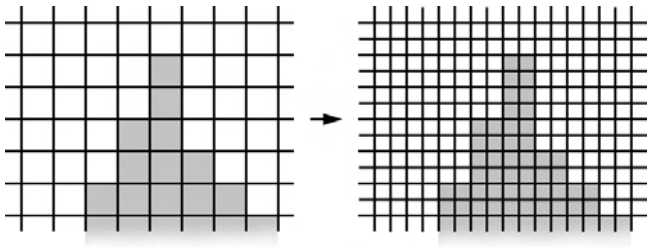


Abbildung 4.7: Überabtastung verdeutlicht an einem zweidimensionalen Datensatz: ein Pixel des Originaldatensatzes wird auf vier Subpixel abgebildet. Im dreidimensionalen Fall wird ein Voxel auf acht Subvoxel abgebildet,

4.2.4 Reduzierung treppenartiger Artefakte

Die im letzten Abschnitt beschriebene Überabtastung bringt einen großen Nachteil mit sich: Die aus der Diskretisierung resultierenden Stufen im Segmentierungsergebnis werden nun durch mehr Voxel beschrieben und somit noch stärker betont. Eine Platzierung der Punkte auf Basis dieses Datensatzes würde zur Repräsentation der Treppen durch mehr Punkte führen. Eine starke Variation der zugehörigen Normalen innerhalb eines kleinen Bereiches wäre eine weitere Folge. Da die Oberflächenrekonstruktion mit MPU Implicits eine möglichst genaue Approximation der Punkte liefert, würden die Treppen in der rekonstruierten Oberfläche erhalten bleiben (siehe Abbildung 4.8 mitte).

Um eine möglichst organische Darstellung zu gewährleisten, ist es deswegen notwendig, die treppenartigen Artefakte bereits in dem überabgetasteten Datensatz $\bar{\mathcal{S}}$ zu verringern. Dies könnte eventuell durch die Nutzung von Filtern erreicht werden. Durch Interpolation

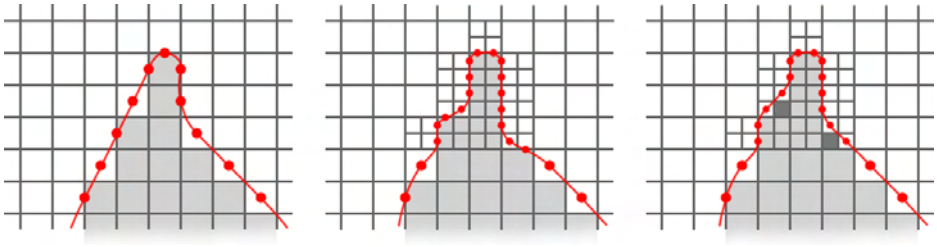


Abbildung 4.8: Durch die Überabtastung werden Stufen durch mehr Punkte beschrieben, was zur Erzeugung einer treppenartigen Oberfläche führen kann (*mitte*). Durch Hinzufügen von Subvoxeln an den Stufen (*dunkelgrau*) werden die Artefakte verringert (*rechts*).

würde so automatisch eine Glättung des Segmentierungsergebnisses durchgeführt. Hierfür wurden verschiedene Filter getestet. Die Überführung des resultierenden Graustufenbildes in ein Binärbild erwies sich jedoch als schwierig. Bei unterschiedlichen Datensätzen mussten unterschiedliche Schwellwerte benutzt werden, um zu einem Ergebnis zu gelangen, bei dem die treppenartigen Artefakte überall ausreichend reduziert wurden. Zudem kann dieser Ansatz zum Verschwinden kleiner Zweige führen.

Für eine robuste Reduzierung der treppenartigen Artefakte wird statt dessen ein fallbasiertes Hinzufügen zusätzlicher Subvoxel in den Treppenstufen genutzt. Basierend auf einer Betrachtung der lokalen Nachbarschaft einzelner Subvoxel wird entschieden, ob diese zu $\bar{\mathcal{S}}$ hinzugefügt werden. Der entsprechende Algorithmus besteht aus zwei Schritten:

- Identifizierung der zu betrachtenden Voxel und
- Untersuchung dieser Voxel auf Subvoxelebene

Identifizierung der zu betrachtenden Voxel

Zuerst muss die Menge der Voxel identifiziert werden, für die die Auffüllung der Treppenstufen zum einen nötig und zum anderen überhaupt möglich ist. Da das höher aufgelöste Voxelgitter nur zur Beschreibung dünner Strukturen genutzt werden soll, müssen Subvoxel auch nur an dünnen Strukturen hinzugefügt werden. Das Hinzufügen von Subvoxeln kann zudem nur zu einer Vergrößerung des beschriebenen Volumens führen, deswegen kommen nur Hintergrundvoxel als Kandidaten für ein Auffüllen in Frage, und zwar nur solche, die an das Objekt grenzen. Aus diesen beiden Beobachtungen folgt, dass nur Hintergrundvoxel, die an dünne Strukturen des beschriebenen Objektes grenzen, eine Untersuchung in Hinsicht auf das Hinzufügen von Subvoxeln erfordern. Diese Voxel wurden bereits identifiziert und sind im Datensatz $\mathcal{S}^*$ mit einer 4 markiert (siehe Abschnitt 4.2.2). Die Menge dieser Voxel sei im Folgenden $\mathcal{V}_1$:

$$\mathcal{V}_1 = \{(x, y, z) : \mathcal{S}^*(x, y, z) = 4\} \tag{4.6}$$

Nun müssen drei Sonderfälle, in denen ein Auffüllen von Subvoxeln unerwünscht ist, beachtet werden. Zum einen sollte eine Vertiefung, die genau die Größe eines Voxels hat, nicht durch zusätzliche Subvoxel verkleinert werden. Eine solche Vertiefung ist letztendlich ein Hintergrundvoxel, das in seiner 3D-6-Nachbarschaft fünf Objektvoxel als Nachbarn hat (siehe Abbildung 4.9 links) und kann somit leicht ausgeschlossen werden.

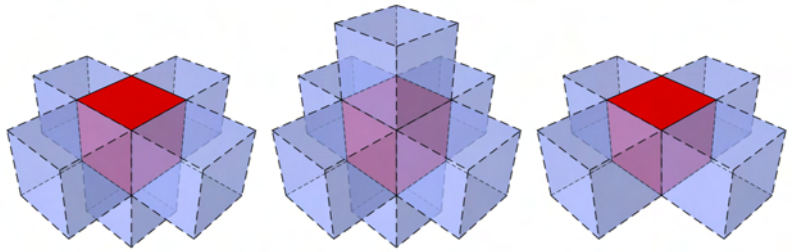


Abbildung 4.9: Sonderfälle, die bei der Auffüllung von Subvoxeln ausgeschlossen werden müssen: Eine Vertiefung (*links*), ein Hohlraum (*mitte*) und ein Loch (*rechts*). Das betrachtete Hintergrundvoxel ist hervorgehoben dargestellt.

Das Gleiche gilt für Hohlräume im Inneren des Objektes, die genau ein Voxel groß sind. Diese lassen sich daran erkennen, dass sie in ihrer 3D-6-Nachbarschaft sechs Objektvoxel besitzen (Abbildung 4.9 mitte). Dieser Fall wird ausgeschlossen, da solche Hohlräume auch bei der späteren Punktplatzierung vernachlässigt werden. Vertiefungen und Hohlräume können auf gleiche Weise erkannt werden:

$$\mathcal{V}_2 = \{(x, y, z) \in \mathcal{V}_1 : num_{nb}(x, y, z) > 4\} \tag{4.7}$$

$num_{nb}(x, y, z)$ ist hierbei die Anzahl der Objektvoxel in der 3D-6-Nachbarschaft des Voxels an der Position (x, y, z). Die Nachbarschaft wird in diesem Fall aus dem Datensatz $\mathcal{S}$ ermittelt.

Der dritte Sonderfall betrifft Hintergrundvoxel, die ein Loch darstellen. Diese Voxel haben genau vier Objektvoxel als Nachbarn, die in einer Ebene liegen (Abbildung 4.9 rechts). Ein Hinzufügen von Subvoxeln würde zu einer Verkleinerung des Loches führen. Zur Erkennung dieses Falls wird der Vektor v_{nb} als Summe der Richtungsvektoren vom Zentrum des betrachteten Voxels zu allen benachbarten Objektvoxeln berechnet:

$$v_{nb}(x, y, z) = \sum_{v \in V} dir(x, y, z, v) \tag{4.8}$$

V ist hierbei die Menge der Richtungsvektoren vom betrachteten Voxel zu seinen sechs direkten Nachbarn:

$$V = \left\{ \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\} \tag{4.9}$$

Die Funktion $dir(x, y, z, v)$ in Formel 4.8 gibt für ein Nachbarvoxel den Richtungsvektor zurück, wenn der Nachbarvoxel ein Objektvoxel ist, ansonsten den Nullvektor.

$$dir(x, y, z, v) = \begin{cases} v & \text{,wenn } \mathcal{S}((x, y, z) + v) > 0 \\ (0, 0, 0) & \text{,sonst} \end{cases} \tag{4.10}$$

v_{nb} entspricht nur dem Nullvektor, wenn die Summe der Richtungsvektoren gegenüberliegender Nachbarvoxel gleich dem Nullvektor ist. Hat das betrachtete Voxel nur vier Objektvoxel als Nachbarn, kann v_{nb} nur dem Nullvektor entsprechen, wenn sich je zwei der

benachbarten Objektvoxel gegenüberliegen. In diesem Fall liegen alle vier benachbarten Objektvoxel in einer Ebene. Das betrachtete Voxel stellt somit ein Loch dar, wenn es genau vier Objektvoxel als Nachbarn besitzt und v_{nb} für dieses Voxel gleich dem Nullvektor ist. Die Menge all dieser Voxel sei:

$$\mathcal{V}_3 = \{(x, y, z) : num_{nb}(x, y, z) = 4 \wedge v_{nb}(x, y, z) = (0, 0, 0)\} \tag{4.11}$$

Die Menge der zu betrachtenden Voxel $\mathcal{V}_{step}$ ergibt sich also aus der Menge der Voxel in $\mathcal{S}^*$, die den Datenwert 4 haben abzüglich der Voxel, die einen der genannten drei Sonderfälle erfüllen:

$$\mathcal{V}_{step} = \mathcal{V}_1 - \mathcal{V}_2 - \mathcal{V}_3 \tag{4.12}$$

Untersuchung der Voxel auf Subvoxelebene

Wurde $\mathcal{V}_{step}$ ermittelt, gilt es, die Subvoxel der in dieser Menge enthaltenen Hintergrundvoxel zu betrachten. Grundlage hierfür ist der höher unterteilte Subvoxeldatensatz $\bar{\mathcal{S}}$. Die acht Subvoxel, die zu einem Voxel aus $\mathcal{V}_{step}$ gehören, werden hierbei unabhängig voneinander betrachtet. Da die Voxel aus $\mathcal{V}_{step}$ Hintergrundvoxel sind, gehören auch diese Subvoxel zum Hintergrund und haben den Wert 0. Für jedes dieser Subvoxel wird auf Basis seiner Nachbarschaft entschieden, ob es in $\bar{\mathcal{S}}$ auf 1 gesetzt wird und somit ein Objektsubvoxel darstellt.

In Abbildung 4.10 wird verdeutlicht, in welchen Fällen ein Subvoxel gesetzt werden soll. Zum einen soll jedes Subvoxel aus $\mathcal{V}_{step}$ gesetzt werden, das mit mindestens zwei Objektsubvoxeln eine Fläche gemeinsam hat (Abbildung 4.10 links). In diesem Fall befindet sich das Subvoxel in einer *direkten Stufe*.

Der zweite Fall betrifft *diagonale Stufen* (Abbildung 4.10 rechts). Dieser Fall liegt vor, wenn das Subvoxel mit einem Objektsubvoxel eine Fläche teilt und mit einem weiteren Objektsubvoxel eine Kante.

Diagonale Stufen können folgendermaßen erkannt werden:

- Zuerst wird ermittelt, welche der sechs benachbarten Subvoxel gesetzt sind. Das zuerst gefundene gesetzte Nachbarsubvoxel bestimmt damit eine Ebene, die der weiteren Betrachtung als Grundlage dient. Die Richtung vom betrachteten Subvoxel zu dem ermittelten Nachbarsubvoxel bildet die Normale dieser Ebene, die durch das betrachtete Subvoxel verläuft (Abbildung 4.11 links).

- Nun werden die vier Nachbarsubvoxel, die in dieser Ebene liegen und nur eine Kante mit dem betrachteten Subvoxel gemeinsam haben (*diagonale Nachbarn*), betrachtet. Ist eines von ihnen gesetzt, liegt das betrachtete Subvoxel in einer diagonalen Stufe und wird zu $\bar{\mathcal{S}}$ hinzugefügt.

Der Algorithmus kann einfach abgewandelt werden, um auch direkte Stufen zu erkennen. Dazu werden in der ermittelten Ebene nicht nur die diagonalen, sondern alle acht Nachbarn betrachtet (Abbildung 4.12).

Mit Hilfe dieses Algorithmus kann zum einen die Treppenbildung reduziert werden. Zum anderen werden diagonale Strukturen, die nur ein Voxel dick sind, an den Stellen verstärkt, an denen die Voxel nur an einer Kante benachbart sind und die beschriebene Struktur somit eine Stärke von 0 hat (Abbildung 4.13). Das beschriebene Volumen nimmt dabei nur minimal zu. Diese Zunahme ist unproblematisch, da nicht anzunehmen ist, dass das beschriebene Objekt an dieser Stelle tatsächlich eine Dicke von 0 hat.

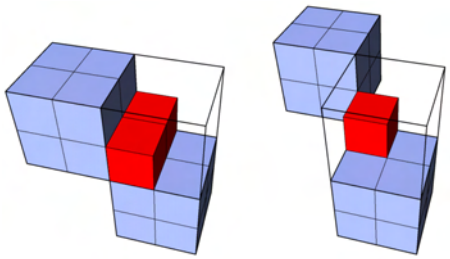


Abbildung 4.10: Fälle, in denen ein Subvoxel gesetzt werden soll: Subvoxel in einer *direkten Stufe* (*links*) und in einer *diagonalen Stufe* (*rechts*). Das betrachtete Voxel ist als Umriss dargestellt; das neu gesetzte Subvoxel ist rot.

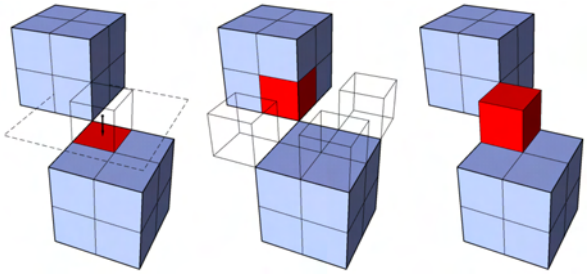


Abbildung 4.11: Setzen eines Subvoxels in einer diagonalen Stufe: Zuerst wird ein gesetztes Nachbarsubvoxel (*rot*) in der 3D-6-Nachbarschaft des betrachteten Subvoxels (*nur umrandet*) ermittelt (*links*). Der Vektor zu diesem Subvoxel bestimmt mit dem Zentrum des betrachteten Subvoxels eine Ebene (*gestrichelt dargestellt*). Die vier diagonalen Nachbarn in dieser Ebene werden nun betrachtet (*mitte*). Ist einer von ihnen gesetzt (*rot*), so wird das betrachtete Subvoxel gesetzt (*rechts*). Das neu gesetzte Subvoxel ist rot gefärbt.

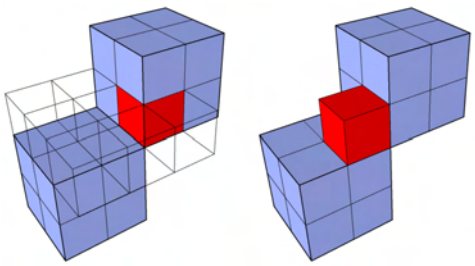


Abbildung 4.12: Betrachtet man alle Nachbarn in der Ebene (*links*), kann der Algorithmus auch genutzt werden, um Subvoxel in direkten Stufen hinzuzufügen (*rechts*).

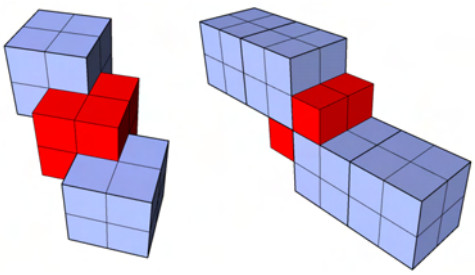


Abbildung 4.13: Beispiele für das Hinzufügen zusätzlicher Subvoxel. Treppen werden hierbei reduziert; dünne Strukturen werden verstärkt. Die ursprünglich vorhandenen Voxel sind blau dargestellt, die neu hinzugefügten Subvoxel rot.

4.2.5 Extraktion der Punkte und Normalenvektoren

Grundlage für die Positionierung der Punkte sind der klassifizierte Datensatz $\mathcal{S}^*$ sowie $\bar{\mathcal{S}}$, das höher aufgelöste Segmentierungsergebnis, das um zusätzliche Subvoxel erweitert wurde. Für die Berechnung der Normalen werden das originale Segmentierungsergebnis $\mathcal{S}$ und wiederum $\bar{\mathcal{S}}$ benötigt.

Die Platzierung der Punkte sowie die Berechnung der Normalenvektoren läuft auf Voxel- und Subvoxelebene auf ähnliche Weise ab. Grundlage für die Platzierung sind die äußeren Randvoxel. Für die Platzierung eines Punktes sowie die Berechnung der zugehörigen Normale wird die Position eines äußeren Randvoxels sowie dessen Nachbarschaft benötigt. Auf Subvoxelebene werden die Subvoxel eines äußeren Randvoxels als Grundlage genutzt. Basierend auf der zuvor getroffenen Klassifizierung wird lokal entschieden, welche Ebene benutzt wird.

Zur Platzierung der Punkte sowie der Berechnung der zugehörigen Normalen werden folgende Daten benötigt:

- Position des (Sub-)Voxels in (Sub-)Voxelkoordinaten: Vektor p_{vox}
- die *3D-26-Nachbarschaft* des betrachteten (Sub-)Voxels auf (Sub-)Voxelebene.

Im Folgenden wird zuerst betrachtet, wie diese Daten auf Voxel- sowie Subvoxelebene ermittelt werden. Anschließend wird gezeigt, wie die Position eines oder mehrerer Punkte sowie die zugehörigen Normalenvektoren basierend auf diesen Daten berechnet werden.

Ermitteln der Daten auf Voxelebene

Die Beschreibung solcher Objektteile, die nicht zu dünnen Strukturen gehören, wird durch Punkte realisiert, die auf Basis des originalen Voxelgitters platziert werden. Die zugehörigen äußeren Randvoxel wurden bereits ermittelt und haben in $\mathcal{S}^*$ den Wert 3. Für jedes dieser Voxel werden nun ein oder mehrere Punkte sowie die zugehörigen Normalen berechnet. p_{vox} entspricht hierbei den Koordinaten des Voxels in $\mathcal{S}$. Die 3D-26-Nachbarschaft wird aus $\mathcal{S}$ ermittelt.

Ermitteln der Daten auf Subvoxelebene

Die äußeren Randvoxel, die an dünne Strukturen grenzen, haben in $\mathcal{S}^*$ den Wert 4. Für jedes dieser Voxel werden nun die acht Subvoxel in $\bar{\mathcal{S}}$ betrachtet. Zuerst muss ermittelt werden, welche dieser Subvoxel die Rolle eines *äußeren Randsubvoxels* einnehmen. Ein äußeres Randsubvoxel spielt auf Subvoxelebene die selbe Rolle, wie ein äußeres Randvoxel auf Voxelebene, es ist also zu einem Objektsubvoxel benachbart. Für jedes dieser Subvoxel wird nun getestet, ob sich in ihrer 3D-6-Nachbarschaft in $\bar{\mathcal{S}}$ ein Objektsubvoxel befindet. Hierbei sei darauf hingewiesen, dass $\bar{\mathcal{S}}$ bereits die hinzugefügten Subvoxel enthält. Befindet sich in der Nachbarschaft ein Objektsubvoxel, wird das betrachtete Subvoxel zur Platzierung eines oder mehrerer Punkte benutzt. p_{vox} ist hierbei die Koordinate des Subvoxels in $\bar{\mathcal{S}}$. Die 3D-26-Nachbarschaft wird aus $\bar{\mathcal{S}}$ ermittelt.

Ermittlung der Punktpositionen

Bis zu diesem Punkt wurden Voxel und Subvoxel ermittelt, die an die Oberfläche grenzen. Es sollen jedoch Punkte extrahiert werden. Im nächsten Schritt muss also entschieden werden, wo innerhalb des Volumens des (Sub-)Voxels ein oder mehrere Punkte platziert werden sollen. Hierbei wird die Position eines (Sub-)Voxels sowie dessen Nachbarschaft genutzt, um ein oder mehrere Punkte auf dem Teil der Oberfläche, der diesen (Sub)-Voxel schneidet, zu positionieren. Im Folgenden wird verallgemeinernd für Voxel und Subvoxel der Begriff Voxel genutzt, da die Behandlung bis auf die spätere Überführung in Weltkoordinaten identisch ist. Der jeweils betrachtete Punkt wird mit p bezeichnet.

Für die Bestimmung der Position wird lediglich die 3D-6-Nachbarschaft des betrachteten Voxels benötigt. Diese ist vollständig in der 3D-26-Nachbarschaft enthalten, und kann somit leicht ermittelt werden. Um die 3D-6-Nachbarschaft auswerten zu können, wird die Anzahl der Objektvoxel in dieser Nachbarschaft $num_{nb}(x, y, z)$ (siehe Abschnitt 4.2.4) ermittelt. Desweiteren wird die Summe der Richtungsvektoren vom Zentrum des Voxels zu allen benachbarten Objektvoxeln v_{nb} berechnet (siehe Formel 4.8).

Ist in der Nachbarschaft nur ein Objektvoxel enthalten ($num_{nb} = 1$), so grenzt das Voxel an eine ebene oder konvexe Struktur. Der Punkt wird in diesem Fall im Mittelpunkt der Grenzfläche zu diesem Nachbarvoxel platziert (Abbildung 4.14). Somit entspricht v_{nb} der Richtung zu diesem Punkt und kann zur Ermittlung der Position des Punktes benutzt werden:

$$p = p_{vox} + 0.5 * v_{nb} \tag{4.13}$$

Auf dieselbe Weise kann auch ein Voxel mit fünf benachbarten Objektvoxeln behandelt werden. Ein solches Voxel stellt eine Vertiefung dar. Der Punkt soll in diesem Fall auf dem „Boden“ der Vertiefung platziert werden (siehe Abbildung 4.15). v_{nb} zeigt in Richtung des Voxels, das den Boden darstellt, weil die restlichen vier Voxel in einer Ebene liegen, und sich die Vektoren vom betrachteten Voxel zu ihren Zentren gegenseitig aufheben (siehe dazu auch Abschnitt 4.2.4).

Besitzt das Voxel zwei sich gegenüberliegende Objektvoxel als Nachbarn, so befindet es sich zwischen zwei Strukturen. Dieser Fall liegt vor, wenn $num_{nb} = 2$ und $v_{nb} = (0, 0, 0)$ ist. Damit an dem betrachteten Randvoxel beide angrenzenden Strukturen beschrieben werden, müssen für dieses Voxel zwei Punkte platziert werden. Die Punkte werden hierbei wiederum in den Zentren der Grenzflächen platziert (Abbildung 4.16 links).

Hat das Voxel vier Objektvoxel als Nachbarn, die in einer Ebene liegen ($num_{nb} = 4$ und $v_{nb} = (0, 0, 0)$, siehe auch Abschnitt 4.2.4), so beschreibt das Voxel ein Loch. In diesem Fall werden vier Punkte jeweils in den Mittelpunkten der Grenzflächen platziert (Abbildung 4.16 rechts).

Voxel, in deren 3D-6-Nachbarschaft sechs Objektvoxel enthalten sind, werden ausgeschlossen, da ein solch winziger Hohlraum nicht dargestellt werden soll. Sollte eine Darstellung erwünscht sein, würden analog zu den beiden vorherigen Fällen sechs Punkte auf den Grenzflächen platziert werden.

In allen anderen Fällen sind die Voxel in der Nachbarschaft so angeordnet, dass sich das betrachtete Voxel in einer Art Treppenstufe befindet. In diesem Fall wird ein einzelner Punkt im Zentrum des Voxels platziert (Abbildung 4.17).

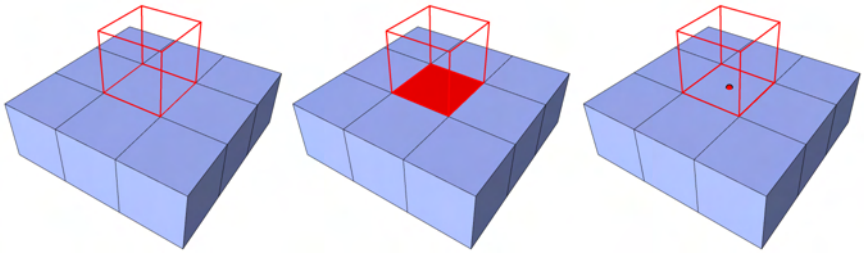


Abbildung 4.14: Platzierung eines Punktes, wenn das betrachtete Voxel (*rot umrandet*) nur ein Objektvoxel in seiner 3D-6-Nachbarschaft hat: die Grenzfläche (*rot*) zu diesem Objektvoxel wird bestimmt (*mitte*). Ihr Zentrum wird als Position für den Punkt gewählt (*links*).

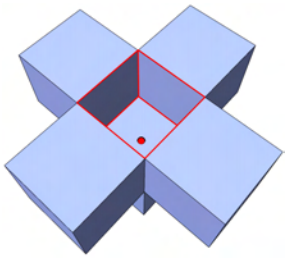


Abbildung 4.15: Platzierung eines Punktes, wenn das betrachtete Voxel (*rot umrandet*) fünf Objektvoxel in seiner 3D-6-Nachbarschaft hat: Der Punkt wird mittig auf dem Boden der Vertiefung platziert.

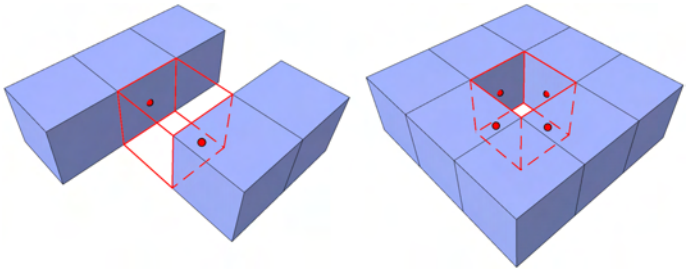


Abbildung 4.16: Platzierung mehrerer Punkte für ein betrachtetes Voxel: Die Punkte werden jeweils in den Zentren der Grenzflächen zu den Nachbarvoxeln platziert.

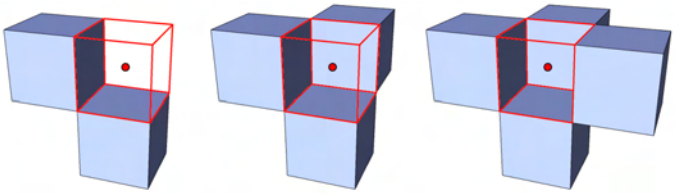


Abbildung 4.17: Platzierung eines Punktes, wenn sich das betrachtete Voxel in einer treppenförmigen Umgebung befindet: Der Punkt wird im Zentrum des Voxels platziert.

Auf diese Weise kann auch für einen Zweig, der nur eine Stärke von einem Voxel hat, ein Volumen beschrieben werden (siehe Abbildung 4.18).

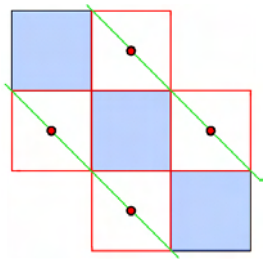


Abbildung 4.18: Bei einer treppenförmigen Anordnung der benachbarten Objektvoxel (*blau*) wird der Punkt im Zentrum des betrachteten Voxels (*rot umrandet*) platziert. Die durch die Punkte verlaufende Oberfläche (*grün*) beschreibt somit ein Volumen.

