Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik

Diplomarbeit

SinusEndoscopy
Ein skalierbares GPU Volume Renderingsystem für die virtuelle Endoskopie

Christoph Kubisch

Institut für Simulation und Graphik

SinusEndoscopy
Ein skalierbares GPU Volume Renderingsystem für die virtuelle Endoskopie

Diplomarbeit

an der
Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von:	Christoph Kubisch
geb. am:	29. April 1982
in:	Deggendorf
Matrikelnummer:	165040

1. Gutachter:	Prof. Dr.-Ing. Bernhard Preim
2. Gutachter:	Prof. Dr.-Ing. Holger Theisel

Betreuer:	Prof. Dr.-Ing. Bernhard Preim
	Dipl.-Ing. Arno Krüger

Zeit der Diplomarbeit:	4.07.2007 - 4.01.2008

Danksagung

Für die Unterstützung in all meinem Schaffen, insbesondere den Jahren des Studiums, möchte ich meinen Eltern und meiner Schwester danken, die mir immer zur Seite stehen und jeden Berg zum Hügel werden lassen.

Meinen beiden Betreuern Prof. Dr.-Ing. Bernhard Preim und Dipl.-Ing. Arno Krüger gilt besonderer Dank für ihre hervorragende Betreuung. Sie gingen konstruktiv auf die Entwicklungen dieser Arbeit ein und waren stets für Besprechungen verfügbar.

Weiterhin danke ich Dr.med. Gero Strauß vom Universitätsklinikum Leipzig und Prof. Dr.-Ing. Dirk Bartz vom ICCAS, die mir ihre fachlichen Ratschläge zukommen ließen und sich die Zeit nahmen, die Software zu testen. Ignacio Castaño und Ignacio Llamas von NVIDIA danke ich für ihre Hilfe bei technischen Anfragen.

Nicht zuletzt möchte ich mich auch bei Eike Decker bedanken, mit dessen Arbeiten LUXINIA entscheidend an Qualität und Stabilität gewinnt, was Implementierungen erleichtert und stets motiviert.

Christoph Kubisch, Dezember 2007

Inhaltsverzeichnis

1 Einleitung **3**

2 Medizinische Grundlagen **6**
2.1 Anatomie und Bildgebung der Nasennebenhöhlen . . . 7
2.2 Endoskopie . . . 8
2.2.1 Aufbau des Endoskops . . . 9
2.2.2 Anwendungsgebiet . . . 10
2.3 Bildgebende Verfahren der NNH . . . 11
2.4 Virtuelle Endoskopie . . . 14
2.4.1 Einordnung innerhalb der Medizin . . . 14
2.4.2 Studien zur VE-Anwendung in der Nasenhöhle oder den Nasennebenhöhlen 16
2.4.3 Grundfunktionen eines Systems . . . 18

3 Techniken für die Darstellung von Volumendaten **20**
3.1 Grundlagen . . . 20
3.1.1 Direktes Volumenrendering . . . 20
3.1.2 Indirektes Volumenrendering . . . 23
3.1.3 Filtering . . . 24
3.1.4 Klassifikation und Transferfunktion . . . 26
3.2 Verfahrensübersicht . . . 27
3.2.1 Isooberflächen aus Dreiecksnetzen . . . 27
3.2.2 Isooberflächen durch CPU Ray-Casting . . . 28
3.2.3 Direktes Volumenrendering mit 2D und 3D Texturen . . . 29
3.2.4 GPU-basiertes Hybrid-Rendering . . . 31
3.3 Algorithmen zur Leistungs- und Qualitätssteigerung . . . 34
3.3.1 Verwaltung großer Datensätze . . . 35
3.3.2 Überspringen von Leerräumen . . . 36
3.3.3 Vorzeitiger Abbruch . . . 36
3.3.4 Deferred Shading . . . 37

4 Konzept und Entwicklung des Renderers **39**
4.1 Anforderungen an das Darstellungsverfahren . . . 39
4.2 Entwicklung der Extraktion . . . 41
4.2.1 Volumendaten und Wahl der Transferfunktion . . . 41
4.2.2 Modifikation des GPU Raycasting . . . 43
4.2.3 Generierung der Oberflächennormalen . . . 48
4.2.4 Glättung der Zwischenergebnisse . . . 50

4.3 Kompositionsschritte . . . 53
4.3.1 Beleuchtung . . . 53
4.3.2 Darstellung von Oberflächenstruktur . . . 55
4.3.3 Eindruck von nasser Oberfläche . . . 57
4.3.4 Projektion von Bildern auf die Oberfläche . . . 57
4.3.5 Interaktion mit polygonaler Geometrie . . . 58
4.3.6 Visualiserung von weniger dichten Materie . . . 59
4.4 Leistung des Systems . . . 61
4.4.1 Vergleich der Renderingvarianten . . . 63
4.4.2 Vergleich von Hardwaresystemen . . . 65

5 Implementierung **68**
5.1 OpenGL als Schnittstelle zur graphischen Darstellung . . . 69
5.1.1 Programmierung der GPU . . . 70
5.1.2 Relevante Textur Funktionen . . . 72
5.2 Luxinia als Entwicklungsplatform . . . 73
5.2.1 Graphical User Interface (GUI) . . . 73
5.2.2 Material- und Shadersystem . . . 74
5.2.3 Aufbau des Renderers . . . 75
5.3 Strukturierung von SinusEndoscopy . . . 76
5.3.1 Steuermodul . . . 79
5.3.2 Ansichten . . . 80
5.3.3 Renderingtechnik . . . 81
5.3.4 Volumendatensatz . . . 81
5.3.5 Animation . . . 82
5.3.6 Zeichenmodul . . . 82

6 Abschließende Bewertung **84**
6.1 Vergleich mit anderen Systemen . . . 84
6.2 Zukünftige Entwicklungen . . . 87
6.2.1 Renderer . . . 87
6.2.2 Interaktion . . . 90
6.3 Weiterführende Techniken . . . 91
6.3.1 Realtime Deformation . . . 92
6.3.2 Realtime Opening . . . 93
6.3.3 Realtime Marching Cube . . . 94
6.4 Klinische Anwendung . . . 95

Literatur **98**

Kapitel 1

Einleitung

Bedingt durch den technischen Fortschritt ist es in der Medizin möglich, chirurgische Eingriffe mit Hilfe von Endoskopen durchzuführen. Diese Methode der „minimal-invasiven-Chirurgie“ bietet eine Reihe von Vorteilen gegenüber der klassischen Radikalchirurgie. Es wird weniger Gewebe zerstört, da nur über wenige kleinere Zugänge in den Anwendungsbereich vorgestoßen wird. Dies hat für den Patienten meist eine kürzere Erholungszeit zur Folge (Konvaleszenzzeit) und ist damit für ihn schonender als Methoden, die erst große Bereiche freilegen müssen. Das Verfahren stellt aber einige besondere Herausforderungen an den Chirurgen. Er hat keinen direkten Blickkontakt mehr mit dem eigentlichen anatomischen Umfeld, sondern nur noch die Bilder, welche die Optik des Endoskops liefert. Dies stellt eine große Reduktion von Information dar, zum einen, weil durch das zweidimensionale Abbild am Bildschirm die Tiefenwahrnehmung erschwert wird, zum anderen, weil das Blickfeld stark eingeschränkt ist. Darüber hinaus treten Verzerrungen bedingt durch die Optik auf, welche zusätzlich die visuelle Wahrnehmung erschweren. Auch die haptische Wahrnehmung wird stark vermindert, indem sämtliche Tätigkeiten über mechanische Werkzeuge erfolgen müssen. Der Arzt kann nicht mehr die Umgebung ertasten und so zusätzliche wichtige Informationen gewinnen. Es bleibt ihm in erster Linie der Bildschirm als Informationsgeber für seine Entscheidungen.
Um sich an dieses Werkzeug zu gewöhnen und eine Operation optimal planen zu können, werden schon seit längerer Zeit erfolgreich computergestützte Systeme genutzt. Allerdings unterliegen diese Systeme einer hochgradigen Spezialisierung auf einzelne medizinische Anwendungsbereiche. Jedes Gebiet stellt besondere Ansprüche und so ist es notwendig, konsequent adäquate Lösungen weiter zu entwickeln. Mit ihnen können sowohl Übungen für die Ausbildung, als auch Planungen konkreter Eingriffe erfolgen. Durch technische Apparaturen, welche über Kraftrückkopplung verfügen, können dann mit Hilfe von geeigneten Datensätzen, verschiedene Eingriffe geübt werde. Bei dieser Art von Simulation eines tatsächlichen Eingriffs wird versucht, die physikalischen Gegebenheiten möglichst exakt nachzubilden. Der Operateur kann so die für das Verfahren notwendigen senso-motorischen Fertigkeiten erlernen und trainieren.
Das zweite Anwendungsfeld der Computerunterstützung ist die virtuelle Endoskopie. Hier steht die Planung und Sichtung des Operationsgebietes im Vordergrund. Die Limitationen des realen Endoskops sind aufgehoben und die Daten können frei exploriert werden. Diese Volumendaten wurden durch Computertomographie direkt am Patienten gemessen und berücksichtigen so die Individualität des Einzelnen. Mit diesen Daten kann sowohl Diagnostik als auch Planung des Eingriffs unterstützt werden. Doch müssen diese Daten erst durch den Computer visuell aufbereitet werden, um für den Operateur nutzbar zu sein. Virtuelle Endoskopiesysteme bieten eine perspektivische Ansicht auf die Daten, in denen eine virtuelle Kamera bewegt wird. Somit

kann der Chirurg sich mit anatomischen Landmarken vertraut machen und Bereiche einsehen, die sonst verborgen wären. Die Systeme erlauben es, den Wertebereich der vorher erstellten Daten unterschiedlich aufzubereiten und versuchen so möglichst viele Informationen zu bieten. Die Methoden der Visualisierungen unterscheiden sich ebenso wie die Qualität und Interaktivität der Verfahren. Zum Teil werden spezielle teure Rechnersysteme benötigt, welche eine erhöhte Verbreitung erschweren. Für den Mediziner wäre es jedoch sinnvoll, die Daten bequem überall darstellen zu können. Vor dem Hintergrund der hohen Leistungssteigerungen von Computerhardware im normalen Konsumentenmarkt wird in dieser Arbeit überprüft, inwiefern die virtuelle Endoskopie davon profitiert und ob mit derart kostengünstiger und weitverbreiteter Hardware ein passendes System entwickelt werden kann. Es gibt viele Anwendungsgebiete der virtuellen Endoskopie in der Medizin. Diese Arbeit konzentrierte sich auf Darstellung von Volumendaten zur Behandlung chronischer Entzündungen in den Nasennebenhöhlen. Vor allem Krankheitsfälle im Zusammenhang mit Störungen des Sekretflusses sind von Bedeutung, da diese noch nicht hinreichend in anderen Systemen berücksichtigt wurden.

Zielsetzung

Aus dem Anwendungsgebiet ergeben sich einige spezifische Anforderungen, die es zu berücksichtigen gilt. Ziel der Arbeit ist es ein System zu entwickeln, das zur virtuellen Endoskopie genutzt werden kann. Der Kern der Forschung gilt der Entwicklung eines Renderingmoduls, das wirtschaftlich auf herkömmlichen Rechnern genutzt werden kann und dennoch skalierbar in Leistung und Qualität ist. Interaktive Bildraten auch auf älteren Systemen zu erzielen, sowie auf modernen mobilen Rechnern ist hierfür wichtig. Allerdings sollen leistungsstarke Systeme qualitativ profitieren und eine der Realität angelehnte Darstellung des Gewebes erzielen. Für den Anwendungsfall wichtig ist vor allem die Darstellung krankhaften Gewebes. Die Anwendung dient zwar nicht zur Simulation, das Renderingmodul ließe sich aber in andere Systeme einbetten. Wegen dieser Abschwächung an die Realismusanforderung sollte der vorgestellte Renderingalgorithmus flexibel erweiterbar sein, um sowohl realistischere als auch abstraktere Bilder zu liefern. Das Renderingsystem muss auch den Vergleich verschiedener Verfahren ermöglichen, um deren Eignung für den Anwendungsfall zu bewerten. Neben diesen visuellen Zielen muss die Anwendung grundlegende Interaktionsmöglichkeiten wie Kamerasteuerung und Veränderung des dargestellten Wertebereichs vollführen können. Ein schnelles und einfaches Einlesen der CT-Daten, und eine geringe CPU-Belastung waren weitere Ziele, um eine effiziente Nutzung der Software zu ermöglichen.

Gliederung der Arbeit

Zunächst wird in **Kapitel 2** ein genauerer Überblick über den medizinisch und technischen Hintergrund gegeben. Dabei wird das reale Endoskop in Aufbau und Funktion erklärt, sowie wichtige Konzepte der virtuellen Endoskopie vorgestellt. Diese Informationen stellen die grundlegenden funktionellen Anforderungen dar, welche für die Entwicklung der Softwarelösung wichtig waren.

Kapitel 3 stellt die programmtechnischen Grundlagen vor, die für die Erstellung des Renderingverfahrens für die virtuelle Endoskopieansicht notwendig sind. Da es möglich ist, diese Ansicht auf verschiedene Weise zu erzeugen, wird auf Gemeinsamkeiten und Unterschiede der Verfahren eingegangen. Jene Algorithmen können darüber hinaus weiter zur Qualitäts- und

Leistungssteigerung verbessert werden, worauf ebenfalls Bezug genommen wird.

In **Kapitel 4** wird der Kern der Anwendung, der Volumenrenderer, detailliert erklärt. Sowohl konzeptioneller Aufbau als auch verschiedene Leistungs- und Qualitätsstufen werden erläutert. Da sich das Verfahren aus verschiedenen Eingabedaten zusammensetzt, die in einer finalen Bildkomposition zusammengefügt werden, wird auf die Erstellung und Probleme der Eingabedaten hingewiesen. Die farbgebenden Berechnungsschritte wie Beleuchtung und Materialeffekte sind auch hier festgehalten.

Kapitel 5 widmet sich dem Aufbau der gesamten Applikation, die im Rahmen dieser Arbeit entstand. Hierbei wird der Fokus auf softwarespezifischen Aufbau und Funktionsumfang gelegt. Außerdem werden die Mittel, die bei der Entwicklung zum Einsatz kamen und für die Implementation genutzt wurden, vorgestellt. Neben der perspektivischen Darstellung der Volumendaten bietet das System zusätzliche Module zur Kamerasteuerung, Animation und Darstellung zusätzlicher Geometrie.

Abschließend zeigt **Kapitel 6** Verbesserungsmöglichkeiten des Systems auf. Neben unmittelbaren Verbesserungen werden auch kurz komplett neue verfahrenstechnische Möglichkeiten angedeutet, die sich erst durch aktuellste Hardware ergeben. Es wird mit einem anderen modernen System, das auch im Anwendungsbereich der Nasennebenhöhlen verwendet wird, verglichen. Zu der Bewertung des Ergebnisses gehören auch die Einschätzungen von Experten der Medizintechnik und Chirurgen über die Software.

Kapitel 2

Medizinische Grundlagen

Obwohl es schon relativ lange Endoskope gibt, und die ersten Exemplare bereits 1850 [1] im medizinischen Einsatz waren, war es doch ein weiter Weg bis sie in der Chirurgie auf breite Akzeptanz und Nutzung gestoßen sind. Ziel der minimal-invasiven Chirurgie ist es, möglichst geringe Wunden bei der Therapie zu erzeugen, damit der Patient von einer raschen Wundheilung profitieren kann. Die Endoskopie ermöglicht den Blick in Körperhöhlen, entweder mit Hilfe von natürlichen Zugängen oder künstlich geschaffenen. Letztere Zugangsart sollte so klein wie nötig sein, denn je weniger der Körper bei der Operation belastet wird, desto geringer ist die Anfälligkeit für andere Komplikationen im Genesungsprozess. Für die minimal-invasive Chirurgie war unter anderem der technische Fortschritt in drei Bereichen von Bedeutung.

1. Das Endoskop selbst, das sowohl zur Diagnostik, als auch zur Therapie genutzt werden kann.

2. Das Akquirieren von Informationen rund um das Eingriffsgebiet, insbesondere Bildmaterial zur anatomischen Beschaffenheit.

3. Die Nutzung des Computers zur Diagnose und Operationsplanung oder zu Ausbildungszwecken.

Daher wird im Folgenden auf diese drei Teilbereiche eingegangen, doch zu Beginn wird auf das konkrete medizinische Anwendungsgebiet dieser Arbeit eingegangen.

[1] `http://de.wikipedia.org/wiki/Endoskop` (Stand 11.2007)

2.1 Anatomie und Bildgebung der Nasennebenhöhlen

Für diese Arbeit war der Bereich der Nasennebenhöhlen (NNH) besonders im Blickpunkt. In der Medizin werden anatomische Verhältnisse primär über Schnittbilder dargestellt. Dabei wird zwischen drei Schnittebenen unterschieden. Axial bedeutet entlang der Körperachse, beim ste-

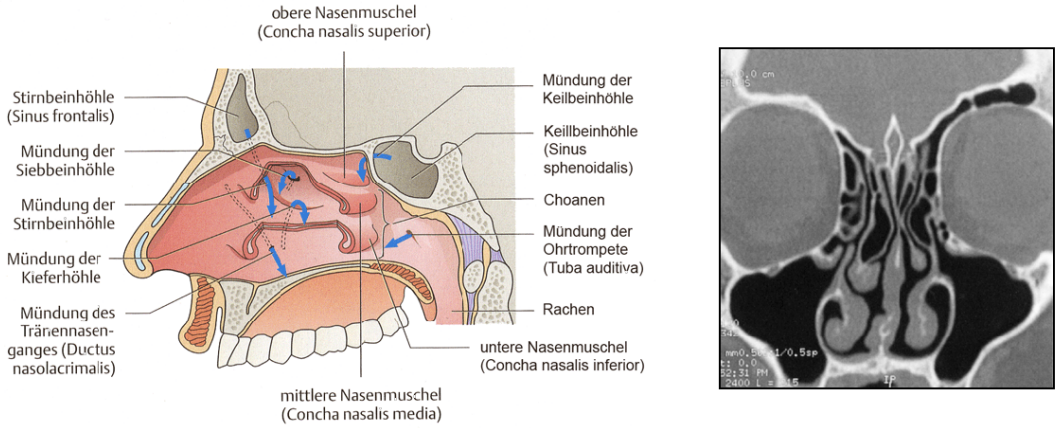

Abbildung 2.1: Links: Übersicht über die Anatomie einer Hälfte der Nasenhöhle. Die blauen Pfeile markieren Zugänge zu den umliegenden Nebenhöhlen. Quelle: [Faller et al., 2004]
Rechts: Koronales CT-Schnittbild mit Blick auf das Siebbeinlabyrinth. Quelle: [Simmen and Jones, 2005]

henden Menschen entspricht dies dem Blick aus der Vogelperspektive. Die koronare Ebene ist equivalent einem Blick auf die Front oder den Rücken, während die Sagittalebene parallel zur Seite liegt. Abbildung 2.1 zeigt eine anatomische Übersicht über die NNH als sagittale Illustration und mit einem koronaren Schnitt, der durch Computertomographie (CT) erstellt wurde. Die Nasenhöhle ist durch die mittlere Nasenscheidewand, das Septum, welches teils knorpelig, teils knöchern ist, in zwei Hälften geteilt. Jede dieser Hälften wird durch mittlere und untere Nasenmuschel weiter in Gänge zergliedert. Die CT Aufnahme daneben zeigt den hohen Grad der Verzweigungen an den Seiten, das Siebbeinlabyrinth. Die Nase dient als Teil des Atmungsorgans zur Luftfilterung, -befeuchtung, -erwärmung und -überprüfung [Faller et al., 2004].

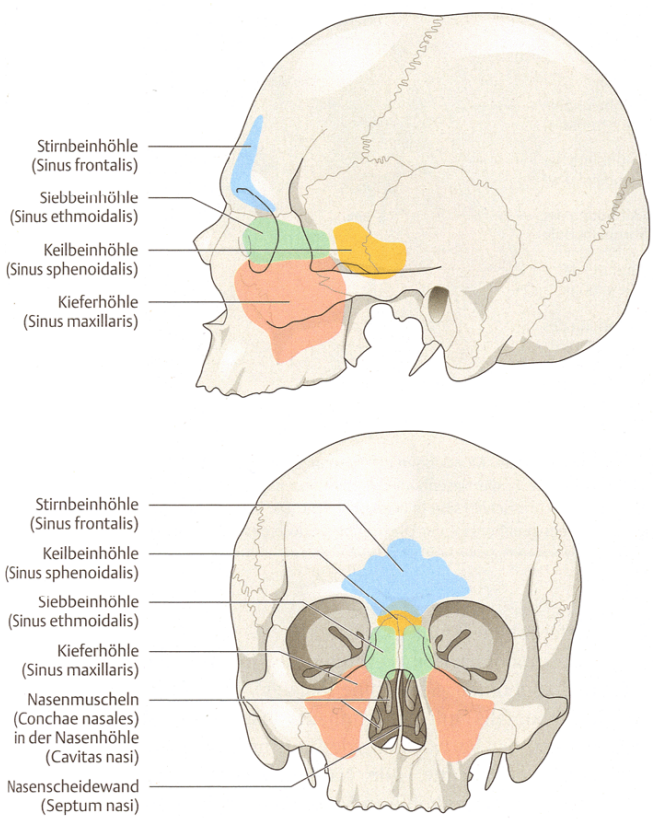

Abbildung 2.2: Nasennebenhöhlen in Seitenansicht (oben) und Frontalansicht (unten). Quelle: [Faller et al., 2004]

In Abbildung 2.2 ist zu sehen, wie die Nasenhöhle in ein größeres System der Nasennebenhöhlen eingegliedert ist. Zu diesen Hohlräumen gehören die Stirnbeinhöhle, die Kieferhöhlen, die Siebbeinhöhlen mit den Siebbeinzellen und die Keilbeinhöhlen. Klangbildung der Stimme sowie die Vorwärmung der Luft sind ihre Hauptaufgaben. Die meisten dieser Hohlräume sind mit einer Schleimhaut, der Mucosa, ausgekleidet und produzieren ein Sekret, welches mit Flimmerhärchen zur Nasenöffnung oder zum Rachen abtransportiert wird. Fehlfunktionen hierbei können zu Entzündungen in den Nasennebenhöhlen führen, die Sinusitis. Liegt eine Beteiligung der Nasenhöhlen vor, wird von einer Rhinosinusitis gesprochen. Derartige Erkrankungen der Atemwege nahmen in den letzten Jahren zu, sind aber meist medikamentös therapierbar. Erst in schweren Fällen muss ein HNO-Arzt chirurgisch eingreifen und benutzt dabei minimal-invasive Methoden mit Hilfe von Endoskopen (vgl. [Oeken et al., 1993]). Bei der bevorstehenden Visualisierungstechnik dieser Arbeit war es daher notwendig, die feinen Gänge ebenso darzustellen, wie den Zustand der Sekretdichte zu erfassen.

2.2 Endoskopie

Die Erfindung des Endoskops (griechisch: end(o) = innen, darinnen; scopein = betrachten, untersuchen) liegt über 200 Jahre zurück. Der konzeptionelle Aufbau ist jedoch bis heute unverändert. Zur Endoskopie werden drei Baugruppen benötigt:

Lichtquelle Sie produziert das Licht, das in das Hohlorgan oder die Körperhöhle geleitet wer-

den soll. Gerade im frühen Zeitalter der Endoskopie war die Lichtquelle ein limitierender Faktor. Moderne Xenon-Lampen haben Kerze und Gaslampen ersetzt und sorgen für ausreichend starke Beleuchtung.

Lichtleiter Er führt das Licht von der Lichtquelle zum Endoskop, oder sogar bis zur Objektivspitze. Glasfaserkabel dienen als Transportmedium, meist in Kombination mit einer Spiegeltechnik, welche den Infrarotanteil des Lichts mindert. Sogenannte Kaltlichtquellen sollen eine stärkere Wärmeentwicklung am Austrittsort des Lichts verhindern.

Endoskop Es bildet das optische System und kann zum Teil auch mit Hilfe eines Arbeitskanals spezielles Kleinstwerkzeug an der Spitze nutzen. Der nächste Abschnitt geht näher auf den Aufbau des Endoskops ein.

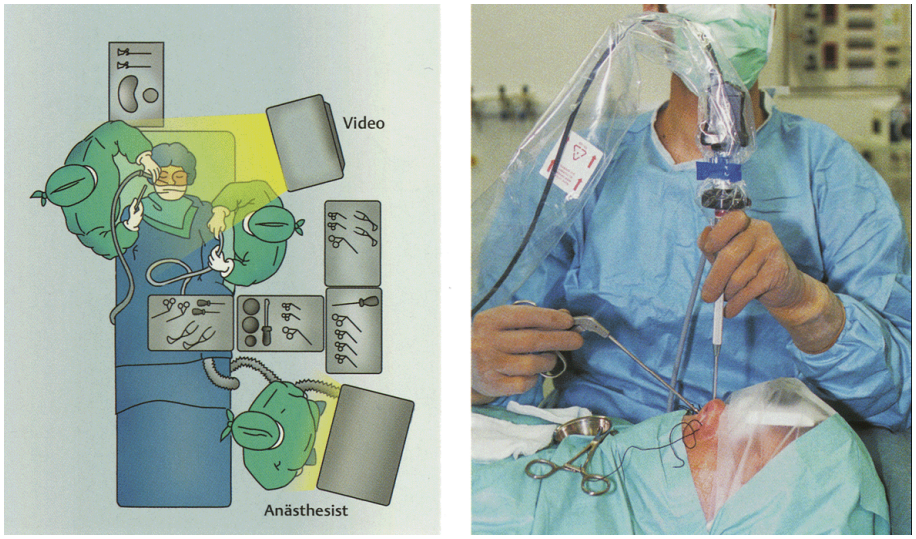

Abbildung 2.3: Nasennebenhöhlen in Seitenansicht (oben) und Frontalansicht (unten). Quelle: [Simmen and Jones, 2005]

Viele Operationen mit Hilfe der Endoskopie erfolgen stationär. Der Operationsaufbau für eine Anwendung im Nasennebenhöhlenbereich mit Hilfe eines Videobildschirms ist in Abbildung 2.3 zu sehen. Bei dieser Operation ist kein Hautschnitt notwendig, sondern Endoskop und Operationsbesteck können über die natürliche Öffnung der Nasenlöcher eingeführt werden [Stampe, 2006].

2.2.1 Aufbau des Endoskops

Je nach Anwendungsgebiet unterscheiden sich Endoskope in Aufbau und Größe. Grundsätzlich gibt es zwei verschiedene Endoskoparten, starre und flexible. Beide werden gleichermaßen an Lichtleiter und Lichtquelle angeschlossen und können mit digitalen CCD-Chips nachgerüstet werden, so dass auch eine Übertragung auf Videobildschirme möglich ist [Manegold and Groitl, 1998]. Der Bildschirm ermöglicht mehreren Ärzten einen Einblick auf das Operationsgebiet.

Starre Endoskope sind technologisch bedingt einfacher und daher schon länger im Einsatz (Abbildung 2.4 links). Die Länge des Endoskops kann zwischen 20-40 cm variieren, auch

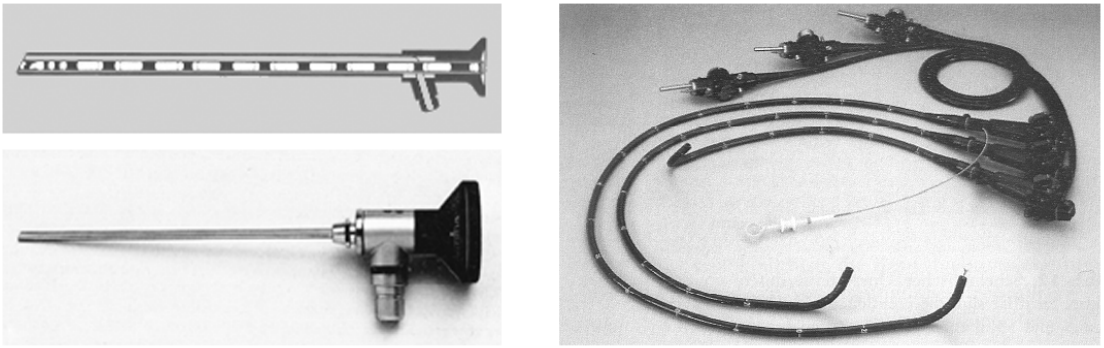

Abbildung 2.4: Links: Starres Endoskop mit Einblick auf das Stablinsensystem im Inneren (oben). Rechts: Flexibles Endoskop. Quelle: [Zylka-Menhorn and Koch, 1996]

der Durchmesser des Zylinders ist unterschiedlich. Ein Objektiv, ein Stablinsensystem und das Okular sind für den Lichttransport zum Betrachter verantwortlich. Jenes Stablinsensystem, das 1953 von Hopkins entwickelt wurde, bricht das Licht an Linsen aus Luft und transportiert es mit Glaszylindern [Stampe, 2006]. Das Objektiv kann verschiedene Öffnungswinkel besitzen und die Blickrichtung kann von der Zylinderachse des Stabes abweichen. Gerade wenn natürliche Zugänge vorliegen oder mit Schnitten neue Zugänge geschaffen werden können, die schnell zum Arbeitsbereich führen, sind starre Endoskope nützlich.

Flexible Endoskope dagegen sind von Vorteil, wenn gewundene Zugänge genutzt werden sollen (Abbildung 2.4 rechts). Insbesondere beim Einsatz im Verdauungssystem finden diese Endoskope viele Anwendungsmöglichkeiten [Manegold and Groitl, 1998]. Neben der Optik und dem Lichtemitter ist auch ein Arbeitskanal vorhanden. In diesem können verschiedene Werkzeuge genutzt werden, so dass Eingriffe wie Fremdkörperextraktion oder Gefäßerweiterungen vorgenommen werden können. Die Beweglichkeit des Endoskops wird über ein Seilzugsystem realisiert und ermöglicht dem Benutzer, die Blickrichtung beliebig zu verändern, sofern die Umgebung dies zulässt. Das Endoskop kann Längen von zwei Meter erreichen und ist damit für den Einsatz im Magen-Darmtrakt geeignet.

2.2.2 Anwendungsgebiet

Zur medizinischen Diagnostik kann das Endoskop genutzt werden, um Gewebeproben zu entnehmen oder andere Messungen, wie Gefäßverengungen, durchzuführen. Die Oberflächenstruktur krankhafter Organe oder Schleimhäute kann dabei ebenfalls untersucht werden. Dies hilft dem Mediziner, weitere Informationen über den Patienten zu gewinnen und den medizinischen Befund zu untermauern. Schließlich kann das Endoskop auch zur Therapie genutzt werden, dabei ist von minimal-invasiver oder endoskopischer Chirurgie die Rede. Mit Hilfe von zusätzlichen Werkzeugen kann krankes Gewebe behandelt werden. Oft werden Gefäßerkrankungen behandelt, indem Erweiterungen oder Stützungen von Gefäßkanälen vorgenommen werden. Es können aber auch Steine entfernt, oder mittels Laser Gewebe abgetragen werden [Stampe, 2006]. Sollte der Arbeitskanal des Endoskops zur Werkzeugnutzung nicht reichen, können weitere Kanäle angelegt werden. In beiden Fällen, Therapie und Diagnostik, ist die Handhabung jener Werkzeuge besonders schwer, weil dem Mediziner nur die Endoskopsicht zur Verfügung steht.

Bereich	Körperteil	Bezeichnung
Innere Medizin	• Speiseröhre	Ösophagoskopie
	• Magen	Gastroskopie
	• Zwölffingerdarm	Dudensoskopie
	• Gallenblase und Bauchspeicheldrüse	retrograde Cholangio-Pankreatikographie
	• Luftröhre	Bronchoskopie
	• Dickdarm	Koloskopie
	• Enddarm	Rektoskopie
Chirurgie	• Bauchhöhle	Laparoskopie
Gynäkologie	• Gebärmutter	Hysteroskopie
Urologie	• Harnröhre	Urethoskopie
	• Harnblase	Zystoskopie
Orthopädie	• Gelenke	Arthroskopie
Hals-Nasen-Ohrenheilkunde (HNO)	• Mundhöhle, Rachen, Nasenhöhle, Nasennebenhöhle	Panendoskopie

Tabelle 2.1: Übersicht verschiedener Anwendungsgebiete der medizinischen Endoskopie. Quelle: [Stampe, 2006]

Der Name der Endoskopie hängt von der Körperhöhle oder dem Hohlraumorgan ab, das untersucht oder behandelt werden soll. Die Endoskopie wird in vielen Bereichen bereits genutzt und Tabelle 2.1 zeigt eine Übersicht über einige davon. Der für diese Arbeit relevante Teilbereich ist die Panendoskopie. Abbildung 2.5 gibt einen Einblick in Handhabung des Geräts und wie das Bild für den Chirurg ist.

2.3 Bildgebende Verfahren der NNH

Um die Diagnostik, aber auch die Planung eines Eingriffs zu erleichtern, kommen in der Medizin mehrere Verfahren zur Anwendung, die dem Arzt Bildmaterial zur Analyse liefern. Grundsätzlich gibt es drei wichtige Verfahrensgruppen, die auf Gammastrahlen-basierten Röntgenverfahren, die Magnetresonanztomographie(MRT) und die Schall-basierte Sonographie. Alle Techniken werden stetig weiterentwickelt und erlauben mit Hilfe des Computers dreidimensionale Abbildungen aus den Messdaten. Letzteres ist für Abbildung knöcherner Strukturen, wie in den Nasennebenhöhlen, eher ungeeignet, deswegen sei es an dieser Stelle vernachlässigt. Es folgt eine Zusammenfassung der verbleibenden Techniken.

In der Diagnostik sind die physikalischen Eigenschaften der Gammastrahlen erwünscht, da sie für das menschliche Auge unsichtbare Dinge sichtbar machen können. Allerdings haben die elektromagnetischen Wellen, welche kurzwelliger und energetisch höher sind als das Licht, chemische und biologische Nebenwirkungen, die unerwünscht sind. Auf unterschiedliche Art und Weise werden diese Strahlen zur Bildgewinnung genutzt und können zweidimensionale Bilder, sowie dreidimensionale Daten liefern. Der Strahlenschutz ist ein wichtiger Punkt im Umgang mit der Gammastrahlung. Heutige Geräte sind sehr effektiv und können die Strahlendosis für den

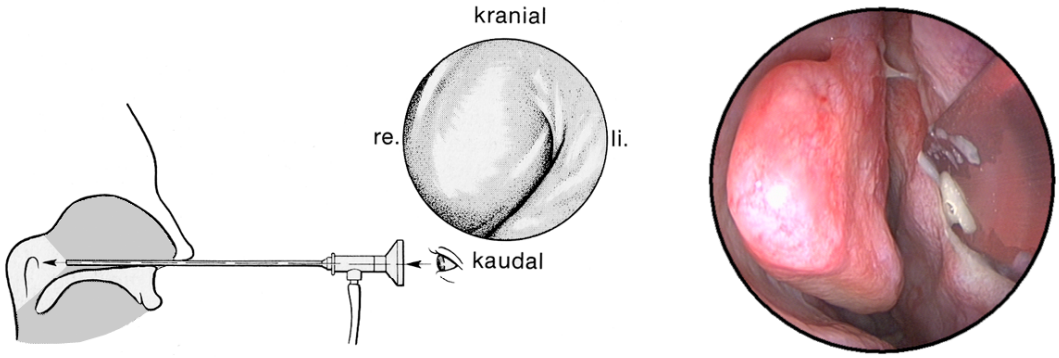

Abbildung 2.5: Links: Endoskopische Untersuchung des Nasenrachens. Quelle: [Becker, 1983]
Rechts: Aufnahme während eines Eingriffs (anderes Operationsgebiet).