Alternativ könnten in diesem Fall die Punkte ebenfalls auf den Grenzflächen zu den benachbarten Objektvoxeln platziert werden. Auf diese Weise würde das Segmentierungsergebnis äußerst genau beschrieben. Allerdings würde dies zu einer wenig organischen Rekonstruktion führen (siehe Abbildung 4.19)

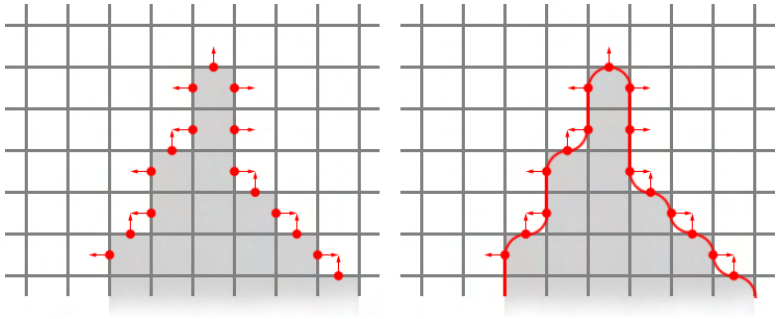


Abbildung 4.19: Alternative Punktplatzierung: Eine Platzierung der Punkte direkt auf den Außenflächen der Randvoxel führt zu einer wenig organsichen, treppenförmigen Rekonstruktion.

Darüber hinaus könnten dünne, diagonale Zweige so nicht zuverlässig beschrieben werden. So kann es zum einen zu Zweideutigkeiten bei der Interpretation durch die MPU Implicits kommen (Abbildung 4.20 rechts). Zum anderen würden die Zweige selbst bei einer korrekten Rekonstruktion unter starken treppenartigen Artefakten leiden (Abbildung 4.20 links).

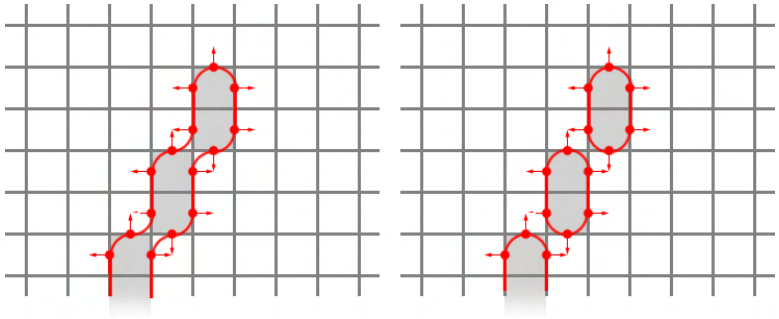


Abbildung 4.20: Alternative Punktplatzierung: Die Rekonstruktion würde unter starken treppenförmigen Artefakten leiden (*links*) oder gar zu Abtrennungen führen (*rechts*).

Berechnung der Normalenvektoren

Für jeden betrachteten Voxel wurden ein oder mehrere Punkte ermittelt. Für jeden Punkt p wird nun der zugehörige Normalenvektor n ermittelt. Hierbei müssen zwei Fälle beachtet werden:

- **Fall 1:** Für einen Voxel wurde nur ein Punkt platziert.
- **Fall 2:** Für einen Voxel wurden mehrere Punkte platziert.

Fall 1: Wurde für einen Voxel nur ein Punkt platziert, wird die zugehörige Normale auf Basis des zu dem Voxel gehörenden *Grauwertgradienten* berechnet. Der Grauwertgradient gibt die Richtung der stärksten Änderung der Grauwerte in der Nachbarschaft eines Voxels an [HB86]. Somit verläuft diese Richtung orthogonal zu Schichten von Voxeln, die den selben Wert haben (siehe Abbildung 4.21). Da angenommen wird, dass die Oberfläche des beschriebenen Objektes durch Voxel mit ähnlichen Datenwerten beschrieben wird, ist diese Richtung auch orthogonal zu der Oberfläche und kann somit als eine Schätzung für den Normalenvektor genutzt werden.

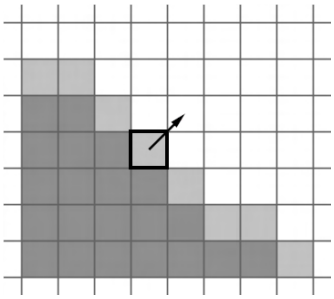


Abbildung 4.21: Der Grauwertgradient eines Voxels zeigt in die Richtung, in der sich die Werte in der Nachbarschaft am stärksten ändern.

Der Grauwertgradient $\vec{G}(x, y, z)$ besteht aus den partiellen Ableitungen der Intensitätswerte $f(x, y, z)$ in Richtung der drei Dimensionen:

$$\vec{G}(x, y, z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right) \tag{4.14}$$

Die partiellen Ableitungen können durch Differenzenquotienten approximiert werden:

$$\begin{aligned} \vec{G}_x(x, y, z) &= \frac{\partial f(x, y, z)}{\partial x} = \frac{f(x+1, y, z) - f(x-1, y, z)}{2} \\ \vec{G}_y(x, y, z) &= \frac{\partial f(x, y, z)}{\partial y} = \frac{f(x, y+1, z) - f(x, y-1, z)}{2} \\ \vec{G}_z(x, y, z) &= \frac{\partial f(x, y, z)}{\partial z} = \frac{f(x, y, z+1) - f(x, y, z-1)}{2} \end{aligned} \tag{4.15}$$

Da die Bestimmung der Normalen jedoch auf Basis binärer Daten erfolgen soll, würde diese Approximation für die partiellen Ableitungen jeweils nur drei verschiedene Werte liefern (-1, 0 und 1). Somit könnten nur 27 verschiedene Normalen berechnet werden. Die Berechnung der Normalenvektoren wäre somit sehr ungenau. Für eine artefaktfreie Rekonstruktion mit MPU Implicits sind akkurate Normalenvektoren jedoch äußerst wichtig [Bra05] (siehe Abbildung 4.22). In [Bra05] wird das Segmentierungsergebnis deswegen vor

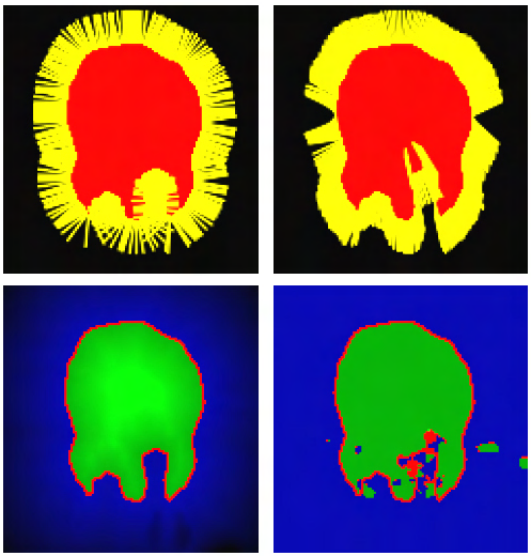


Abbildung 4.22: Auswirkung ungenauer Normalen auf die Rekonstruktion mit MPU Implicits verdeutlicht an einem Schnitt durch das Skalarfeld: Die Nutzung ungenauer Normalenvektoren (*gelb*) kann bei der Oberflächengenerierung (*unten*) zur Entstehung von Artefakten führen (*unten rechts*). *Quelle:* [Bra05]

der Gradientenberechnung mit Hilfe eines *Gauss*-Filters geglättet. Anschließend werden die Gradienten mit Hilfe eines *Sobel*-Filters ermittelt. Bei dieser Form der Gradientenbestimmung wird eine größere Nachbarschaft betrachtet. So werden zur Berechnung von $\vec{G}_x(x, y, z)$ alle Voxel in der 3D-26-Nachbarschaft hinzugezogen, deren x-Wert um 1 vom x-Wert des betrachteten Voxels abweicht. Diese insgesamt 18 Voxel müssen nun unterschiedlich gewichtet werden. Nach [Pom04] werden die Gewichte folgendermaßen verteilt: Die beiden Voxel, die über eine Fläche zu dem betrachteten Voxel benachbart sind, erhalten das größte Gewicht. Dieses Gewicht wird mit w_6 bezeichnet, da diese Voxel in der 3D-6-Nachbarschaft enthalten sind. Die Verteilung wird in Abbildung 4.23 verdeutlicht.

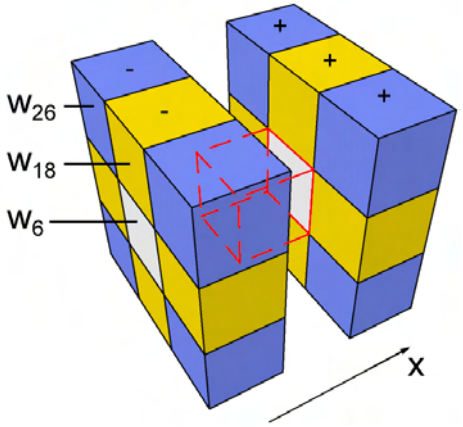


Abbildung 4.23: Verteilung der Gewichte bei der Berechnung des partiellen Grauwertgradienten des betrachteten Voxels (*rot umrandet*) in x-Richtung. Voxel mit kleineren x-Werten erhalten negative Gewichte.

Die Voxel mit den w_6-Gewichten sind weiß dargestellt. Das nächstgrößere Gewicht erhalten jene Voxel, die eine Kante mit dem betrachteten Voxel gemeinsam haben. Da es sich hierbei um die Voxel handelt, die in der 3D-18-Nachbarschaft zusätzlich zu den 3D-6-Voxeln enthalten sind, wird das entsprechende Gewicht w_{18} genannt (in Abbildung 4.23 gelb dargestellt). Das geringste Gewicht erhalten die Voxel, die lediglich einen Eckpunkt mit dem betrachteten Voxel teilen (in Abbildung 4.23 blau dargestellt). Diese Voxel kommen in der 3D-26-Nachbarschaft hinzu, weswegen das entsprechende Gewicht w_{26} heißt.

Die so verteilten Gewichte werden nun für die Voxel, deren x-Wert um 1 kleiner ist als der des betrachteten Voxels, negiert. Die Voxel, deren x-Wert größer ist als der des betrachteten Voxels, erhalten positive Gewichte.

Die Berechnung der partiellen Ableitung erfolgt nach [Tie99] folgendermaßen:

$$\vec{G}_x(x,y,z) = \sum_{i=-1}^{1}\sum_{j=-1}^{1}\sum_{k=-1}^{1} W_x(i,j,k) f(x+i, y+j, z+k) \tag{4.16}$$

W_x gibt hierbei die Gewichte für die partielle Ableitung in x-Richtung zurück. Die Verteilung der Gewichte wird für die beiden anderen partiellen Ableitungen analog vorgenommen.

Für die Belegung der Wichtungsfaktoren w_6, w_{18} und w_{26} wurden in der Literatur verschiedene Werte vorgeschlagen. Die Werte des *Zucker-Hummel*-Operator basieren auf den euklidischen Abständen vom betrachteten Voxel zu den Nachbarn [ZH81]. Der Operator approximiert damit eine Kugelform, betrachtet aber nicht das Volumen der Voxel. Tiede [Tie99] erweitert dieses Prinzip und berechnet die Gewichte auf Basis des Schnittvolumens der Nachbarvoxel mit einer in dem betrachteten Voxel zentrierten Kugel. Die 3D-Erweiterung des *Sobel*-Operators, die auch in [Bra05] genutzt wird, wählt die Gewichte hingegen willkürlich. Die Gewichte der Operatoren sind in Tabelle 4.1 zusammengefasst. Sie wurden so normiert, dass w_6 1 entspricht.

Tabelle 4.1: Verschiedene Belegungen der Gewichte für eine auf der 3D-26-Nachbarschaft basierende Gradientenberechnung.

Gewicht	Zucker-Hummel	Tiede	Sobel-3D
w_6	1	1	1
w_{18}	0.707	0.54	0.5
w_{26}	0.577	0.183	0.25

Die vorgeschlagenen Gewichte unterscheiden sich stark. Die resultierenden Normalen weichen jedoch nur gering voneinander ab [Pom04]. Die Herleitung der Gewichte nach Tiede ist jedoch am plausibelsten und wird deswegen für die Bestimmung der Normalenvektoren benutzt. Auf eine Glättung des Segmentierungsergebnisses vor der Gradientenberechnung, wie sie in [Bra05] beschrieben wurde, wird verzichtet, da die Gradientenberechnung mit dem Tiede-Operator durch Betrachtung der 3D-26-Nachbarschaft bereits eine Glättung enthält. Eine zusätzliche Glättung würde die Berechnungsdauer zusätzlich erhöhen.

Wurden die drei partiellen Ableitungen berechnet, dienen sie als Komponenten des Normalenvektors:

$$n = (\vec{G}_x(x,y,z), \vec{G}_y(x,y,z), \vec{G}_z(x,y,z)) \tag{4.17}$$

Normalenvektoren sollten immer die Länge 1 haben. Der Vektor wird jedoch erst später normalisiert, nachdem er in Weltkoordinaten überführt wurde (siehe Abschnitt 4.2.6).

Fall 2: Wurden für ein äußeres Randvoxel mehrere Punkte platziert, können die Normalen nicht auf Basis des Gradienten dieses Voxels berechnet werden, da diese Punkte so den gleichen Normalenvektor hätten. Für die Bestimmung der Normalen könnten alternativ die Gradienten der nächstgelegnen Nachbarvoxel genutzt werden. Diese Möglichkeit wurde jedoch verworfen, da diese Voxel zu Strukturen gehören könnten, die nur ein Voxel dick sind. Für solche Voxel kann kein verlässlicher Gradient bestimmt werden.

Statt dessen werden die Normalenvektoren der Grenzflächen genutzt, in deren Mittelpunkt die Punkte platziert wurden (Abbildung 4.24). Dieser Vektor entspricht der Richtung von dem zu der Grenzfläche gehörenden Nachbarvoxel zum betrachteten Voxel.

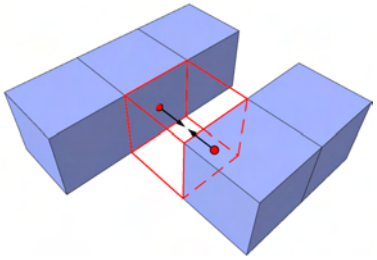


Abbildung 4.24: Wenn zu einem Voxel (*umrandet*) mehrere Punkte erzeugt wurden, werden die Normalenvektoren der Grenzflächen als Normalen für die Punkte genutzt.

4.2.6 Überführung in Weltkoordinaten

Bis zu diesem Punkt wurden alle Schritte im Koordinatensystem der Voxeldatensätze durchgeführt. Zudem wurden zwei verschiedene Voxelkoordinatensysteme genutzt. Die Oberfläche soll jedoch im Weltkoordinatensystem erzeugt werden. Um die Punkte und Normalen in das Weltkoordinatensystem zu überführen, werden zuerst die Positionen der Punkte, die auf Subvoxelebene erzeugt wurden, in Voxelkoordinaten überführt. Dazu wird der Positionsvektor mit 0.5 multipliziert, da der Subvoxeldatensatz eine doppelt so große Auflösung hat wie der Voxeldatensatz. Zusätzlich wird eine Verschiebung ausgeführt, da die Zentren der Subvoxel in jeder Dimension um 0.25 zu den Zentren der Voxel verschoben sind:

$$p_v = 0.5 * p - \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \end{bmatrix} \tag{4.18}$$

Im Folgenden werden Punkte im Voxelkoordinatensystem mit p_v benannt. Wurde ein Punkt auf Voxelebene erzeugt, ist p_v gleich p. Nun gilt es, jeden Punkt p_v in das Weltkoordinatensystem zu überführen. Dafür wird die homogene Repräsentation von p_v benötigt. Die vierte Komponente des Vektors ist dabei 1. Die Matrix M_w dient der Überführung in das Weltkoordinatensystem und ist ein Ergebnis des bildgebenden Verfahrens. Die Position p_w eines Punktes im Weltkoordinatensystem ergibt sich somit folgendermaßen:

$$p_w = M_w p_v \tag{4.19}$$

Als nächstes werden die Normalenvektoren in das Weltkoordinatensystem transformiert. Die Transformation von Voxelkoordinaten in Weltkoordinaten ist oft anisotrop. Das heißt, dass die Skalierung in den drei Dimensionen unterschiedlich stark ist. Die Normalenvektoren müssen dabei anders transformiert werden als die Positionen, da sie sonst nach der Transformation nicht mehr senkrecht zu der Oberfläche stehen würden (siehe Abbildung 4.25 mitte).

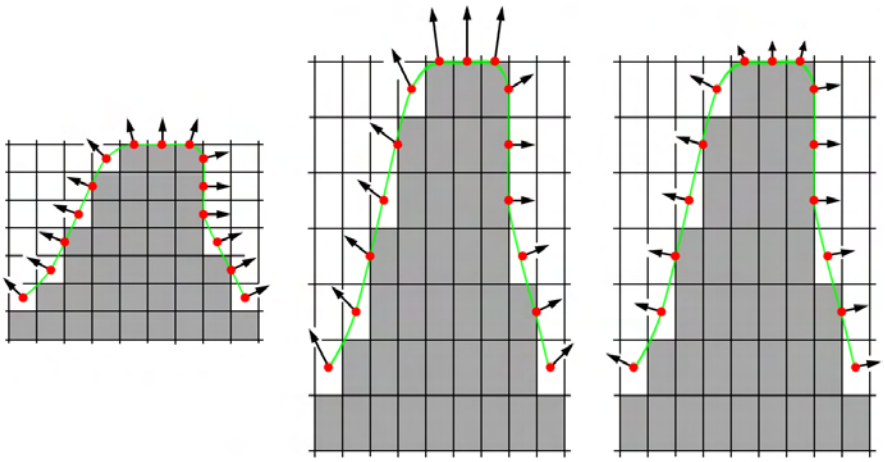


Abbildung 4.25: Überführung der Normalen in das Weltkoordinatensystem: Werden die Normalen auf die selbe Weise transformiert wie die Punkte, so stehen sie nicht mehr senkrecht zu der Oberfläche (*mitte*). Bei einer korrekten Transformation bleiben die Normalenvektoren senkrecht zu der Oberfläche (*rechts*).

Um die Normalenvektoren so zu transformieren, dass sie weiterhin senkrecht zu der Oberfläche stehen, müssen sie mit der Transponierten der Inversen von M_w multipliziert werden [Tur90]:

$$n_w = n_v M_w^{-1T} \tag{4.20}$$

n_v ist hierbei die zuvor ermittelte Normale in homogenen Koordinaten, wobei der vierte Eintrag des Vektors 0 sein muss. Anderenfalls würde der Vektor als Punkt interpretiert und zusätzlich einer Translation unterzogen.

Wurden die Normalen in das Weltkoordinatensystem transformiert, müssen sie noch normalisiert werden:

$$n'_w = \frac{n_w}{\mid n_w \mid} \tag{4.21}$$

Das Ergebnis liegt nun in der Menge $\mathcal{P}$ aller Punkte p_w sowie der Menge $\mathcal{N}$ aller zugehörigen Normalen n'_w im Weltkoordinatensystem vor.

4.3 Ermittlung geeigneter Parameter

Sowohl die Generierung der impliziten Funktion als auch deren Polygonalisierung können durch eine Vielzahl von Parametern beeinflusst werden. Eine Anpassung der Parameter durch eine große Anzahl von Versuchen wäre für den angestrebten Einsatz im klinischen Alltag jedoch hinderlich. Deswegen ist es wichtig, dass sinnvolle Belegungen für die Parameter automatisch ermittelt werden. Im Folgenden wird für die einzelnen Parameter ein Weg zur Herleitung sinnvoller Werte dargelegt und begründet. Die so errechneten Werte sind als Richtwerte zu verstehen, die in einzelnen Fällen eventuell vom Benutzer angepasst werden müssen.

4.3.1 Parameter für die Generierung der Impliziten Funktion

Die Parameter, die die Aufstellung der impliziten Funktion steuern, wurden in Abschnitt 3.3.2 erläutert. Eine Zusammenfassung der Parameter ist in Tabelle 3.2 gegeben. Die Auswirkungen, die verschiedene Belegungen dieser Parameter mit sich bringen, wurden in Abschnitt 3.4.1 untersucht.

Für die Startgröße des kugelförmigen Bereichs α, den Wachstumsfaktor λ sowie die minimale Punktanzahl N_{min} wurden von Ohtake et al. [OBA+03b] Standardwerte vorgeschlagen, die sich für einen großen Teil der von ihnen getesteten Datensätze als geeignet erwiesen. Für die Rekonstruktion einer Gefäßoberfläche auf Basis von Punktwolken, die mit dem in Abschnitt 4.2 beschriebenen Verfahren erzeugt wurden, erwiesen sich diese Werte jedoch als ungeeignet. Sie führen zur Rekonstruktion einer Oberfläche, die kleine Vertiefungen und Erhebungen aufweist (Abbildung 4.26 links). Die Erhöhung der Werte für diese drei Parameter begünstigt eine glattere Darstellung, da sie zur Verwendung von mehr Punkten für die lokalen Approximationen führt (siehe Abschnitt 3.4.1).

[Bra05] schlägt für N_{min} den Wert 100 vor. Da durch die in der vorliegenden Arbeit vorgeschlagene Überabtastung die Punktwolke teilweise dichter ist als bei [Bra05], wird für N_{min} ein noch höherer Wert genutzt. In zahlreichen eigenen Versuchen wurde eine Parameterbelegung ermittelt, die bei den meisten Punktwolken, die mit dem in Abschnitt 4.2 beschriebenen Verfahren erzeugt wurden, zur Rekonstruktion einer glatten, organischen Oberfläche führte. Diese Belegung ist in Tabelle 4.2 zusammengefasst und den von Ohtake vorgeschlagenen Werten gegenübergestellt. In Abbildung 4.26 sind zudem Ergebnisse für beide Parameterbelegungen zu sehen. Die höheren Werte führen zu einer organischeren Darstellung (Abbildung 4.26 rechts).

Tabelle 4.2: Ermittelte Parameterbelegungen für α, λ und N_{min} (*recht Spalte*) sowie die Werte aus [OBA+03b] (*mittlere Spalte*).

Prameter	Bedeutung	Standardbelegung nach [OBA+03b]	vorgeschlagener Wert
α	Startgröße des kugelförmigen Bereiches	0.75	0.8
λ	Wachstumsfaktor	0.1	0.2
N_{min}	Mindestanzahl der für eine lokale Approximation benötigten Punkte	15	200

Die Parameter ϵ_0 und *max_level* müssen für jeden Datensatz einzeln bestimmt werden, da sie in Abhängigkeit von der Ausdehnung des Datensatzes zu unterschiedlichen Ergebnissen führen.

ϵ_0 gibt den maximalen Fehler an. Der Wert bezieht sich auf die Diagonale der AABB der Punktwolke. Hat die Punktwolke eine Größe von $100 * 100 * 100$ Einheiten, so hat die Diagonale eine Länge von 173.205 Einheiten. Soll die maximale Abweichung der Oberfläche von den Punkten 0.5 Einheiten betragen, muss ϵ_0 gleich $0.5/173.205 = 0.00289$ sein. Für eine Oberfläche, die aus Volumendaten gewonnen wird, ist es nahe liegend, die maximale Abweichung in Relation zur Größe der Voxel zu setzen. Die Abweichung sollte kleiner als ein Voxel sein, aber auch groß genug, damit die generierte Oberfläche sich

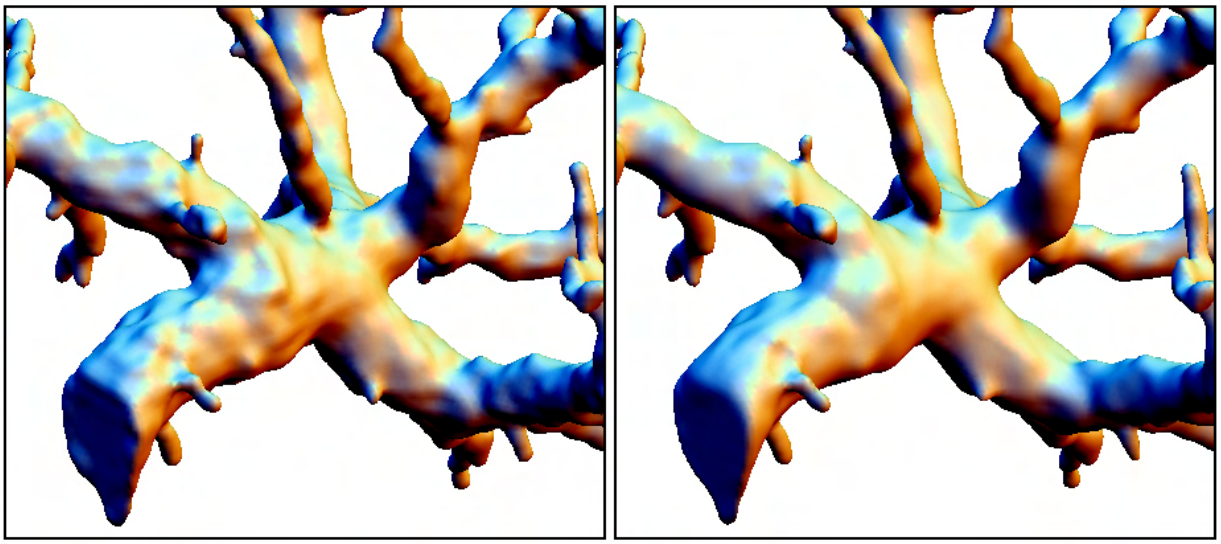


Abbildung 4.26: Die mit den vorgeschlagenen, höheren Werten für α, λ und N_{min} erzeugte Oberfläche (*rechts*) wirkt glatter und organischer als die mit der Standardbelegung generierte Oberfläche (*links*).

nicht zu genau an den treppenartig angeordneten Punkten orientiert. Als geeigneter Wert für die maximale Abweichung konnte in Versuchen die Hälfte der Diagonale eines Voxels ermittelt werden. Da ϵ_0 einen Anteil der Diagonale der Punktwolke darstellt, kann seine Berechnung in Voxel- oder Weltkoordinaten erfolgen; der Anteil ist ein beiden Fällen gleich. Die Berechnung erfolgt in Voxelkoordinaten, da die Diagonale eines Voxels in Voxelkoordinaten vorliegt und die Positionen der Punkte in beiden Koordinatensystemen. Die Berechnung von ϵ_0 erfolgt nun folgendermaßen:

$$\epsilon_0 = \frac{voxDiag}{2 * diag} = \frac{\sqrt{3}}{2 * diag} \tag{4.22}$$

Die Diagonale eines Voxels $voxDiag$ hat im Voxelkoordinatensystem die Länge $\sqrt{3}$. $diag$ ist die Diagonale der AABB der Punktwolke in Voxelkoordinaten.

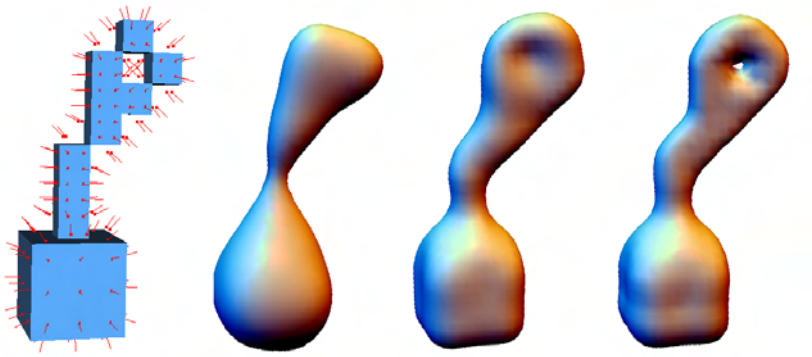
max_level beschränkt die maximale Unterteilungstiefe des Octrees. Eine zu tiefe Unterteilung kann zu der Entstehung von Artefakten führen (siehe Abschnitt 3.4.1). Eine zu geringe Unterteilung führt hingegen zu einer ungenauen Rekonstruktion.

Die maximale Unterteilung sollte so eingegrenzt werden, dass die Größe der Octree-Zellen in etwa der Größe der Voxel entspricht. Da die Anzahl der Octree-Zellen im Bereich der AABB der Punktwolke in allen drei Dimensionen gleich, und zudem eine Zweierpotenz ist, ist es nicht möglich, die Größe der Zellen genau an die Voxel anzupassen. max_level kann jedoch so berechnet werden, dass die Ausdehnung der Octree-Zellen in der Dimension, in der die Ausdehnung der Punktwolke am geringsten ist, etwa der Ausdehnung der Voxel entspricht:

$$max\_level = \lceil \log_2(\min(ext_{xv}, ext_{yv}, ext_{zv})) \rceil \tag{4.23}$$

ext_{xv}, ext_{yv} und ext_{zv} sind hierbei die Abmessungen der AABB der Punktwolke in Voxelkoordinaten. Bei Datensätzen, in denen die Abmessungen in den drei Dimensionen stark voneinander abweichen, kann diese Belegung jedoch dazu führen, dass die Octree-Unterteilung in der Richtung, in der die Ausdehnung des Datensatzes am größten ist,

nicht ausreicht und somit Details verloren gehen (Abbildung 4.27 mitte links). Der Wert in Formel 4.23 stellt also eine untere Schranke für *max_level* dar und wird im Folgenden mit max_level_{min} bezeichnet.