Patienten minimieren. Das Risiko für eine einzelne Aufnahme, gewebezerstörende Krankheiten zu induzieren, ist im Tausendstel Promille Bereich.

Kernprinzip der Röntgendiagnostik ist, durch die Röntgenröhre erzeugte Gammastrahlung mit einem speziellen Film aufzufangen. Zwischen Strahlenemitter und Film ist der Körperbereich des Patienten, der untersucht werden soll. Gammastrahlen werden von Materie mit hoher Dichte, wie Knochen, stärker und von Materie geringerer Dichte schwächer absorbiert und beeinflussen so die Schwärzung des Films. Mittlerweile wurde das System aus Filmfolien, das die Lumineszenz beim Auftreffen der Strahlen nutzte, durch eine besondere Speicherfolie ersetzt. Diese nutzt den Halbleitereffekt der Gammastrahlen und wird über ein Lasersystem ausgelesen. Die Daten der Abtastung können dann auf Computermonitoren für den Arzt als „Bild“ sichtbar gemacht werden, oder über optische Printer (Laserimager) in ein echtes Filmnegativ umgewandelt werden. Die Archivierung der digitalen Daten ist auch möglich und erlaubt einen schnellen Vergleich von Aufnahmen am Computer. Da es sich nur um ein zweidimensionales Abbild handelt, benötigt man für eine räumliche Zuordnung mehrere Bilder unter unterschiedlichen Winkeln. Meist wird hier eine Aufnahme im 90° Winkel zur Erstaufnahme angefertigt. Es gibt auch die Möglichkeit, das Verfahren als Echtzeit-Film anzuwenden, doch ist dabei die Auflösung geringer als im Einzelbildverfahren und die Strahlenbelastung deutlich höher. Wird die Strahlung nicht außerhalb erzeugt, um den Patienten zu durchleuchten, sondern im Inneren, ist von Szintigraphie die Rede. In diesem Fall werden radioaktive Stoffe dem Patienten verabreicht und deren Strahlung trifft auf das Aufnahmegerät. Diese Methode wird in erster Linie für Stoffwechselanalysen genutzt, da es zeitliche und örtliche Messergebnisse liefert. Im Anwendungsgebiet dieser Arbeit ist es allerdings auch nicht relevant.

Die Computertomographie (CT) kommt häufig zum Einsatz für die Gewinnung räumlicher Dichteverteilungen. Sender und Empfänger der Strahlen sind so montiert, dass sie sich während des Aufnahmevorgangs auf einer Kreisbahn um den Messbereich bewegen, was der Aufbau in Abbildung 2.6 verdeutlichen soll. Die Auswertung aller Ergebnisse des radial-erfassten Schwächungsprofils, ermöglicht die Speicherung der Daten in einer räumlichen Anordnung. Diese Grauwerte werden normiert in Houndsfield-Einheiten angegeben und entsprechen bei -1000 einem Vakuum und +1000 Calcium. Durch Aneinanderreihung von Schnittebenen lassen sich komplette räumliche Daten gewinnen. Moderne Geräte können eine Translationsbewegung des

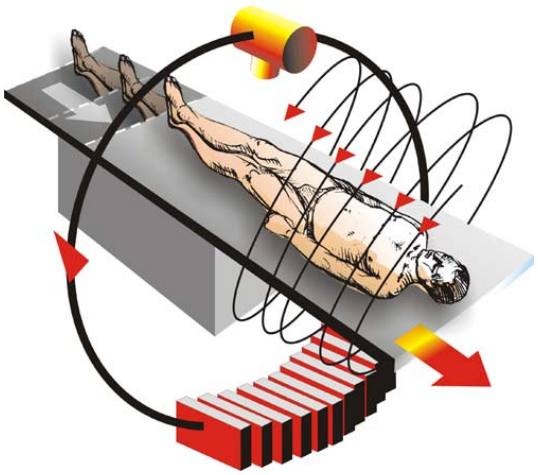

Abbildung 2.6: Spiral-CT Ablauf. Der Emitter bewegt sich auf einer Kreisbahn, während der Patient bewegt wird. Quelle: [Kabayim, 2007]

Patienten vollführen und somit eine spiralförmige Messung vornehmen, die ebenfalls zu dreidimensionalen Daten führt. Diese Daten sind für die virtuelle Endoskopie besonders wichtig. In der Medizin werden meist nur die einzelnen Schnittbilder, wie bereits in Abbildung 2.1 zu sehen war, zur Diagnostik genutzt.

Ein weiteres Verfahren, das ebenfalls räumliche Daten liefert, ist die Kernspintomographie oder auch Magnetresonanztomographie genannt (MRT). Allerdings wird hier nicht mit der Abschwächung von Gammastrahlung operiert, sondern die Verteilung von Protonen im Magnetfeld gemessen. Das Verfahren nutzt die Eigenschaft des Kernspins von Atomkernen. Diese drehen sich um eine zufällig ausgerichtete Raumachse. Durch ein starkes Magnetfeld kann diese Raumachse entlang der Kraftlinien des Feldes ausgerichtet werden. Ein zweites Hochfrequenzfeld kann, je nach Kernart, die Raumachsen auf eine Kreiselbewegung bringen. Beim Ausschalten des Feldes rotiert die Achse zurück in die ausgerichtete Position und gibt dabei einen messbaren Strom ab. Die finalen Messwerte hängen damit von Substanz und Geschwindigkeit der Atomkerne ab. Die Möglichkeiten dieses Verfahrens, das so vielseitig einsetzbar ist, sind aber noch nicht ausgeschöpft. Vorteile gegenüber dem CT sind bessere Differenzierung von Weichteilprozessen in einigen Bereichen, wie Herz und Gehirn. Zwar tritt keine Strahlenbelastung auf, doch birgt das starke Magnetfeld Risiken für Patient und Personal. Ferromagnetische Stoffe können in Bewegung gebracht werden und die Gesundheit gefährden. Das Verfahren verfügt im Moment noch nicht über eine so hohe Auflösung wie das Spiral-CT und erzeugt eine schlechtere Darstellung von knöchernen Strukturen. Die für diese Arbeit genutzten Daten stammen daher aus CT Messungen. Genauere Informationen über die Verfahren im praktischen Einsatz in der Chirurgie sind [Mantke, 1998] und [Raab and Hau, 1998] zu entnehmen, deren Ausführungen als Grundlage für diesen Abschnitt genutzt wurden.

2.4 Virtuelle Endoskopie

Die Virtuelle Endoskopie (VE) versucht mit Hilfe von gewonnenen 3D Volumendaten des Patienten oder allgemeinen Datensätzen wie dem *Visible Human* [2] die Endoskopie zu unterstützen. Die Anforderung an den Operateur bei der Endoskopie sind hoch. Er hat keinen direkten Sichtkontakt und keine direkte haptische Interaktionsmöglichkeit durch seine Hände innerhalb des Operationsbereichs. Nur über ein Okular oder einen Monitor kann er die Geschehnisse beobachten, wobei er mit Verzerrungen der Optik, geringem Lichteinfall und dem Verlust des räumlichen Sehens umgehen muss. Bei flexiblen Endoskopen kommt dabei noch eine zusätzliche Herausforderung in der Steuerung des Geräts hinzu, dies ist jedoch nicht relevant für NNH-Operationen. Die virtuelle Endoskopie kann hier helfen, dass sich der Mediziner an die erschwerten Sichtbedingungen gewöhnen kann. Was ein derartiges System leisten kann und welche anderen Funktionen es noch bietet wird im Folgenden erläutert.

2.4.1 Einordnung innerhalb der Medizin

[DiGioia et al., 1998] unterscheiden zwischen drei Systemarten in der Computer-unterstützten Chirurgie:

Aktiv Diese Systeme können durch den Einsatz von Robotern Eingriffe autonom durchführen. Der Chirurg überwacht dabei die Tätigkeit der Maschine und kann im Notfall eingreifen.

Semi-Aktiv Hierbei wird der Handlungsspielraum des Chirurgen eingeschränkt und er wird von der Technik bei seinem Einsatz geführt. Er kann auch hier jederzeit die Unterstützung des Systems übergehen.

Passiv Im Gegensatz zu den anderen beiden Arten greifen diese Systeme nicht in die Operation ein. Sie dienen lediglich dazu zusätzliche Informationen zu liefern. Sie können auch zur Planung, Ausbildung und Simulation eines Eingriffs genutzt werden. Diese Arbeit gliedert sich in diesen Teilbereich der Forschung ein, welcher bereits eine breite praktische Akzeptanz und Nutzung gefunden hat.

Ein wichtiger Bereich der passiven Systeme ist die Simulation von Eingriffen. In der Medizin wird im Tierversuch oder an Präparaten trainiert. Neue technische Möglichkeiten erlauben aber auch die Simulation am Computer. Ähnlich wie in einem Flugsimulator wird ein Umfeld eines realen Eingriffs geschaffen, jedoch sind die Bilddaten des Endoskops komplett vom Computer berechnet. Neben der visuellen Aufbereitung müssen ebenso die Auswirkungen auf das Gewebe simuliert werden. Die dabei verwendeten Berechnungsmodelle können sehr komplex sein und zusätzliche Rechenleistung verbrauchen. Nicht für alle Arten von Operationen gibt es ausreichend realistische Simulationsmöglichkeiten. Aber die Handhabung des Geräts und die Gewöhnung an das Umfeld konnten laut [Satava and Jones, 1998] eine Übertragungsrate von 28-35% für den realen Einsatzfall erreichen. Dieser Wert ergab sich aus dem Vergleich von Präzision und Geschwindigkeit eines am Patienten oder im Tierversuch erlernten Eingriffs im Verhältnis zum Simulator. Bei Flugsimulatoren lag die Simulatorleistung in etwa bei 45-55% im Vergleich zur realen Flugstunde. Die Autoren gingen davon aus, dass diese Übertragungsraten durch Fortschritte der Simulatortechnik erhöht werden.

Die am weitesten verbreitete Form eines passiven Systems, ist die virtuelle Endoskopie zur visuellen Aufbereitung der Messdaten. Hier spielt die Simulation des Eingriffs eine untergeordnete Rolle, und die Exploration der Daten des Patienten ist von größerer Bedeutung. Es ist bereits

[2] `http://www.nlm.nih.gov/research/visible/visible_human.html` (Stand 11.2007)

in den meisten medizinischen Bildbearbeitungsprogrammen möglich, mit einer perspektivischen Sicht, wie mit einem Endoskop, sich innerhalb des Volumens zu bewegen [Bartz, 2005].

[Vilanova et al., 2000] geben eine Übersicht an Vor- und Nachteilen der virtuellen Endoskopie:
Vorteile:

- Die Daten sind ohnehin vorhanden, weshalb keine zusätzlichen Eingriffe notwendig sind, und folglich die Methode nicht invasiv ist.
- Nicht nur Oberflächeninformation auch zusätzliche Organe können sichtbar gemacht werden. [Satava and Jones, 1998] erwähnen, dass es besonders nützlich ist, Gefäße oder Tumore sichtbar zu machen, die hinter der eigentlichen Gewebeoberfläche liegen.
- Die Handhabung eines realen Endoskops ist schwer und umständlich, es ist leichter sich virtuell ein Bild von der Situation zu machen.
- Ein reales Endoskop hat einen festen Öffnungswinkel und kann, bedingt durch die Bauart, nicht alles sichtbar machen. [Satava and Jones, 1998] geben an, dass es auch praktisch ist in solche Regionen vorzustoßen, in denen ein echtes Endoskop nicht vordringen kann, wie zum Beispiel in das Innenohr.

Nachteile:

- Durch CT-Daten können nur Dichtewerte sichtbar gemacht werden, was keine umfangreichen Aussagen zulässt, z.B. ob Blutungen vorliegen. Außerdem ist die Auflösung der Daten begrenzt und so können Anormalien kleiner als etwa 1mm nicht erfasst werden [Satava and Jones, 1998].
- Durch schlechte Wahl von Parametern bei der Darstellung der Daten kann weiterer Informationsverlust entstehen.
- Navigation und Berechnung sind sehr aufwändig und benötigen Spezialisten. Dieser Punkt verliert aber zunehmend an Relevanz durch Verbesserung der Mensch-Maschine Schnittstellen und Leistungssteigerungen der Hardware.

Die Vorteile der virtuellen Endoskopie lassen einige Anwendungsszenarien zu. In dem State-of-the-Art Report von [Bartz, 2005] über derartige Systeme werden diese verglichen und die gängigsten Anwendungsfunktionen verdeutlicht. In [Satava and Jones, 1998] lässt sich eine ähnliche Auflistung des Gebrauchseinsatzes finden.

Aufklärung Da es keine Einschränkungen in der Sichtweise auf die Anatomie des Patienten gibt, kann man virtuell leichter einen Einblick in diese erhalten. Problematische Stellen können dann dem Patienten gezeigt und erklärt werden. Ebenso zu Ausbildungszwecken können die Daten genutzt werden, um anatomische Gegebenheiten und Variationen zu lernen.

Diagnose Neben den CT-Schichtbildern ist es möglich, durch die virtuelle 3D Sicht, Formdefekte an Organen oder anderem Gewebe zu erkennen.

Interventionsplanung Vor einem Eingriff kann der Chirurg die individuellen anatomischen Merkmale des Patienten berücksichtigen, um so einen optimalen Zugang zu finden. Komplikationen, die durch solche Unregelmäßigkeiten entstehen können, werden erkannt. Auch die individuelle Lage von benachbarten Organen kann verdeutlicht werden, damit der Operateur möglichst viele Informationen zur Planung zur Verfügung hat.

Interoperative Navigation Jene zusätzlichen Daten, die bereits zur Planung dienten, können während des Eingriffs verfügbar gemacht werden. Mittels Objekt-Tracking kann die Position des realen Endoskops mit dem virtuellen Datensatz abgeglichen werden. Es ist dann während der Operation möglich, zusätzliche Markierung oder Gefahrenstellen hervorzuheben und mit dem realen Bild zu überlagern. Ebenso kann eine virtuelle Rundumsicht an der derzeitigen Position des Endoskops ermöglicht werden.

Die abschließende Bewertung in dem State-of-the-Art Report geht auch auf die Problematik der verglichenen Systeme ein. Ein Kernelement, die Visualisierung der Volumendaten, bereitet nach wie vor große Schwierigkeiten. Es gelingt nicht allen Systemen, ein optimales Gleichgewicht aus Bildqualität und Interaktivität herzustellen. Oft ist eines der beiden unzureichend umgesetzt. Eine mangelnde Darstellung hat jedoch einen weiteren Informationsverlust zur Folge, der den Nutzen der Technik in Frage stellt. Andererseits ist eine flüssige und schnelle Berechnung der Bilder notwendig, damit der Benutzer in Echtzeit die Daten sichten kann. Dafür sollten mindestens 15 Bilder pro Sekunde erreicht sein, idealerweise 25-30 wie beim Fernsehen. [Bartz, 2005] stellt auch die Wichtigkeit der Spezialisierung der einzelnen Systeme auf einen Teilbereich heraus. Da in den verschiedenen medizinischen Anwendungsgebieten unterschiedliche anatomische Gegebenheiten vorliegen und die relevanten Wertebereiche der Daten variabel sein können, ist es kaum möglich, eine generelle Lösung für alle Endoskopiebereiche anzubieten. Erst durch die Spezialisierung können die Systeme ihren Zweck hinreichend erfüllen. Im nächsten Abschnitt wird auf zwei Studien eingegangen, die sich der virtuellen Endoskopie im Bereich der Nasenhöhlen annahmen, dem gleichen Anwendungsgebiet wie für diese Arbeit.

2.4.2 Studien zur VE-Anwendung in der Nasenhöhle oder den Nasennebenhöhlen

In [Rogalla, 2001] wird über den Einsatz von virtueller Endoskopie im Bereich der Nase und den Nasennebenhöhlen diskutiert. Dabei wird unter anderem eine Studie von 1998 zitiert, welche den Einsatz und Nutzen von VE im Vergleich zur reinen Nutzung von CT-Schichtbildern thematisierte. Wichtig für die Einschätzung der Ergebnisse ist jedoch die Tatsache, dass zu dem Zeitpunkt der Ausführung keine interaktive Exploration möglich war, und die Vorverarbeitung der Daten sowie die Bilderstellung noch recht zeitraubend und aufwändig war. Es kam ein fest definierter Schwellenwert zum Einsatz, der fortan die Dichtedaten in Leerraum und Gewebe klassifizierte. Dem Arzt standen mehrere Bildserien entlang eines automatisch generierten Pfades zur Verfügung, die an jeder Position Bilder für verschiedene Blickrichtungen beinhalteten. Abbildung 2.7 zeigt zwei solcher Bilder.