$max\_level = 2$ $max\_level = 3$ $max\_level = 4$

Abbildung 4.27: Auswirkung von *max_level* auf die Rekonstruktion einer Oberfläche: Die Punktwolke (*links*) hat eine Ausdehnung von $3 \times 10 \times 5$ Einheiten. Bei Nutzung von $max\_level_{min} = 2$ wird das Loch nicht rekonstruiert (*mitte links*). Wird hingegen $max\_level_{max} = 4$ genutzt, wird selbst dieses kleine Detail korrekt dargestellt (*rechts*).

Eine obere Schranke ergibt sich, wenn die Octree-Unterteilung so angepasst wird, dass die Zellen in der Dimension, in der die Ausdehnung der Punktwolke am größten ist, kleiner sind als die Voxel:

$$max\_level_{max} = \lceil \log_2(\max(ext_{xv}, ext_{yv}, ext_{zv})) \rceil \tag{4.24}$$

Die Nutzung von max_level_{max} führt zu einer besseren Rekonstruktion sehr feiner Details, kann jedoch zur Bildung von Artefakten führen. max_level_{min} hingegen kann zu einer unzureichenden Darstellung von sehr kleinen Details, wie etwa winzigen Löchern, führen. Als Startwert wird max_level_{min} empfohlen, da dieser Wert bei allen getesteten medizinischen Datensätzen zu guten Ergebnissen geführt hat. Bei Bedarf kann *max_level* aber erhöht werden, bis max_level_{max} erreicht ist. Der Datensatz aus Abbildung 4.27 stellt ein extremes Beispiel dar, das zum einen sehr stark voneinander abweichende Ausdehnungen in den drei Dimensionen besitzt, und zum anderen eine äußerst filigrane Struktur beschreibt.

4.3.2 Parameter für die Polygonaliserung

Die Polygonalisierung der impliziten Funktion erfolgt durch einen Bloomenthal-Polygonizer ([Blo94], siehe auch Abschnitt 3.2.2). Der Algorithmus kann über 2 Parameter eingestellt werden. Zum einen kann die Größe der für die Polygonalisierung benutzten Zellen angegeben werden. Zum anderen kann der Isowert eingestellt werden.

Die für die Polygonalisierung benutzten Zellen haben eine Würfelform. Ihre Größe kann über den Parameter *grid_size* eingestellt werden. Der Parameter gibt an, wie groß die Zellen in Bezug auf die größte Ausdehnung der Punktwolke in den drei Dimensionen sind. Er kann so gewählt werden, dass die resultierenden Zellen in der Größe mit der originalen Voxelausdehnung übereinstimmen:

$$grid\_size = \frac{1}{\max(ext_{xv}, ext_{yv}, ext_{zv})} \tag{4.25}$$

ext_{xv}, ext_{yv} und ext_{zv} sind hierbei die Abmessungen der AABB der Punktwolke in Voxelkoordinaten Für die Darstellung sehr dünner Zweige ist eine etwas geringere Zellgröße notwendig. In Versuchen hat sich eine Multiplikation des Wertes mit 0.7 als geeignet erwiesen:

$$grid\_size = \frac{0.7}{\max(ext_{xv}, ext_{yv}, ext_{zv})} \tag{4.26}$$

Der Isowert *iso*, der die durch die Punktwolke beschriebene Oberfläche repräsentiert, ist 0. Eine Modifikation dieses Wertes führt ein Wachstum ($iso > 0$) oder Schrumpfen ($iso < 0$) des beschriebenen Objekts herbei. Ohtake et al. haben in ihrer Arbeit nicht genauer untersucht, ob die generierten Offset-Oberflächen einen konstanten Abstand zu der Oberfläche mit dem Isowert 0 haben. In ihrer Arbeit haben sie jedoch Versuche durchgeführt, die vermuten lassen, dass dies der Fall ist. Die entsprechenden Ergebnisse sind in Abbildung 4.28 zu sehen.

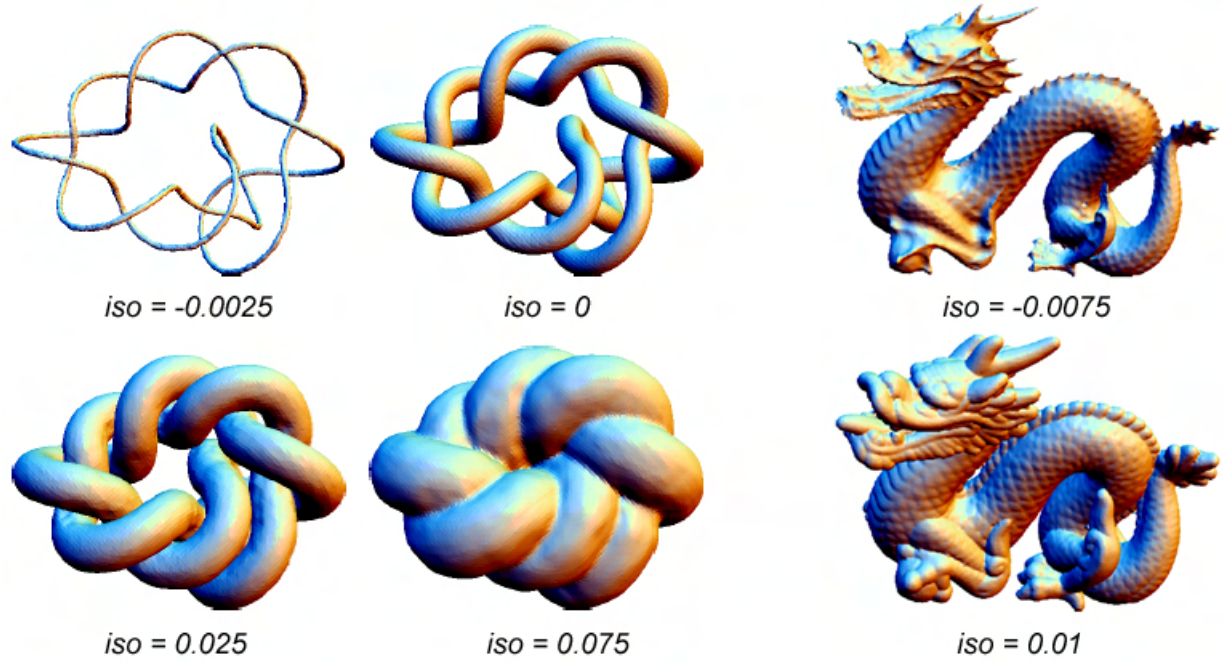


Abbildung 4.28: Auswirkung des Isowertes auf die Rekonstruktion der Oberfläche: positive Werte führen zu einem Wachstum, negative Werte zu einem Schrumpfen des Objektes. Die erzeugten Offset-Oberflächen wachsen dabei an allen Punkten um den gleichen Betrag. *Quelle:* [OBA+03b]

Ein Wachstum oder Schrumpfen der Gefäßoberfläche scheint im ersten Moment nicht erwünscht, da eine möglichst genaue Rekonstruktion im Vordergrund steht. Durch eine Erhöhung des Wertes und das damit verbundene Wachstum des Objektes kann jedoch die Abtrennung von Zweigen verhindert werden. Zu dieser Abtrennung kann es kommen, wenn die Verbindung zu einem Objektteil extrem dünn ist (Abbildung 4.29 mitte). Der Objektteil ist in der impliziten Funktion vorhanden, wird aber aufgrund des Surface Tracking-Ansatzes des Polygonizers (siehe Abschnitt 3.2.2) nicht mit polygonalisiert. Durch ein geringes Wachstum kann die Verbindung verstärkt werden, so dass der zuvor abgetrennte Objektteil von der Polygonalisierung erkannt wird (Abbildung 4.29 rechts).

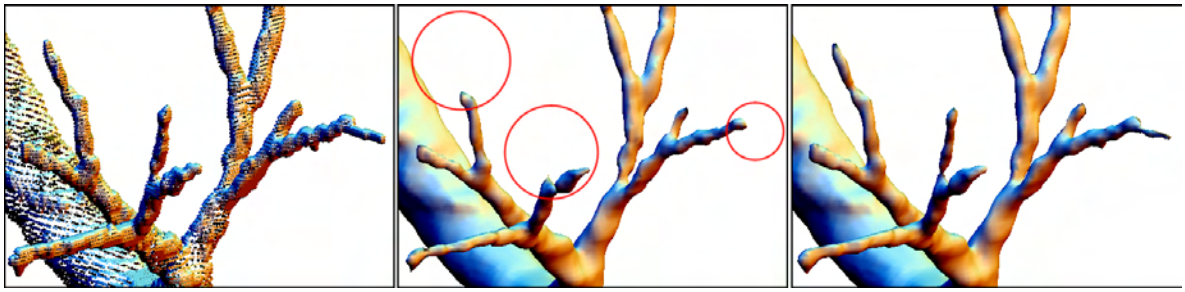


Abbildung 4.29: Auswirkung des Isowertes auf die Rekonstruktion der Gefäßoberfläche: Bei der Nutzung des Isowertes 0 (*mitte*) werden einige Objektteile abgetrennt (*durch Kreise hervorgehoben*), die in der Punktwolke (*links*) enthalten sind. Eine geringe Erhöhung des Isowertes führt zur Erhaltung dieser Objektteile (*rechts*).

Ein geeigneter Isowert sollte jedoch nicht zu groß sein, damit die Genauigkeit nicht zu stark verringert wird. Der Wert muss wiederum von den Ausmaßen der Punktwolke abhängig gemacht werden und sollte bei unterschiedlichen Datensätzen zu einem ähnlichen Wachstum führen. Um dies zu gewährleisten, wird der Isowert wiederum von der Diagonale der AABB der Punktwolke abhängig gemacht. Eine Schätzung für den Isowert kann nun leicht auf Basis des bereits errechneten maximalen Fehlers ϵ_0 erfolgen:

$$iso = \epsilon_0 * \beta \tag{4.27}$$

Als geeigneter Wert für β hat sich 0.1 erwiesen (Abbildung 4.29 rechts).

4.4 Registrierung

Unter *Registrierung* ist im Kontext dieser Arbeit die Zuordnung der erzeugten Polygone zu Teilen eines Gefäßmodells zu verstehen. Die Umsetzung eines entsprechenden Algorithmus war auf Grund des begrenzten Zeitraumes nicht möglich. Dennoch soll im Folgenden kurz eine mögliche Strategie für einen solchen Algorithmus dargelegt werden.

Erweitertes Gefäßmodell

Grundlage für das Verfahren ist einen Erweiterung des in [SSPP00], [HPSP01] und [SPSP02] vorgestellten Gefäßmodells. Das Modell liegt in Form der Datenstruktur *MLGraph* vor, die Bestandteil der Visualisierungsplattform MeVisLab[1] ist. Das Modell unterscheidet sich von dem etablierten Modell ([SSPP00], [HPSP01] und [SPSP02]) darin, dass für jeden Skelettvoxel zusätzlich eine Liste der Voxel enthalten ist, die den lokalen Gefäßquerschnitt repräsentieren. Zu jedem Skeletvoxel ist also eine zum Gefäßverlauf orthogonale Schicht von Voxeln gegeben (Abbildung 4.30 links). Die Summe all dieser Schichten ergibt das Segmentierungsergebnis (Abbildung 4.30 rechts).

MLGraph ist hierarchisch aufgebaut. Zu einem Baum gehört eine Menge von Kanten. Diese Kanten repräsentieren Teile eines Gefäßes zwischen Wurzel und Verzweigung, zwischen zwei Verzweigungen oder zwischen einer Verzweigung und dem Ende eines Gefäßes.

[1] MeVisLab wird am Forschungsinstitut MeVis – Center for Medical Diagnostic Systems and Visualization in Bremen entwickelt; www.mevis.de, letzter Stand: 06.07.2006

Zu jeder der Kanten gehört die Menge der Skelettvoxel, die den Verlauf der Kante repräsentieren. Die Skelettvoxel enthalten wiederum eine Liste der Voxel, die in der zugehörigen Schicht liegen.

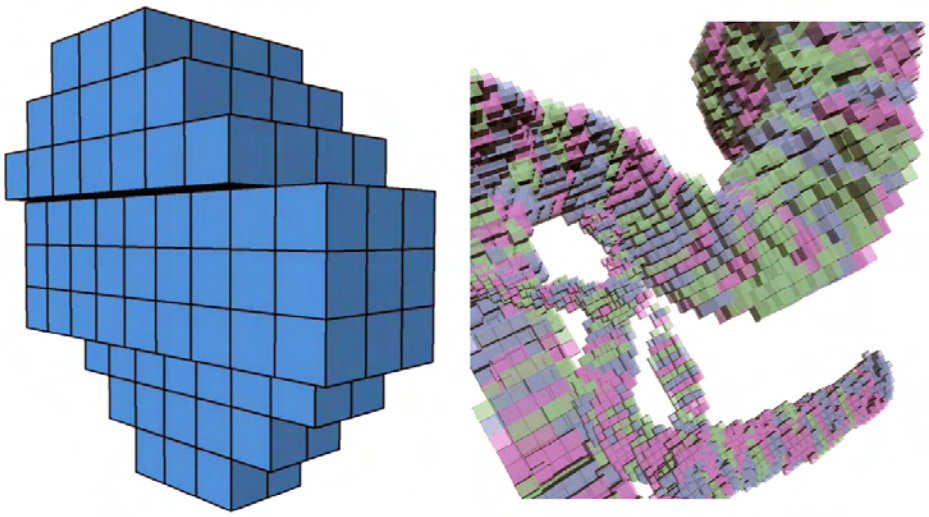


Abbildung 4.30: Repräsentation des Segmentierungsergebnisses durch die Datenstruktur MLGraph: Zu jedem Skelettvoxel ist eine Schicht von Voxeln gegeben (*links*). Die Menge all dieser Voxel ergibt das Segmentierungsergebnis (*rechts*). Zur Verdeutlichung sind die Schichten benachbarter Skelettvoxel in unterschiedlichen Farben dargestellt. *Quelle:* [Sch05]

Zuordnung der Punkte zu den Kanten des Modells

Durch den hierarchischen Aufbau der Datenstruktur MLGraph kann leicht die Menge aller Voxel, die zu einer Kante gehören, ermittelt werden. Wurde dies für alle Kanten getan, kann ein Volumendatensatz erzeugt werden, dessen Datenwerte die Zuordnung der Voxel zu den einzelnen Kanten kodieren. Hintergrundvoxel erhalten den Wert -1. Im nächsten Schritt muss nun für jeden Punkt des Polygonalisierungsergebnisses ermittelt werden, zu welchem Voxel er gehört. Dazu werden die Punkte in das Voxelkoordinatensystem transformiert. Ist der Wert des zugehörigen Voxels größer als -1, liegt der Punkt direkt in einem zu einer Kante gehörenden Voxel und kann somit dieser Kante zugeordnet werden. Ist der Wert hingegen -1, so liegt der Punkt außerhalb des Segmentierungsergbenisses. In diesem Fall kann ein zugehöriges Voxel des Segmentierungsergebnisses ermittelt werden, indem entlang der negativen Normale nach Voxeln gesucht wird, deren Wert größer als -1 ist. Hierfür eignen sich *Voxeltraversing*-Techniken wie z.B. das Verfahren von Cleary und Wyvill [CW88].

Zuordnung der Polygone zu den Kanten des Modells

Wurden alle Punkte den Kanten zugeordnet, muss dies auch für die Polygone getan werden. Für Polygone, deren Punkte zu der gleichen Kante gehören ist dies trivial. In diesem Fall wird das Polygon ebenfalls der Kante zugeordnet. Polygone, deren Punkte zu verschiedenen Kanten gehören, müssen einer der Kanten zugeordnet werden. Hierbei sollte

darauf geachtet werden, dass in einem solchen Grenzbereich für alle Polygone die gleiche Entscheidung getroffen wird, damit die Grenze zwischen den Polygonen zweier benachbarter Kanten nicht unnötig stark gezackt ist.

Das Ergebnis dieses Verfahrens ist eine Zuordnung der Polygone zu den Kanten des Gefäßbaumes. Das entsprechende Polygonnetz kann nun so aufgeteilt werden, dass zu jeder Kante ein Polygonnetz gehört. Diese Repräsentation des Gefäßbaumes würde sich nun für explorative Aufgaben, wie zum Beispiel das selektive Einfärben oder Ausblenden von Zweigen, eignen.

4.5 Zusammenfassung

In diesem Kapitel wurde eine Visualisierungspipeline vorgestellt, die die Erzeugung einer Gefäßoberfläche auf Basis des Segmentierungsergebnisses erlaubt. Ein wichtiger Schritt dieser Pipeline ist die Extraktion einer Punktwolke basierend auf dem Segmentierungsergebnis. Bei diesem Vorgang wird besondere Sorgfalt auf die Beschreibung dünner Strukturen verwandt. Dazu wird das Segmentierungsergebnis adaptiv überabgetastet, um die Erzeugung von genügend Punkten zur Beschreibung dünner Strukturen zu gewährleisten. Dabei werden zusätzliche Subvoxel hinzugefügt, um die Entstehung treppenartiger Artefakte bei der Oberflächenrekonstruktion zu verhindern. In Abbildung 4.31 wird das Verfahren für einen Beispieldatensatz verdeutlicht.

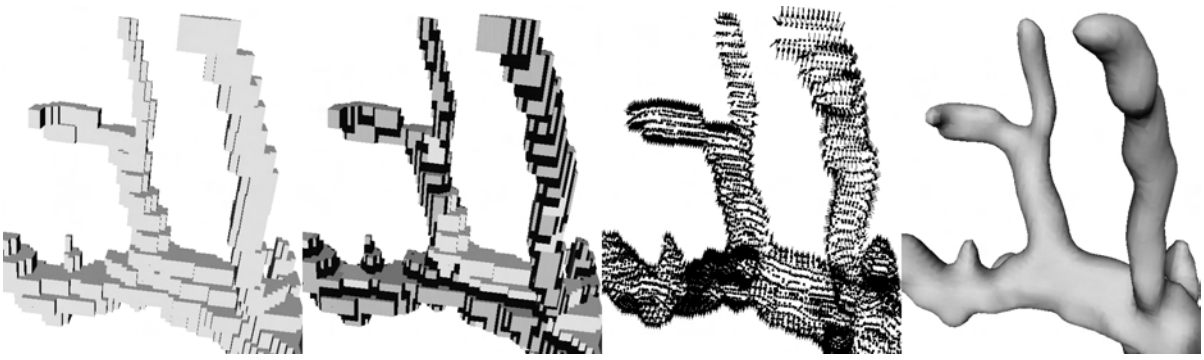


Abbildung 4.31: Ablauf der Oberflächenrekonstruktion verdeutlicht am Ausschnitt eines Beispieldatensatzes: Das Segmentierungsergebnis (*links*) wird überabgetatstet (*mitte rechts*). Dabei werden an dünnen Strukturen (*dunkelgrau*) Subvoxel (*schwarz*) hinzugefügt, um die Bildung treppenartiger Artefakte zu unterdrücken. Die erzeugte Punktwolke (*mitte rechts*) weist eine variierende Dichte auf, um dünne Strukturen mit ausreichend Punkten zu beschreiben. Die durch die MPU Implicits rekonstruierte Oberfläche (*rechts*) approximiert die Punktwolke.

Die Erzeugung der Oberfläche auf Basis der Punktwolke wird durch MPU Implicits realisiert. Die Anwendung dieses Verfahrens wird durch die in Abschnitt 4.3 beschriebene automatische Ermittlung geeigneter Parameterwerte stark vereinfacht.

Das Resultat dieses Verfahrens ist ein einzelnes Polygonnetz, das sich nur eingeschränkt für die Exploration von Gefäßbäumen eignet. Um diese Oberfläche für eine größere Anzahl von Anwendungen nutzbar zu machen, wurde ein Verfahren zur Aufteilung der Oberfläche in Teiloberflächen vorgeschlagen, die zu den Kanten des Gefäßbaumes gehören.

5 Implementierung

Das in Kapitel 4 beschriebene Verfahren zur Visualisierung von Gefäßsystemen mit MPU Implicits wurde im Rahmen dieser Arbeit umgesetzt. Lediglich die Registrierung konnte aufgrund des begrenzten Bearbeitungszeitraumes dieser Diplomarbeit nur konzeptionell erarbeitet werden. In diesem Kapitel werden wichtige Aspekte der Realisierung der vorgeschlagenen Gefäßvisualisierungspipeline erläutert. Im ersten Abschnitt werden die für die Implementierung genutzten Werkzeuge beschrieben. In Abschnitt 5.2 wird eine Übersicht über die für die Umsetzung der Pipeline benötigten Komponenten gegeben. In den darauf folgenden Abschnitten werden diese Komponenten vorgestellt.

5.1 Entwicklungswerkzeuge

Als Entwicklungsumgebung für die Umsetzung der Visualisierungspipeline wurde die Softwareplattform MeVisLab[1] gewählt, da sie sich durch ihr Modulakonzept und eine leichte Erweiterbarkeit hervorragend für die schnelle Umsetzung von Bildverarbeitungs- und Visualisierungsaufgaben eignet. Die vorgeschlagene Visualisierungsmethode wird in MeVisLab unter Nutzung von C++ und Open Inventor umgesetzt.

5.1.1 Die Programmiersprache C++

Die Umsetzung der zu entwickelnden Komponenten erfolgte mit der Programmiersprache C++ innerhalb der Programmierumgebung Microsoft Visual Studio C++ 6.0[2]. C++ basiert auf der Sprache C, erweitert diese aber um eine Vielzahl neuer Konzepte, wie zum Beispiel die objektorientierte Programmierung. Dieser Ansatz erlaubt die effiziente Entwicklung und Erweiterung von Applikationen sowie die Wiederverwendung von Funktionalitäten. Applikationen, die mit C++ entwickelt wurden, können zudem relativ einfach auf verschiedene Betriebssysteme portiert werden.

5.1.2 Open Inventor

Open Inventor ist eine hierarchisch aufgebaute, objektorientierte Grafikbibliothek. Die einzelnen Klassen repräsentieren zum Beispiel geometrische Objekte, Materialeigenschaften, Animationskomponenten oder Werkzeuge zur Interaktion. Die Klassen werden hierbei als *Knoten* bezeichnet. Neue Knoten können durch Ableitung neuer Klassen erzeugt werden.

Innerhalb einer Applikation können Instanzen solcher Knoten zu einem *Szenengraph* kombiniert werden. Die Visualisierung der Szene erfolgt durch Traversieren des Szenengraphs und eine Darstellung der Geometrie der einzelnen Knoten auf Basis von OpenGL.

[1] MeVisLab wird am Forschungsinstitut MeVis – Center for Medical Diagnostic Systems and Visualization in Bremen entwickelt; www.mevis.de, letzter Stand: 06.07.2006

[2] www.microsoft.de, letzter Stand: 06.07.2006

Ein Überblick zur objektorientierten Applikationsentwicklung mit Open Inventor ist in [Wer94] gegeben.

5.1.3 Die Softwareplattform MeVisLab

MeVisLab ist eine erweiterbare Softwareplattform für Bildverarbeitung und Visualisierung im Anwendungsbereich der Medizin. Die Software wird am *Centrum für Medizinische Diagnosesysteme und Visualisierung (MeVis)* in Bremen entwickelt. Sie ist sowohl für Windows- als auch Linux-Betriebssysteme erhältlich. MeVisLab wurde im Jahr 2004 als Nachfolger der Plattform ILAB4 vorgestellt. Für die Umsetzung der vorgestellten Gefäßvisualisierungsmethode wurde Version 1.2 der Sotware genutzt.

Die grundlegenden Konzepte von MeVisLab werden im Folgenden beschrieben.

Modularität

Die Menge der in MeVisLab implementierten Funktionen und Algorithmen ist in Module aufgeteilt. Dieser Ansatz gewährleistet, dass Teilfunktionen in vielfältigen Kombinationen benutzt werden können. Für komplexe Algorithmen müssen so nicht wiederholt die selben Teilalgorithmen implementiert werden. Die Bibliothek der in MeVisLab enthaltenen Module gliedert sich in drei Grupppen:

- **ML-Module** dienen der Verarbeitung von Bilddatensätzen. Hierzu zählen zum Beispiel Importer, Filter und Module zur Darstellung.
- **Open Inventor-Module** dienen der dreidimensionalen Darstellung von Objekten und bieten Möglichkeiten zur Benutzerinteraktion.
- **Makro-Module** kombinieren beliebig viele ML- und Open Inventor-Module.

Module unterschiedlicher Gruppen können miteinander kombiniert werden, wenn die Module entsprechende Schnittstellen besitzen.

Visuelle Programmierung

Für die Kombination der Module zur Lösung komplexer Aufgaben wird in MeVisLab das Konzept der visuellen Programmierung genutzt. Ein Überblick über Konzepte der visuellen Programmierung ist zum Beispiel in [BD04] gegeben. Module stellen in MeVisLab Teile eines Programms dar, die Daten einlesen, verarbeiten und ausgeben. Die Module werden grafisch als voneinander getrennte Einheiten dargestellt. Das Programm wird beschrieben, indem die Module miteinander verknüpft werden. Die Ausgabe eines Moduls dient dabei als Eingabe eines anderen Moduls. Die Verknüpfung wird vom Benutzer mit Hilfe weniger Mausklicks hergestellt und in Form einer Linie visuell repräsentiert. Ein durch Module und Verknüpfungen beschriebenes Programm wird als *Netzwerk* bezeichnet. Ein einfaches Beispiel für ein Netzwerk ist in Abbildung 5.1 gegeben.

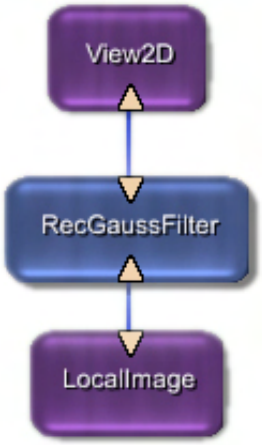


Abbildung 5.1: Beispiel für die Visuelle Programmierung in MeVisLab: Das Modul `LocalImage` dient dem Import von Bilddaten. Diese werden mit dem Filter `RecGaussFilter` geglättet und durch das Modul `View2D` dargestellt.

Erweiterbarkeit

MeVisLab kann um neue Module erweitert werden. Die Definition neuer Module kann auf zwei Arten erfolgen:

- ML- und Open Inventor-Module werden in C++ entwickelt und in Form von Dynamic Link Libraries (DLL) zur Laufzeit in MeVisLab eingebunden. Für die Entwicklung von ML-Modulen ist im MeVisLab SDK (Software Development Kit) die MeVis Image Processing Library (kurz ML) enthalten. Neue Open Inventor-Module können von jedem Open Inventor-Knoten abgeleitet werden.
- Makro-Module erfordern keine Programmierung in C++ und werden aus Netzwerken erstellt. Diese Zusammenfassung komplexer Netzwerke erhöht die Lesbarkeit und kapselt wichtige Funktionalität.

Mit diesen Konzepten ist MeVisLab äußerst gut für die Realisierung der beschriebenen Gefäßvisualisierungspipeline geeignet. Die Umsetzung wird im Folgenden erläutert.

5.2 Umsetzung der Visualisierungspipeline

Die in Abschnitt 4.1 beschriebene Gefäßvisualisierungspipeline wurde mit Hilfe vorhandener Software sowie neu implementierter Komponenten umgesetzt. Alle neuen Komponenten wurden als Module für die Softwareplattform MeVisLab umgesetzt.

Für die Extraktion einer Punktwolke aus Volumendaten wurde das MeVisLab-Modul `MPUPrepare` entwickelt. Der Zugriff auf das Segmentierungsergebnis innerhalb von MeVisLab erfolgt durch das bereits vorhandene Modul `LocalImage`. Die extrahierte Punktwolke wird in eine Datei des Formats *pwn* exportiert. Die Generierung und Polygonalisierung der MPU Implicits wird mit einem von Ohtake et al. bereitgestellten Programm durchgeführt [OBA+03a]. Das Programm trägt keinen Titel und wird im Folgenden MPUI-Programm genannt.

Die erzeugte Oberfläche wird in eine Datei des Formats *poly* geschrieben. Für die Überführung dieser Datei in eine in MeVisLab nutzbare Oberfläche wurde das Modul `PolyImport` entwickelt. Diese Pipeline ist in Abbildung 5.2 dargestellt. Die Komponenten der Pipeline werden in den folgenden Abschnitten vorgestellt.