Bei der Diagnose verschiedener Krankheiten waren in vielen Fällen CT-schichtbildbasierte Betrachtungen und VE ähnlich erfolgreich, insgesamt schnitt die CT-Bilder jedoch besser ab. Derartige Vergleiche hängen jedoch sehr stark von den konkreten Aufgaben und der Vertrautheit des Arztes mit den beiden Techniken ab. Parametereinstellungen können die Ergebnisse stark positiv oder negativ beeinflussen. Ein negativer Punkt, der unabhängig von diesen Problemen eine Rolle spielte, waren Verdeckungen in den Ansichtsbildern. Die Kameraeinstellungen

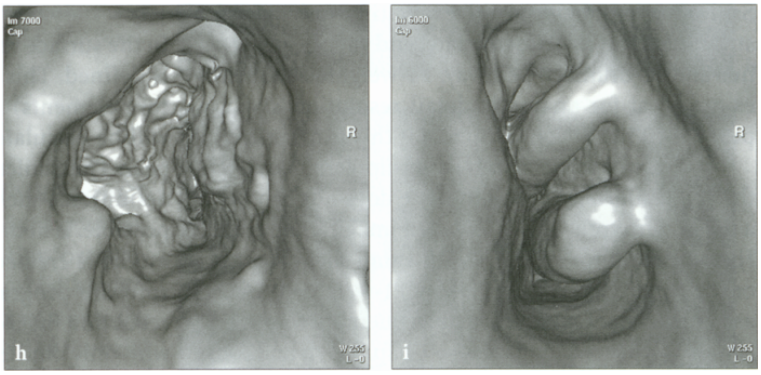

Abbildung 2.7: Nicht-Interaktive Virtuelle Endoskopie mit vorberechneten Bildern. Quelle: [Rogalla, 2001]

der Bilder wurden vorher fest definiert, woraus sich zwangsläufig ergab, dass nicht alle Bereiche eingesehen werden konnten. Die Ärzte bewerteten den interoperativen Nutzen der virtuellen Bilder durchweg positiv. Im post-operativen Anwendungsfall konnten die Auswirkungen des Eingriffs deutlich wahrgenommen werden. Besonders nützlich war die Rundumsicht für die Ärzte, um einen Blick entgegen der Endoskoprichtung, auf den Zugangseintritt, zu haben. Geschaffene Durchgänge, die auf CT-Bildern übersehen werden können, seien mit der VE sichtbarer gewesen. Für die pre-operative Nutzung war es möglich, den optimalen Zugangsweg zu finden, um möglichst wenig Schäden zu hinterlassen. Allerdings war das CT-Bild hier noch sehr wichtig, weil z.B. die Dicke der Mucosa besser zu erkennen war. Dies lag an dem Visualisierungsverfahren des VE Systems, das nur die Gewebewand selbst als festes Objekt darstellen konnte. An dieser Stelle sei auf die Motivation dieser Arbeit hingewiesen, gerade diese Sekretschichten ebenfalls zu berücksichtigen. Für posttraumatische Diagnosen waren die CT-Schichtbilder noch besser geeignet, da sie klare Schnittbilder der Frakturen zeigten, und gerade kleine Frakturen in dem VE System unter dem Gewebe nicht erkannt werden konnten.

[Rogalla, 2001] fasste zusammen, dass für diesen Anwendungsbereich VE sehr gut geeignet ist, weil die vielen luftgefüllten Hohlräume einen idealen Kontrast bilden, und somit sehr gute Ergebnisse bei CT-Messungen ergeben. Dies erleichtert die erwähnte Unterteilung in solide und leere Bereiche. Für die Diagnose waren aber gerade die Sekretverstopfungen problematisch, da diese den Kontrast herabsetzen und so die Klassifizierung erschwerten. Auch bei CT-Aufnahmen unmittelbar vor der OP, die für die interoperative Informationen genutzt werden sollten, war die Erstellung des Gewebeoberflächenmodells erschwert. Denn der Nasenraum wurde schon mit zusätzlichen Stoffen, wie Gleitmittel für das Endoskop, behandelt. Zur Patientenaufklärung sei vor allem eine höhere Interaktivität wünschenswert, um den Vorgang des Eindringens und die Anatomie deutlicher zu machen.

Insgesamt sei die perspektivische Ansicht aber praktischer, weil die Anatomie schneller wahrgenommen wird, und kein so trainiertes Auge wie bei dem Vergleich mehrere CT-Schichtbilder benötigt. Diese Form der Datenverdichtung, aus vielen Schichtbildern ein perspektivisches Bild zu erstellen, wäre für die klinische Kommunikation vorteilhaft. Damit würde die Kommunikation zwischen Radiologen und Spezialisten erleichtert und so indirekt die medizinische Leistung verbessert.

Eine zweite Studie wurde von [Wolfsberger et al., 2004] durchgeführt. Dabei kam ein fortschrittlicheres System zum Einsatz (STEPS), das interaktive Exploration ermöglichte (siehe Abbildung 2.8). Es konnten zum Teil Eingriffe bereits simuliert werden. Auch hier waren die Ärzte

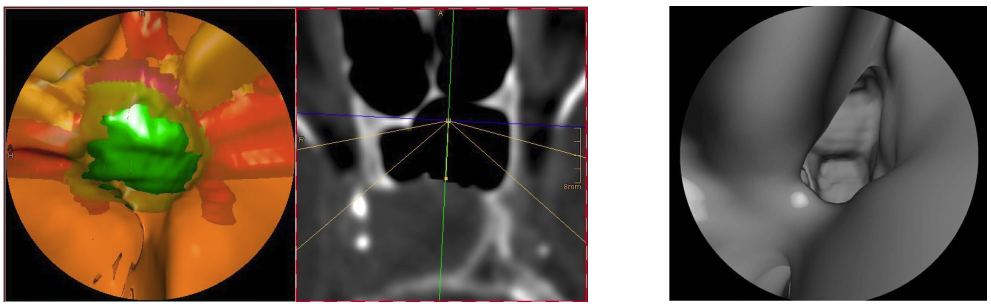

Abbildung 2.8: Interaktive Endoskopie mit STEPS. Rechts ist eine andere Position und ein anderer Darstellungsmodus gewählt worden. Quelle: [Wolfsberger et al., 2004]

vom Nutzen der VE überzeugt. Verbesserungsmöglichkeiten sah man im Bereich der Steuerung. Dies war mit einem Joystick mit Kraftrückkopplung realisiert worden, der mittels Kollisionserkennung ein Stoßen gegen die Gewebewände vermitteln konnte. Diese harte Kollision sei aber nicht dem weichen Gewebe angemessen. Ebenso fiel die aufwändige Segmentierung der Volumendaten negativ auf. Dies ist aber notwendig, um einzelne Organe und Gefäße hervorzuheben. Außerdem war das Freilegen von Bereichen unzureichend simuliert worden, und es konnten unnatürliche Hohlräume entstehen, wenn die Segmentierung nicht vollständig war. Auf das System wird später in Kapitel 6.1 noch weiter eingegangen und es wird mit den Resultaten dieser Arbeit verglichen.

Trotz dieser Mängel weisen auch [Wolfsberger et al., 2004] auf die Vorteile bei interoperativer Nutzung hin. Hinzu kommt, dass in der Planungsphase eine zusätzliche Sicherheit für den Eingriff gewonnen werden würde. Einerseits, weil genauer auf anatomische Variationen des Einzelnen eingegangen werden kann und andererseits, weil besondere Gefäße zusätzlich hervorgehoben werden können und damit ihre räumliche Beziehung besser wahrgenommen wird.

2.4.3 Grundfunktionen eines Systems

Die meisten Systeme bieten einen ähnlichen Funktionsumfang. Der Grad der Interaktivität bei der Exploration der Daten und die Darstellungsqualität sind jedoch meist unterschiedlich. [Bartz, 2005] nutzt zum Vergleich der Systeme die Form der perspektivischen Ansicht und das darunterliegende Kameramodell. Erstere kann auf verschiedene Arten erzeugt werden. Es ist möglich über CPU-basierte Cluster interaktiv Volumendaten zu rendern [You et al., 1997] oder auch Videoclips zu generieren, welche dann abgespielt werden können [Vilanova et al., 2000]. Primär ist für eine Verbesserung des visuellen Bestandteils eines VE Systems die Leistungssteigerung der Hardware verantwortlich. Allerdings ist neben der reinen Leistung auch der Grad der Verbreitung dieser Hardware wichtig, um die Technik möglichst kostengünstig weitverbreitet einsetzen zu können. Modernste Systeme wie von [Scharsach et al., 2006] benutzen die Grafikhardware heutiger Personal Computer.

Der zweite wichtige Bestandteil des Systems ist die Kamerasteuerung. Dabei stellen sich die Fragen, auf welche Weise diese erfolgen soll und ob die Exploration irgendwelchen Limitationen unterliegt. Beliebt sind Methoden, die eine geführte Kamera anbieten, auch wenn diese eine Einschränkung in der Translation haben [Vilanova et al., 2000]. Die Art der Führung kann zum Beispiel entlang eines Pfades im Volumen sein oder eine Kollisionserkennung mit Gewebewänden. Es ist außerdem möglich Kraftfelder zu erzeugen, welche auf das Eingabegerät einwirken [Krüger et al., 2007].

Weitere Interaktionsformen sind die Veränderung des Wertebereichs, bzw. die Form der Repräsentation der Werte. Es sind daneben andere Erweiterungen möglich, wie zusätzliche Simulationen von Werkzeugen, Einschränkung der Kamerarotation und -bewegung wie beim realen Eingriff.

Für diese Arbeit war die freie Exploration der Nasennebenhöhlen relevant. Die Volumeninformationen hierfür stammen aus CT-Daten und ihre perspektivische Darstellung sollte es dem Chirurgen ermöglichen, Planungen von Eingriffen vorzunehmen und ihn bei der Patientenaufklärung unterstützen. Falls eine ausreichende Präzision der Daten vorliegt ist auch die Diagnostik möglich, und es kann auf endoskopische Eingriffe vor der Therapie verzichtet werden. Das nächste Kapitel wird sich dem technischen Hintergrund für die visuelle Grundfunktion des Systems widmen.

Kapitel 3

Techniken für die Darstellung von Volumendaten

Die Darstellung der Volumendaten ist der Kernbestandteil eines jeden virtuellen Endoskopiesystems. Allerdings gibt es eine Vielzahl technischer Möglichkeiten diese Darstellung zu erreichen. Diese Verfahren mussten sich meist technischer Limitierungen unterwerfen, und nur in seltenen Fällen war spezielle Hardware zur Darstellung von Volumendaten verfügbar. Teil dieser Arbeit ist, solche Verfahren zu analysieren, die mit robusten Methoden auf Hardware arbeiten, die kostengünstig für den Massenmarkt verfügbar ist. An diesem Punkt sei erwähnt, dass diese einfachere Hardware in erster Linie auf das Zeichnen von Dreiecken spezialisiert ist. Doch auch speziell-entwickelte Hardware unterliegt einigen methodischen Grundlagen. Die Problematik ist in allen Fällen die gleiche, das Volumen ist eine Funktion, die Skalarwerte repräsentiert, abhängig von dreidimensionalen Raumkoordinaten:

$$\phi : \mathbb{R}^3 \quad \rightarrow \quad \mathbb{R}. \tag{3.1}$$

Zur Darstellung muss nun eine Mapping Funktion genutzt werden, welche einer bestimmten Blickrichtung ρ von einem Betrachtungspunkt a aus einen Farbwert am Bildschirm zuweist:

$$f(a, \rho) \quad \rightarrow \quad \text{color} \in \mathbb{R}^3. \tag{3.2}$$

3.1 Grundlagen

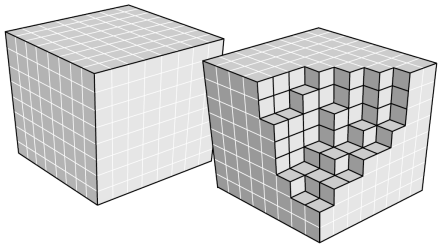

Abbildung 3.1: Uniformes kartesisches Gitter in dem die Volumendaten gespeichert sind. Quelle: [Engel et al., 2006]

Dieser Zuweisungsprozess kann prinzipiell auf zwei verschiedene Arten geschehen, dem direkten und indirekten Volumenrendering. In den meisten Fällen, wie auch im medizinischen Anwendungsfall, liegen die Volumendaten als uniformes 3D Gitter vor, wie in Abbildung 3.1 veranschaulicht, d. h. eine begrenzte Anzahl an Punktwerten ist räumlich verteilt und Werte, die zwischen diesen Punkten liegen, müssen durch den Prozess des *Filterings* erst wieder zurückgewonnen werden. Abschließend werden dem skalaren Dichtewert im Klassifikationsschritt optische Attribute wie Farbe und Transparenz zugeordnet. Zusätzliche farbgebende Funktionen wie Beleuchtung und Schattierung sind ebenfalls möglich.

3.1.1 Direktes Volumenrendering

Wird das Volumen als solches mit seinen Dichtwerten als eine Art gasförmige Materie angesehen, kommt das direkte Volumenrendering zum Einsatz. Der Bildgenerierung in der Computergrafik liegt das physikalische Modell des Lichttransports zu Grunde. In dem Modell bewegt sich Licht in geraden Linien so lange, bis es mit einem Medium interagiert [Engel et al., 2006, Kapitel 1.2]. Es gibt drei wesentliche Interaktionsarten:

Emission. Das Gas sendet Licht aktiv aus und erhöht so die abgegebene Strahlung. Beim vereinfachten Modell zur lokalen Beleuchtung wird die Emission auch um den Faktor des Beleuchtungseinflusses anderer Lichtquellen erweitert. Die Intensität dieses Einflusses hängt dabei vom Gradienten des Volumenpunktes und der einfallenden Lichtrichtung ab. Der Gradient fungiert hier als Oberflächennormale, wie im Phong'schen Beleuchtungsmodell.

Absorption. Das Medium absorbiert einfallende Strahlung bis zu einem gewissen Grad, so dass die Lichtenergie reduziert wird.

Streuung (*Scattering*). Der Lichtstrahl kann durch ein Medium abgelenkt werden. Auf diese Art kann Energie verloren gehen, indem der Strahl weggelenkt wird (*Out-Scattering*). Es kann aber auch Energie dazugewonnen werden, durch andere abgelenkte Strahlen (*In-Scattering*). Zusätzlich kann die Wellenlänge des Lichts durch Streuung verändert werden, was aber meist vernachlässigt wird.

Die Lichtenergie wird als Strahldichte I beschrieben, die sich aus der Strahlung Q pro Einheitsfläche A, dem Winkel θ zwischen Flächennormale und Lichtrichtung, pro Winkel Ω und pro Zeit t zusammensetzt:

$$I = \frac{\mathrm{d}Q}{\mathrm{d}A \cos\theta \mathrm{d}\Omega \mathrm{d}t} \tag{3.3}$$

Die Transferfunktion des Lichts für eine bestimmte Lichtrichtung ω an der Stelle x ist wie folgt definiert:

$$\omega \cdot \nabla_x I(x,\omega) = -(\kappa(x,\omega) + \sigma(x,\omega))I(x,\omega) + q(x,w) + \int_{\text{sphere}} \sigma(x,\omega')p(x,\omega',\omega)I(x,\omega')\mathrm{d}\Omega'. \tag{3.4}$$

κ ist die Absorbtion der Lichtenergie, σ der Energieverlust durch Streuung und q der Emissionswert. Alle drei sind optische Eigenschaften des Mediums. Der „nabla" Operator gibt den Gradienten an der Stelle x an ($\nabla_x = (\delta/\delta x, \delta/\delta y, \delta/\delta z)$). Das Integral steht für den Energiegewinn durch Streuung und beinhaltet den Faktor p, der die Wahrscheinlichkeit angibt,

mit der ein Strahl ω' in Richtung ω abgelenkt wird. Um Kosten bei der Berechnung zu sparen, wird dieses Integral jedoch oft vernachlässigt. Durch Weglassen diverser Faktoren dieser Formel lässt sich das optische Modell vereinfachen. Das beim Volumenrendering am meisten verbreitete Modell besteht aus Emission und Absorption. Um die Strahlungsdichte I von einem Startpunkt s_0 bis zu einem Endpunkt D zu berechnen, muss ein Integral gelöst werden, das als *Volume Rendering Integral* bekannt ist:

$$I(D) = I_0 e^{-\int_{s_0}^{D} \kappa(t)\mathrm{d}t} + \int_{s_0}^{D} q(s) e^{-\int_{s}^{D} \kappa(t)\mathrm{d}t} \mathrm{d}s.$$

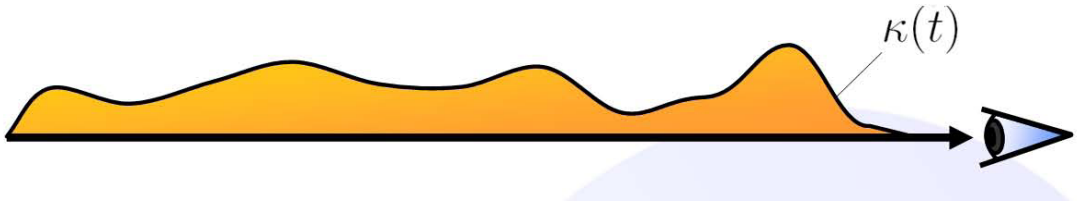

Abbildung 3.2: *Volume Rendering Integral* für einen Sichtstrahl. Quelle: [Engel et al., 2006]

I_0 gibt hier die Lichtenergie an, die beim Eintritt in das Volumen an der Hintergrundposition s_0 anliegt. Das Gesamtergebnis gibt schließlich die Menge Energie an, die beim Betrachter ankommt, das Integral ist in Bild 3.2 dargestellt.

Durch die Definition von Transparenz T als

$$T(s_1, s_2) = e^{-\tau(s_1, s_2)} = e^{-\int_{s_1}^{s_2} \kappa(t)\mathrm{d}t} \tag{3.5}$$

lässt sich das Integral auch wie folgt ausdrücken:

$$I(D) = I_0 T(s_0, D) + \int_{s_0}^{D} q(s) T(s, D)\mathrm{d}s. \tag{3.6}$$

Dieses Integral ist Basis für das direkte Volumenrendering. Da es nur durch Diskretisierung am Computer gelöst werden kann, wird es in kleinen Schritten fester oder variabler Schrittweite evaluiert. Das Gesamtintegral wird in mehrere Segmente unterteilt. Der Einfluss eines solchen Segments auf die Transparenz oder Intensität wird folgend festgehalten:

$$T_i = T(s_{i-1}, s_i), \qquad c_i = \int_{s_{i-1}}^{s_i} q(s) T(s, s_i)\mathrm{d}s. \tag{3.7}$$

Benutzt man diese Notation, kann das Integral als Summe und Produkt von Farb- und Transparenzwerten geschrieben werden.

$$I(D) = \sum_{i=0}^{n} c_i \prod_{j=i+1}^{n} T_j, \quad \text{mit} \quad c_o = I(s_0) \tag{3.8}$$

Für eine effiziente Lösung durch den Computer ist es sinnvoll, diese Formel iterativ zu lösen, so dass sich das Gesamtergebnis durch viele Kompositionsschritte, die bei jedem Segment ausgeführt werden, ergibt. Hierfür gibt es zwei Kompositionsverfahren, die den eigenen Wert (src) mit dem Ergebnis der vorherigen Berechnungen (dst) zusammenführen. Die Transparenz sei mit Hilfe der Opazität α durch (1-α) definiert. Für die Evaluierung in Blickrichtung (*front-to-back*) ergibt sich für die Farbe C die Kompositionsformel:

$$\begin{aligned} C_{\text{dst}} &\leftarrow C_{\text{dst}} + (1 - \alpha_{\text{dst}})C_{\text{src}}, \\ \alpha_{\text{dst}} &\leftarrow \alpha_{\text{dst}} + (1 - \alpha_{\text{dst}})\alpha_{\text{src}} \end{aligned} \tag{3.9}$$

Für die umgekehrte Richtung gilt die Formel:

$$C_{\text{dst}} \leftarrow (1 - \alpha_{\text{src}})C_{\text{dst}} + C_{\text{src}} \tag{3.10}$$

Letztere Formel 3.10 wird meist bevorzugt, weil es keine so großen Präzisionsprobleme gibt. Denn bei Formel 3.9 hängt die Ausgabe auch von α_dst ab. Da die dst-Werte nach der Speicherung an Präzision verlieren könnten, würde sich die Ungenauigkeit mit jedem Schritt vergrößern.