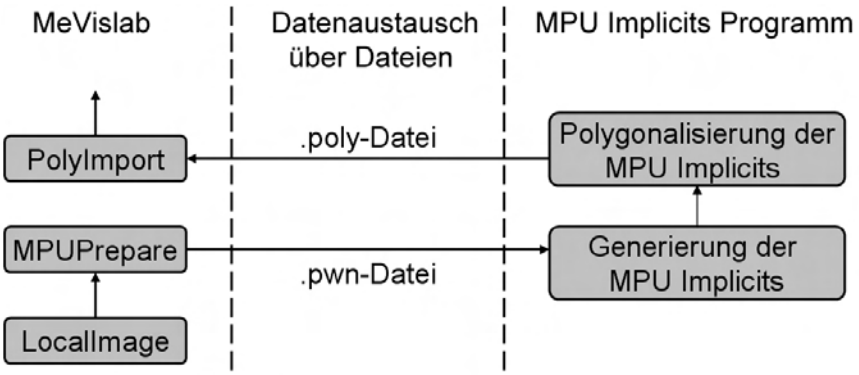


Abbildung 5.2: Umsetzung der Pipeline: Die Punktwolkenextraktion erfolgt in MeVisLab im Modul `MPUPrepare`. Die Punktwolke wird über eine *pwn*-Datei in das MPUI-Programm transferiert, wo sie als Grundlage der Oberflächengenerierung dient. Die polygonalisierte Oberfläche wird in Form einer *poly*-Datei nach MeVisLab transportiert. Das Modul `PolyImport` extrahiert aus der Datei eine in MeVisLab nutzbare Oberfläche.

5.3 Das Modul MPUPrepare

Das MeVisLab-Modul `MPUPrepare` wurde für die Extraktion einer Punktwolke aus dem Segmentierungsergebnis entwickelt. Da das Modul aufgrund seiner Position in der Pipeline keine Bilddaten oder Open Inventor-Knoten zurückgeben muss, konnte frei entschieden werden, ob die Umsetzung in einem ML- oder Open Inventor-Modul erfolgt. Da während der Entwicklung eine Visualisierung der Punktwolke benötigt wurde, um das Ergebnis beurteilen zu können, fiel die Entscheidung zugunsten eines Open Inventor-Moduls.In den folgenden Abschnitten werden die Bedienung des Moduls sowie wichtige Details der Implementierung der Punktwolkenextraktion beschrieben.

5.3.1 Die Bedienoberfläche

Die Bedienoberfläche des Moduls besteht aus zwei Oberflächen, die über einen Karteireiter gewechselt werden können (Abbildung 5.3 rechts). Auf der Oberfläche *Compute* wird mit der Schaltfläche *Calculate PWN* die Berechnung der Punktwolke gestartet. Die Punktwolkenextraktion wird in den folgenden Abschnitten genauer beschrieben.
Nach Abschluss der Punktwolkenextraktion werden die Parameterbelegungen auf Basis der in Abschnitt 4.3 aufgestellten Formeln berechnet und im Bereich *MPUI Parameters* der Bedienoberfläche ausgegeben.

Auf der Oberfläche *Export* der Bedienoberfläche kann die Punktwolke in Form einer Datei des Formats *.pwn* exportiert werden. Auf den Export wird in Abschnitt 5.3.9 eingegangen.

5.3.2 Repräsentation des Segmentierungsergebnisses

Wenn die Punktwolkenextraktion gestartet wird, wird zuerst das Segmentierungsergebnis eingelesen. Dabei gelangen die Bilddaten von dem Modul `LocalImage` über den Eingang *sgmImage* in das Modul MPUPrepare (Abb. 5.3 links). Die Daten liegen dabei in Form eines Objektes der Klasse `SoSFMLImage` vor. Die Klasse ist jedoch nicht für einen Zugriff

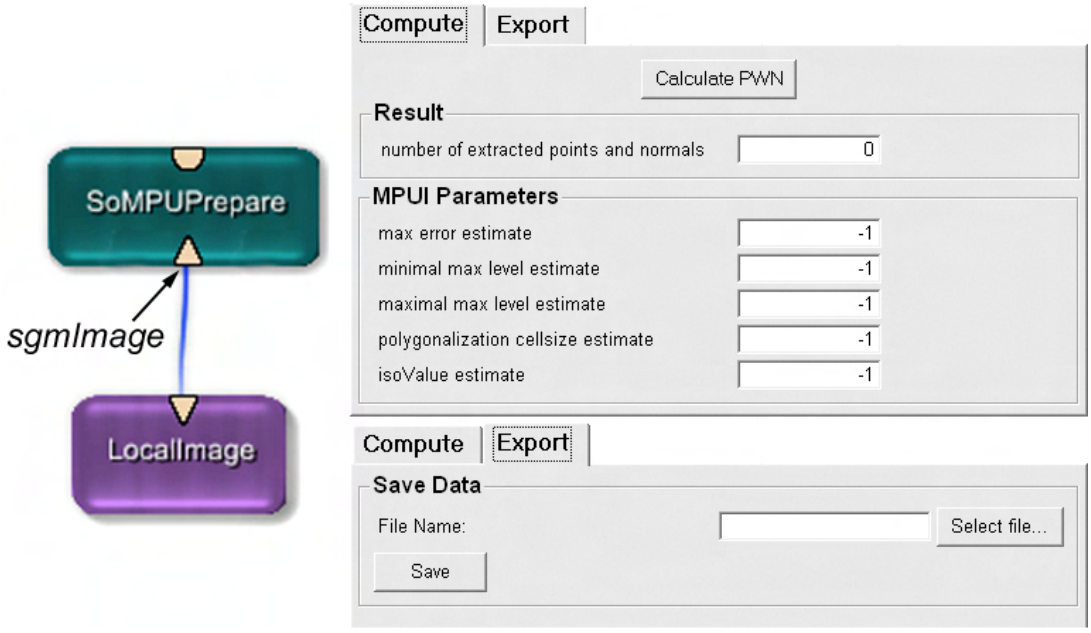


Abbildung 5.3: Das Modul `MPUPrepare` erhält als Eingabe die Bilddaten, die das Modul `LocalImage` liefert (*links*). Die Bedienoberfläche des Moduls besteht aus zwei Oberflöchen (*rechts*). Die erste Oberfläche *Compute* dient der Berechnung der Punktwolke. Die Anzahl der extrahierten Punkte sowie die ermittelten Parameter werden ausgegeben. Die zweite Oberfläche *Export* dient der Ausgabe der Punktwolke in Form einer *pwn*-Datei.

auf einzelne Voxel vorgesehen; es wird nur der Zugriff auf Teilvolumen ermöglicht. Um einen schnellen Zugriff auf einzelne Voxel zu ermöglichen, wurde eine speziell für diesen Zweck gedachte Datenstruktur in Form der Klasse `VoxelData` entwickelt.

Die VoxelData-Instanz *sgmData* repräsentiert das Segmentierungsergebnis. Mit Hilfe der Methode `readInput(SoSFMLImage* img)` werden die Daten der übergebenen `SoSFMLImage`-Instanz auf *sgmData* übertragen. Dabei werden die Daten in das interne `Integer`-Feld *array* der `VoxelData`-Instanz kopiert. Die Reihenfolge der Daten wird dabei beibehalten. Es handelt sich hierbei um ein eindimensionales Feld. Die Abbildung der dreidimensionalen Voxelkoordinaten auf eindimensionale Indizes ist in Abbildung 5.4 veranschaulicht.

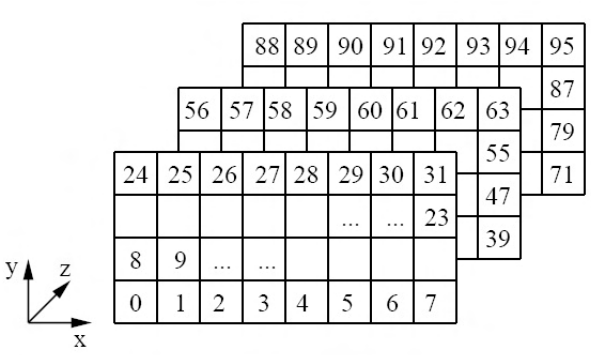


Abbildung 5.4: Zuordnung der Voxelkoordinaten eines $8 \times 4 \times 3$ Volumens auf eindimensionale Indizes: Das Volumen wird eine Schicht nach der anderen in dem Feld gespeichert. Die Zeilen einer Seite werden nacheinander gespeichert.

Das Voxel mit den Koordinaten $(0, 0, 0)$ erhält den Index 0. Der Index wächst mit steigendem x-Wert, bis das Ende der ersten Zeile erreicht wird. Nun wird y erhöht und die nächste Zeile betrachtet. Der Index steigt weiter bis y den maximalen Wert erreicht hat und so alle Zeilen für z gleich 0 betrachtet wurden. Dies wird nun für alle nachfolgenden Schichten des Datensatzes fortgeführt, bis z ebenfalls den maximalen Wert erreicht hat.

Eine Instanz der Klasse `SoSFMLImage` kann Bilddaten mit bis zu vier Kanälen speichern, jeweils ein Kanal für rot, grün, blau und Transparenz. Da jedoch nur eine Grauwertkodierung benötigt wird, wird nur der erste Kanal auf `VoxelData` übertragen. Somit kann der benötigte Speicherplatz stark reduziert werden. Alternativ hätten auch der zweite oder dritte Kanal gewählt werden können, da die Werte für rot, grün und blau bei einem Segmentierungsergebnis gleich sind. Es wurde jedoch der erste Kanal gewählt, weil dieser auf jeden Fall in der `SoSFMLImage`-Instanz enthalten ist.

Die Übertragung der Daten von der `SoSFMLImage`-Instanz wird auf folgende Weise realisiert:

```
1  float* pBuffer;
2  pBuffer=(float*)malloc(sizeof(float)*numberOfVoxels);
3
4  XVImageSize position;
5  position.x = 0;
6  position.y = 0;
7  position.z = 0;
8
9  XVImageSize imageSize;
10 img->getSize(imageSize);
11 imageSize.c = 1;
12
13 img->getTile6D(pBuffer, MLfloatType, pos, imageSize);
```

In den Zeilen 1 und 2 wird das Feld *pBuffer* erzeugt, das die Daten aufnehmen kann. *numberOfVoxels* wurde zuvor aus den Abmessungen des Voxeldatensatzes ermittelt. Mit dem Befehl `getTile6D` können Teilvolumen aus einem Bilddatensatz kopiert werden. Hier wird er genutzt, um das gesamte Volumen auf *pBuffer* zu übertragen. Die Objekte *position* und *imageSize* enthalten dabei die Startposition und die Größe des zu übertragenden Volumens. Als Startposition wird der Voxel mit den Koordinaten $(0, 0, 0)$ gewählt (Zeile 5-7). Als Größe wird die Größe des ursprünglichen Datensatzes genutzt (Zeile 10). In Zeile 11 wird die Anzahl der zu übertragenden Kanäle auf 1 gesetzt.

Da für die spätere Bearbeitung `Integer`-Werte benötigt werden, werden die Werte aus *pBuffer* in einer Schleife auf das `Integer`-Feld *array* übertragen.
Wurde *sgmData* mit den Bilddaten gefüllt, müssen diese Daten normalisiert werden. Bei diesem Vorgang erhalten alle Hintergrundvoxel den Wert 0 und alle Objektvoxel den Wert 1. Dies ist notwendig, weil die Bilddatensätze je nach Herkunft unterschiedliche Werte für Objekt- und Hintergrundvoxel besitzen können. Für die Normalsierung wird die Funktion `normalize()` der Klasse `VoxelData` genutzt.

5.3.3 Identifizierung dünner Strukturen

Die für die Identifizierung der dünnen Strukturen notwendigen morphologischen Operatoren sind in Form der Methoden `erode(int sElement[ ], int num)` und `dilate(int sElement[ ], int num)` in der Klasse `VoxelData` implementiert. Das benutzte Strukturelement wird in Form des eindimensionalen `Integer`-Feldes *sElement* definiert. Je drei aufeinander folgende Einträge definieren die relative Position eines Voxels innerhalb des Strukturelements. Das verwendete $3 \times 3 \times 3$ -Strukturelement benötigt demnach 81 Einträge. Dieses Strukturelement ist im Anhang A.1 definiert. Zusammen mit diesem `Integer`-Feld wird die Anzahl der im Strukturelement enthaltenen Voxel übergeben. Das benutzte $3 \times 3 \times 3$ -Strukturelement enthält 27 Voxel.

Für die Identifizierung der dünnen Strukturen wird eine neue Instanz der Klasse `VoxelData` angelegt. Der komplette Prozess zur Identifizierung dünner Strukturen ist folgendermaßen umgesetzt:

```
1 VoxelData* thickData = new VoxelData();
2 thickData->init(*sgmData);
3 thickData->erode(se3x3x3,27);
4 thickData->dilate(se3x3x3,27);
```

thickData wird als Kopie von *sgmData* initialisiert. Hierfür wird die Methode `init(VoxelData& vd)` genutzt. Sie kopiert das Feld *array* sowie alle Eigenschaften von *sgmData* auf *thickData*. Die Instanz wird somit zu einer Kopie des Segmentierungsergebnisses.

Daraufhin wird ein Opening auf Basis eines $3 \times 3 \times 3$ -Strukturelements durchgeführt (Zeilen 3 und 4). In *thickData* sind somit jene Voxel auf 1 gesetzt, die in sgmData den Wert 1 haben und nicht zu einer dünnen Struktur gehören. Alle anderen Voxel haben den Wert 0. *thickData* entspricht somit $\mathcal{O}'$ aus Abschnitt 4.2.1. Die Menge $\mathcal{O}''$ der Voxel, die zu dünnen Strukturen gehören, wird nicht explizit berechnet. Für einen Voxel kann auf Basis von *sgmData* und *thickData* ermittelt werden, ob er zu einer dünnen Struktur gehört.

5.3.4 Klassifizierung der Voxel

Als nächstes folgt die Klassifizierung des Volumendatensatzes. Grundlage hierfür sind die beiden Volumandatensätze *sgmData* und *thickData*. *sgmData* entspricht dem Segmentierungsergebnis $\mathcal{S}$ in Abschnitt 4.2.2. Da der klassifizierte Datensatz $\mathcal{S}^*$ auch alle Informationen des Segmentierungsergebnisses enthällt, werden diese beiden Datensätze in der Implementierung nicht getrennt. Die Markierungen werden somit direkt in *sgmData* eingetragen.

Bei dem Vorgang zur Klassifizierung müssen einzelne Voxel gelesen und gesetzt werden. Für das Lesen einzelner Voxel wird dabei die Methode `getVoxel(int x, int y, int z)` der Klasse `VoxelData` genutzt. Der Index für den Zugriff auf das interne lineare Feld *array* wird dabei folgendermaßen ermittelt:

$$index = z * extX * extY + y * extX + x \tag{5.1}$$

Die Variablen *extX*, *extY* und *extZ* sind hierbei die Ausdehnungen des Datensatzes in den drei Dimensionen. Befinden sich die Koordinaten außerhalb dieser Ausdehnungen, wird der Wert 0 zurückgegeben.

Für das Setzen von Voxeln wurde in `VoxelData` analog die Methode `setVoxel(int x, int y, int z, int g)` implementiert. Die Variable *g* repräsentiert hiebei den zu setzenden Wert.

Bei der Klassifizierung muss die Nachbarschaft einzelner Voxel betrachtet werden. Dies kann folgendermaßen effizient umgesetzt werden:

```
1  vec3 nbDir[6];
2
3  nbDir[0][0] = -1;    nbDir[0][1] =  0;    nbDir[0][2] =  0;
4  nbDir[1][0] =  1;    nbDir[1][1] =  0;    nbDir[1][2] =  0;
5  nbDir[2][0] =  0;    nbDir[2][1] =  1;    nbDir[2][2] =  0;
6  nbDir[3][0] =  0;    nbDir[3][1] = -1;    nbDir[3][2] =  0;
7  nbDir[4][0] =  0;    nbDir[4][1] =  0;    nbDir[4][2] =  1;
8  nbDir[5][0] =  0;    nbDir[5][1] =  0;    nbDir[5][2] = -1;
9
10 for(int i=0;i<6;i++){
11  xn=x+nbDir[i][0];
12  yn=y+nbDir[i][1];
13  zn=z+nbDir[i][2];
14  val=sgmData->getVoxel(xn,yn,zn);
15 }
```

Das Feld *nbDir* enthält die relativen Positionen der Voxel aus der 3D-6-Nachbarschaft (Zeilen 1-8). Eine Iteration über das Feld erlaubt die Ermittlung der absoluten Positionen der benachbarten Voxel (Zeilen 10-15). Diese Methode wird immer eingesetzt, wenn die Nachbarschaft eines Voxels betrachtet werden muss.

Für die Klassifizierung müssen alle Voxel in *sgmData* betrachtet werden. Dazu wird in drei geschachtelten Schleifen über alle Voxel von *sgmData* iteriert:

```
for(int z = 0;z<sgmData->getExtZ();z++){
 for(int y = 0;y<sgmData->getExtY();y++){
  for(int x = 0;x<sgmData->getExtX();x++){
   val=sgmData->getVoxel(x,y,z);
   ...
}}}
```

Die Methoden `getExtX()`, `getExtX()` und `getExtX()` liefern hierbei die Ausdehnungen von *sgmData*. Innerhalb dieser geschachtelten Schleife findet die Betrachtung der Nachbarschaft eines jeden Voxels statt, um ihn zu klassifizieren.

5.3.5 Überabtastung der Voxel

Der in Abschnitt 4.2 beschriebene Algorithmus erfordert eine Überabtastung des Segmentierungsergebnisses (Abschnitt 4.2.3). In der Praxis kann dies jedoch nicht durch einen Datensatz mit doppelter Auflösung realisiert werden, da das erforderliche Datenvolumen auf das achtfache ansteigen würde. Für eine effiziente Überabtastung wird die Belegung der acht zu einem Voxel gehörenden Subvoxel deswegen im Datenwert des Voxels gespei-

chert. Somit kann eine Instanz der Klasse `VoxelData` mit der gleichen Auflösung wie das Segmentierungsergebnis genutzt werden, um die Überabtastung durchzuführen.

Die Kodierung der Belegung der Subvoxel erfolgt hierbei binär. Es wird nur gespeichert, ob ein Subvoxel gesetzt ist oder nicht. Es werden demnach acht Bits benötigt. Sind alle acht Bits gesetzt, beträgt der Datenwert des Voxels 255. Dies bedeutet, dass alle acht Subvoxel Objektsubvoxel sind. Gehören alle acht Subvoxel zum Hintergrund, beträgt der Datenwert des Voxels 0.

Für die überabgetastete Repräsention des Segmentierungsergebnisses wird die `VoxelData`-Instanz *subData1* angelegt. Ihre Abmessungen werden mit Hilfe der Methode `constInput(int ext_x, int ext_y, int ext_z, int g)` gesetzt. Die variable g gibt hierbei den Wert an, mit dem die Voxel initialisiert werden. Für die Initialisierung von *subData1* ist g gleich 0. Anschließend erhält in einer Schleife jeder Voxel, der in *sgmData* den Wert 1 oder 2 hat, und somit zu dem beschriebenen Objekt gehört, in *subData1* den Wert 255. Somit sind für jeden Objektvoxel alle acht Subvoxel gesetzt.

5.3.6 Reduzierung treppenartiger Artefakte

Für die Reduzierung der treppenartigen Artefakte wird ein Zugriff auf die einzelnen Subvoxel des überabgetasteten Segmentierungsergebnisses benötigt. Dafür wurden in VoxelData die Methoden `setSubVoxel(int x, int y, int z, int subVoxelvalue)` und `getSubVoxel(int x,int y,int z)` implementiert. (x, y, z) sind hierbei die Subvoxelkoordinaten. Da die Speicherung der Subvoxel auf Basis der Voxel, zu denen sie gehören, erfolgt, müssen in beiden Methoden zuerst die Koordinaten dieses Voxels ermittelt werden:

$$\begin{aligned} x_{vox} &= \lfloor x/2 \rfloor \\ y_{vox} &= \lfloor y/2 \rfloor \\ z_{vox} &= \lfloor z/2 \rfloor \end{aligned} \tag{5.2}$$

Für das Setzen und Lesen einzelner Subvoxel ist der Zugriff auf einzelne Bits des Datenwerts dieses Voxels erforderlich. Jeder Subvoxel wird durch ein Bit repräsentiert. Es ist also erforderlich, für den zu setzenden Subvoxel zu ermitteln, von welchem Bit dieser repräsentiert wird. Dazu wird zuerst die relative Position der Subvoxel innerhalb des ermittelten Voxels ermittelt. Sie sei im Folgenden $(x_{sub}, y_{sub}, z_{sub})$, wobei die Einträge des Tripels nur die Werte 0 und 1 annehmen können:

$$\begin{aligned} x_{sub} &= x - 2 * x_{vox} \\ y_{sub} &= y - 2 * y_{vox} \\ z_{sub} &= z - 2 * z_{vox} \end{aligned} \tag{5.3}$$

Der Index des zu setzenden Bits wird auf Basis dieser Werte ermittelt:

$$bitIndex = 4 * z_{sub} + 2 * y_{sub} + x_{sub} \tag{5.4}$$

bitIndex kann so Werte zwischen 0 und 7 annehmen.

In der Methode `setSubVoxel(int x, int y, int z, int subVoxelvalue)` wird *bitIndex* genutzt, um das entsprechende Bit zu setzen:

```
if(subVoxelvalue > 0){
        voxelVal = voxelVal OR pow(2,bitIndex);
}else{
        voxelVal = voxelVal AND NOT pow(2, bitIndex);
}
```

Der Wert des Voxels *voxelVal* wird auf Basis des nach Formel 5.1 ermittelten Index in *array* gesetzt. Bei der getSubVoxel-Methode wird das durch *bitIndex* spezifizierte Bit aus dem Wert des Voxels (voxelVal) gelesen:

```
int pot = pow(2, bitIndex);
if((voxelVal AND pot) == pot) return 1;
else return 0;
```

Diese Methoden werden nun genutzt, um die Nachbarschaft der äußeren Subvoxel in *subData1* zu untersuchen. Wurde für einen Subvoxel entschieden, dass er gesetzt werden soll, so geschieht dies jedoch nicht in *subData1*, denn auf diese Weise würde die Nachbarschaft der benachbarten Subvoxel verändert. Statt dessen werden die zu setzenden Subvoxel in einem weiteren Datensatz *subData2* gesetzt, der die selben Amessungen wie *subData1* hat. Wurden alle äußeren Randsubvoxel betrachtet, werden die Subvoxel aus *subData2* auf *subData1* übertragen. Dazu wird die Methode `add(VoxelData vd)` der VoxelData-Klasse genutzt. Die Addition der beiden Datensätze entspricht der Übertragung der Subvoxel, weil in *subData2* nur in solchen Voxeln Subvoxel gesetzt wurden, die in *subData1* äußere Randvoxel sind und somit in *subData1* den Wert 0 haben. Die Addition eines Wertes ungleich 0 führt somit dazu, dass der Voxel in *subData1* die Subvoxel des entsprechenden Voxels aus *subData2* übernimmt.

5.3.7 Extraktion der Punkte und Normalenvektoren

Für die Platzierung der Punkte und die Berechnung der Normalen auf Basis eines aüßeren Voxel oder äußeren Subvoxels wird die 3D-26-Nachbarschaft dieses Voxels bzw. Subvoxels benötigt. Sie wird in dem dreidimensionalen `Integer`-Feld *neighborhood* mit $3 \times 3 \times 3$ Einträgen gespeichert.

Auf Voxelebene wird die Nachbarschaft durch Iteration über die benachbarten Voxel in *sgmData* ermittelt. Für jeden Nachbarvoxel wird der Datenwert mit der `getVoxel`-Methode gelesen. Handelt es sich bei dem Wert um eine 1 oder 2, ist der Nachbarvoxel ein Objektvoxel. In diesem Fall wird an der entsprechenden Position in *neighborhood* der Wert 1 eingetragen, ansonsten der Wert 0. Aus dem klassifizierten Datensatz wird so für die Nachbarschaft wieder das Segmentierungsergebnis gewonnen.

Auf Subvoxelebene können die Werte der benachbarten Subvoxel direkt mit der `getSubVoxel`-Methode aus *subdata1* ermittelt werden.

Für jeden der äußeren Randvoxel bzw. Randsubvoxel wird daraufhin die Methode `extractPoints(vec3 voxelCoord, bool subVoxel, int neighborhood[3][3][3])` aufgerufen. Diese Methode dient der Extraktion der zu einem äußeren Rand(sub-)voxel gehörenden Punkte und Normalen. Der Methode wird die zuvor ermittelte Nachbarschaft des (Sub-)Voxels übergeben. Der Vektor *voxelCoord* bezeichnet die Koordinaten des Voxels

bzw. Subvoxels in Bezug auf den Datensatz, aus dem er stammt. Die für die Repräsentation des Vektors genutzte Klasse `vec3` ist Bestandteil der Bibliothek `mlLinearAlgebra` der MeVis Image Processing Library. Die `Boolean`-Variable *subVoxel* gibt an, ob es sich um einen Voxel (`false`) oder Subvoxel (`true`) handelt. Handelt es sich um einen Subvoxel, müssen für jeden extrahierten Punkt die Koordinaten in das Voxelkoordinatensystem transformiert werden (siehe Formel 4.18).

Wurde nur ein Punkt für einen (Sub-)voxel platziert, erfolgt die Berechnung des Normalenvektors auf Basis des Grauwertgradienten. Dazu wird die Methode `getNormal(int neighborhood[3][3][3])` aufgerufen. Ihr wird als einziges Argument *neighborhood* übergeben; das Ergebnis ist ein Vektor. Wurden mehrere Punkte ermittelt, werden die Richtungen zu den Grenzflächen, auf denen die Punkte platziert wurden, als Normalenvektoren verwendet.

Für die Speicherung der extrahierten Punkte und Normalenvektoren wird das Objekt *pntdata* genutzt. Es ist eine Instanz der Klasse `pwnData`, die für die Verwaltung einer Liste von Punkten und Normalen entwickelt wurde. Ein Punkt und die zugehörige Normale werden hierbei zusammen in einem Datentyp gespeichert:

```
struct POINT{
        vec3 position;
        vec3 normal;
        vec3 worldPosition;
        vec3 worldNormal;
};
```

Punktposition und Normalenvektor werden dabei in Voxel- und Weltkoordinaten gespeichert. Die Speicherung der `POINT`-Instanzen erfolgt in `pwnData` in dem Klassenelement *pwnList* vom Typ `Vector`.

Für das Hinzufügen eines Punktes und der zugehörigen Normale wurde in der Klasse `pwnData` die Methode `addPoint(vec3 position,vec3 normal)` implementiert. Die Vektoren *position* und *normal* werden hierbei in Voxelkoordinaten übergeben. Für die Berechnung der Weltkoordinaten erhält die `pwnData`-Instanz die entsprechende Matrix. Sie ist Bestandteil der von dem `LocalImage`-Modul übergebenen `SoSFMLImage`-Instanz und liegt in Form eines Objektes der Klasse `mat4` vor. die Klasse `mat4` ist ebenfalls Bestandteil der Bibliothek `mlLinearAlgebra` und dient der Repräsentation und Manipulation von 4×4-Matritzen. Bestandteil der Klasse sind auch Methoden zur Ermittlung der inversen und der transponierten Matrix. Die für die Transformation der Normalenvektoren benötigte Matrix kann somit ermittelt werden (siehe Abschnitt 4.2.6).

5.3.8 Ermittlung der Parameter

Für die Ermittlung der Parameter ϵ_0, *max_level* und *grid_size* werden die Abmessungen den Punktwolke benötigt (siehe Abschnitt 4.3). Diese können folgendermaßen ermittelt werden:

```
pntData->computeExtensions();
float extX = pntData->maxX - pntData->minX;
float extY = pntData->maxY - pntData->minY;
float extZ = pntData->maxZ - pntData->minZ;
```

Die Methode `computeExtensions()` wird nach Beendigung der Punktwolkenextraktion aufgerufen. In dieser Methode werden die minimalen und maximalen X, Y und Z-Werte ermittelt, die bei den Koordinaten der Punkte der Punktwolke in vorkommen (`minX`, `minY`, `minZ`, `maxX`, `maxY` und `maxZ`). Dabei werden die Werte in Voxelkoordinaten ermittelt. Durch Bildung der Differenz zwischen dem maximalen und dem minimalen Wert für eine Dimension wird die Ausdehnung in dieser Dimension berechnet.

Auf Basis dieser Werte werden die Parameter entsprechend den in Abschnitt 4.3 gegebenen Formeln berechnet und im Bereich *MPUI Parameters* der Bedienoberfläche des `MPUPrepare`-Moduls angezeigt (siehe Abbildung 5.3).

5.3.9 Export der Punktwolke

Das MPUI-Program liest Punktwolken in Form von *pwn*-Dateien ein. Dabei scheint es sich um ein speziell für das Programm entwickeltes Format zu handeln. Die Abkürzung steht vermutlich für *Points with Normals*. Um die extrahierte Punktwolke in das Programm transportieren zu können, wurde in dem Modul `MPUPrepare` ein Export der Punktwolke in dieses Format implementiert. Der Aufbau des *pwn*-Formats ist wie folgt:

V
$p_1x\ p_1y\ p_1z$
$\vdots$
$p_Vx\ p_Vy\ p_Vz$
$n_1x\ n_1y\ n_1z$
$\vdots$
$n_Vx\ n_Vy\ n_Vz$

In der ersten Zeile wird die Anzahl der Punkte V angegeben. Die nächsten V Zeilen enthalten die Koordinaten für je einen Punkt. Im Anschluss daran werden auf die selbe Weise die zugehörigen Normalen angegeben.