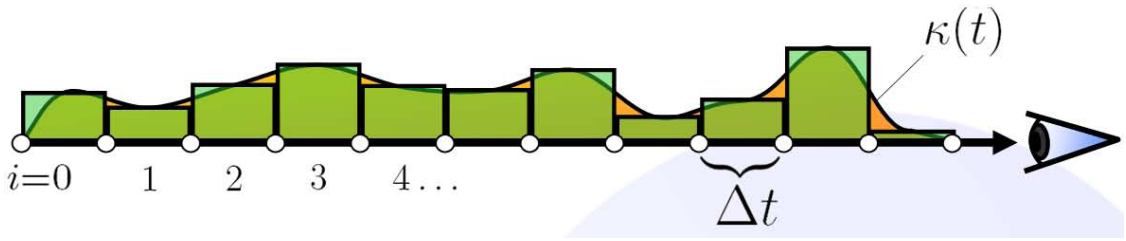

Abbildung 3.3: Approximation der Integralsegmente mit Rechtecken. Quelle: [Engel et al., 2006]

Meist wird zur Beschleunigung aber nicht das Integral der Segmente berechnet, sondern als Rechtecksfläche approximiert, wie in Abbildung 3.3 zu sehen. Dies lässt darauf schließen, dass höhere Schrittzahlen, bzw. geringere Schrittweiten zu genaueren Ergebnissen führen. Die meisten Algorithmen dieser Methodengruppe versuchen ein adäquates Gleichgewicht aus Geschwindigkeit und Genauigkeit des Algorithmus zu finden. Weitere Informationen zu den hier erwähnten Grundlagen des Volumenrendering sind in [Engel et al., 2006, Kapitel 1.2] zu finden.

3.1.2 Indirektes Volumenrendering

Im Gegensatz zu diesem gasförmigen Modell geht man beim indirekten Volumenrendering von einem festen Stoff aus. Der Vorteil ist, dass man nicht entlang des gesamten Sichtstrahls Berechnungen ausführen muss, sondern lediglich der Übergang von Leere zu festem Stoff von Bedeutung ist. Die Klassifizierung eines Volumenwertes ergibt eine klare Zuweisung in soliden oder leeren Bereich. Durch Segmentierungsverfahren können die soliden Bereiche weiter unterteilt werden. Mit Hilfe von Dreiecksnetzen können die soliden Bereiche umschlossen und so geometrisch beschrieben werden. Diese werden dann mit weit verbreiteten Verfahren gezeichnet.

Grafikkarten benutzen hierbei einen Rasterisierungsprozess, der die Geometrie des Dreiecks in Bildpunkte diskretisiert. In Bild 3.4 ist dieses Pipelinemodell abgebildet. Die Grafikkarten können die Transformation der Dreieckspunkte ebenso wie die Rasterisierung und Schattierung der Fragmente äußerst effizient erledigen.

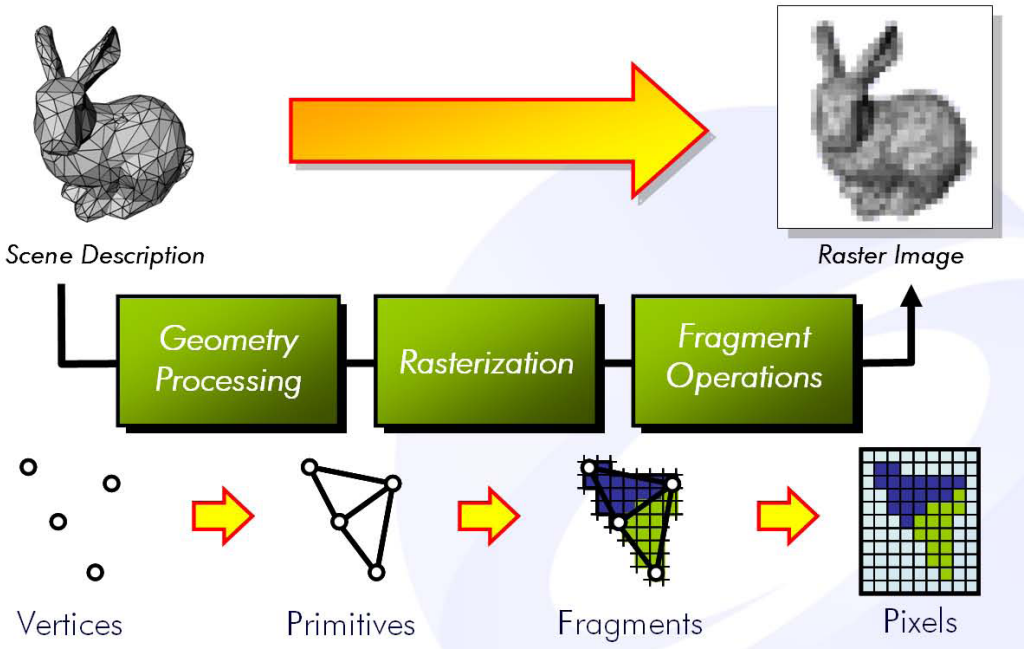

Abbildung 3.4: Pipelinemodell für Grafikkarten, die mit Rasterisierung arbeiten. Quelle: [Engel et al., 2006]

Eine andere Methode ist das Ray-Casting, bei dem analytisch der Schnittpunkt von Sichtstrahl und Dreiecksgeometrie bestimmt wird. Jeder Bildpixel ist Startpunkt eines solchen Sichtstrahls und bekommt nach der Evaluierung einen Farbwert zugewiesen. Abbildung 3.5 illustriert die Funktionsweise dieses Verfahrens, bei dem der Schnittpunkt mit zusätzlichen Informationen, wie der Oberflächennormale, weitere Berechnungen von Spiegelungen oder Beleuchtungseffekten zulässt. Beim Ray-Casting kann aber auch auf die Beschreibung durch Dreiecke verzichtet werden und entlang des Sichtstrahls selbst der Übergang in die feste Materie gesucht werden. Diese Art des Ray-Castings ist sehr verbreitet. [You et al., 1997] kombinieren ein indirektes Verfahren als „Startpunkt“ für ein direktes Rendering des Volumens hinter den Gewebewänden.

3.1.3 Filtering

Damit beide Verfahren auch Volumenwerte benutzen können an Positionen, die zwischen den Gitterpunkten liegen, muss durch Interpolation dieser Wert erzeugt werden. In [Engel et al., 2006, Seite 21-25] werden einige mathematischen Grundlagen zur Rekonstruktion dieser Werte erläutert. Im einfachsten Fall wird lediglich der Wert des nächsten Gitterpunkts übernommen, was zu qualitativ minderwertigen Ergebnissen führt. Dies wird auch als nearest-neighbor Filtering bezeichnet. Eine bessere Methode wäre die trilineare Interpolation, wie sie auch von den Grafikkarten unterstützt wird.

Zunächst wird bilinear in den Ebenen unter und über des Punktes ein Wert ermittelt, welcher dann linear interpoliert den finalen Wert ergibt (Abbildung 3.6). Mit Hilfe des sinus cardinalis,

$$\operatorname{sinc}(t) = \begin{cases} \frac{\sin(t)}{t} & \text{wenn } t \neq 0 \\ 1 & \text{wenn } t = 0 \end{cases}$$

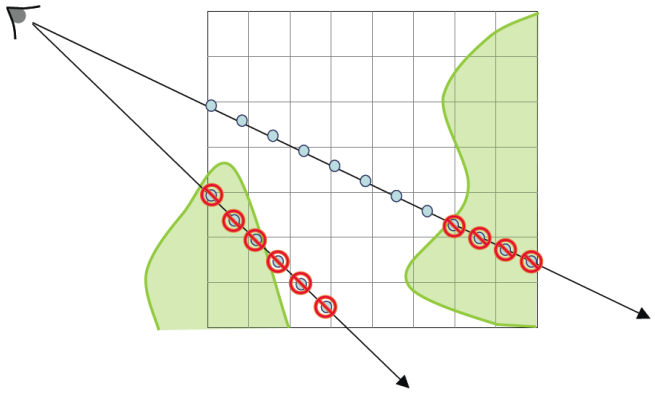

Abbildung 3.5: Ray-Casting innerhalb des Volumens. Die Isooberfläche wurde entweder vorher aus Dreiecken erstellt, oder sie wird iterativ gesucht. Quelle: [Tariq and Llamas, 2007]

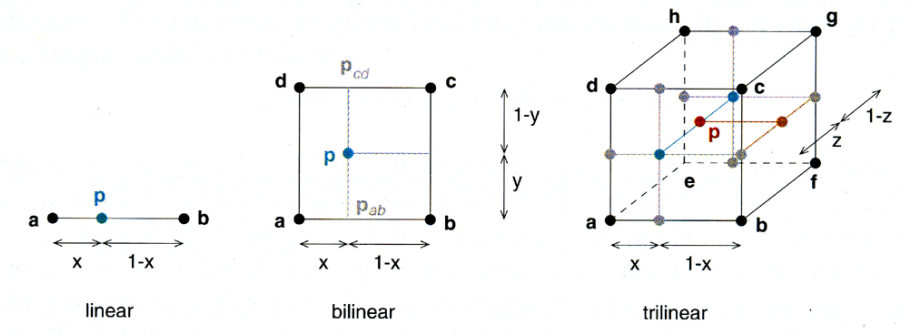

Abbildung 3.6: Funktionsweise des trilinearen Tensorprodukt Filterings. Quelle: [Engel et al., 2006]

lassen sich noch präzisere Ergebnisse erzielen, die den ursprünglichen Signalwerten noch näher kommen. Doch wie in Abbildung 3.7 zu sehen, sind im Vergleich zum linearen und nearest neighbor- Filtering nicht nur die unmittelbaren Nachbarn entlang einer Dimension notwendig, sondern alle Gitterpunkte, weswegen dieser Filter nur dann zum Einsatz kommen kann, wenn die Berechnungszeit eine untergeordnete Rolle spielt. Da das Ziel dieser Arbeit aber gerade eine hohe Interaktivität ist, müssen hier Vereinfachungen vorgenommen werden. Bei der Entwicklung des Renderers wird in Abschnitt 4.2.1 noch einmal auf diese Problematik eingegangen.

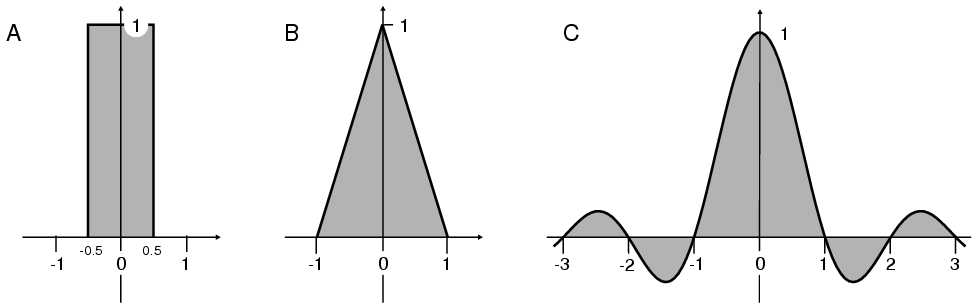

Abbildung 3.7: Einfluss der benachbarten Pixelwerte verschiedener Filter. Nearest-Neighbor (A), Linear (B) und Sinc (C). Quelle: [Engel et al., 2006]

3.1.4 Klassifikation und Transferfunktion

Schließlich folgt die Klassifikation durch eine Transferfunktion, so dass der skalare Wert des Volumens eine Farbe und Transparenz erhält. Hierbei können ein oder mehrdimensionale Transferfunktionen zum Einsatz kommen [Kniss, 2003]. Eindimensionale Funktionen werden meist als „Rampen“ Funktion genutzt, d. h. innerhalb eines Wertebereichs nimmt die Opazität linear entweder ab oder zu. Da von Volumendaten ausgegangen wird, deren Nullwert Leere repräsentiert, werden zunehmende Rampenfunktionen genutzt. Neben der Transparenz werden auch die Intensitätsverläufe der einzelnen Farbkanäle angegeben. Mehrdimensionale Funktionen benutzen neben dem Volumenwert noch weitere Eigenschaften wie Gradientenstärke oder Krümmungsrichtungen.

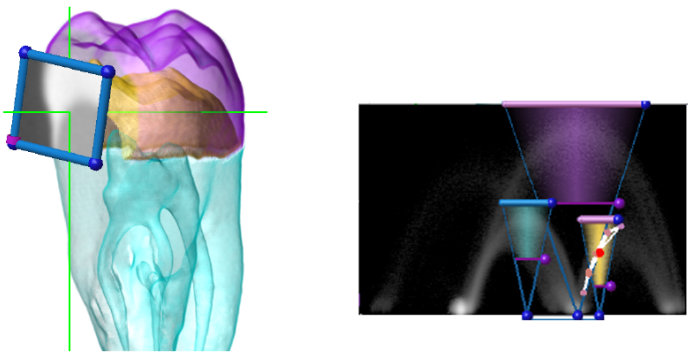

Abbildung 3.8: Mehrdimensionale Transfer Funktion. Quelle: [Engel et al., 2006]

Der Zahn in Abbildung 3.8 zeigt wie durch eine 2D Transferfunktion die verschiedenen Strukturen besser differenziert werden können, indem neben Volumenwerten auch die Gradientenstärke als Eingabe genutzt wird. Allerdings ist die Erstellung solcher Transferfunktionen aufwändiger und bedarf oft zusätzlicher Unterstützung. In [Engel et al., 2006, Kapitel 10] wird auf die Herausforderungen und technischen Hintergründe dieser eigenständigen Problematik eingegangen.

Die Transferfunktion könnte auch auf den Originaldaten angewendet werden, man spricht dabei von Pre-Klassifikation. Wie [Engel, 2003] zeigt, sind die Ergebnisse dabei jedoch schlechter. Transferfunktionen mit höheren Frequenzen liefern im Vergleich zur Post-Klassifikation einen zu großen Qualitätsverlust (siehe Abbildung 3.9), da die interpolierten Werte zwischen den Gitterpunkten durchaus verschieden klassifiziert werden können, als ihre unmittelbaren Nachbarn. Deswegen und weil die Volumendaten nicht verändert werden müssen, eignen sich Post-Klassifikationen für qualitativ bessere Ergebnisse. Durch *Pre-Integrated Classification* können die Ergebnisse noch weiter verbessert werden, ohne die Anzahl der Samplingpunkte zu erhöhen.

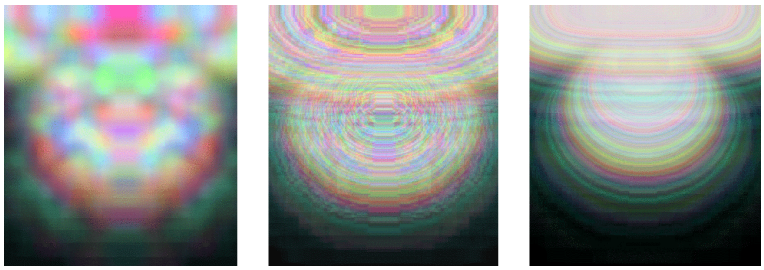

Abbildung 3.9: Klassifikationsverfahren im Vergleich. Die Pre-Klassifkation (links) hat das schlechteste Ergebnis, während die anderen beiden Post-Klassifkationen höherwertig sind. Rechts im Bild wurde *Pre-Integrated Classification* angewandt.

3.2 Verfahrensübersicht

Aufbauend auf diesen Methoden ergeben sich nun viele mögliche Implementierungen. Im Folgenden wird auf solche eingegangen, die den höchsten Verbreitungsgrad besitzen. Jedes Verfahren besitzt Vor- und Nachteile, die im Hinblick auf die Themenstellung bewertet werden. Zum Teil erfolgten praktische Umsetzungen und Tests an typischen medizinischen Volumendatensätzen, andererseits wurden veröffentlichte Verfahren theoretisch überprüft.

3.2.1 Isooberflächen aus Dreiecksnetzen

Die erste Verfahrensgruppe nutzt Dreiecksnetze zur Visualisierung. Das Volumen muss zunächst in diese umgewandelt werden. Ein beliebter Algorithmus hierfür ist der *Marching Cube* Algorithmus [Lorensen and Cline, 1987]. Das Volumen wird in Kuben unterteilt, deren Eckpunkte die Gitterpunkte des Volumens sind. Je nachdem, ob die 8 Gitterpunkte nun innerhalb oder außerhalb des Volumens liegen, werden entsprechende Dreiecke innerhalb des Kubus erzeugt. Die Dreickspunkte liegen dabei immer auf festen Unterteilungspunkten, so dass benachbarte Zellen eine geschlossene Oberfläche bilden.