Der entsprechende Schreibvorgang wurde in der Datenstruktur `pwnData` in Form der Methode `saveToFile(const char* path)` implementiert. Da *pwn*-Dateien unverschlüsselt sind, können die Daten mit einer Instanz der Klasse `ofstream` geschrieben werden. `ofstream` ist Bestandteil der Standard Template Library von C++ [Str00].

Entsprechend dem beschriebenen Aufbau des *pwn*-Formats wird zuerst die Anzahl der Punkte in die Datei geschrieben. Diese kann mit der `getCount()`-Methode der `pwnData`-Instanz ermittelt werden. Anschließend werden in einer Schleife alle Punkte geschrieben. Dabei werden die Weltkoordinaten der Punkte genutzt. In einer weiteren Schleife werden die zugehörigen Normalenvektoren ebenfalls in Weltkoordinaten in die Datei geschrieben.

5.4 MPUI-Programm

Die Autoren von [OBA+03b] stellen auf ihrer Homepage ein Programm zur Erzeugung und Polygonalisierung von MPU Implicits zur Verfügung [OBA+03a]. Da das Programm zu

Testzwecken entwickelt wurde und nur einen kryptischen Namen[3] hat, wird es im Rahmen dieser Arbeit als MPUI-Programm bezeichnet.

Nach dem Import einer *pwn*-Datei wird die entsprechende Punktwolke im Viewport des Programms dargestellt und kann von verschiedenen Seiten betrachtet werden. Über die zwei Eingabefenster *Function Properties* und *Polygonization* können die in Abschnitt 3.4.1 vorgestellten Parameter eingegeben werden (Abbildung 5.5).

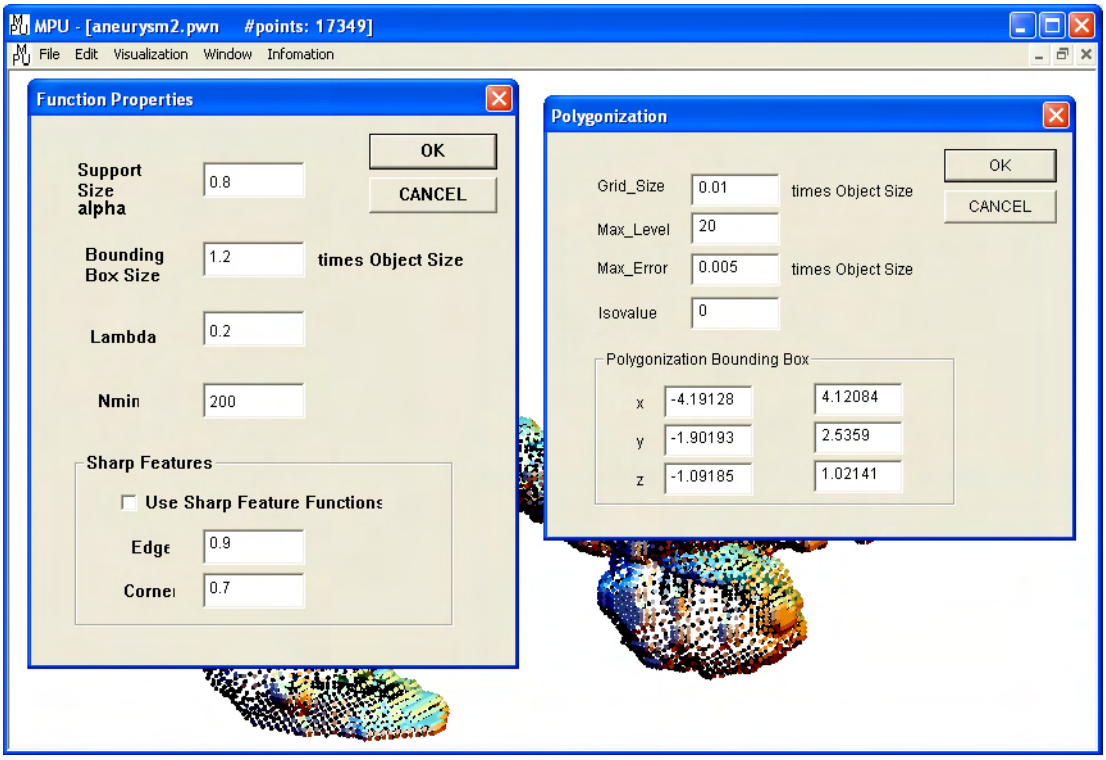


Abbildung 5.5: Bedienoberfläche des MPUI-Programms: Im Viewport wird die Punktwolke dargestellt. Die für die Berechnung benötigten Parameter werden über die zwei Eingabefenster *Function Properties* und *Polygonization* eingegeben.

Für α, λ und N_{min} werden die vorgeschlagenen Werte aus Abschnitt 4.3.1 genutzt ($\alpha = 0.8$, $\lambda = 0.2$ und $N_{min} = 200$). Die Erkennung von Kanten (*Use Sharp Feature Functions*) wird deaktiviert, da diese bei organischen Formen nicht benötigt wird. Die restlichen Parameter werden von den entsprechenden Ausgabefeldern des `MPUPrepare`-Moduls übernommen (siehe Abbildung 5.3). Da die Bezeichnungen der Parameter nicht exakt übereinstimmen ist in Tabelle 5.1 eine Zuordnung der Bezeichnungen gegeben.

Durch Betätigen der Schaltfläche *OK* im Fenster *Polygonization* wird die Berechnung der Oberfläche gestartet. Das Aufstellen der impliziten Funktion und deren Polygonalisierung werden hierbei automatisch nacheinander durchgeführt. Die erzeugte Oberfläche wird in eine Datei des Formats *poly* gespeichert. Dieses Format wurde vermutlich ebenfalls extra für das Programm entwickelt. Die Beschreibung der Oberfläche in einer *poly*-Datei erfolgt durch Definition der Polygone durch Zeiger auf eine Knotenliste. Diese Form der Beschreibung polygonaler Oberflächen ist zum Beispiel in [FDFH97] beschrieben.

[3]Die für das zip-Archiv genutzte Bezeichnung *MPU_OCT_03* nimmt Bezug darauf, dass das Programm vom Oktober des Jahres 2003 stammt.

Tabelle 5.1: Zuordnung der Parameterbezeichnungen ziwschen MPUPrepare und dem MPUI-Programm.

Modul MPUPrepare	MPUI-Programm
max error estimate	Max_Error
minimal max level estimate	Max_Level
polygonization cellsize estimate	Grid_Size
isoValue estimate	Isovalue

Eine *poly*-Datei ist wie folgt aufgebaut:

V
P
$v_1x\ v_1y\ v_1z$
$\vdots$
$v_Vx\ v_Vy\ v_Vz$
$n_1\ v_{1\_1} \ldots v_{1\_n_1}$
$\vdots$
$n_P\ v_{P\_1} \ldots v_{P\_n_P}$

In der ersten Zeile wird die Anzahl der *Verticies* (V) definiert. Ein Vertex ist ein Eckpunkt eines Polygons. In der folgenden Zeile wird die Anzahl der Polygone angegeben (P). Es folgt eine Liste von V Verticies. Jede Zeile enthält dabei die Koordinaten je eines Vertcies. Anschließend daran folgt eine Liste von P Polygonen. Die Definition eines Polygons beginnt mit der Anzahl der Verticies n_k, die zu diesem Polygon gehören, gefolgt von den n_k Indizes auf die entsprechenden Verticies.

poly-Dateien sind genau wie *pwn*-Dateien unverschlüsselt. Ein Lesen der Dateien ist somit ebenfalls unproblematisch.

5.5 Das Modul PolyImport

Das MeVisLab-Modul `PolyImport` wurde entwickelt, um die von dem MPUI-Programm exportierten Oberflächen in MeVislab zu importieren. Das Lesen der Daten erfolgt durch eine Instanz der Klasse `ifstream`. Für die Beschreibung der Oberfläche werden folgende Knoten verwendet:

- `SoCoordinate3` *coord*
- `SoShapeHints` *sHint*
- `SoNormalBinding` *normBind*
- `SoIndexedFaceSet` *ifs*

Das Koordinatenfeld *coord* wird mit den Verticies der *poly*-Datei gefüllt. Der Knoten *sHint* wird genutzt, um den `creaseAngle` zu definieren. Ist der Winkel zwischen den

Normalen zweier benachbarter Polygone größer als dieser Winkel, so wird die Schattierung zwischen den beiden Polygonen unterbrochen. Die entsprechende Kante wird somit in der Schattierung sichtbar. Um einen organischen Eindruck zu gewährleisten, wird dieser Winkel auf 180° gesetzt. Somit wird die Schattierung in keinem Fall unterbrochen.

Knoten der Klasse `SoNormalBinding` definieren, wie die Zuordnung von Normalen zu Verticies erfolgt. Da Normalen nicht explizit gegeben sind, wird das Feld `value` des Knotens *normBind* auf `DEFAULT` gesetzt. Somit werden automatisch Normalenvektoren für jeden Vertex generiert.

Das `SoIndexedFaceSet` *ifs* enthält die darzustellenden Polygone. Über sein Feld `coordIndex` wird definiert, welche der Verticies, die in *coord* definiert sind, zu einem Polygon gehören. Diese Information kann direkt aus der *poly*-Datei übernommen werden. Die Oberflächenbeschreibung mit je einem Knoten der Klassen `SoCoordinate3` und `SoIndexedFaceSet` entspricht somit der Definition von Polygonen durch Zeiger auf eine Knotenliste.

Die Bedienoberfläche des `PolyImport`-Moduls ist in Abbildung 5.6 dargestellt. Der Benutzer kann die zu importierende *poly*-Datei auswählen (Schaltfläche *Select File...*) und den Lesevorgang starten (Schaltfläche *Open*). Das Modul `PolyImport` besitzt einen einzigen Ausgang, der die oben aufgelisteten Knoten ausgibt. Zur Darstellung der importierten Oberfläche kann der Ausgang zum Beispiel mit einem `SoExaminerViewer` verbunden werden (siehe Abbildung 5.6 links).

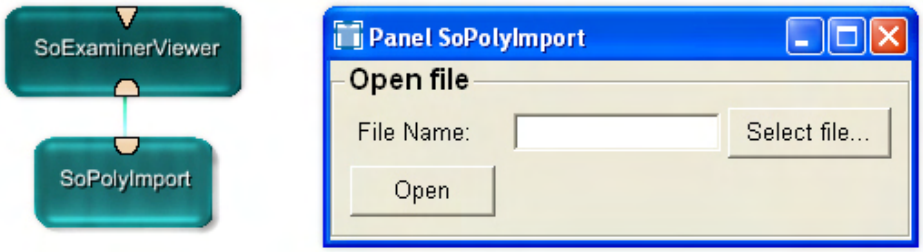


Abbildung 5.6: Der Ausgang des Moduls `PolyImport` stellt die Geometrie in Form eines `SoGroup`-Knotens zur Verfügung. Die Geometrie kann so zum Beispiel mit einem `SoExaminerViewer` dargestellt werden (*links*). *Rechts:* Die Bedienoberfläche des Moduls erlaubt die Auswahl einer Datei (*Select File...*) und das Starten des Lesevorgangs (*Open*).

6 Ergebnisse und Validierung des Verfahrens

Dieses Kapitel befasst sich mit der Auswertung des vorgestellten Verfahrens. Im ersten Abschnitt werden die Ergebnisse für einige Oberflächenrekonstruktionen gezeigt. Dabei wird das Verfahren in Hinblick auf die in der Einleitung aufgestellten Anforderungen bewertet. In Abschnitt 6.2 wird das Verfahren bezüglich der erreichten Genauigkeit untersucht.

6.1 Ergebnisse

In diesem Abschnitt werden einige Resultate des vorgestellten Verfahrens präsentiert. Dabei wird die Geschwindigkeit des Verfahrens sowie die Qualität der erzeugten Oberflächen betrachtet. Zusätzlich zu den Gefäßbäumen wird ein Lungenflügel rekonstruiert, um zu zeigen, dass das vorgestellte Verfahren grundsätzlich auch zur Oberflächenextraktion beliebiger anatomischer Strukturen geeignet ist.

6.1.1 Beispiele

Im Folgenden wird die Rekonstruktion mit der vorgeschlagenen Methode am Beispiel von vier verschiedenen anatomischen, baumartigen Strukturen demonstriert. Die verwendeten Datensätze repräsentieren ein breites Spektrum von unterschiedlichen Gefäßbäumen. Sie unterscheiden sich stark in ihrer Komplexität voneinander.

Neben Abbildungen, die die Ergebnisse zeigen, werden vor allem Geschwindigkeitsmessungen präsentiert. Es wird die für die Erzeugung der Punktwolke benötigte Zeit gemessen. Diese Zeit kann auf eine zehntel Sekunde genau gemessen werden. Die unterschiedlichen Zeiten für Generierung und Polygonalisierung der MPU Implicits können nicht getrennt ermittelt werden, da das MPUI-Programm nur die Summe der Zeitspannen ausgibt. Zudem wird diese Zeitspanne nur in ganzen Sekunden ausgegeben. In den folgenden Abschnitten wird deswegen von einem zusammenhängenden Prozess gesprochen und die Bezeichnung *Oberflächenrekonstruktion* genutzt.

Die ermittelten Zeitspannen gelten für einen PC mit Pentium4 Prozessor, einer Taktfrequenz von $3,2$ GHz und 1 GB Arbeitsspeicher.

Datensatz „Bronchialbaum“

Der Bronchialbaum des Menschen ist ein relativ komplexes Gefäßsystem. Er zeichnet sich durch eine tief verzweigte Struktur, stark variierende Radien sowie eine stark von der Kreisförmigkeit abweichende Morphologie aus (siehe dazu auch Abschnitt 2.1).

In Abbildung 6.1 wird die Rekonstruktion eines Bronchialbaumes gezeigt. Der verwendete Datensatz hat eine Auflösung von $343 \times 193 \times 259$ Voxeln. Der gesamte Prozess zur Generierung der Oberfläche nimmt $38,1$ Sekunden in Anspruch. Davon werden $11,1$ Sekunden für die Generierung der Punktwolke benötigt. Die Oberflächenrekonstruktion dauert 27 Sekunden.

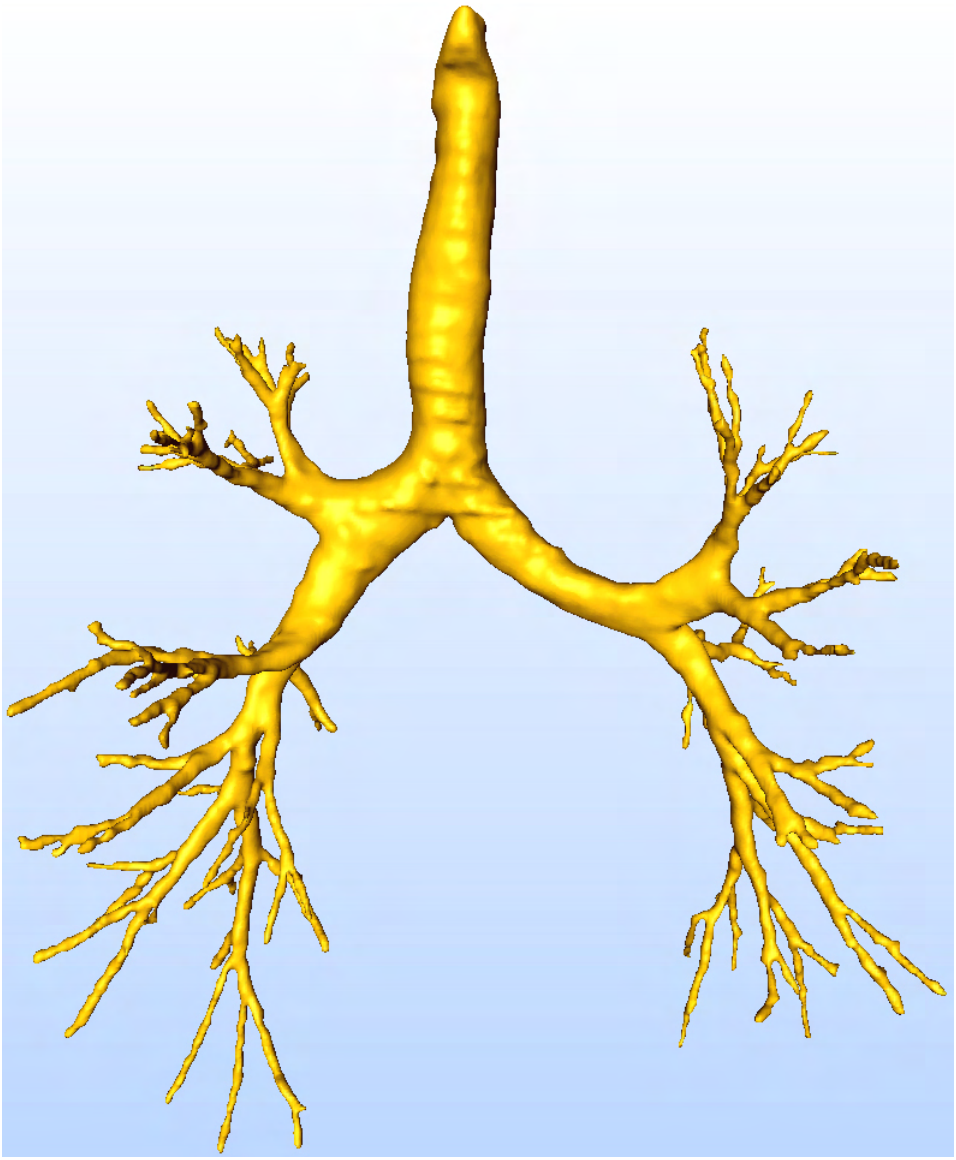


Abbildung 6.1: Rekonstruktion eines Bronchialbaumes mit Hilfe des beschriebenen Verfahrens.

Die generierte Oberfläche wirkt organisch und weist keine treppenartige Artefakte auf. Die vereinzelt sichtbaren Stufen an der Luftröhre sind keine aus dem Algorithmus resultierenden Artefakte, sondern anatomische Merkmale, die in dem Datensatz enthalten waren. Es handelt sich hierbei um die *Cartilagines tracheales*, Spangen aus Knorpel, die in der Wand der Luftröhre liegen.

Datensatz „Lebergefäßbaum“

Der verwendete Lebergefäßdatensatz zeichnet sich durch eine geringe Auflösung sowie eine starke Anisotropie aus. Der Datensatz hat eine Auflösung von $342 \times 256 \times 81$ Voxeln. Das Seitenverhältnis der Voxel ist $0.63 : 0.63 : 2$. Um zu zeigen, dass das Verfahren dennoch ein gutes Ergebnis liefert, ist in Abbildung 6.2 ein Vergleich mit dem Ergebnis des Marching Cubes-Verfahrens sowie der Rekonstruktion mit Convolution Surfaces gegeben.

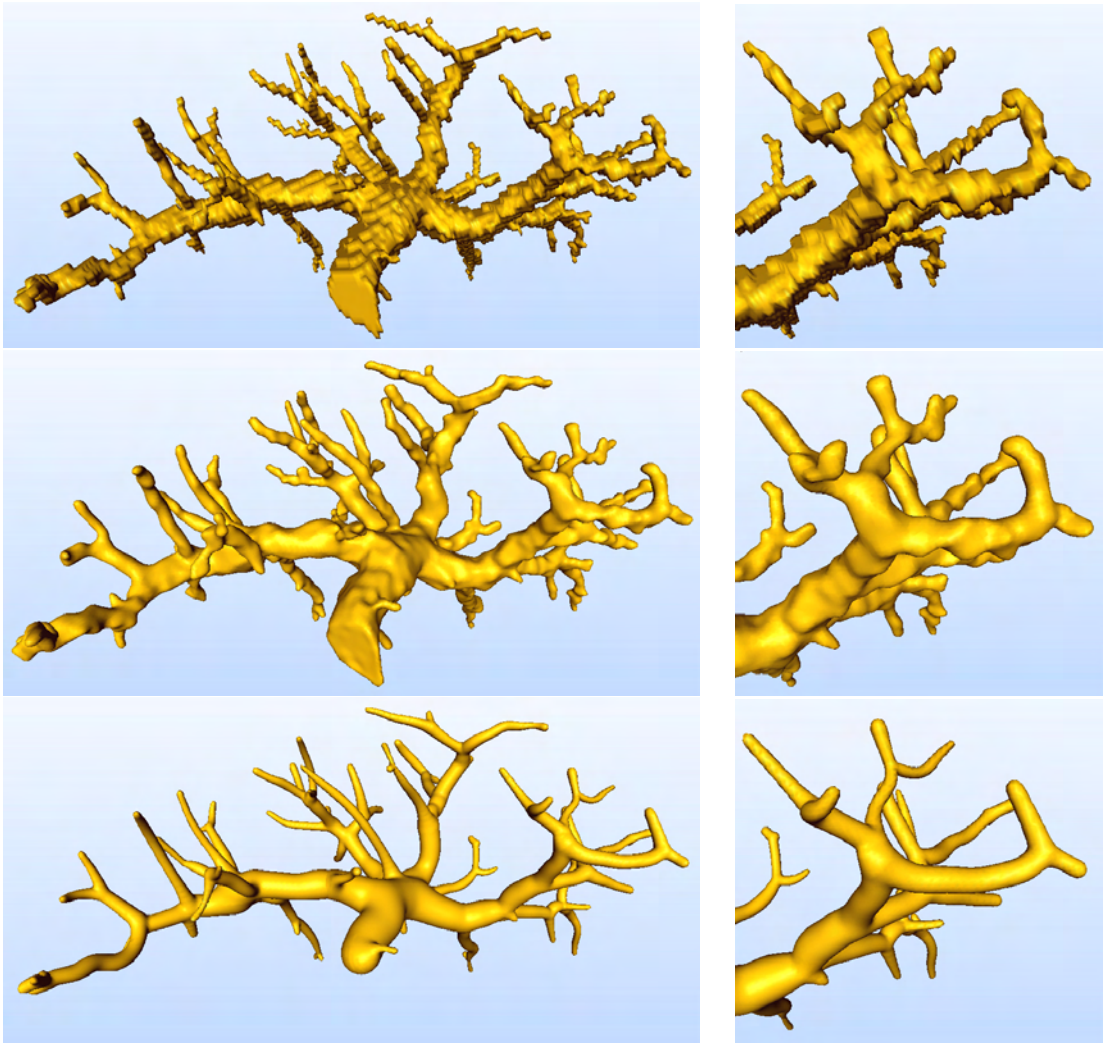


Abbildung 6.2: Rekonstruktion eines Lebergefäßbaumes: Das Ergebnis der MPU Implicits-basierten Rekonstruktion (*mitte*) wirkt weitaus organischer als das Ergebnis des Marching Cubes-Algorithmus (*oben*) und gibt zudem das Segmentierungsergebnis genauer wieder als die Rekonstruktion mit Convolution Surfaces (*unten*).

Das Resultat des Marching Cubes-Algorithmus leidet unter starken treppenförmigen Artefakten. Die MPU Implicits-basierte Rekonstruktion ist frei von solchen Artefakten und wirkt weitaus organischer. In der Abbildung wird aber auch deutlich, dass das Ergebnis der MPU Implicits-basierten Rekonstruktion die Morphologie des Gefäßes genauer wiedergibt als die Convolution Surfaces.

Die Extraktion der Punktwolke ist nach $5,7$ Sekunden abgeschlossen. Die Oberflächenrekonstruktion nimmt weitere 8 Sekunden in Anspruch.

Datensatz „Zerebrales Gefäßsystem“

Dieser Datensatz ist durch sehr lange Gefäße mit fast gleichbleibendem Radius gekennzeichnet. Er hat eine Auflösung von $322 \times 233 \times 180$ Voxeln. Das Ergebnis der Rekonstruktion ist in Abbildung 6.3 zu sehen.

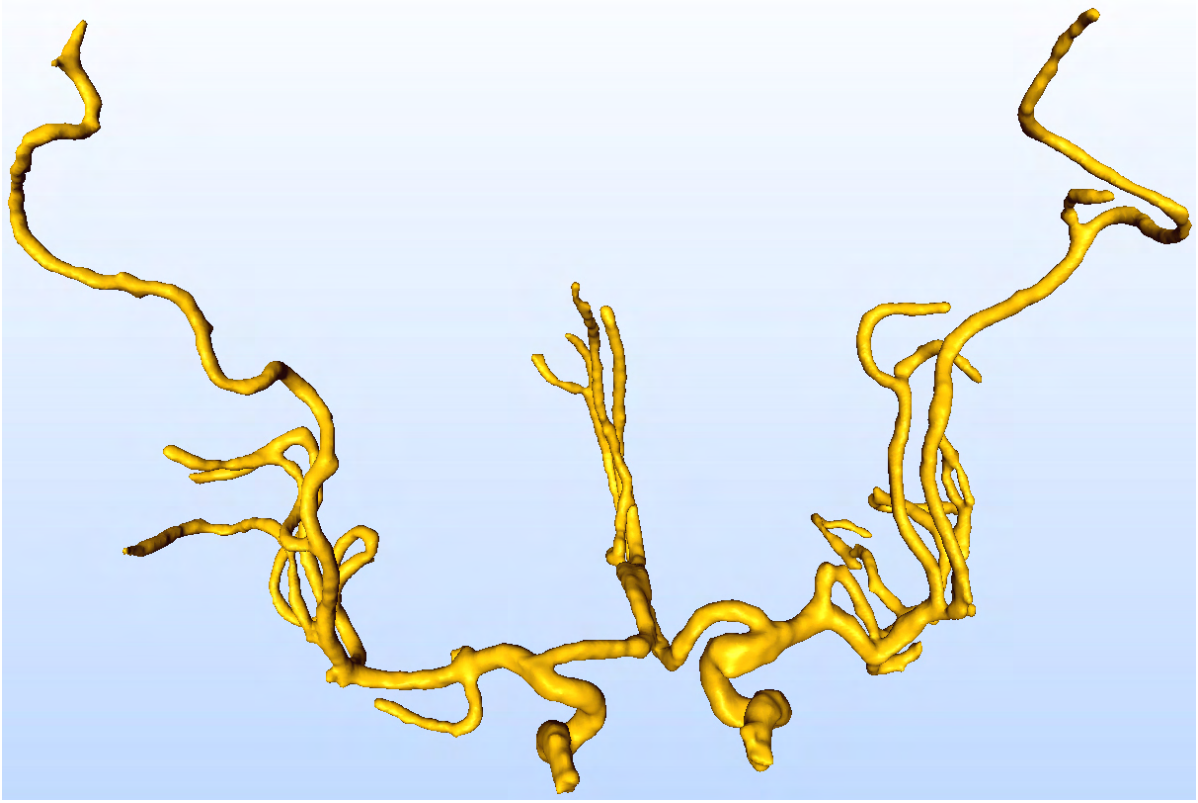


Abbildung 6.3: Rekonstruktion eines zerebralen Gefäßsystems mit der vorgeschlagenen Methode.

Das rekonstruierte Gefäß wirkt organisch. Auch Zyklen im Gefäßverlauf werden korrekt rekonstruiert (Abbildung 6.4 links). Der Datensatz zeigt jedoch auch, dass Gefäße, die im Segmentierungsergebnis direkt aneinander grenzen, in der Rekonstruktion miteinander verschmelzen (Abbildung 6.4 rechts).

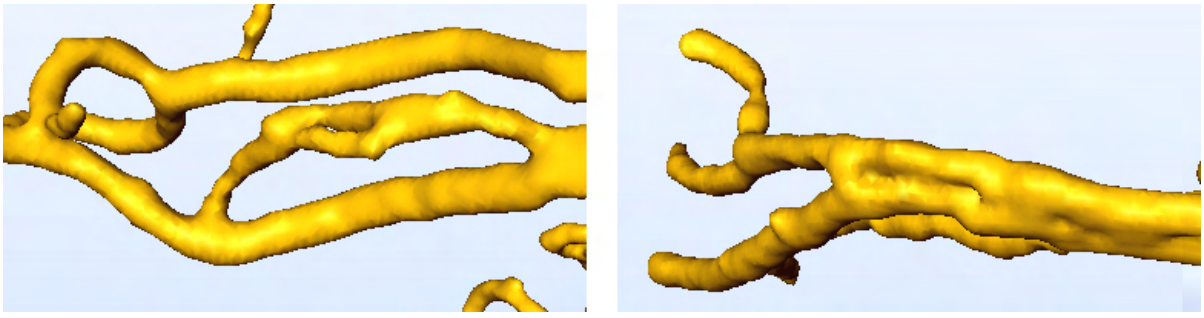


Abbildung 6.4: Zyklen im Gefäßverlauf werden korrekt rekonstruiert (*links*). Direkt benachbarte Gefäße verschmelzen (*rechts*)

Sollen die Gefäße getrennt rekonstruiert werden, muss eine Trennung bereits im Segmentierungsergebnis erfolgen, damit die entsprechende Grenzfläche zum Hintergrund vorhanden ist. Nur so können entsprechende äußere Randvoxel gefunden werden, die für die Platzierung von Punkten notwendig sind.

Die Generierung der Punktwolke erfolgt innerhalb von 8, 4 Sekunden. Für die Oberflächenrekonstruktion werden 20 Sekunden benötigt.

Datensatz „Aneurysma“

Eine wichtige Anforderung an das Verfahren ist die korrekte Wiedergabe von pathologischen Veränderungen. Beispielhaft für pathologische Gefäße wurde ein zerebrales Gefäß mit einem Aneurysma rekonstruiert (Abbildung 6.5).

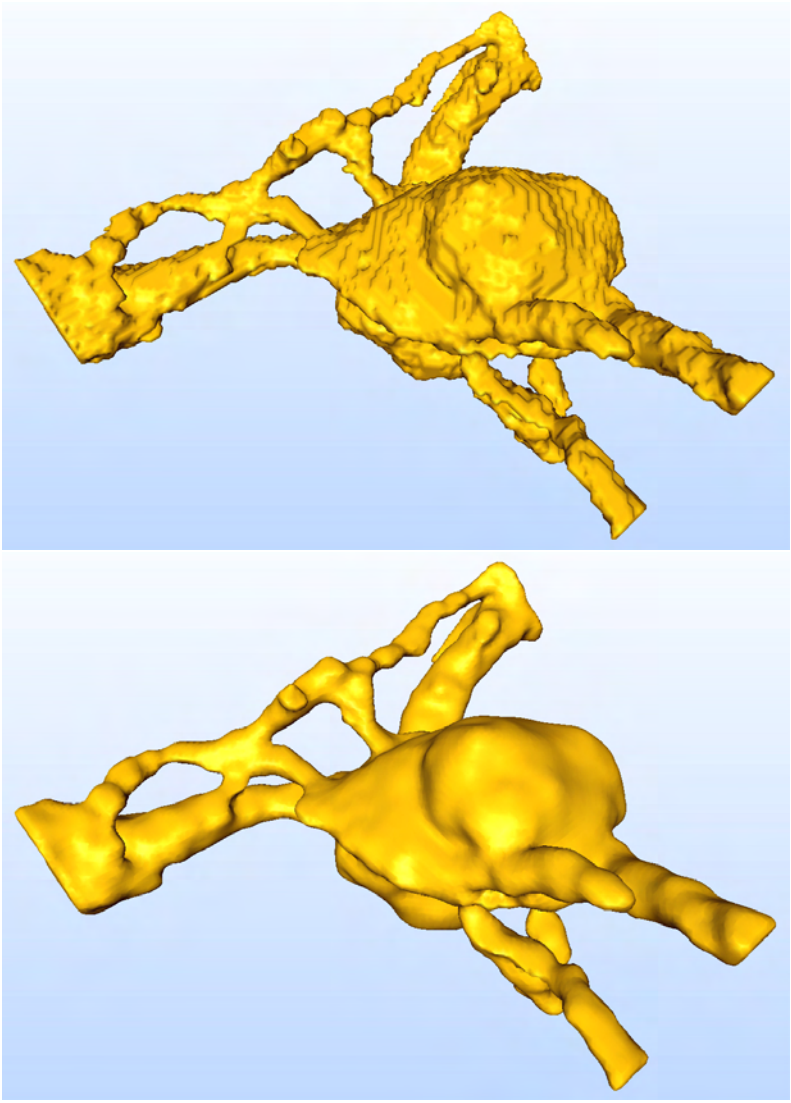


Abbildung 6.5: Pathologische Veränderungen wie Aneurysmen werden korrekt rekonstruiert (*unten*). Zum Vergleich ist das Segmentierungsergebnis in Form des Marching Cubes-Resultats gegeben (*oben*).

Der Datensatz liegt in einer Auflösung von $129 \times 107 \times 45$ Voxeln vor. Die Punktwolke wird innerhalb von $1,3$ Sekunden extrahiert. Die Oberflächenrekonstruktion nimmt 4 Sekunden in Anspruch.

Gefäß und Aneurysma werden korrekt dargestellt. Einige kleine, nur wenige Voxel große Vertiefungen werden allerdings nur andeutungsweise und nicht in ihrer ganzen Tiefe rekonstruiert. Auf diese Einschränkung des Verfahrens wurde bereits in Abschnitt 4.5 eingegangen. Daraus resultierende Auswirkungen auf die Genauigkeit der Rekonstruktion werden in Abschnitt 6.2 betrachtet.

In Tabelle 6.1 sind die relevanten Charakteristiken zu den Datensätzen zusammengefasst.

Tabelle 6.1: Relevante Charakteristiken der verwendeten Datensätze. Voxelgröße und Voxeldiagonale sind in Millimeter gegeben.

Datensatz	Auflösung	Voxelgröße	Voxeldiagonale
Bronchialbaum	$343 \times 193 \times 259$	$0,57 \times 0,57 \times 1$	1,2847
Lebergefäßbaum	$342 \times 256 \times 81$	$0,633 \times 0,633 \times 2$	2,1911
Zerebrales Gefäßsystem	$322 \times 233 \times 180$	$1 \times 1 \times 1$	1,7321
Aneurysma	$129 \times 107 \times 45$	$1 \times 1 \times 1$	1,7321

6.1.2 Rekonstruktion dünner Zweige

Eine wichtige Anforderung an das Verfahren ist die genaue und organisch wirkende Rekonstruktion von dünnen Strukturen. Die durchgeführten Versuche haben gezeigt, dass das vorgestellte Verfahren dazu in der Lage ist. Auch Zweige, die nur aus wenigen, diagonal versetzten Voxeln bestehen, werden rekonstruiert (Abbildung 6.6 links und mitte). Die meisten der dünnen Zweige wirken zudem organisch.

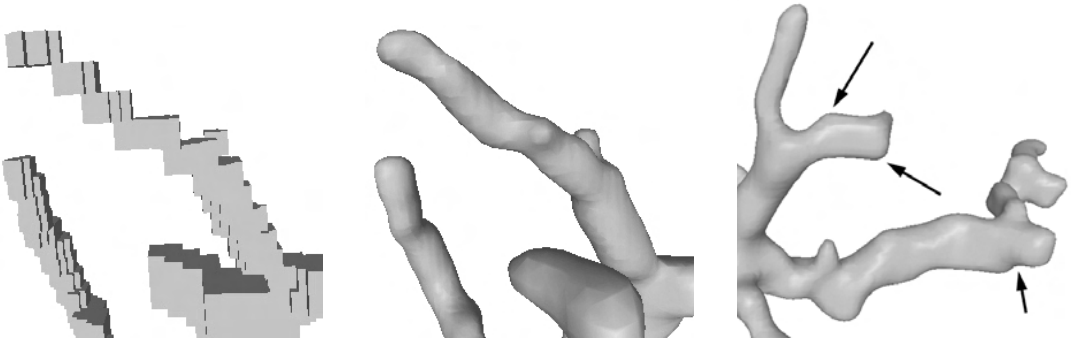


Abbildung 6.6: Dünne Zweige werden erfolgreich aus dem Segmentierungsergebnis (*links*) rekonstruiert (*mitte*). Leichte, treppenartige Artefakte treten nur äußerst selten auf und sind in diesen Fällen stark abgerundet (*rechts, durch Pfeile hervorgehoben*). Der organische Eindruck wird so kaum beeinträchtigt.

Einige der dünneren Zweige weisen jedoch eine leicht treppenförmige Geometrie auf. Durch die starke Abrundung der Formen ist dies aber nicht optisch störend (Abbildung 6.6 rechts). Die Ursache hierfür liegt in der Anordnung der Punkte, die trotz der Reduzierung der „Treppenstufen“ durch das Hinzufügen von Subvoxeln am Voxelgitter ausgerichtet sind. Durch eine Erhöhung der Parameter α und N_{min} (siehe Abschnitt 4.3.1) könnte dies weiter reduziert werden. Die Berechnungszeit für die Generierung der MPU Implicits würde so aber weiter steigen. Dies erscheint aufgrund der organischen Erscheinung jedoch nicht nötig.

6.1.3 Rekonstruktion von Verzweigungen

Ein wichtiger Aspekt für die Beurteilung eines Gefäßvisualisierungsverfahrens liegt in der Darstellung von Verzweigungen. Die vorgestellte, MPU Implicits-basierte Oberflächenrekonstruktion stellt Verzweigungen auf organische Weise dar (Abbildung 6.7). Dies resultiert aus der Orientierung des Verfahrens an den Volumendaten und der glättenden Natur der MPU Implicits. Die Volumendaten beschreiben die tatsächliche Morphologie der Verzweigungen. Da das Verfahren die in den Volumendaten beschriebene Morphologie möglichst genau wiedergibt und dabei eine glatte Oberfläche erzeugt, werden auch Verzweigungen automatisch natürlich darsgestellt.

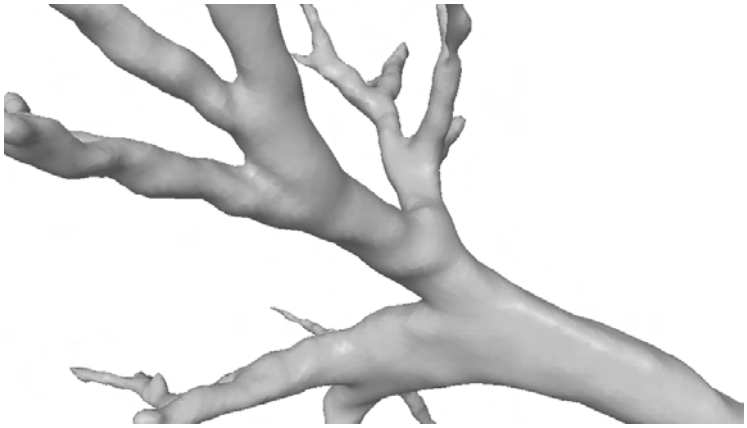


Abbildung 6.7: Verzweigungen werden von dem Verfahren organisch dargestellt.

6.1.4 Ermittelte Parameter

Für die Rekonstruktionen werden die Parameter nach Abschnitt 4.3 ermittelt und unverändert benutzt (siehe Tabelle 6.2). Es ist keine manuelle Anpassung erforderlich; die Ergebnisse werden beim ersten Versuch erfolgreich generiert. Für max_level wird ausschließlich die untere Schranke max_level_{min} verwendet, wie in Abschnitt 4.3.1 empfohlen. Es kommt bei keinem der Datensätze zur Entstehung der in Abschnitt 3.4 beschriebenen Artefakte (siehe auch Abbildung 3.18).

Tabelle 6.2: Ergebnisse im Überblick: In Spalte 1 sind die Namen der Datensätze gegeben. Spalte 2 gibt die ermittelten Werte für den maximalen Fehler ϵ_0 an. Spalte 3 enthält die ermittelten Werte für die maximale Unterteilungstiefe des Octrees. Die letzten beiden Spalten zeigen die für die Polygonalisierung ermittelten Parameter.

Datensatz	ϵ_0	max_level_{min}	Zellgröße	Isowert
Bronchialbaum	0,00184473	8	0,00204678	0,000184473
Lebergefäßbaum	0,00265447	6	0,0028	0,000265447
Zerebrales Gefäßsystem	0,00199254	8	0,00218069	0,000199254
Aneurysma	0,00503742	6	0,00546875	0,000503742

Aus den Belegungen wird deutlich, wie unterschiedlich die Parameter je nach Datensatz gewählt werden müssen. Ohne die vorgeschlagene Parameterermittlung wäre das Verfahren kaum im klinischen Alltag einsetzbar, da eine Vielzahl von Versuchen notwendig wäre, um eine artefaktfreie Rekonstruktion zu erhalten.

6.1.5 Glattheit der Rekonstruktion

MPU Implicits erzeugen eine glatte Oberfläche. Auch eine visuelle Inspektion vermittelt den Eindruck, dass die generierte Oberfläche glatt ist. Im Folgenden soll dieser Eindruck durch einen Vergleich der Glattheit der erzeugten Oberflächen mit dem etablierten Marching Cubes-Verfahren untermauert werden. Dieser Vergleich wird mit Hilfe der Software AMIRA der Firma *Mercury Computer Systems*[1] durchgeführt. Es kommt Version 3.1 des Programms zum Einsatz. Das Programm dient der Analyse und Bearbeitung von Volumendatensätzen und polygonalen Objekten.

Für die direkte Auswertung der Glattheit der erzeugten Oberflächen stehen in AMIRA keine Möglichkeiten zur Verfügung. Eine Auswertung kann jedoch erfolgen, indem die Verteilung von Krümmungswerten auf der Oberfläche betrachtet wird. Die Krümmung kann in AMIRA mit dem Modul *GetCurvature* berechnet werden. Das Modul bietet verschiedene Krümmungsmaße an. Davon wird die maximale Krümmung gewählt. Diese stellt das Maximum der Beträge der beiden Hauptkrümmungen dar und ist sehr gut für die Bewertung der Glattheit geeignet ([Haa05], Seite 65).

In Abbildung 6.8 ist die Verteilung der Krümmungswerte auf der Gefäßoberfläche zu sehen. Hierbei wird das Ergebnis der MPU Implicits-basierten Rekonstruktion (rechts) dem Resultat des Marching Cubes-Verfahrens (links) am Beispiel des Bronchialbaums gegenübergestellt. Es ist deutlich zu sehen, dass die Krümmungswerte bei der Marching Cubes-Rekonstruktion überall auf der Oberfläche stark variieren. Dies ist den treppenförmigen Artefakten zuzuschreiben, die der Algorithmus erzeugt. Bei der MPU Implicits-Rekonstruktion beschränken sich die hohen Krümmungswerte hingegen auf Zweige mit kleinem Radius. Die Luftröhre sowie die beiden Hauptbronchien weisen fast überall sehr niedrige Krümmungswerte auf. Bei der MPU Implicits-basierten Rekonstruktion liegt die Ursache für hohe Krümmungswerte also allein in der Morphologie der beschriebenen Struktur, nicht in Artefakten, die durch die Rekonstruktionsmethode verursacht wurden.

Der Unterschied der erreichten Qualität der beiden Verfahren wird auch an den Histogrammen der Verteilung der Krümmungswerte deutlich (Abbbildung 6.9). Das Histogramm zeigt bei dem Marching Cubes-Ergebnis einige Häufungspunkte. Dies resultiert aus der begrenzten Zahl von Fällen zur Rekonstruktion der Teiloberfläche innerhalb einer Zelle bei binären Daten. Die generierten Dreiecke können nur eine von wenigen Neigungen einnehmen. Die hohen Krümmungswerte sowie deren starke Variation drücken sich in dem relativ hohen Mittelwert von 1,23 und einer hohen Standardabweichung von 0,94 aus. Das Histogramm für die MPU Implicits-basierte Rekonstruktion weist hingegen einen kontinuierlichen Verlauf auf. Dies ist für das dargestellte Gefäßsystem zu erwarten, da die Radien der Gefäße langsam variieren. Die Verteilung der Krümmungswerte entspricht also in etwa der Verteilung der Radien. Das Histogramm zeigt, dass weit mehr Punkte mit geringen Krümmungswerten vorhanden sind. Das ist darauf zurückzuführen, dass für die beschreibung von Gefäßabschnitten mit großen Radien weitaus mehr Punkte benötigt werden.

[1] www.amiravis.com; letzter Stand: 12.07.2006

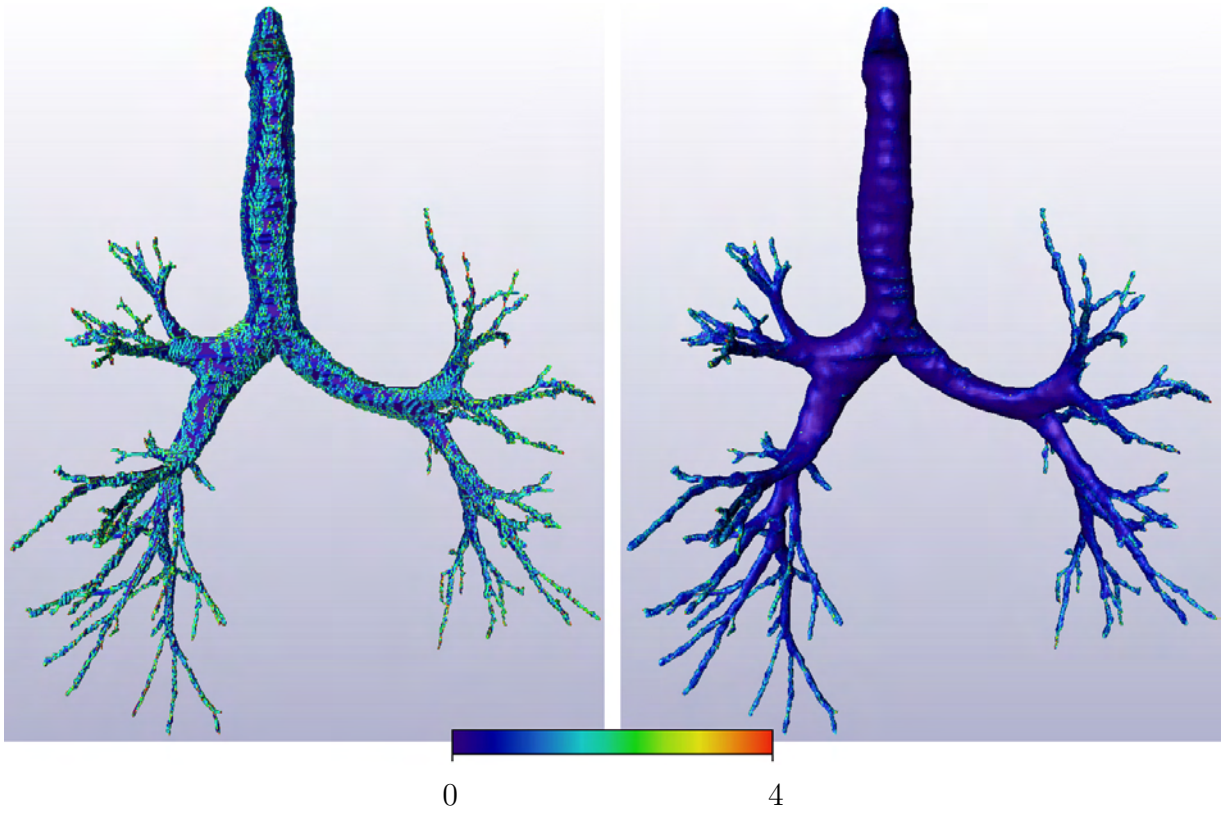


Abbildung 6.8: Visualisierung der Krümmung: das Ergebnis des Marching Cubes-Verfahrens (*links*) im Vergleich mit dem Ergebnis der MPU Implicits-basierten Rekonstruktion (*rechts*). Die Krümmung der Oberfläche ist durch Farben kodiert. Die verwendete Skala ist in der Mitte unten zu sehen. Blau steht für eine geringe Krümmung.

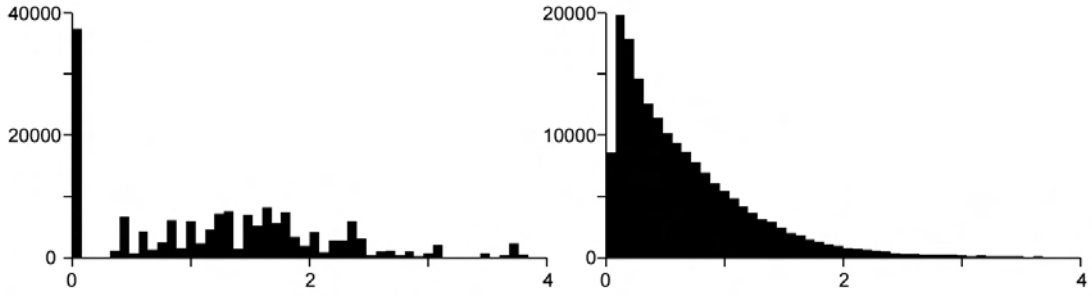


Abbildung 6.9: Histogramm der Krümmungswerte für das Ergebnis des Marching Cubes-Verfahrens (*links*) und das Ergebnis der MPU Implicits-basierten Rekonstruktion (*rechts*). Auf der Abszisse sind die Krümmungswerte abgetragen.

Die hohe Qualität der mit dem vorgeschlagenen Verfahren generierten Oberfläche wird durch geringe Krümmungswerte (Mittelwert: 0,68) und eine geringe Variation (Standardabweichung: 0,61) unterstrichen.

6.1.6 Komplexitätsbetrachtungen

Tabelle 6.3 enthält die für die Berechnung benötigten Zeitspannen sowie die Anzahl der bei der Punktwolkenextraktion generierten Punkte. Zusätzlich zu den bereits genannten Berechnungszeiten sind die für die Teilprozesse benötigten Zeiträume angegeben. Überabtastung und Reduzierung treppenartiger Artefakte sind hierbei zusammengefasst, da die Überabtastung selbst nur einen minimalen Zeitraum benötigt. Aus der Tabelle wird ersichtlich, dass bei der Punktwolkengenerierung der größte Teil der Zeit auf die Positionierung der Punkte und die Berechnung der Normalen verwandt wird.

Tabelle 6.3: Die Gesamtdauer der Punktwolkenextraktion (T_{PW}) setzt sich zusammen aus der Dauer für: die Identifizierung dünner Strukturen (T_d), die Klassifizierung (T_k), die Überabtastung und Reduzierung dünner Strukturen (T_r) sowie die Platzierung der Punkte und die Ermittlung der Normalen (T_p). In Spalte 7 ist die Anzahl der generierten Punkte gegeben. Die Gesamtdauer T_{gesamt} ergibt sich aus der Gesamtdauer der Punktwolkenextraktion T_{PW} und der für die Erzeugung und Polygonalisierung der MPU Implicits benötigten Dauer T_{MPU}. Die in Klammern angegebenen Werte für T_{MPU} wurden bei der Verwendung der von Ohtake et al. [OBA+03b] vorgeschlagenen Werte für α, λ und N_{min} (siehe Tabelle 4.2) erreicht. Alle Zeitangaben sind in Sekunden gegeben.

Datensatz	T_{PW}	T_d	T_k	T_r	T_p	Punkte	T_{MPU}	T_{gesamt}
Bronchialbaum	11,1	1,5	2,2	2,3	5,1	144.160	27 (15)	38,1
Lebergefäßbaum	5,7	0,6	0,9	1,2	3,0	84.223	8 (6)	13,7
Zerebrales Gefäßsystem	8,4	1,1	1,8	1,7	3,8	104.320	20 (9)	28,4
Aneurysma	1,3	0,2	0,1	0,2	0,8	32.408	4 (2)	5,3

Die Komplexität der Erkennung der dünnen Strukturen wächst nur mit der Auflösung des Segmentierungsergebnisses. Die restlichen Teilprozesse verwenden den größten Teil der benötigten Zeit für Berechnungen auf Basis der äußeren Randvoxel. Deren Anzahl ist von dem Flächeninhalt der beschriebenen Oberfläche abhängig. Die Berechnungsdauer der Reduzierung der treppenartigen Artefakte sowie die Dauer der Platzierung der Punkte und Berechnung der Normalen steigt zudem mit der Anzahl der Voxel, die als zu einer dünnen Struktur gehörend erkannt wurden. Die für die Punktextraktion benötigte Zeitspanne ist somit von drei Faktoren abhängig: von der Auflösung des Segmentierungsergebnisses, vom Flächeninhalt der beschriebenen Oberfläche und vom Anteil der dünnen Strukturen an der Oberfläche. Die letzten beiden Faktoren charakterisieren die Komplexität der beschriebenen Oberfläche.

Der größte Teil der Gesamtdauer wird für die Generierung und Polygonalisierung der MPU Implicits benötigt. Die Wahl der Parameter α, λ und N_{min} ist auf ein qualitativ hochwertiges Ergebnis ausgelegt. Versuche haben gezeigt, dass bei Verwendung der von

Ohtake et al. [OBA+03b] vorgeschlagenen Werte (siehe Tabelle 4.2) diese Zeitspanne um fast 50% reduziert werden kann. Die entsprechenden Zeitspannen sind in Tabelle 6.3 in der Spalte T_{MPU} in Klammern angegeben. Die Nutzung dieser Parameterbelegung führt jedoch zu großen Qualitätseinbußen (siehe dazu Abbildung 4.26). Da die in der vorliegenden Arbeit vorgeschlagene Parameterbelegung bei allen Datensätzen innerhalb eines kurzen Zeitraumes zu einer hervorragenden Qualität führt, wird diese Parameterbelegung als geeignet betrachtet.

Die für die Berechnung der MPU Implicits benötigte Dauer ist von der Komplexität der beschriebenen Oberfläche abhängig, denn für die Rekonstruktion von Details muss der Octree tiefer unterteilt werden als für die Rekonstruktion von großen Flächen [OBA+03b].

Da die für die Generierung der MPU Implicits benötigte Zeitspanne weitaus höher ist als die Dauer der Punktwolkenextraktion, ist die Berechnungskomplexität der vorgestellten Gefäßrekonstruktionsmethode vorrangig von der Komplexität des beschriebenen Gefäßes abhängig.

In Tabelle 6.4 ist ein Vergleich der benötigten Rechendauer mit dem Marching Cubes-Verfahren sowie der Gefäßrekonstruktion mit Convolution Surfaces gegeben. Die Werte für die Convolution Surfaces wurden hierbei ohne die Generierung eines Gefäßmodells gemessen. Eine Rekonstruktion des Aneurysmas war mit Convolution Surfaces nicht möglich. Die Genauigkeit der Zeitmessung für die Rekonstruktion mit Marching Cubes und Convolution Surfaces ist auf Sekunden beschränkt.

Tabelle 6.4: Vergleich der Berechnungsdauer für die Rekonstruktion mit Marching Cubes, Convolutions Surfaces und MPU Implicits. Alle Zeitangaben erfolgen in Sekunden.

Datensatz	Marching Cubes	Convolution Surfaces	MPU Implicits
Bronchialbaum	3	36	38,1
Lebergefäßbaum	2	15	13,7
Zerebrales Gefäßsystem	3	9	28,4
Aneurysma	1	-	5,3

Das Marching Cubes-Verfahren benötigt bei allen Datensätzen die geringste Zeitspanne für die Rekonstruktion. Die Zeitspannen zur Rekonstruktion mit Convolution Surfaces sowie MPU Implicits sind für einige Datensätze ähnlich. Die Rekontruktion des zerebralen Gefäßbaumes erfolgt mit Convolution Surfaces jedoch weitaus schneller, obwohl dieses Gefäß ähnlich komplex wie der Bronchialbaum zu sein scheint. Bei diesem Datensatz zeigt sich vermutlich besonders deutlich die Wirkung der in [Oel04] vorgeschlagenen Optimierung der Funktionswertberechnung bei der Polygonalisierung von Convolution Surfaces. Dabei werden Hüllkörper benutzt, um die Anzahl der bei der Funktionswertbestimmung zu betrachtenden Liniensegmente zu reduzieren. Wo sich Hüllkörper überschneiden benötigt die Funktionswertbestimmung mehr Zeit, da mehr Liniensegmente betrachtet werden müssen. Dies ist vor allem an Verzweigungen der Fall. Die Anzahl der Verzweigungen ist bei dem zerebralen Gefäßsystem relativ gering. Zudem enthält es sehr lange, dünne Gefäße, in deren näheren Umgebung keine anderen Gefäße liegen. Die geringe Berechnungsdauer bei der Rekonstruktion dieses Gefäßsystems mit Convolution Surfaces ist also

darauf zurückzuführen, dass es bei einem großen Teil der zu berechnenden Funktionswerte zu keiner Überschneidung der Hüllkörper kommt.

Generell sind die benötigten Zeitspannen aufgrund der unterschiedlichen Herangehensweise der Verfahren nicht vergleichbar. Aus der Tabelle wird aber ersichtlich, dass sich die Berechnungszeiten für das vorgeschlagene Verfahren in einem ähnlichem Bereich bewegen wie bei der Rekonstruktion mit Convolution Surfaces.

6.1.7 Auflösung des Polygongitters

In Tabelle 6.5 ist ein Vergleich der Komplexität der erzeugten Polygonnetze für Marching Cubes, Convolution Surfaces und MPU Implicits gegeben. Die Anzahl der generierten Polygone bleibt bei allen drei Verfahren in einem Rahmen, der eine Darstellung in Echtzeit erlaubt. Alle erzeugten Oberflächen können bei Nutzung einer über 3 Jahre alten *ATI Radeon 9600*-Grafikkarte interaktiv betrachtet werden.

Tabelle 6.5: Komplexität der erzeugten Polygonnetze.