Abbildung 3.10 zeigt eine Auswahl der 256 verschiedenen Fälle der Dreieckskombinationen innerhalb einer Zelle. Durch Ausnutzung von Symmetrien müssen nicht alle manuell erstellt wer-

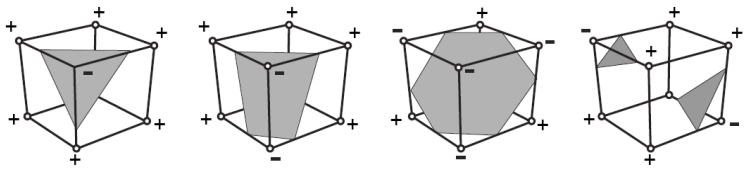

Abbildung 3.10: Einige Kombinationen die beim Marching Cube Algorithmus entstehen können. Quelle: [Rezk-Salama, 2002]

den, sondern eine Tabelle mit allen 256 Dreicksnetzen kann auch automatisch erstellt werden. Zur Laufzeit muss dann lediglich der Index in diese Tabelle aus den Daten der acht Gitterpunkte berechnet werden, um die passende Geometrie zu erzeugen. Durch Verschmelzen von Punkten auf den Gitterlinien kann die Punktmenge verringert werden. Die Summe der Dreicke bleibt jedoch sehr hoch. Durch die Art der Anordnung der Dreiecke innerhalb der Zelle, entsteht eine Oberfläche mit Flächen im 45° Winkel zueinander, was im Hinblick auf die organischen Strukturen der NNH keine angemessene Form ist. Durch kleinere Zellen innerhalb des Volumengitters und dem dabei benötigten Filtering, können feinere Strukturen besser repräsentiert werden. In jedem Fall sollten die Dreieckspunkte iterativ untereinander angenähert werden, um die Strukturen zu glätten.

Durch die feinere Auflösung entstehen aber enorm viele Punkte und Dreiecke, ebenso sind Flächen gleicher Orientierung unnötig stark zerteilt. Es gibt eine Reihe von Algorithmen, die Dreiecksnetze vereinfachen können und geometrische Beziehungen wie Krümmungsstärke beachten, doch entstehen dadurch wiederum höhere Berechnungszeiten.
Die fertig aufbereiteten Dreiecksdaten können dann mit Grafikkarten beschleunigt schattiert dargestellt werden, oder durch diverse CPU-gestützte Verfahren, die sich meist auf das Ray-Casting stützen. Letzteres wird bei sehr komplexen Modellen vorgezogen. Moderne Grafikkarten haben zwar sehr hohe theoretische Nennleistung, was das Darstellen von Dreiecken angeht, aber diese Werte sind praktisch nur schwer zu erzielen. Daher entwickelten [Wald et al., 2004] ein System, das mit Hilfe von speziellen Prozessoren sehr große Polygondatenmengen in Echtzeit bewältigen kann. In beiden Fällen wird die CPU belastet um Beschleunigerstrukturen zu verwalten, die die berechnete Menge der Dreiecke auf die tatsächlich Sichtbaren reduziert.
Selbst wenn eine effiziente Darstellung gewährleistet werden kannn, haben die Dreiecke jedoch einen entscheidenden Nachteil: Ihr Erstellungsprozess ist zu aufwändig und damit ist eine interaktive Änderung der Transferfunktion ausgeschlossen. Diese ist aber gerade für den medizinischen Anwendungsfall von großer Bedeutung, damit der Wertebereich der Daten leicht justiert werden kann. Somit ist dieses Verfahren, trotz der Vorzüge des Shadings von Dreiecken, für diesen medizinischen Anwendungsfall nicht geeignet. Hinzu kommt, dass es nicht um eine reine binäre Klassifikation in fest und leer geht, sondern auch um mehrere Bereiche, wie die Sekretstärke.

3.2.2 Isooberflächen durch CPU Ray-Casting

Die Isooberflächen können auch direkt gewonnen werden ohne vorher Dreiecke zu erstellen, was den Einsatz von dynamisch veränderbaren Transferfunktionen wieder ermöglicht. Die besondere Herausforderung besteht hierbei darin, die Zellen innerhalb des Volumengitters zu finden, in denen eine genaue Analyse zur Gewinnung des Schnittpunktes mit dem Sichtstrahl durchgeführt werden muss. Dabei kommen einige wichtige Prinzipien zur Anwendung, wie sie von [Neubauer et al., 2002a] beschrieben werden:

Überspringen von Leerräumen. Ein wichtiger Beschleunigungsfaktor ist das effiziente Überspringen von Leerräumen, in denen es garantiert keinen Schnitt mit der Isooberfläche geben kann.

Datenreduktion. Es müssen nur die Zellen analysiert werden, die auch innerhalb des Sichtfeldes liegen.

Cache Koherenz. Beim Arbeiten mit großen Datenmengen ist es sinnvoll, Ergebnisse zu teilen und wieder zu verwerten. Die räumliche Nähe von Sichtstrahlen untereinander kann zu ähnlichen Zellen im Volumen führen und damit das Speicher-caching des Computers besser nutzen.

Viele Fortschritte wurden in einzelnen Bereichen bereits gemacht und es gibt eine Reihe von Systemen [Neubauer et al., 2002b, Parker et al., 1999, Wald et al., 2005], die sich auch um die Darstellung medizinischer Volumendaten bemühen. Doch sind die Hardwareanforderungen meist sehr hoch und benötigen entweder spezielle Hardware oder Cluster Systeme aus mehreren Computern, um interaktive Bildraten zu erzeugen. [Koloszár and Jae-Young, 2003] und [Neubauer et al., 2004] sind hierbei besonders zu erwähnen, da sie auch für endoskopische Ansichten innerhalb des Volumens entwickelt wurden. Ersteres hat allerdings relativ geringe Bildraten auf durchschnittlichen Rechnersystemen, während das zweite System eine spezielle Clusterhardware nutzt. Für das Ziel dieser Arbeit sind jedoch interaktive Bildraten bei normaler Hardware wichtig, auch bietet eine reine Isooberfläche zu wenig Informationen über das Volumen. Denn die viskosen Bereiche sind bei Gewebekrankheiten und diversen Einschlüssen in den Nasennebenhöhlen wichtig.

3.2.3 Direktes Volumenrendering mit 2D und 3D Texturen

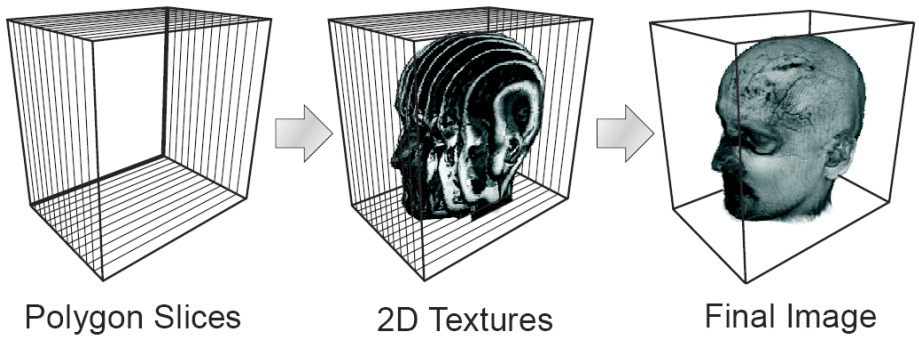

Abbildung 3.11: Schnittebenen sind an den Hauptachsen ausgerichtet. Es können Lücken zwischen den Ebenen entstehen. Quelle: [Engel et al., 2006]

Um alle Wertebereiche vollständig darzustellen, wird häufig das direkte Volumenrendering mit 2D und 3D Texturen verwendet. Texturen erlauben es Grafikkarten, auf vorher gespeicherte Daten zuzugreifen oder Zwischenergebnisse in ihnen zu speichern, die später wiederverwendet werden sollen. Normalerweise werden sie genutzt um zweidimensionale (2D) RGB-Bilddaten darzustellen. Zunächst dominierten die Verfahren mit 2D Texturen, denn diese wurden als erstes von den Grafikkarten beschleunigt. Das Volumen wird aus einer Reihe von 2D Texturen zusammengesetzt, die jeweils die Daten entlang der Hauptachsen enthalten [Cabral et al., 1994].

So ergeben sich drei Sätze von 2D Texturen, die der Dimension Breite×Höhe, Höhe×Länge, Länge×Breite. Je nachdem, welche Blickrichtung gerade dominant ist, wird der passende Datensatz in Form von Rechtecken dargestellt, da Grafikkarten nur Polygone rastern können. Meist ist die Reihenfolge des Zeichnens der Rechtecke von hinten nach vorne und mit dem in Formel 3.10 beschriebenen Kompositionsblendmodus, der von der Hardware beschleunigt durchgeführt werden kann. Es kommt bei dieser Methode allerdings zu Aliasingartefakten, und erst wenn die Schnittebenen im Volumen erhöht werden, wie beim Erläutern der Diskretisierung bereits erwähnt, werden diese geringer. Da die Abstände zwischen den Ebenen entlang der Blickrichtung variabel sind, wie in Abbildung 3.11 zu sehen, können die Schnitte nicht akkurat die Integration zwischen zwei Ebenen speichern. Eine höhere Anzahl an Schnittebenen bedeutet aber auch einen höheren Speicherverbrauch, da mehr Texturen benötigt werden. Ebenso erhöht sich die Füllratenbelastung der Grafikkarte.

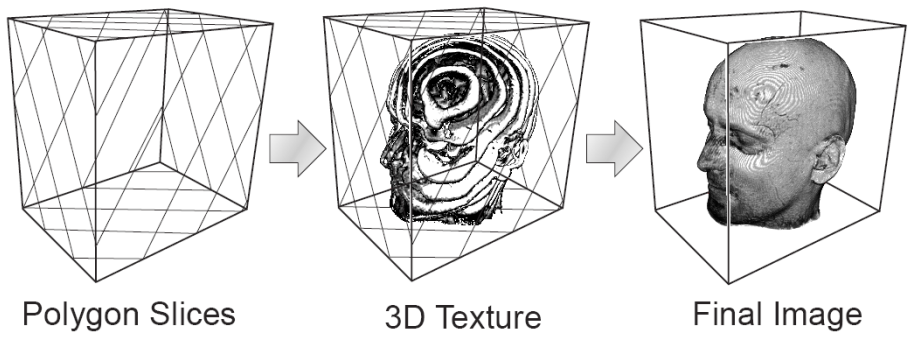

Abbildung 3.12: Schnittebenen sind parallel zur Bildebene. Quelle: [Engel et al., 2006]

Seit 2001 sind 3D Texturen mit trilinearer Interpolation innerhalb des Volumens auf dem Massenmarkt verfügbar [1]. Derartige Hardware gab es zwar bereits schon vorher [Cullip and Neumann, 1994], aber nicht für den normalen Konsumenten. Allerdings wurde dabei ein Algorithmus entwickelt, der diese Fähigkeit nutzt. In [Cullip and Neumann, 1994] werden Ebenen parallel zur Bildebene erstellt (Abbildung 3.12). Da nun alle Punkte innerhalb dieser Rechtecke durch trilineare Interpolation aus dem Volumen gewonnen werden, ist das Verfahren einfacher umzusetzen und ebenso performant. Das Volumen muss nur einmal abgespeichert werden, so dass die Speicherkosten sinken. Da alle Bereiche, die Transformation der Rechteckspunkte, die Berechnung der Koordinaten für den Volumentexturzugriff, dieser selbst und die finale Komposition von der Hardware beschleunigt werden, ist dieses Verfahren heute sehr beliebt.

Mit der Erweiterung der Hardware Renderingpipeline um die Programmierung der Vertex- und Fragmentprozessoren konnten auch komplexere Transferfunktionen realisiert werden, die sich auch nach Belieben dynamisch verändern lassen. Außerdem verfügen moderne Grafikkarten über höhere Präzision, sowohl in der Berechnung als auch in Speicherung, so dass deutliche Qualitätsverbesserungen erreicht werden konnten.

[1] GeForce3: `http://www.nvidia.com/page/geforce3.html` (Stand 11.2007)
Radeon8500: `http://ati.amd.com/products/radeon8500/radeon8500/index.html` (Stand 11.2007)

3.2.4 GPU-basiertes Hybrid-Rendering

Die Erweiterung der Renderingpipeline, wie sie in Abbildung 3.13 zu sehen ist, ermöglichte es, die Vorteile von direktem und indirektem Verfahren zu vereinen. Daher wird an dieser Stelle genauer auf die Veränderung der Pipeline und den damit verbundenen Auswirkungen auf das Volumenrendering eingegangen.

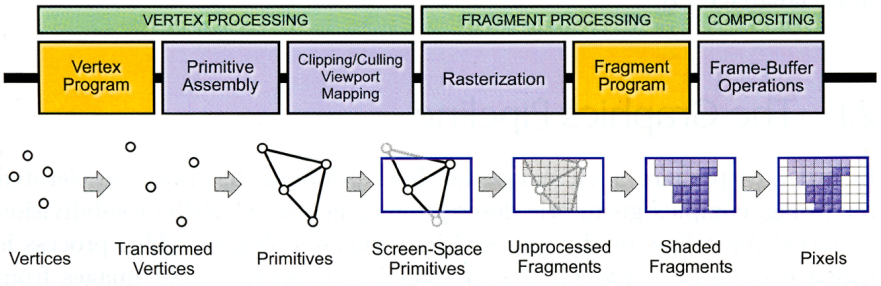

Abbildung 3.13: Die Programmierbare Renderingpipeline. Gelb hervorgehoben sind die Teile im Prozess, die programmiert werden können. Quelle: [Engel et al., 2006]

Pipelinemodell

Moderne Grafikprozessoren sind als hoch-optimierte Streamingprozessoren konzipiert, das heißt, sie zeichnen sich durch parallele Bearbeitung der Daten aus. Es werden meist mehrere geometrische Punkte, sowie später Fragmente gleichzeitig berechnet. Bedingt durch das Rasterisierungsprinzip fallen keine Daten an, die wiederverwendet oder länger gespeichert werden müssen. Jedes Dreieck wird isoliert betrachtet und dargestellt. Dazu müssen erst die Punktdaten (Vertex), sowie deren Attribute, wie Texturkoordinaten und Farbe berechnet werden. Danach wird das Dreieck gerastert und die Fragmente berechnet, deren Eingangswerte die linear interpolierten Attribute der Vertices sind. Schließlich werden die Fragmentfarben zu Pixeln in einem Ausgabepuffer. Es gibt nur zwei relativ kleine Caches, den Vertex- und Texturecache. Ersterer speichert Ergebnisse der Evaluation der letzten Vertices ab, so dass ein Dreieck, welches einen Vertex im Cache benutzt, davon profitieren kann. Letzterer beschleunigt nach ähnlichem Prinzip die Texturzugriffe, so dass das Lesen von Texturdaten bei örtlich nahen Texturkoordinaten beschleunigt wird. Dieses Datenflussmodell aus Eingangswerten in Form von Geometriepunkten und deren Attributen und mehrdimensionalen Texturen, die vorher gespeicherte Werte beinhalten, ist sehr starr aufgebaut und daher äußerst effizient. Anfangs konnten die Operationen auf Vertex- und Fragmentebene nur auf fest definierten Wegen beeinflusst werden, die sogenannte Fixed-Function Pipeline. Einige der Vertex- und Fragmentoperationen wurden später durch die Vertex- und Fragmentprozessoren programmierbar.

Früher unterschieden sich noch die beiden Prozessortypen in ihren Operationsfähigkeiten und sie waren meist auch physisch verschieden, heute implementieren die Grafikkarten eine *Unified Shader Architecture*, in der alle Prozessoren gleich sind[2]. Die Prozessoren werden dabei vom Benutzer über *SIMD* (Single Instruction Multiple Destinations) Befehle programmiert, die auf vierdimensionalen Skalarvektoren operieren. Der Befehlsumfang wurde ständig erweitert und so können die Prozessoren auch für andere parallelisierbare Berechnungen genutzt werden und nicht nur zur Bilddarstellung. In Abbildung 3.14 sieht man den Ablaufprozess eines solchen

[2] `http://en.wikipedia.org/wiki/Unified_shader_model` (Stand 11.2007)

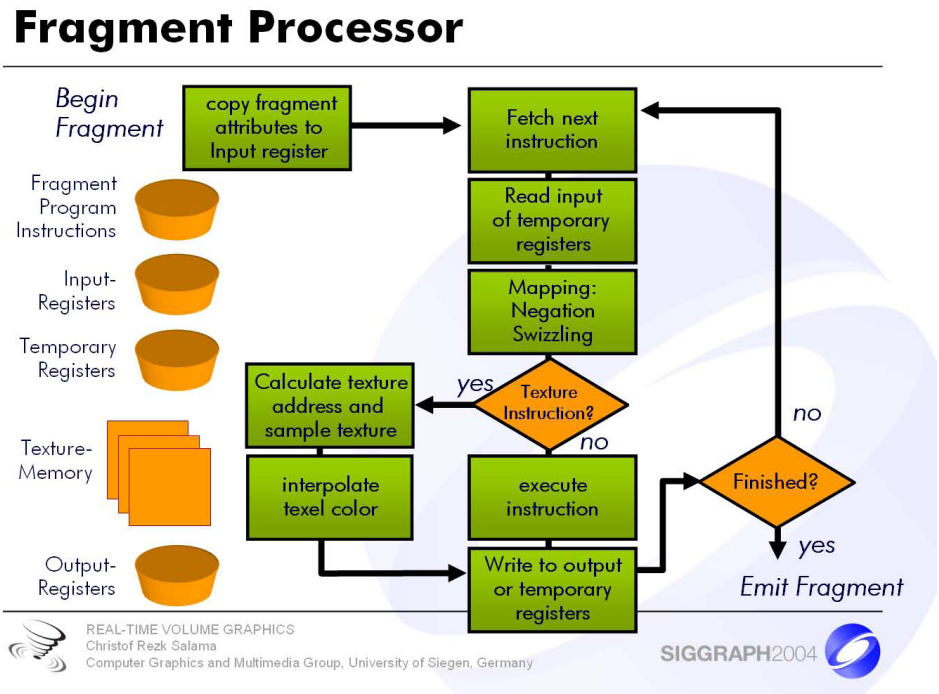

Abbildung 3.14: Ablauf eines Programms für den Fragmentprozessor.