Datensatz	Marching Cubes	Convolution Surfaces	MPU Implicits
Bronchialbaum	165.544	201.464	179.626
Lebergefäßbaum	80.020	124.928	166.748
Zerebrales Gefäßsystem	114.804	92.900	141.758
Aneurysma	53.948	-	61.314

Keines der Verfahren hebt sich in Bezug auf die Polygonanzahl stark von den anderen Verfahren ab. Dies ist darauf zurückzuführen, dass sich bei allen drei Verfahren die Erzeugung der polygonalen Oberfläche an der Größe der Voxel orientiert. Die für das Marching Cubes-Verfahren benutzten Zellen haben die Größe eines Voxels (siehe Abschnitt 3.1.1). Die Gefäßrekonstruktion auf Basis von Convolution Surfaces nutzt genau wie die MPU Implicits-basierte Rekonstruktion einen Bloomenthal-Polygonizer für die Überführung der impliziten Funktion in ein Polygonnetz. Die für die Polygonalisierung benutzte Zellgröße wird für beide Verfahren automatisch auf Basis der Voxelgröße bestimmt.

Die Ergebnisse zeigen, dass das Verfahren hinsichtlich der Menge der erzeugten Polygone mit etablierten Verfahren vergleichbar ist. Die Bestimmung der für die Polygonalisierung genutzten Zellgröße erfüllt somit ihren Zweck und erlaubt die effiziente Nutzung des Bloomenthal-Polygonizers.

Die Qualität der durch den Bloomenthal-Polygonizer generierten Oberfläche leidet jedoch unter vielen degenerierten Dreiecken. Zudem führt die nicht-adaptive Natur dieses Polygonalisierungsverfahrens zu einer wenig effektiven Nutzung der Polygone. Die Polygone haben überall etwa die selbe Größe. Strukturen mit einer kleinen Krümmung werden mit genauso vielen Polygonen dargestellt wie Strukturen mit einer großen Krümmung. Abbildung 6.10 zeigt das Polygonalisierungsergebnis des Bronchialbaums. Die gleichbleibende Größe der Polygone ist deutlich zu erkennen. Eine effizientere Darstellung wäre durch Nutzung einer adaptiven Polygonalisierung möglich. So könnten kleine Zweige feiner polygonalisiert werden, ohne die Gesamtanzahl der Polygone in die Höhe zu treiben. Ein vielversprechendes Verfahren hierfür ist die adaptive Polygonalisierung nach Araujo [AJ04], die bereits erfolgreich auf MPU Implicits angewendet wurde.

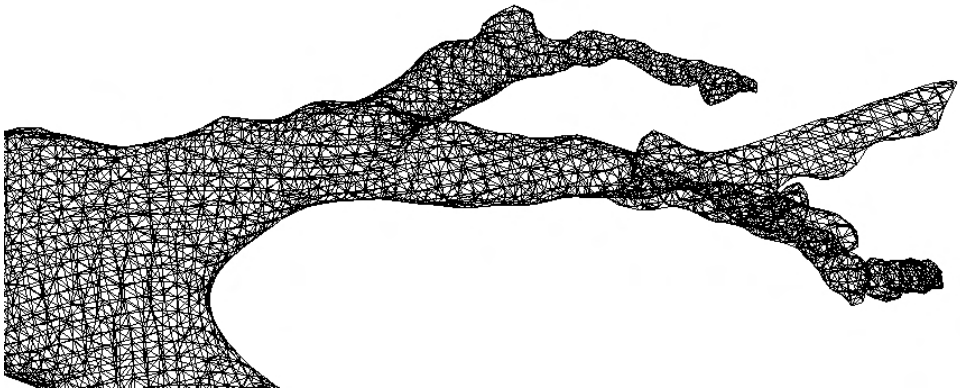


Abbildung 6.10: Die Nutzung eines Bloomenthal-Polygonizer führt zu einer krümmungsunabhängigen Polygonalisierung. Die Größe der Polygone ist bei leicht und stark gekrümmten Strukturen gleich.

6.1.8 Rekonstruktion anderer anatomischer Strukturen

Da das Verfahren nicht modellbasiert arbeitet, sondern sich ausschließlich an den Volumendaten orientiert, kann jegliche Morphologie rekonstruiert werden. Daraus folgt, dass das Verfahren auch zur Rekonstruktion anderer Objekte auf Basis von Volumendaten geeignet ist. In Abbildung 6.11 ist das Ergebnis der MPU Implicits-basierten Rekonstruktion eines Lungenflügels zu sehen. Der Datensatz ist mit $266 \times 332 \times 438$ Voxeln sehr groß.

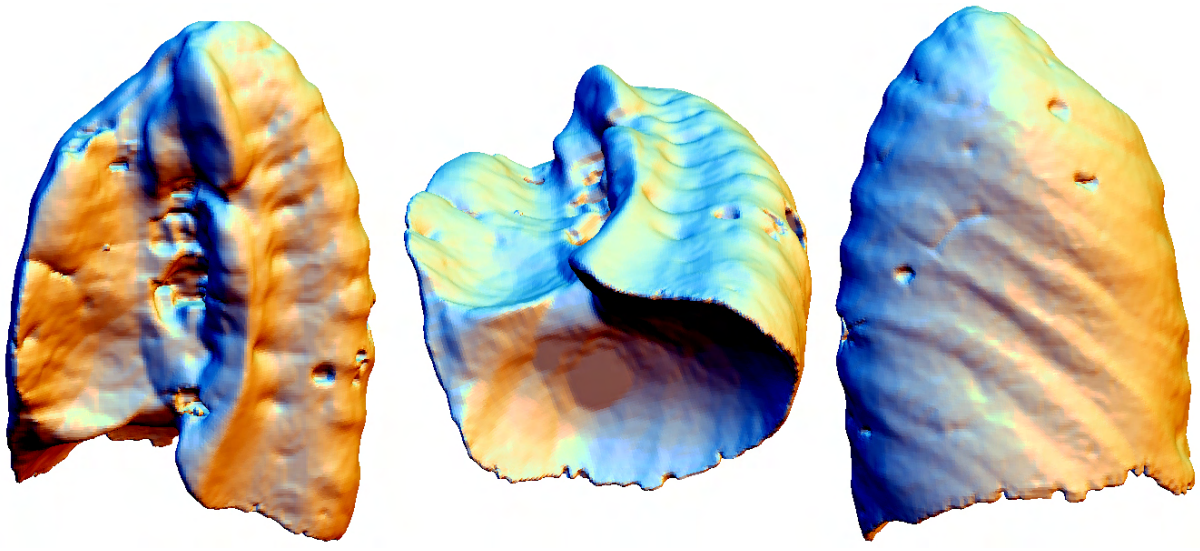


Abbildung 6.11: Rekonstruktion eines Lungenflügels mit dem vorgestellten Verfahren.

Die Generierung der Punktwolke benötigt 28,1 Sekunden. Die erzeugte Punktwolke enthällt 286848 Punkte. Die ermittelten Parameter können auch bei diesem Beispiel genutzt werden. Lediglich die Zellgröße für den Polygonalisierungsvorgang wird erhöht, da keine kleinen Details enthalten sind, die eine solch feine Polygonalisierung erfordern würden. Die Rekonstruktion der Oberfläche nimmt 36 Sekunden in Anspruch und resultiert in einem aus 256216 Polygonen bestehenden Polygonnetz.

Die Oberfläche des Lungenflügels weist eine vergleichsweise einfache Topologie auf. Dennoch ist die Berechnungsdauer weitaus höher als bei den komplexesten getesteten Gefäßbäumen. Deswegen wäre für solche Datensätze, die keine dünnen Strukturen enthalten, evtl. die Herleitung einer anderen Parameterbelegung sinnvoll.

6.2 Validierung

Zur Überprüfung der Genauigkeit des Verfahrens wird die Abweichung des Ergebnisses zum entsprechenden Segmentierungsergebnis gemessen. Das Segmentierungsergebnis wird hierbei durch das Resultat des Marching Cubes-Algorithmus ohne nachträgliche Glättung repräsentiert und im Folgenden als *MC-Oberfläche* bezeichnet. Das Ergebnis der MPU Implicits-basierten Rekonstruktion erhält analog die Bezeichnung *MPUI-Oberfläche*.

Darüber hinaus wird das Verfahren in Hinsicht auf die erreichte Genauigkeit mit der Oberflächenrekonstruktion auf Basis von Convolution Surfaces (im Folgenden *CS-Oberfläche*) verglichen. Dazu wird ebenfalls die Abweichung der CS-Oberfläche zu der MC-Oberfläche gemessen und der Abweichung zur MPUI-Oberfläche gegenübergestellt.

Für die Messung der Abweichung wird der lokale Ortsfehler zwischen MC-Oberfläche und MPUI-Oberfläche sowie zwischen MC-Oberfläche und CS-Oberfläche mit Hilfe von Distanzmessungen bestimmt. Dies erfolgt wiederum in *AMIRA*. Mit dem Modul *Surface-Distance* kann die Entfernung zwischen zwei Oberflächen gemessen werden. Dazu wird für jeden Punkt auf der ersten Oberfläche der nächstgelegene Punkt auf der zweiten Oberfläche ermittelt und die entsprechende Entfernung gespeichert. Als erste Oberfläche wird hierbei die MC-Oberfläche gewählt, da sie jedes Detail des Segmentierungsergebnisses enthält. Wird die Messung in diese Richtung durchgeführt, werden auch Ausreißer erkannt, also die Teile der zweiten Oberfläche, die stark von der MC-Oberfläche abweichen. Würde die Messung in die andere Richtung durchgeführt, würden für diese stark abweichenden Punkte der zu bewertenden Oberfläche die nächstgelegenen Punkte der MC-Oberfläche für die Distanzbestimmung genutzt. Somit würde die eigentlich größere Entfernung zu den weiter entfernten Punkten des unzureichend genau rekonstruierten Details nicht mit ausgewertet. Dieser Sachverhalt ist in Abbildung 6.12 verdeutlicht.

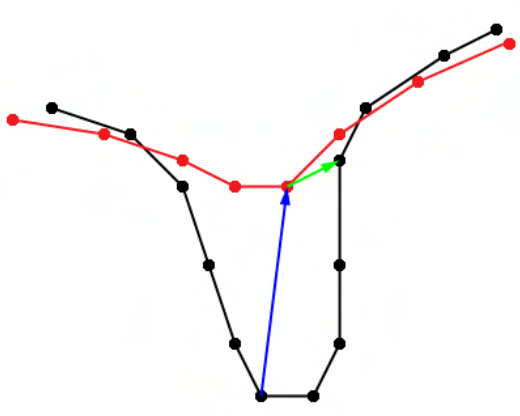


Abbildung 6.12: Eine Distanzmessung zwischen der MC-Oberfläche (*schwarz*) und der zu bewertetenden Oberfläche (*rot*) kann in zwei Richtungen durchgeführt werden. Die Messung sollte von der MC-Oberfläche aus erfolgen, da nur so die maximale Abweichung ermittelt werden kann (*blauer Pfeil*). Bei einer Messung in die andere Richtung würde die Entfernung zwischen zwei anderen Punkten bestimmt und die maximale Abweichung nicht erfasst werden (*grüner Pfeil*).

Die Ergebnisse der Messungen für die vier beschriebenen Datensätze sind in Tabelle 6.6 aufgelistet. Die angegebenen Werte sind in Bezug auf die Länge der Diagonale eines Voxels angegeben, da die Platzierung der Punkte ausschließlich im Voxelkoordinatensystem

erfolgt und sich die Bestimmung der Parameter für die Erstellung und Polygonalisierung der MPU Implicits ebenfalls auf die Diagonale eines Voxels bezieht. Abstandsangaben in Millimeter würden wenig Sinn machen, da die Stärke der Abweichung auch von der Größe der Voxel abhängig ist.

Tabelle 6.6: Ergebnisse der Distanzmessungen: Alle Angaben beziehen sich auf die Länge einer Voxeldiagonale. In Spalte eins sind die Namen der Datensätze gegeben. Die Spalten zwei und drei enthalten den Mittelwert ϕ und die Standardabweichung σ. Die nächsten drei Spalten zeigen den mittleren quadratischen Fehler, den Median und den Maximalwert. Die letzte Spalte enthält den Prozentsatz der Punkte, die weiter als eine halbe Voxeldiagonale von der MC-Oberfläche entfernt sind. Die Werte in Klammern beziehen sich auf die CS-Oberfläche.

Datensatz	ϕ MPUI (CS)	σ MPUI (CS)	Mittl. quadr. Fehler MPUI (CS)	Median MPUI (CS)	Max MPUI (CS)	$> D_{thresh}$ MPUI (CS)
Bronchialbaum	0,171 (0,537)	0,114 (0,57)	0,205 (0,783)	0,158 (0,37)	1,398 (5,534)	0,69% (37,9%)
Lebergefäßbaum	0,165 (0,425)	0,114 (0,519)	0,201 (0,671)	0,151 (0,298)	0,844 (6,868)	0,82% (27,4%)
Zerebrales Gefäßsystem	0,201 (0,393)	0,129 (0,335)	0,239 (0,517)	0,195 (0,315)	1,677 (2,697)	1,70% (28,8%)
Aneurysma	0,214 (-)	0,164 (-)	0,269 (-)	0,189 (-)	1,912 (-)	4,12%/ - (-)
Durchschnitt	0,188 (0,452)	0,130 (0,475)	0,229 (0,657)	0,173 (0,328)	1,458 (5,033)	1,84% (31,4%)

Ein wichtiger Wert dieser Tabelle ist die durchschnittliche Abweichung der getesteten Oberfläche von der MC-Oberfläche. Diese Abweichung ist bei den CS-Oberflächen mehr als doppelt so hoch wie bei den MPUI-Oberflächen. Die durchschnittliche Abweichung der MPUI-Oberfläche von der MC-Oberfläche ist mit $0,188$ Voxeldiagonalen äußerst gering. Diese Abweichung ist auf das durch die Veränderung des Isowerts resultierende Wachstum der MPUI-Oberfläche (siehe Abschnitt 4.3.2) sowie die treppenförmigen Artefakte der MC-Oberfläche zurückzuführen.

Zur Erfassung von Ausreißern, also Punkten mit extremen Abweichungen, wird der Schwellwert D_{thresh} als die Hälfte der Diagonale eines Voxels definiert. Der Anteil der Ausreißer ist bei den CS-Oberflächen mit über 30% wiederum weitaus höher als bei den MPUI-Oberflächen $(1,84\%)$.

Bei den MPUI-Oberflächen ist bei dem Anteil der Ausreißer trotz des niedrigen Durchschnitts eine hohe Schwankung zu erkennen. Bei den ersten beiden Datensätzen sind weniger als ein Prozent aller Punkte der MPUI-Oberfläche weiter als D_{thresh} von der MC-Oberfläche entfernt. Bei den letzten beiden Datensätzen ist der Anteil der Ausrei-

ßer größer. Bei dem zerebralen Gefäßsystem ist dies auf die Verschmelzung benachbarter Gefäße zurückzuführen. Die größte Anzahl von Ausreißern ist bei dem Aneurysma zu beobachten. Der Grund hierfür sind die bereits erwähnten Vertiefungen (Abbildung 6.13). Die Ursachen sind bei beiden Datensätzen die gleichen: enge Zwischenräume zwischen Strukturen werden nicht durch genügend Punkte beschrieben. Zudem schrumpfen Vertiefungen durch das Wachstum des Objektes. Dieses Wachstum sollte dazu dienen, dünne Zweige bei der Polygonalisierung zu erhalten. Das Schrumpfen von Vertiefungen ist eine natürliche Folge. Dies ist für das relevante Anwendungsgebiet, nämlich die Gefäßsdiagnose, jedoch nicht von Bedeutung.

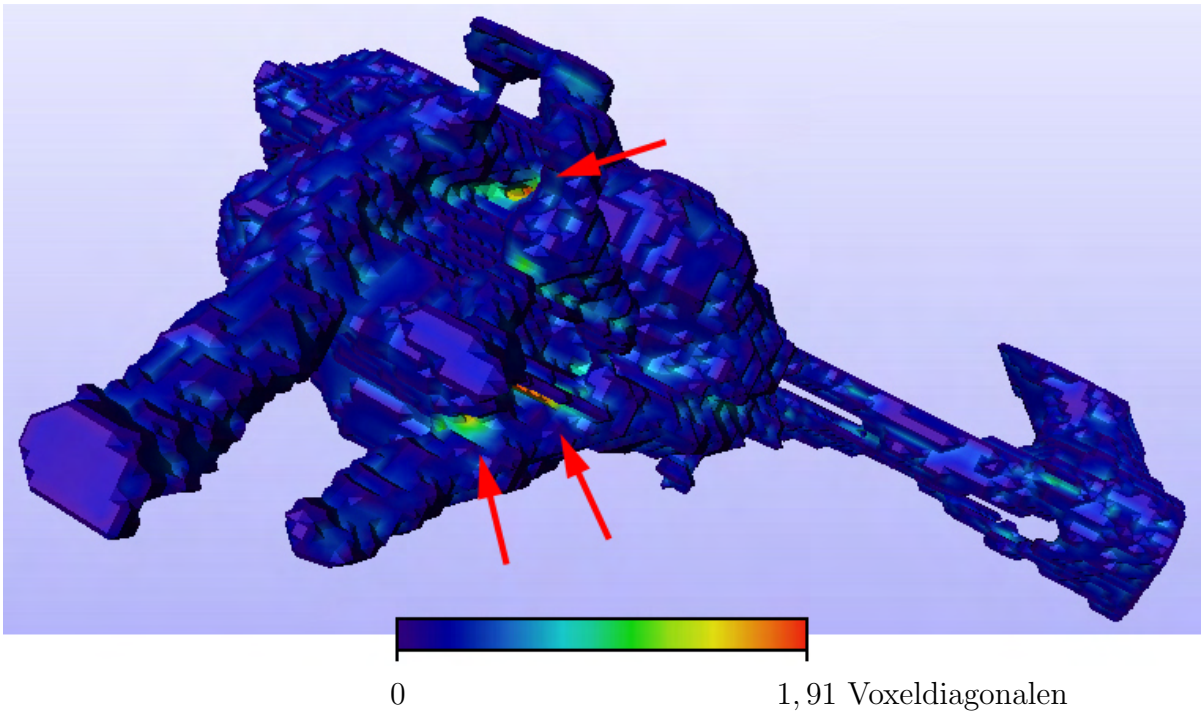


Abbildung 6.13: Visualisierung der Distanzmessungen auf der Oberfläche des Aneurysmas: die Abweichungen sind in kleinen Vertiefungen besonders stark.

Die gemessenen Werte zeigen, dass die MPUI-basierte Rekonstruktion weitaus genauer ist als die Rekonstruktion mit Convolution Surfaces. Dies resultiert aus dem Verzicht auf eine Modellannahme und die Orientierung des Verfahrens an den Volumendaten.

7 Zusammenfassung und Ausblick

Für die Gefäßvisualisierung im Anwendungsbereich der Therapieplanung und medizinischen Ausbildung sind *modellbasierte Gefäßvisualisierungsverfahren* hervorragend geeignet. Das Gefäßmodell, das ihnen zugrunde liegt, ermöglicht verschiedenste analytische und explorative Anwendungen. Ein Nachteil dieser Verfahren ist die oftmals wenig natürlich wirkende Darstellung der Gefäßstrukturen. Neuere Verfahren, wie zum Beispiel die Gefäßrekonstruktion mit *Convolution Surfaces*, widmen sich speziell der Rekonstruktion organisch wirkender Gefäßoberflächen. Für die Diagnostik sind modellbasierte Gefäßvisualisierungsverfahren ungeeignet, weil sie die Patientendaten zu ungenau wiedergeben und somit pathologische Veränderungen der Gefäße nicht kommunizieren können. Damit ein Visualisierungsverfahren im Bereich der Gefäßdiagnostik erfolgreich eingesetzt werden kann, muss es das Segmentierungsergebnis möglichst genau wiedergeben. Diese Anforderung erfüllt beispielsweise der *Marching Cubes*-Algorithmus. Die von diesem Verfahren erzeugte Oberfläche weist jedoch starke treppenförmige Artefakte auf. Eine nachträgliche Glättung ist vor allem bei Gefäßsystemen äußerst problematisch. Dünne Zweige können hierbei entfernt werden.

Ziel der vorliegenden Arbeit war deshalb die Konzipierung und Umsetzung eines Gefäßvisualisierungsverfahrens, das sich möglichst nah am Segmentierungsergebnis orientiert und eine Oberfläche erzeugt, deren Qualität ähnlich hoch ist wie bei der modellbasierten Gefäßrekonstruktion mit Convolution Surfaces. Durch Abbildung der generierten Oberfläche auf ein Gefäßmodell sollte das Verfahren zudem für die angesprochenen analytischen und explorativen Aufgaben nutzbar werden und sich somit sowohl für die Gefäßdiagnostik als auch für die Therapieplanung und medizinische Ausbildung eignen.

Für die Generierung der Gefäßoberfläche wurde mit *MPU Implicits* [OBA+03b] eine implizite Oberflächenrekonstruktionsmethode gewählt, die ein Objekt auf Basis einer Punktwolke rekonstruiert. Die Genauigkeit und Glattheit kann dabei durch eine Vielzahl von Parametern beeinflusst werden.

Ein entscheidender Beitrag der vorliegenden Arbeit liegt in der Entwicklung eines Algorithmus zur Extraktion einer Punktwolke auf Basis des Segmentierungsergebnisses, die als Eingabe für die Oberflächenrekonstruktion mit MPU Implicits dient. Die Grundlage für die Platzierung der Punkte bildet das Voxelgitter des Segmentierungsergebnisses. Das Verfahren ähnelt der in [Bra05] vorgestellten Rekonstruktionsmethode, die jedoch nicht auf die Rekonstruktion von Gefäßsystemen ausgerichtet ist. Die Platzierung der Punkte orientiert sich in der vorliegenden Arbeit an der Anforderung, auch dünne Strukturen zu beschreiben. Um eine artefaktfreie Rekonstruktion der dünnen Strukturen durch MPU Implicits zu ermöglichen, müssen diese Strukturen durch eine ausreichende Anzahl Punkte beschrieben werden. Hiefür wurde eine adaptive Überabtastung des Voxelgitters entwickelt. Ein höher aufgelöstes Voxelgitter wird hierbei für die Platzierung der Punkte genutzt, die dünne Strukturen beschreiben. Bei der Platzierung der Punkte wird besondere Sorgfalt darauf verwandt, die Punkte nicht treppenförmig anzuordnenn. Hierfür werden dem höher aufgelösten Voxelgitter Subvoxel hinzugefügt. Darüber hinaus wird auf Basis

der lokalen Nachbarschaft entschieden, wo die Punkte innerhalb eines Voxels platziert werden.

Für die Abbildung der generierten Oberfläche auf ein Gefäßmodell wurde der als *Registrierung* bezeichnete Prozess vorgeschlagen. Ein möglicher Algorithmus wurde skizziert, konnte im Rahmen der zur Verfügung stehenden Zeit jedoch nicht implementiert werden.

Im Rahmen der Arbeit wurde das Verfahren prototypisch umgesetzt. Die Ergebnisse zeigen, dass das Verfahren die in der Einleitung aufgestellten Anforderungen erfüllt. Die rekonstruierte Oberfläche wirkt organisch. Dieser subjektive Eindruck wurde durch einen Vergleich der Glattheit mit dem entsprechenden Marching Cubes-Ergebnis untermauert.

Dünne Zweige, Verzweigungen und pathologische Veränderungen werden korrekt rekonstruiert. Um die Abweichung vom Segmentierungsergebniss zu messen, wurden im Rahmen einer Validierung die Oberflächen von vier Datensätzen mit den entsprechenden Marching Cubes-Ergebnissen verglichen. Die durchschnittliche Abweichung liegt bei 0,188 Voxeldiagonalen und ist somit äußerst gering. Die durchschnittliche Abweichung der Convolution Surfaces zu den Marching Cubes-Ergebnissen entspricht fast einer halben Voxeldiagonale.

Durch die vorgeschlagene automatische Parameterberechnung entfällt die sonst zeitaufwändige manuelle Anpassung der Parameterwerte für die Erzeugung und Polygonalisierung der MPU Implicits für verschiedene Datensätze. Die von dem Verfahren benötigte Berechnungsdauer liegt im Bereich von Sekunden. Ein Einsatz des Verfahrens im klinischen Alltag scheint daher möglich.

Bereits während der Konzeptuierung des Verfahrens, als auch bei der Umsetzung wurden einige Aspekte deutlich, die Potential für Verbesserungen oder Weiterentwicklungen bieten:

Einsatzbereich

Das beschriebene Verfahren verzichtet auf die Nutzung von Modellannahmen, um jegliche Morphologie darstellen zu können. Es kann somit als ein allgemeines Verfahren zur Oberflächenrekonstruktion auf Basis von Volumendaten betrachtet werden. Somit ergibt sich auch die Möglichkeit, das Verfahren auf andere anatomische Strukturen anzuwenden.

In diesem Fall wäre es wünschenswert, die adaptive Überabtastung zu deaktivieren. Ein vollkommen automatischer Prozess wäre denkbar, wenn auf Basis morphologischer Operatoren ermittelt würde, ob überhaupt dünne Strukturen vorhanden sind. Im Fall des Nichtvorhandenseins dünner Strukturen könnten zudem andere Parameterbelegungen vorgeschlagen werden, da die in dieser Arbeit beschriebene Parameterberechnung auf die Rekonstruktion dünner Strukturen ausgerichtet ist.

Rekonstruktion dünner Zweige

Auch wenn die Rekonstruktion dünner Zweige von dem Verfahren verhältnismäßig gut gehandhabt wird, wurden doch einige Probleme deutlich. Zum einen wird teilweise eine leicht treppenförmige Geometrie erzeugt, die nicht störend, aber dennoch nicht für Gefäße typisch ist. Zum anderen können dünne Zweige in der impliziten Funktion als abgetrennt rekonstruiert werden. Um eine zusammenhängende Polygonalisierung zu ermöglichen musste deswegen der Isowert angepasst werden, wodurch die Genauigkeit des Verfahrens reduziert wurde.

Eine qualitativ hochwertigere Rekonstruktion dieser feinen Strukturen könnte durch eine Kombination von MPU Implicits und Convolution Surfaces ermöglicht werden. Zweige mit einem geringen Durchmesser würden hierbei durch die Convolution Surfaces rekonstruiert, das restliche Volumen durch MPU Implicits. Ein Blending der beiden Repräsentation könnte auf Basis von *Partition of Unity*-Funktion erfolgen, die den Convolution Surfaces zugewiesen werden.

Auf diese Weise würden dünne Zweige zwar wiederum auf Basis der Annahme kreisrunder Gefäßquerschnitte rekonstruiert, allerdings ist die Genauigkeit der Segmentierung für solche Strukturen auf Grund des Partialvolumeneffekts anzweifelbar. Ein genauer Querschnitt kann hier durch die Volumendaten somit nicht mehr transportiert werden. Zudem sind dünne Strukturen diagnostisch weniger relevant, so dass eine genaue Wiedergabe des Querschnitts für solche Strukturen nicht notwendig ist.

Rekonstruktion von Vertiefungen

Eine Einschränkung bei der darstellbaren Morphologie betrifft kleine Löcher und Vertiefungen mit einem Querschnitt von ein bis zwei Voxeln. Wenn die umgebende Struktur nicht als dünn klassifiziert wurde, wird das Loch durch sehr wenige Punkte beschrieben. Dieser Fall ist vergleichbar mit dünnen Zweigen. Die adaptive Überabtastung könnte so angepasst werden, dass auch Vertiefungen, die nur wenige Voxel groß sind, durch mehr Punkte beschrieben werden. Dies hätte die genauere Rekonstruktion dieser Vertiefungen zur Folge. Es wurde jedoch kein Aufwand betrieben, Vertiefungen mit mehr Punkten zu beschreiben, da dieser Fall für die Darstellung von Gefäßen nur geringe Relevanz besitzt.

Polygonalisierung

Der verwendete Polygonlisierungsalgorithmus weist einige Nachteile auf. Es wurde bereits darauf eingegangen, dass der Bloomenthal-Polygonizer nur eine zusammenhängende Struktur polygonalisieren kann. Es wurde ein Weg beschrieben, wie auch mehrere in einem Datensatz enthaltene Gefäßbäume rekonstruiert werden können. Da die Einschränkung eine Folge der Polygonalisierung ist, wäre es jedoch sinnvoller, die Polygonalisierung anzupassen oder gar ein anderes Verfahren zu wählen.

Eine Anpassung des Bloomenthal-Polygonizers würde eine Durchführung des Polygonalisierungsvorgangs für jede Struktur erfordern. Hierfür müsste jeweils für jede Struktur ein Saatpunkt ermittelt werden. Die Identifizierung der Saatpunkte könnte im Segmentierungsergebnis mit *Connected Component Labeling* erfolgen. Auch die in Abschnitt 4.4 beschriebene Datenstruktur *MLGraph* scheint zur Ermittlung der Saatpunkte geeignet, da sie auch mehrere voneinander unabhängige Gefäßbäume beschreiben kann.

Ein weiterer Nachteil ist die krümmungsunabhängige Polygonalisierung. Die Nutzung einer adaptiven Polygonalisierung wäre wünschenswert, da so die Anzahl der erzeugten Polygone reduziert werden könnte, ohne, dass dünne Zweige durch zu wenig Polygone beschrieben würden.

Eine weiterführende Betrachtung dieser Aspekte scheint sinnvoll und könnte vor allem den Einsatzbereich des beschriebenen Verfahrens erweitern.

Abbildungsverzeichnis

1.1 Beispiel für die Visualisierung mit MPU Implicits 3

2.1 Morphologie von Bronchialbäumen 6
2.2 Querschnittsänderung in Abhängigkeint vom transmuralen Druck 6
2.3 Form von Aneurysmen 1 7
2.4 Form von Aneurysmen 2 7
2.5 Querschnittsveränderung durch Stenosen 8
2.6 Direct Volume Rendering und Surface Rendering 9
2.7 Bestimmung des optimalen Schwellwertes 12
2.8 Beispiele für Pruning 13
2.9 Glättung des Skeletts 13
2.10 Ablauf der Gefäßrekonstruktion 14
2.11 Visualisierung mit einfachen Primitiven 1 15
2.12 Visualisierung mit einfachen Primitiven 2 15
2.13 Topologische Dualität bei Simplex Meshes 16
2.14 Erzeugung eines Simplex Meshes 16
2.15 Beispielhafte Visualisierung mit Simplex Mesh 17
2.16 Rekonstruktion einer Trifurkation mit SDS 18
2.17 Ergebnisse einer Visualisierung mit SDS 18
2.18 Genauigkeitstest SDS 19
2.19 Visualisierung mit Convolution Surfaces 20
2.20 Vergleich Convolution Surface mit Isosurface 21
2.21 Visualisierung mit Freiformflächen 22
2.22 Verwendung von Textur in der Gefäßvisualisierung 23

3.1 Glättung eines Polygonalisierungsergebnisses 26
3.2 Glättung eines Constrained Elastic Surface Nets 27
3.3 Constrained Elastic Surface Nets: Ergebnisse in 2D 27
3.4 Constrained Elastic Surface Nets: Ergebnisse in 3D 28
3.5 Beispiel für Quadriken 30
3.6 Beispiel für Blobby Objects 31
3.7 Adaptive Polygonalisierung nach Araújo 1 34
3.8 Adaptive Polygonalisierung nach Araújo 2 34
3.9 Rekonstruktion einer Niere mit Impliziten Oberflächen 35
3.10 Rekonstruktion eines verzweigten Objekts mit Impliziten Oberflächen . . . 36
3.11 Rekonstruktion einer Leber mit Impliziten Oberflächen 36
3.12 Ergebnisse einer Rekonstruktion mit MPU Implicits 37
3.13 Überblendung lokaler Approximationen 42
3.14 Anpassen einer Quadrik 45
3.15 Lokale Approximation 46

3.16 Auswirkung der Parameter 1 47
3.17 Auswirkung der Parameter 2 48
3.18 Auswirkung der Parameter 3 48
3.19 Auswirkung der Parameter 4 49
3.20 Anforderung an Punktwolke 49
3.21 Anwendungsbeispiele für MPU Implicits 50

4.1 Überblick über Visualisierungspipeline 52
4.2 Beispiel für die Bildung von Artefakten an dünnen Strukturen 53
4.3 Überblick über die Punktextraktion 55
4.4 Opening angewendet auf Gefäßbaum 56
4.5 Nachbarschaft in einem Voxelgitter 57
4.6 Klassifizierung der Voxel 57
4.7 Überabtastung des Datensatzes 58
4.8 Folgen der Überabtastung 59
4.9 Sonderfälle bei der Reduzierung von treppenartigen Artefakten 60
4.10 Fälle beim Setzen von Subvoxeln 62
4.11 Setzen eines Subvoxels in einer diagonalen Stufe 62
4.12 Setzen eines Subvoxels in einer direkten Stufe 62
4.13 Beispiele für das Setzen von Subvoxeln 62
4.14 Platzierung der Punkte 1 65
4.15 Platzierung der Punkte 2 65
4.16 Platzierung der Punkte 3 65
4.17 Platzierung der Punkte 4 65
4.18 Platzierung der Punkte 5 66
4.19 Alternative Punktplatzierung 1 66
4.20 Alternative Punktplatzierung 2 66
4.21 Grauwertgradient eines Voxels 67
4.22 Auswirkung ungenauer Normalen auf die Rekonstruktion 68
4.23 Wichtungen der Voxel bei der Berechnung des Grauwertgardienten 68
4.24 Bestimmung der Normalenvektoren bei multiplen Punkten 70
4.25 Skalierung der Normalenvektoren 71
4.26 Parameterbelegung 1 73
4.27 Parameterbelegung 2 74
4.28 Offset-Oberflächen durch unterschiedliche Isowerte 75
4.29 Parameterbelegung 3 76
4.30 Registrierung: Repräsentation des Segmentierungsergebnisses durch ML-Graph 77
4.31 Zwischenergebnisse der Pipeline 78

5.1 Beispiel für die Visuelle Programmierung in MeVisLab 81
5.2 Implementierung der Pipeline 82
5.3 GUI von MPUPrepare 83
5.4 Umwandlung der Voxelkoordinaten in eindimensionale Indizes 83
5.5 GUI des MPUPI-Programms 91
5.6 Modul PolyImport 93

6.1 Ergebnis: Bronchialbaum 96
6.2 Ergebnis: Lebergefäßbaum 97
6.3 Ergebnis: Zerebrales Gefäßsystem 1 98
6.4 Ergebnis: Zerebrales Gefäßsystem 2 98
6.5 Ergebnis: Aneurysma 99
6.6 Darstellung von dünnen Strukturen 100
6.7 Darstellung von Verzweigungen 101
6.8 Glattheit der rekonstruierten Oberfläche 1 103
6.9 Glattheit der rekonstruierten Oberfläche 2 103
6.10 Auflösung des Polygonalisierungsergebnisses 107
6.11 Rekonstruktion eines Lungenflügels 107
6.12 Distanzmessung 108
6.13 Distanzmessung am Beispiel des Aneurysmas 110

Literaturverzeichnis

[AG01] Akkouche, S. ; Galin, E.: Adaptive Implicit Surface Polygonization using Marching Triangles. In: *Computer Graphics Forum* 20 (2001), Nr. 2, S. 67–80

[AJ04] Araújo, R. B. ; Jorge, A. J.: Curvature Dependent Polygonization of Implicit Surfaces. In: *Proceedings of the 17th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI 2004/SIACG 2004)*. Curitiba, Brasilien, Oktober 2004

[AJ05] Araújo, R. B. ; Jorge, A. J.: A Calligraphic Interface for Interactive Free-Form Modeling with Large Datasets. In: *Proceedings of the 18th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI 2005)*. Natal Brasilien, Oktober 2005

[BD04] Boshernitsan, M. ; Downes, M.: Visual Programming Languages: A Survey / University of California, Computer Science Division (EECS). Berkeley, California, Dezember 2004 (UCB/CSD-04-1368). – Forschungsbericht

[Ben75] Bentley, J. L.: Multidimensional Binary Search Trees used for Associative Searching. In: *Communications of the ACM* 18 (1975), Nr. 9, S. 509–517

[Ben79] Bentley, J. L.: Multidimensional Binary Search Trees in Database Applications. In: *IEEE Transactions on Software Engineering* 5 (1979), Juli, Nr. 4, S. 333–340

[BFC04] Bühler, K. ; Felkel, P. ; Cruz, A. L.: Geometric Methods for Vessel Visualization and Quantification - A Survey. In: G. Brunnett, H. M. (Hrsg.): *Geometric Modelling for Scientific Visualization*. Kluwer Academic Publishers, 2004, S. 399–420

[BGSC85] Barillot, C. ; Gibaud, B. ; Scarabin, J. ; Coatrieux, J.: 3d Reconstruction of Cerebral Blood Vessels. In: *IEEE Computer Graphics and Applications* 5 (1985), Nr. 12, S. 13–19

[BHP06] Bade, R. ; Haase, J. ; Preim, B.: Comparison of Fundamental Mesh Smoothing Algorithms for Medical Surface Models. In: *Simulation und Visualisierung (2006)*, SCS, 2006, S. 289–304

[Bli82] Blinn, J. F.: A Generalization of Algebraic Surface Drawing. In: *ACM Transactions on Graphics* 1 (1982), Nr. 3, S. 235–256

[Blo88] Bloomenthal, J.: Polygonization of Implicit Surfaces. In: *Computer Aided Geometric Design* 5 (1988), S. 341–355

[Blo94] Bloomenthal, J.: An Implicit Surface Polygonizer. In: *Graphics Gems IV* (1994), S. 324–349

[Blo97] Bloomenthal, J.: *Introduction to Implicit Surfaces*. San Francisco, California : Morgan Kaufmann Publishers, Inc., 1997 (The Morgan Kaufmann Series in Computer Graphics and Geometric Modelling)

[Blu67] Blum, H.: A Transformation for Extracting New Descriptors of Shape. In: Wathen-Dunn, W. (Hrsg.): *Models for the Perception of Speech and Visual Form*. Cambridge : MIT Press, 1967, S. 362–380

[Bra05] Braude, I.: *Smooth 3D Surface Reconstruction from Contours of Biological Data with MPU Implicits*. http://dspace.library.drexel.edu/handle/1860/545, Drexel University, Diplomarbeit, August 2005

[BRB05] Bornik, A. ; Reitinger, B. ; Beichel, R.: Reconstruction and Representation of Tubular Structures using Simplex Meshes. In: *Proceedings of WSCG (2005) - Short Papers*, 2005, S. 61–65

[BS91] Bloomenthal, J. ; Shoemake, K.: Convolution Surfaces. In: *Computer Graphics* 25 (1991), Nr. 4, S. 251–256

[CBC+01] Carr, J. C. ; Beatson, R. K. ; Cherrie, J. B. ; Mitchell, T. J. ; Fright, W. R. ; McCallum, B. C. ; Evans, T. R.: Reconstruction and Representation of 3D Objects with Radial Basis Functions. In: *SIGGRAPH (2001): Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press, 2001, S. 67–76

[CC78] Catmull, E. ; Clark, J.: Recursively generated B-Spline Surfaces on Arbitrary Topological Surfaces. In: *Computer-Aided Design* 10 (1978), November, Nr. 6, S. 350–355

[CW88] Cleary, J. G. ; Wyvill, G.: Analysis Of An Algorithm For Fast Ray Tracing Using Uniform Space Subdivision. In: *The Visual Computer* 4 (1988), Juli, Nr. 2, S. 65–83

[Del99] Delingette, H.: General Object Reconstruction based on Simplex Meshes. In: *International Journal on Computer Vision* 32 (1999), September, Nr. 2, S. 111–146

[DKT98] DeRose, T. ; Kass, M. ; Truong, T.: Subdivision Surfaces in Character Animation. In: *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press, 1998, S. 85–94

[DTC96] Desbrun, M. ; Tsingos, N. ; Cani, M. P.: Adaptive Sampling of Implicit Surfaces for Interactive Modeling and Animation. In: *Computer Graphics Forum* 15 (1996), Dezember, Nr. 5, S. 319–325

[DZ00] Kap. 2 In: Dawant, B.M. ; Zijdenbos, A.P.: *Handbook of Medical Imaging.* Bd. 2. SPIE Press, 2000, S. 100

[EDKS94] Ehricke, H.-H. ; Donner, K. ; Koller, W. ; Strasser, W.: Visualization of Vasculature from Volume Data. In: *Computers and Graphics* 18 (1994), Nr. 3, S. 395–406

[ES04] Kap. 5 In: Ehrlich, A. ; Schroeder, C. L.: *Medical Terminology for Health Professions.* Thomson Delmar Learning, August 2004, S. 136–137

[FCA96] Ferley, E. ; Cani, M. P. ; Attali, D.: Skeletal Reconstruction of Branching Shapes. In: Hart, J. C. (Hrsg.) ; van Overveld, K. (Hrsg.): *Implicit Surfaces.* Eindhoven, Niederlande, Oktober 1996, S. 127–142

[FDFH97] Foley, J. D. ; van Dam, A. ; Feiner, S. K. ; Hughes, J. F.: *Computer Graphics, Principles and Practice, Second Edition in C.* Addison-Wesley, 1997

[FFKW02] Felkel, P. ; Fuhrmann, A.L. ; Kanitsar, A. ; Wegenkittl, R.: Surface Reconstruction of the Branching Vessels for Augmented Reality Aided Surgery. In: *BIOSIGNAL (2002)* Bd. 16, 2002, S. 252–254

[FN80] Franke, R. ; Nielson, G. M.: Smooth Interpolation of Large Sets of Scattered Data. In: *International Journal for Numerical Methods in Engineering* 15 (1980), Nr. 11, S. 1691–1704

[Gib98] Gibson, S. F. F.: Constrained Elastic Surface Nets: Generating Smooth Surfaces from Binary Segmented Data. In: *MICCAI '98: Proceedings of the First International Conference on Medical Image Computing and Computer-Assisted Intervention.* London, UK : Springer-Verlag, 1998, S. 888–898

[GKS+93] Gerig, G. ; Koller, T. ; Székely, G. ; Brechbühler, C. ; Kübler, O.: Symbolic Description of 3-D Structures applied to Cerebral Vessel Tree obtained from MR Angiography Volume Data. In: *IPMI '93: Proceedings of the 13th International Conference on Information Processing in Medical Imaging.* London, UK : Springer-Verlag, 1993, S. 94–111

[GQ05] Guo, X. ; Qin, H.: Real-Time Meshless Deformation. In: *Computer Animation and Virtual Worlds* 16 (2005), Nr. 3-4, S. 189–200

[Haa05] Haase, J.: *Glättung von Polygonnetzen in medizinischen Visualisierungen*, Otto-von-Guericke-Universität Magdeburg, Institut für Simulation und Graphik der Fakultät für Informatik, Diplomarbeit, Dezember 2005

[Han06] Hansen, C.: *Verwendung von Textur in der Gefäßvisualisierung*, Otto-von-Guericke-Universität Magdeburg, Institut für Simulation und Graphik der Fakultät für Informatik, Diplomarbeit, April 2006

[HB86] Hoehne, K.H. ; Bernstein, R.: Shading 3D-Images from CT Using Gray-Level Gradients. In: *IEEE Transactions on Medical Imaging* MI-5 (1986), März, Nr. 1, S. 45–47

[Hin06] von Hintzenstern, V.: *Interaktionstechniken zur Exploration von Gefäßbäumen für die Therapieplanung*, Otto-von-Guericke-Universität Magdeburg, Institut für Simulation und Graphik der Fakultät für Informatik, Diplomarbeit, 2006. – unveröffentlicht

[HPSP01] Hahn, H. K. ; Preim, B. ; Selle, D. ; Peitgen, H. O.: Visualization and Interaction Techniques for the Exploration of Vascular Structures. In: *VIS '01: Proceedings of the conference on Visualization '01*. Washington, DC, USA : IEEE Computer Society, 2001, S. 395–402

[HSEP00] Hahn, H. K. ; Selle, D. ; Evertsz, C. J. G. ; Peitgen, H. O.: Interaktive Visualisierung von Gefäßsystemen auf der Basis von Oberflächenprimitiven. In: *Simulation und Visualisierung (2000)*, 2000, S. 105–118

[HSIW96] Hilton, A. ; Stoddart, A. ; Illingworth, J. ; Windeatt., T.: Marching Triangles : Range Image Fusion for Complex Object Modelling. In: *International Conference on Image Processing*, 1996

[JR06] Jalba, A. C. ; Roerdink, J. B. T. M.: Efficient Surface Reconstruction from Noisy Data using Regularized Membrane Potentials. In: *Eurographics/IEEE-VGTC Symposium on Visualization*, 2006, S. 83–90

[KBSS01] Kobbelt, L. P. ; Botsch, M. ; Schwanecke, U. ; Seidel, H.-P.: 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 Press, 2001, S. 57–66

[KFW+02] Kanitsar, A. ; Fleischmann, D. ; Wegenkittl, R. ; Felkel, P. ; Gröller, M. E.: CPR - Curved Planar Reformation. In: *VIS '02: Proceedings of the conference on Visualization '02*. Washington, DC, USA : IEEE Computer Society, 2002, S. 37–44

[KQ04] Kirbas, C. ; Quek, F. K. H.: A Review of Vessel Extraction Techniques and Algorithms. In: *ACM Computing Surveys* 36 (2004), Juni, Nr. 2, S. 81–121

[LC87] Lorensen, W. E. ; Cline, H. E.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In: *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press, 1987, S. 163–169

[LC06] La Cruz, A.: *3D Modelling and Reconstruction of Peripheral Vascular Structure*. Vienna, Östereich, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Diss., März 2006

[Lev96] Kap. 16 In: Levy, S.: *CRC Standard Mathematical Tables and Formulae*. 30th Edition. CRC Press, Februar 1996. – Digitale Version: http://www.geom.uiuc.edu/docs/reference/CRC-formulas/book.html

[Mas02] Masutani, Y.: RBF-Based Representation of Volumetric Data: Application in Visualization and Segmentation. In: *MICCAI '02: Proceedings of the*

5th International Conference on Medical Image Computing and Computer-Assisted Intervention-Part II. London, UK : Springer-Verlag, 2002, S. 300–307

[MMD96] Masutani, Y. ; Masamune, K. ; Dohi, T.: Region-Growing Based Feature Extraction Algorithm for Tree-Like Objects. In: *VBC '96: Proceedings of the 4th International Conference on Visualization in Biomedical Computing*. London, UK : Springer-Verlag, 1996, S. 161–171

[Mün02] von Münster, T.: *Der Einfluss der Körperposition auf die zerebrale venöse Drainage. Eine duplexsonographische Untersuchung der Vena jugularis interna und Vena vertebralis*, Neurologische Klinik der Medizinischen Fakultät Charité, Humboldt-Universität zu Berlin, Diss., November 2002

[NHK+85] Nishimura, H. ; Hirai, M. ; Kawai, T. ; Kawata, T. ; Shirakawa, I. ; Omura, K.: Object Modelling by Distribution Function and a Method of Image Generation. In: *The Transactions of the Institute of Electronics and Communication Engineers of Japan* Bd. J68-D, 1985, S. 718–725

[NSE99] Niessen, W.J. ; van Swihjndregt, ADM ; Elsmann, BHP et a.: Improved Arterial Visualization in Blood Pool Agent MRA of the Peripheral Vasculature. In: H.U. Lemke, K. Inamura A.G. F. (Hrsg.): *Computer-Assisted Radiology and Surgery*, Elsevier, 1999, S. 119–123

[OBA+03a] Ohtake, Y. ; Belyaev, A. ; Alexa, M. ; Turk, G. ; Seidel, H.-P. *MPU Implicits Software*. http://www.mpi-sb.mpg.de/∼ohtake/mpu_implicits/, letzter Stand: 30.05.2006. Oktober 2003

[OBA+03b] Ohtake, Y. ; Belyaev, A. ; Alexa, M. ; Turk, G. ; Seidel, H.-P.: Multi-level Partition of Unity Implicits. In: *ACM Transactions on Graphics* 22 (2003), Nr. 3, S. 463–470

[OBS04] Ohtake, Y. ; Belyaev, A. ; Seidel, H.-P.: Ridge-Valley Lines on Meshes via Implicit Surface Fitting. In: *ACM Transactions on Graphics* 23 (2004), Nr. 3, S. 609–612

[Oel04] Oeltze, S.: *Visualisierung baumartiger anatomischer Strukturen mit Convolution Surfaces*, Otto-von-Guericke-Universität Magdeburg, Institut für Simulation und Graphik der Fakultät für Informatik, Diplomarbeit, Februar 2004

[OM95] Opalach, A. ; Maddock, S.: An Overview of Implicit Surfaces. In: *Introduction to Modelling and Animation Using Implicit Surfaces* (1995), S. 1.1 – 1.13

[OP05] Oeltze, S. ; Preim, B.: Visualization of Vascular Structures: Method, Validation and Evaluation. In: *IEEE Transactions on Medical Imaging* 24 (2005), April, Nr. 4, S. 1–9

[Osb99] Osborn, A. G.: *Diagnostic Cerebral Angiography*. Second Edition. Lippincott Williams & Wilkins, Januar 1999

[OW93] van Overveld, K. ; Wyvill, B.: Potentials, Polygons and Penguins. An Efficient Adaptive Algorithm for Triangulating an Equi-Potential Surface. In: *5th Annual Western Computer Graphics Symposium (SKIGRAPH 93)*, 1993, S. 31–62

[PB06] Preim, B. ; Bartz, D.: *Visualization in Medicine*. Morgan-Kaufman, 2006. – unveröffentlicht

[PFTV92] Kap. 15 In: Press, W. H. ; Flannery, B. P. ; Teukolsky, S.l A. ; Vetterling, W. T.: *Numerical Recipes in C: The Art of Scientific Computing*. 2. Edition. Cambridge University Press, Oktober 1992, S. 657–661

[Poh04] Pohle, R. *Computergestützte Bildanalyse zur Auswertung medizinischer Bilddaten*. Habilitationsschrift, Otto-von-Guericke Universität Magdeburg. März 2004

[Pom04] Pommert, A.: *Simulationsstudien zur Untersuchung der Bildqualität für die 3D-Visualisierung tomographischer Volumendaten*, Universität Hamburg, Fachbereich Informatik, Diss., Juni 2004

[PTN97] Puig, A. ; Tost, D. ; Navazo, I.: An Interactive Cerebral Blood Vessel Exploration System. In: *VIS '97: Proceedings of the 8th conference on Visualization '97*. Los Alamitos, CA, USA : IEEE Computer Society Press, 1997, S. 443–ff.

[Sch05] Schumann, C. *Validierung der kreisförmigen Approximation von Gefäßquerschnitten durch Convolution Surfaces*. Otto-von-Guericke-Universität Magdeburg, Institut für Simulation und Graphik der Fakultät für Informatik, Laborpraktikum. Dezember 2005

[SH97] Stander, B. T. ; Hart, J. C.: Guaranteeing the Topology of an Implicit Surface Polygonization for Interactive Modeling. In: *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press/Addison-Wesley Publishing Co., 1997, S. 279–286

[SLT04] Kap. 6 In: Schmidt, R. F. ; Lang, F. ; Thews, G.: *Physiologie des Menschen mit Pathophysiologie*. 29. Auflage. Springer Medizin Verlag, August 2004, S. 612 – 616

[SPSP02] Selle, D. ; Preim, B. ; Schenk, A. ; Peitgen, H.-O.: Analysis of Vasculature for Liver Surgery Planning. In: *IEEE Transactions on Medical Imaging* 21 (2002), November, Nr. 11, S. 1344–1357

[SSPP00] Selle, D. ; Spindler, W. ; Preim, B. ; Peitgen, H.-O.: Mathematical Methods in Medical Image Processing: Analysis of Vascular Structures for Preoperative Planning in Liver Surgery. In: B. Engquist, W. S. (Hrsg.):

Springer's Special Book for the World Mathematical Year 2000: Mathematics Unlimited - 2001 and Beyond., Springer, 2000, S. 1039–1059

[Str00] Stroustrup, B.: *Die C++ Programmiersprache.* 4. Auflage. Addison-Wesley, 2000

[Str05] Strand, R. *The Face-Centered Cubic Grid and the Body-Centered Cubic Grid : a Literature Survey.* CBA internal report 35, Centre for Image Analysis, Uppsala University, Sweden. 2005

[Tau91] Taubin, G.: Estimation of Planar Curves, Surfaces, and Nonplanar Space Curves Defined by Implicit Equations with Applications to Edge and Range Image Segmentation. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13 (1991), Nr. 11, S. 1115–1138

[TBG95] Tsingos, N. ; Bittar, E. ; Gascuel, M.-P.: Implicit Surfaces for Semi-Automatic Medical Organs Reconstruction. In: *Computer Graphics International'95.* Leeds, UK, 1995, S. 3–15

[Tie99] Tiede, U.: *Realistische 3D-Visualisierung multiattributierter und multiparametrischer Volumendaten*, Universität Hamburg, Fachbereich Informatik, Diss., 1999

[Tur90] Kap. 10 In: Turkowski, K.: *Graphics GEMS.* Bd. 1. Academic Press, 1990, S. 539–547

[VGF02] Velho, L. ; Gomes, J. ; de Figueiredo, L. H.: *Implicit Objects in Computer Graphics.* Springer-Verlag Ltd., 2002

[Wer94] Wernecke, J.: *The Inventor Mentor: Progamming Object-Oriented 3D Graphics with Open Inventor, Release 2.* Addison-Wesley Publishing Group, 1994

[WMW86] Wyvill, G. ; McPheeters, C. ; Wyvill, B.: Data Structure for Soft Objects. In: *The Visual Computer* 2 (1986), Nr. 4, S. 227–234

[ZH81] Zucker, S. W. ; Hummel, R. A.: A Three-Dimensional Edge Operator. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 3 (1981), Nr. 3, S. 324–331

[Zui95] Zuiderveld, K.J.: *Visualization of Multimodality Medical Volume Data using Object-Oriented Methods*, Utrecht University, Diss., 1995

A Anhang

A.1 Definition eines Strukturelements

Die Klasse `Voxeldata` bietet mit den Methoden `erode(int sElement[], int num)` und `dilate(int sElement[], int num)` die Möglichkeit, den Volumendatensatz zu erodieren bzw. zu dilatieren. Der Aufbau des dafür genutzten Strukturelements wurde in Abschnitt 5.3.3 erläutert.
Im Folgenden ist zur Verdeutlichung die Definition eines $3 \times 3 \times 3$-Strukturelements gegeben:

```
1  int se3x3x3[81];
2  //obere Schicht, y = 1
3  se3x3x3[0]  = -1;      se3x3x3[1]  =  1;      se3x3x3[2]  = -1;
4  se3x3x3[3]  =  0;      se3x3x3[4]  =  1;      se3x3x3[5]  = -1;
5  se3x3x3[6]  =  1;      se3x3x3[7]  =  1;      se3x3x3[8]  = -1;
6  se3x3x3[9]  = -1;      se3x3x3[10] =  1;      se3x3x3[11] =  0;
7  se3x3x3[12] =  0;      se3x3x3[13] =  1;      se3x3x3[14] =  0;
8  se3x3x3[15] =  1;      se3x3x3[16] =  1;      se3x3x3[17] =  0;
9  se3x3x3[18] = -1;      se3x3x3[19] =  1;      se3x3x3[20] =  1;
10 se3x3x3[21] =  0;      se3x3x3[22] =  1;      se3x3x3[23] =  1;
11 se3x3x3[24] =  1;      se3x3x3[25] =  1;      se3x3x3[26] =  1;
12 //mitlere Schicht, y = 0
13 se3x3x3[27] = -1;      se3x3x3[28] =  0;      se3x3x3[29] = -1;
14 se3x3x3[30] =  0;      se3x3x3[31] =  0;      se3x3x3[32] = -1;
15 se3x3x3[33] =  1;      se3x3x3[34] =  0;      se3x3x3[35] = -1;
16 se3x3x3[36] = -1;      se3x3x3[37] =  0;      se3x3x3[38] =  0;
17 se3x3x3[39] =  0;      se3x3x3[40] =  0;      se3x3x3[41] =  0;
18 se3x3x3[42] =  1;      se3x3x3[43] =  0;      se3x3x3[44] =  0;
19 se3x3x3[45] = -1;      se3x3x3[46] =  0;      se3x3x3[47] =  1;
20 se3x3x3[48] =  0;      se3x3x3[49] =  0;      se3x3x3[50] =  1;
21 se3x3x3[51] =  1;      se3x3x3[52] =  0;      se3x3x3[53] =  1;
22 //untere Schicht, y = -1
23 se3x3x3[54] = -1;      se3x3x3[55] = -1;      se3x3x3[56] = -1;
24 se3x3x3[57] =  0;      se3x3x3[58] = -1;      se3x3x3[59] = -1;
25 se3x3x3[60] =  1;      se3x3x3[61] = -1;      se3x3x3[62] = -1;
26 se3x3x3[63] = -1;      se3x3x3[64] = -1;      se3x3x3[65] =  0;
27 se3x3x3[66] =  0;      se3x3x3[67] = -1;      se3x3x3[68] =  0;
28 se3x3x3[69] =  1;      se3x3x3[70] = -1;      se3x3x3[71] =  0;
29 se3x3x3[72] = -1;      se3x3x3[73] = -1;      se3x3x3[74] =  1;
30 se3x3x3[75] =  0;      se3x3x3[76] = -1;      se3x3x3[77] =  1;
31 se3x3x3[78] =  1;      se3x3x3[79] = -1;      se3x3x3[80] =  1;
```