Prozessors. Die Anzahl der temporären Register, welche die einzige Möglichkeit zur Speicherung von Ergebnissen innerhalb eines Programms bieten, ist nach wie vor relativ begrenzt.

Die für das Volumenrendering wichtigsten Funktionen wurden 2004 hinzugefügt, dynamische Programmflusskontrolle und Schleifen. Daneben bieten heute die Programme die Möglichkeit, sehr viele Instruktionen in einem Programm zu verwenden. So erlaubt es die Programmflusskontrolle auf Fragmentebene, die Volumenintegration in einem Programmdurchlauf zu realisieren. Es müssen nicht mehr Rechtecks oder andere Schnittgeometrien verwendet werden, sondern die Zugriffe, der Transfer und die Komposition können in einem Programm für jeden finalen Pixel berechnet werden.

Volumenrendering

[Stegmaier et al., 2005] nutzen diese Fähigkeiten um ein Framework zur Volumenberechnung zu erstellen. Ihr Algorithmus, hier in Pseudocode, kann vollständig durch den Einsatz dieser GPU-Programme realisiert werden:

```
Compute volume entry position
Compute ray of sight direction
While in volume
    Lookup data value at ray position
    Accumulate color and opacity
    Advance along ray
```

Sie weisen auf zwei wichtige Vorteile des Verfahrens hin:

Zukunftsorientiert Die Entwicklung der Grafikkarten geht ständig weiter und ihre Rechenleistung hat die der CPU schon weit überflügelt. Es ist anzunehmen, dass dieser Trend weitergeht, oder ähnliche Funktionalität in die CPU integriert wird.

Flexibel und einfach Für das Volumenrendering ist sehr praktisch, dass die komplette Evaluierung in einem Programm geschrieben werden kann und sich nicht aus verschiedenen Renderingschritten zusammensetzen muss. Gerade die Tatsache, dass die Richtung des Strahls beeinflusst werden und die Evaluierung komplett in ausreichender Präzision vorgenommen werden kann, erlaubt es, verschiedene Volumenrenderingtechniken umzusetzen.

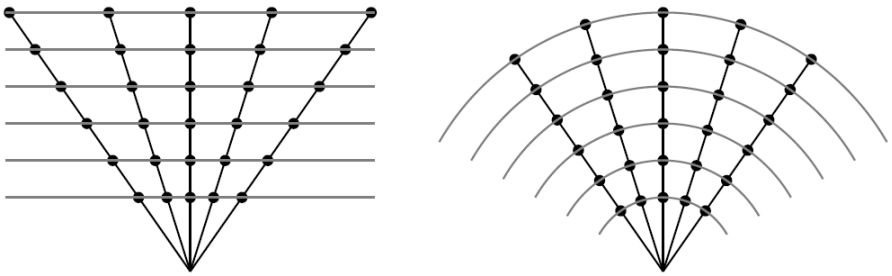

Abbildung 3.15: Ebenenbasiertes (links) oder radiales (rechts) Sampling. Letzteres hat equidistante Schrittweiten. Quelle: [Strengert et al., 2006]

Beide Punkte haben sich in der Praxis bestätigt und so sind verschiedene Techniken entstanden, welche die Vorteile der Berechnung in einem Durchlauf nutzen. Die leichte Beeinflussung der Samplingposition erlaubt sowohl radial equidistantes Sampling als auch die gewohnten Schnittebenen, mit variablen Weiten. Beide Möglichkeiten sind in Abbildung 3.15 illustriert. Es ist ebenso möglich, Effekte wie Reflektion oder Refraktion zu implementieren [Strengert et al., 2006]. [Scharsach, 2005] zeigen eine Reihe von Verbesserungsmöglichkeiten auf, die insbesondere beim direkten Volumenredering von Nutzem sind. Man kann das Verfahren aber auch nützen, um Isooberflächen zu extrahieren [Hadwiger et al., 2005]. Im medizinischen Anwendungsbereich ist daneben eine Hybridlösung interessant, welche Isooberflächen mit direktem Volumenrendering vereint [Scharsach et al., 2006]. Diese Idee wurde bereits vorher schon mit einer Mischung aus Polygonalenflächen und anschließendem direktem Rendering in [You et al., 1997]

realisiert. Doch ist dies mit den modernen Karten in einem Durchlauf möglich und bei voll interaktiver Transferfunktion.

Das Verfahren kann, bedingt durch das strikte Pipelinemodell, nicht ganz so leicht optimiert werden, wie CPU-basierte Renderer. Diese können viel effizienter Zwischenergebnisse speichern und teilen. Darüber hinaus können sie komplexere Verfahren zur Beschleunigung nutzen. Die für diese Arbeit relevante Hardwaregeneration kann ausschließlich Fließkommaberechnungen durchführen und keine Ganzzahlarithmetik. Die Nutzung von Koherenz zwischen den Sichtstrahlen wie in [Koloszár and Jae-Young, 2003] ist daher zum Beispiel in dieser Form nicht realisierbar. Sämtliche Zwischenergebnisse gehen mit jedem fertigen Bild (Frame) verloren. Zwar lassen sich auch mehrere Ausgabewerte realisieren und im nächsten Frame als Eingabewerte verwenden, doch lässt dies den Vorteil des einfachen Aufbaus wieder schwinden. Außerdem ist eine Synchronisation mit der CPU zur Nachbearbeitung der Daten sehr langsam. Daraus ergibt sich jedoch der Vorteil, dass die CPU sonst nicht belastet wird. Sie kann für andere Dinge, wie z.B. für eine Kollisionserkennung, genutzt werden.

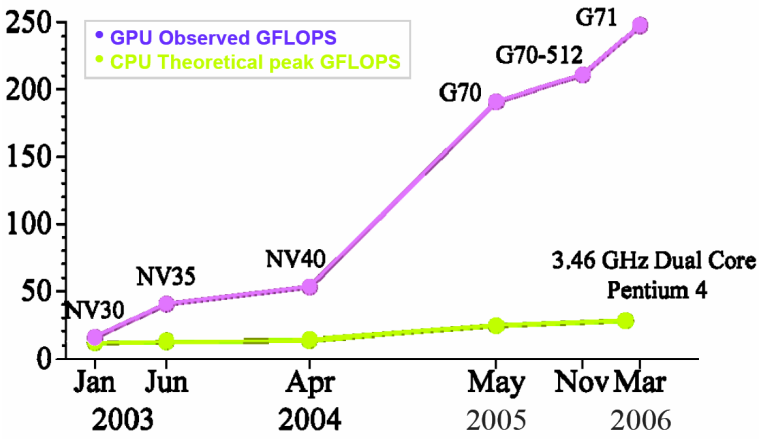

Abbildung 3.16: Leistungsentwicklung von GPU im Vergleich zur CPU. Quelle: [Harris, 2006]

Es gibt eine Reihe von Algorithmen. die sich der Qualitäts- und Leistungssteigerung widmen. Der Grafikhardware-Pipeline angemessen können sie robust das Rendering beschleunigen. Dass diese Algorithmen nun auf die GPU übertragen werden, liegt an den Vorteilen der hohen Leistung (vgl. Abbildung 3.16) und Flexibilität dieser GPU-basierten Verfahrensgruppe. Einige der geeignet erscheinenden Methoden werden im nächsten Abschnitt detaillierter ausgeführt.

3.3 Algorithmen zur Leistungs- und Qualitätssteigerung

Alle vorgestellten Volumenrenderingverfahren profitieren von einigen Optimierungsalgorithmen, die sich generell auf das Volumenrendering anwenden lassen. In welcher Form sie letztendlich implementiert werden, hängt natürlich vom Verfahren selbst ab. Wie bereits erwähnt, ist die Entwicklung mit der CPU leichter, da sie viel universeller einsetzbar ist, und weniger Restriktionen und Limitationen mit sich bringt. Deswegen wurden die meisten Techniken zunächst für CPUs entwickelt und werden nun vermehrt auf das Pipelinemodell der Grafikkarten übertragen.

3.3.1 Verwaltung großer Datensätze

Generell stellen große Datensätze eine besonderer Herausforderung dar. Der Datensatz eines kompletten Menschen bei hoher Präzision kann schnell zu sehr großen Datenmengen führen, z.B. 512 MB bei 512×512×1024 16-Bit. Zugriffe auf einen so großen Datensatz sind generell problematisch. Einerseits kann der GPU- oder Hauptspeicher nicht reichen, andererseits der Zugriff zu langsam sein. Wenn die Daten sequentiell gespeichert sind, ist die Entfernung im Speicher zwischen benachbarten Voxeln sehr groß. Im gewählten Beispiel wäre ein Voxel vom darüber liegenden 512 KB entfernt, was bei älteren CPUs außerhalb der Cachegrößen liegt, und folglich einen langsamen Zugriff darstellt. Um dies zu beschleunigen, ist es sinnvoll, das Volumen in Blöcke zu zerteilen, die jeweils für sich gespeichert sind und damit kleinere Speicherdistanzen zwischen benachbarten Voxeln vorlegen. [Parker et al., 1999] geben einen Algorithmus dafür an, der als *Bricking* bekannt ist. Jeder Block speichert meist noch eine zusätzliche Hülle der adjazenten Voxel, was zwar zu einer gewissen Redundanz führt, aber für Interpolationszwecke nützlich ist.

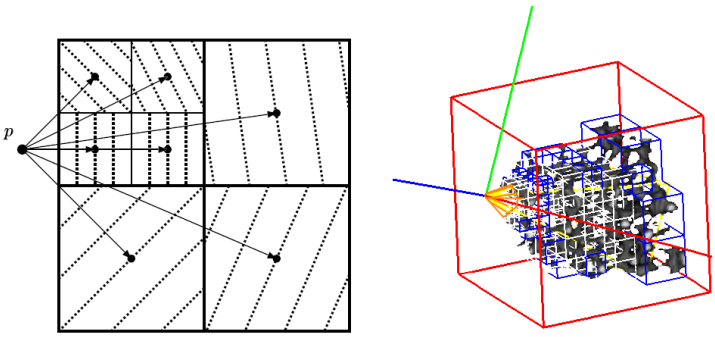

Abbildung 3.17: Bricking mit Level-of-Detail. Das Volumen wird in Blöcke zerteilt und nahe der Kamera werden höher aufgelöste Volumentexturen verwendet werden. [LaMar et al., 1999]

Für GPUs kann dieses Verfahren zusätzlich noch genutzt werden um die Speicherkosten zu minimieren. Denn Grafikkarten sind in ihren Speichergrößen immer noch sehr eingeschränkt. Standardmodelle haben meist 256 MB und in diesen müssen auch der Framebuffer und andere Texturen gespeichert werden, weswegen große Datensätze wie der erwähnte nicht hineinpassen. Für ältere Grafikkarten wurden daher hierarchische Texturen entwickelt, in denen die Blöcke auch noch im Detailgrad organisiert sind [LaMar et al., 1999]. Wichtige Regionen, also Bereiche nahe der Kamera, benutzen höher aufgelöste Volumendaten, als andere, was in Abbildung 3.17 verdeutlicht werden soll. Damit kann Speicher gespart werden, da nicht alle Blöcke in voller Auflösung vorliegen müssen. In ihrem GPU-basierten Isooberflächenrenderer zeigen [Hadwiger et al., 2005] unter anderem eine Methode des *Brickings* für moderne Grafikkarten. Sie benutzen kein Verfahren, das die Auflösung verringert, sondern ein Cache System, das versucht, alle sichtbaren Blöcke im Speicher zu halten. Da viele Bereiche eines Volumens leer sein können, lohnt sich diese Form der Datenverwaltung. In einer Referenzvolumentextur werden die Blöcke gespeichert. In einer zweiten, die niedriger aufgelöst ist, werden die Verschiebungsvektoren in die Referenz gespeichert. Das Packen einer solchen Textur und der Zugriff sind in Abbildung 3.18 dargestellt. Zunächst muss folglich diese Translationstextur gelesen werden, um dann an der richtigen Stelle in der Referenztextur zuzugreifen. Das System erkennt Leerräume ebenso wie noch ungecachte Blöcke und reagiert mit Überspringen. Die CPU versucht ihrerseits

immer die notwendigen Blöcke zur Verfügung zu stellen.

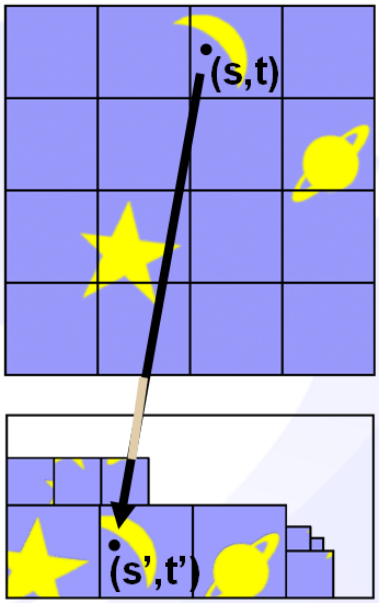

Abbildung 3.18: Kompression durch Verwerfen von Leerräumen. Die Zugriffs-Texturkoordinate muss für die Atlastextur umgerechnet werden. Quelle: [Engel et al., 2006]

Zwar sind große Datensätze, wie am Beispiel ersichtlich, in der Medizin durchaus möglich, doch ist die Region des Interesses viel kleiner. Im Anwendungsfall dieser Arbeit hat sich gezeigt, dass die meisten Datensätze maximal 512×512×128 Voxel groß waren und die eigentliche Region rund um den endoskopischen Engriff bereits mit 256×256×64 ausreichend beschrieben ist. Aus diesem Grund findet kein komplexeres Datenmanagement zur Laufzeit statt.

3.3.2 Überspringen von Leerräumen

Das *Bricking* hatte bereits das Potential gezeigt, Leerräume effizient zu überspringen (*Empty Space Skipping*). [Krüger and Westermann, 2003] nutzen dafür noch feiner aufgelöste uniforme Unterteilung, die aber nicht zur finalen Darstellung benutzt wird, sondern lediglich als Approximation der Volumenhülle fungiert. Die GPU kann durch vorher gespeicherten Minima- und Maximawert pro Block testen, ob dieser mit der aktuellen Transferfunktion sichtbar ist. Falls ja, wird der Schnittpunkt des Sichtstrahls mit dem Block in eine Textur geschrieben und abgebrochen. Der eigentliche Volumrenderingschritt wird dann mit Hilfe dieser Startpunkte durchgeführt. Voraussetzung dafür ist, dass die Startpunkte in ausreichender Präzision gespeichert werden können.
[Li et al., 2003] hatten ein ähnliches, aber komplexeres Verfahren vorgestellt, das eine genauere geometrische Repräsentierung nutzte, und auch die Sichtbarkeit der Blöcke exakter berechnet. Damit war es möglich, nicht nur Leerräume ausfindig zu machen, sondern auch die Verdeckung der Blöcke untereinander zu berücksichtigen.

3.3.3 Vorzeitiger Abbruch

Die im vorherigen Abschnitt erläuterte Technik führt zum nächsten besonders effizienten Beschleunigungsprinzip des Volumenrenderings, dem vorzeitigen Abbruch (*Early Ray Terminati-*

on). Ähnlich wie die vorgestellten Ideen zur Findung genauerer Startpunkte, lassen sich diese für die Endpunkte analog anwenden. Statt den Vorderseiten des ersten sichtbaren Blocks sind es dann die Rückseiten des letzten. Diese Art der Bestimmung der Endpunkte ist aber eine Annahme für den schlimmsten Fall, dass auch der äußerste Volumenpunkt noch einen Einfluss auf den Bildpunkt hat. Wie anfangs gezeigt wurde, ist jedoch die Komposition beim Rendern von vorn nach hinten so definiert, dass der Einfluss je nach Opazität abnimmt. Hierbei lassen sich die bedingten Sprünge der modernen GPU Programme nutzen, um bei Unterschreitung eines definierten Mindesteinflusses auf den Pixel die Integration abzubrechen. Interessiert nur die Isooberfläche, so würde das Programm beim ersten gefundenen Übergang in den festen Bereich stoppen.

3.3.4 Deferred Shading

Für Beleuchtung und Materialeffekte von Isooberflächen hat sich das *Deferred Shading* als sehr nützlich erwiesen. Hierbei werden alle farbgebenden Berechnungen im Bildraum durchgeführt. Die namensgebende „Verzögerung“ kommt zustande, indem zunächst Attribute, die für das Shading relevant sind, in Puffer gespeichert werden, welche die Größe des Ausgabebildes haben. Ursprünglich von [Saito and Takahashi, 1990] für Nicht-Photorealistisches Rendering entwickelt, speichern diese G-Buffer pro Pixel Oberflächennormale, Distanz zur Kamera, Position im Weltraum, Texturkoordinaten und Objektzugehörigkeit ab. Daraus können dann weitere Eigenschaften im Bildraum gewonnen werden, wie Silhouetten oder Krümmungen. Der Vorteil ist, dass die Berechnungen unabhängig von der Objektkomplexität nur von der Anzahl der Ausgabepixel abhängen.

In der Abbildung 3.19, die aus [Sigg et al., 2004] entnommen ist, wird gezeigt, dass auch beim GPU-basierten Rendering lediglich aus dem Tiefenpuffer alle für Materialeffekte wichtigen Parameter erzeugt werden können. Andere Implementationen, wie [Harris, 2004, Valient, 2007, Thibieroz, 2004], stehen für eine steigende Beliebtheit des Verfahrens in Computerspielen, wo es besonders praktisch ist, die Beleuchtung im Bildraum durchzuführen. Denn im Vergleich zum Objekt-basierten Shading kann es dort notwendig sein, die Geometrie immer wieder pro Lichtquelle zu zeichnen, oder es werden Pixel doppelt berechnet, die durch Eigenüberdeckung übermalt werden. Deswegen und wegen der hohen Flexibilität der finalen Komposition der Eingabepuffer ist es entsprechend effizient, die Beleuchtung im Bildraum durchzuführen. Denn hier werden garantiert nur die Pixel einbezogen, die auch am Ende zu sehen sind. Außerdem können die Beleuchtungseffekte auf die Regionen im Bild limitiert werden, die vom Licht erfasst werden. Ein großes Problem ist allerdings, dass das Verfahren nicht mit Transparenzen umgehen kann. Pro Pixel kann nur eine Oberfläche gespeichert werden, bei transparenten Flächen würden aber mehrere unterschiedliche Flächen auf den gleichen Pixel fallen können.

Für den im nächsten Kapitel beschriebenen Renderer wurde als Basis das GPU-Raycasting gewählt. Es bietet, wie in der bisherigen Ausführung deutlich wurde, für das Anwendungsgebiet ein optimales Verhältnis aus Leistung und geringem Implementierungsaufwand. Der vorzeitige Abbruch sowie eine sehr primitive Form des Überspringens von Leerräumen werden darüber hinaus benutzt um die Berechnungzeit zu verringern. Da verschiedene Effekte auf der Oberfläche simuliert werden sollen, wird *Deferred Shading* gewählt, um diese flexibel realisieren und kombinieren zu können.

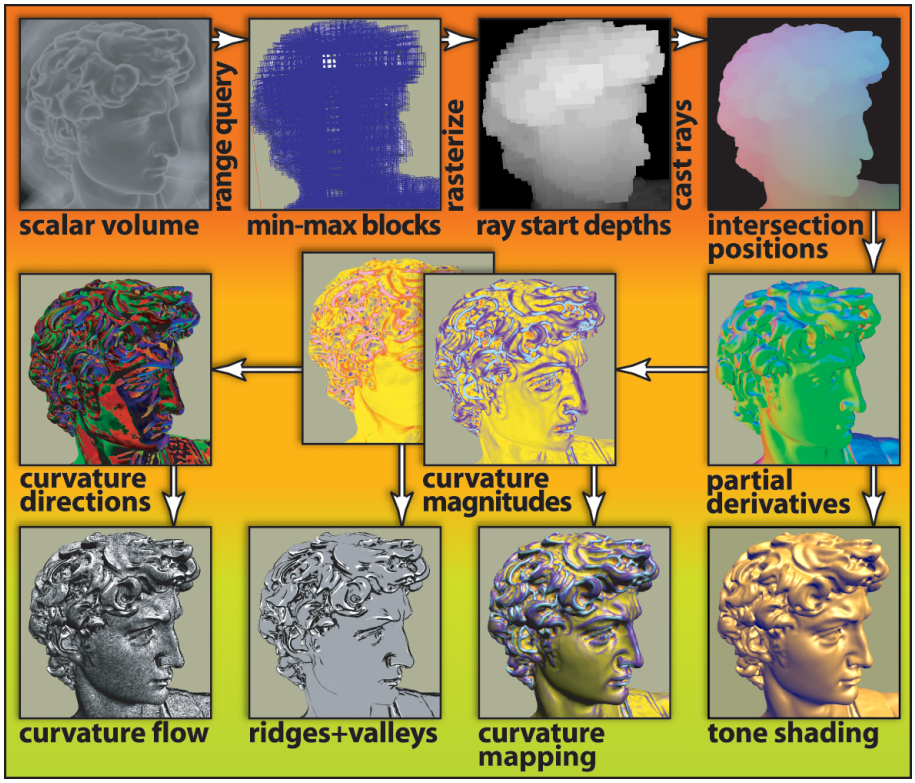

Abbildung 3.19: Erstellung der Zwischenergebnisse und Resultate von Deferred Shading. Nur die Erstellung der Oberflächenpunkte hat Objektkomplexität, alle anderen Operationen sind im Bildraum. Quelle: [Hadwiger et al., 2005]

Kapitel 4

Konzept und Entwicklung des Renderers

Der in dieser Arbeit entwickelte Renderer lässt sich in zwei prinzipielle Schritte zerlegen:

Extraktion. Zunächst werden alle benötigten Bildeigenschaften aus dem Volumen extrahiert. Hierbei kommt primär ein Ray-Casting Verfahren zum Einsatz, das für jeden Pixel im finalen Bild Attribute wie Distanz zur Bildebene, Oberflächennormale, Sekretdichte usw. liefert. Jeder Pixel steht dabei für einen Sichtstrahl.

Komposition. Nachdem alle notwendigen Eigenschaften gefunden wurden, werden diese kombiniert. Das Resultat der Komposition ist der finale RGB-Farbwert, der in den Framebuffer geschrieben wird. Die Komposition erlaubt es, dass Effekte wie Beleuchtung, Interaktion des Volumens mit polygonalen Flächen und dergleichen, den Ausgabewert beeinflussen.

In diesem Kapitel wird auf die Entwicklung dieser Schritte und die Probleme, die sich ergeben, eingegangen. Es gibt zwei Verfahrensgruppen, solche die Extraktion und Komposition in einem Renderingaufruf erledigen (*SinglePass*), und solche die mehrere Schritte brauchen (*MultiPass*). Letztere erzielen die qualitativ besseren Ergebnisse zu Lasten der Leistung. Im Folgenden wird kurz aufgeführt, welche Anforderungen von medizinischer Seite wesentlich sind.

4.1 Anforderungen an das Darstellungsverfahren

Der größte Unterschied zu generischen Volumenrenderern besteht darin, dass die Annahme, immer innerhalb des Volumens zu sein, viele Vereinfachungen nach sich bringt. Dies liegt auch an der Anatomie in den Nasennebenhöhlen, die aus engen Gängen und Kammern bestehen. Deren Darstellung sollte so realistisch wie möglich sein. Andere Systeme bieten bei Betrachtung anderer Daten, insbesondere wenn sich die Kamera außerhalb des Volumens befindet, deutlich bessere Ergebnisse. Optimierungsverfahren wie das EmptySpace Skipping greifen hier besser und der Gesamtanteil der Volumenpixel am Bild ist geringer. Dieser Leistungsgewinn kann in eine höhere Qualität der wenigen verbleibenden Pixel investiert werden.

So ist es zum Beispiel möglich segmentierte Volumendaten direkt darzustellen, wie in [Engel et al., 2006, Kapitel 16] und [Hadwiger, 2004] gezeigt. Dies ermöglicht die Visualisierung komplexer Strukturen und deren Verhältnisse untereinander, wie man Abbildung 4.1 entnehmen kann. Die Möglichkeit, sich diese zusätzlichen Informationen über die Anatomie des Patienten

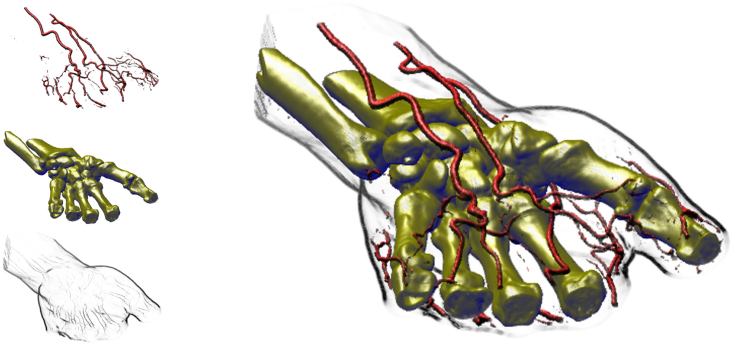

Abbildung 4.1: GPU-Raycasting von segmentierten Volumendaten. Den unterschiedlichen Objekten können verschiedene Materialeffekte zugeordnet werden. Quelle:[Hadwiger, 2004]

anzeigen zu lassen, spielte aber für den konkreten Anwendungsfall dieser Arbeit eine untergeordnete Rolle. Hauptziele der zu entwickelnden Visualisierungstechnik sind:

Realistische Darstellung. Gewebewände und Sekretschichten sollten ähnlich wie bei der realen Endoskopie dargestellt werden können. Wichtig sind vor allem Hinweise auf die Form der Anatomie. Sowohl Beleuchtung als auch Oberflächenstrukturen helfen die Formwahrnehmung zu verbessern. Der Grad des Realismus sollte variabel sein, damit das System auch für die Patientenaufklärung genutzt werden kann. Hierfür können stilisierte Methoden besser geeignet sein. Realistischere Darstellungen dienen in erster Linie dem Arzt eine höhere Vertrautheit zu bieten.

Einfache Benutzbarkeit. Neben der Darstellung ist eine einfache und schnelle Handhabung des Systems wichtig. Das bedeutet Verfahren mit aufwändigen Vorberechnungen wie Segmentierungen, oder Generierung von Dreiecksmodellen sind eher ungeeignet. Um die Planungszeit des Eingriffs zu unterstützen, sollten auch schnelle Ladezeiten erreicht werden, die wenig Interaktionen seitens des Benutzers benötigen.

Planungsunterstützende Funktionen. Das Renderingverfahren sollte mit normalen polygonalen Modellen interagieren können. Zum einen können segmentierte Strukturen hinter den Gewebewänden dargestellt, zum anderen auch polygonale Objekte für Werkzeuge genutzt werden. In [Preim and Bartz, 2007, Kapitel 13.2] werden Techniken erläutert, die es ermöglichen, mit Hilfe von Meßwerkzeugen in der endoskopischen Sicht Größen der anatomischen Verhältnisse zu ermitteln. Eine weitere Möglichkeit, die Planung eines Eingriffs zu dokumentieren oder dem Patienten zu verdeutlichen, wird mit Hilfe von Zeichenfunktionen ermöglicht [Preim and Bartz, 2007, Kapitel 16.3.3]. Durch Aufzeichnen von geplanten Schnitten oder Schattieren von Anomalien, kann der Arzt direkt auf die individuelle Anatomie eingehen. Der Renderer sollte es daher ermöglichen, zusätzliche Farbinformationen auf dem Gewebe anzubringen.

Individuelle Anpassungen. Die Hounsfield Werte des CT-Datensatzes können je nach Fall leicht variieren. Deshalb sollte es möglich sein, die Grenzen der Übergänge von festem Gewebe nach Sekret und Luft dynamisch anzupassen. Mit Hilfe der Variation dieser Werte können auch dynamisch diese Übergänge auf Knochengewebe oder andere Gewebeschichten gelegt werden, um so die gemessenen Daten optimal zu nutzen. Eine weitere Anpassungsmöglichkeit, die der Benutzer haben sollte, sind Parametereinstellungen, welche die

Darstellung beeinflussen. Je nach Chirurg können individuelle Vorlieben bei Beleuchtungseinstellungen oder Gewebefarben berücksichtigt werden.

Für das Modell zur realistischen Darstellung sind folgende Eigenschaften von Bedeutung:

Gewebekoordinaten im Raum Der Übergang von Sekret oder Luft nach Gewebe bildet die Gewebeoberfläche. Die Position des Schnittes mit dem Sichtstrahl im dreidimensionalen Raum kann zur Distanzbestimmung oder anderen Materialparameter genutzt werden. In dieser Arbeit wird die Koordinate primär zur Texturierung mit Oberflächendetails genutzt. Im System können aber auch Oberflächennormalen aus den Positionen gewonnen werden.

Distanz zum Gewebe Um die Tiefenwahrnehmung zu verbessern und den Effekt des Lichtabfalls zu simulieren, lässt sich die Distanz zwischen Gewebe und Kamera nutzen.

Oberflächennormale des Gewebes Die Beleuchtungseffekte benötigen die Normale des Gewebes. So werden die Lichtreflektionen approximiert und Rundungen besser wahrgenommen. Die Normalen werden auch genutzt, um Oberflächenstrukturen nachzubilden.

Distanz zur Sekretschicht Für Materialeffekte ist ähnlich wie beim Gewebe auch bei der Sekretschicht die Distanz zum ersten Übergang von Luft nach Sekret nützlich.

Dichte der Sekretschicht Wie viel Sekret sich entlang eines Sichtstrahls gebildet hat, wird mit dem Sekretdichtewert definiert und visualisiert.

4.2 Entwicklung der Extraktion

Nachdem Erfahrungen mit allen GPU-gestützten Verfahren gesammelt wurden, stellte sich schnell heraus, dass GPU-Raycasting die meisten Vorzüge bietet. Es lassen sich damit alle zuvor erwähnten Attribute gewinnen. Die Extraktion der Attribute, welche dann im Kompositionsschritt verarbeitet werden, erfolgt durch einen Renderingdurchlauf, der diese Attribute in mehrere Ausgabebild-große Puffer schreibt. Als Eingangsdaten werden lediglich die Kameratransformationen sowie eine 3D Volumentextur, welche die Dichtewerte der CT Daten enthält, und ihre metrische Größe benötigt.

4.2.1 Volumendaten und Wahl der Transferfunktion

Die Volumentextur und wie ihre Dichtewerte klassifiziert werden sind die primären Eingangsdaten für die Extraktion. Sowohl medizinisches Anwendungsgebiet, für welches der Wertebereich von Bedeutung ist, als auch die Fähigkeiten der Hardware beeinflussen die angewandten Methoden.

Einlesen der Daten

Obwohl bei klinischen Datensätzen oft sehr große Volumendaten entstehen, sind für den Anwendungsfall meist nur Ausschnitte notwendig. [Engel et al., 2006, Kapitel 17] widmet sich der Probleme, die beim Einladen von großen Datenmengen entstehen können. In dieser Arbeit kamen typische Datensätze aus dem klinischen Alltag zum Einsatz, die nicht sehr hoch aufgelöst waren. Der Wertebereich zwischen -512 und +1024 Hounsfield schien nach Absprache mit Medizinern als geeignet. Dies hielt den Präzisionsverlust in Grenzen, der bei der Konvertierung in 8-Bit pro Voxel vorgenommen wurde. Würde der vollständig gemessene Bereich von -1024 bis 3096,

was 12-Bit entspricht, auf 8-Bit reduziert werden, wären die Resultate beim flexiblen Ändern der Transferfunktion nicht mehr genau genug. Das System kann aber auch mit 16-Bit Präzision arbeiten, wenn dies erwünscht ist.

Filtering

Das Filtering der Werte geschieht durch die Grafikhardware automatisch trilinear. Das Hauptproblem ist die Tatsache, dass die Testdaten sehr anisotrop waren, weil die Auflösung entlang der Höhe deutlich geringer ist als in der Ebene. Die meisten Testdaten hatten ein Verhältnis von 6:1 was die Skalierung entlang der Höhe anging. Daraus resultiert ein hoher Interpretationsspielraum durch die Interpolation. [Hadwiger, 2004] gibt eine Methode an, wie moderne Grafikhardware auch höherwertigere Filterfunktionen einsetzen können. Eine weniger optimierte Fassung von trikubischen Filtering wurde auf Anfrage durch einen NVIDIA Mitarbeiter bereitgestellt und im Rahmen dieser Arbeit implementiert. In Abbildung 4.2 werden die Resultate der verschiedenen Filteringverfahren gezeigt. Für den interaktiven Einsatz eignet sich das trilineare Filtering am besten, Standbilder können aber mit komplexeren Filtern erzeugt werden. Um diese echtzeitfähig zu machen, muss jedoch erst eine optimierte Fassung wie in [Hadwiger, 2004] implementiert werden.

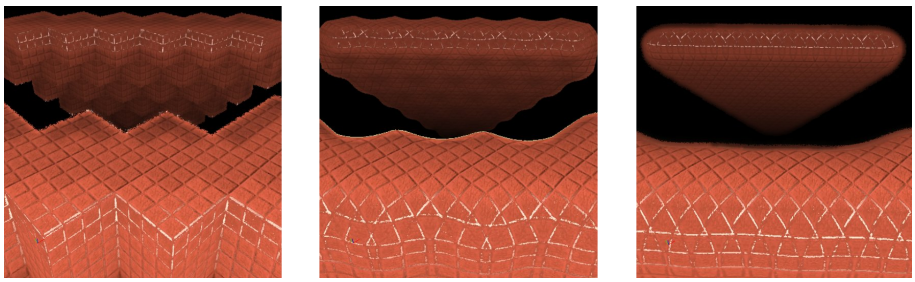

Abbildung 4.2: Unterschied zwischen nearest-neighbor (links), trilinearer (mittig) und trikubischer (rechts) Interpolation.

Transferfunktion

Nach dem Filtering erfolgt die Transferfunktion mit Post-Klassifikation, durch eine 1D Rampen Funktion wie in [Kniss, 2003] und [Engel, 2003] vorgestellt. Sie besteht aus einer Breite w und einem Zentrum c. Allerdings erfolgt die Klassifikation nicht durch das Sampling einer 1D Textur, welche die Klassifizierung vornimmt, sondern durch mathematische Evaluierung der Funktion:

$$\text{saturate}(v) = \begin{cases} 0 & \text{wenn } v < 0 \\ 1 & \text{wenn } v > 1 \\ v & \text{sonst} \end{cases} \tag{4.1}$$

$$\text{transfer}(v) = \text{saturate}((v - (c - w/2))/w) \tag{4.2}$$

Moderne Hardware besitzt eine ausreichend hohe Leistung für mathematische Operationen. Jedoch sind Texturzugriffe im Vergleich zu einfachen Berechnungen langsamer. Die Rampenfunktion liefert Werte im Intervall [0,1], diese werden danach interpretiert als:

$<$ **0.01** Luftgefüllter Raum

[0.01,0.99] Sekretschicht

$>$ **0.99** Festes Gewebe

4.2.2 Modifikation des GPU Raycasting

Das Sampling des Volumens geschieht im Extraktionsschritt. Für den konkreten Anwendungsfall wurde in dieser Arbeit eine eigene Lösung dafür entwickelt. Sie basiert auf Modifikation von Methoden, die bereits praktische Anwendung fanden.

Start- und Endpositionen des Sichtstrahls

Im Vergleich zum GPU Raycasting von [Krüger and Westermann, 2003] und [Scharsach, 2005] werden keine Puffer angelegt, in denen Start- und Endpositionen kodiert werden. In Abbildung 4.3 ist zu sehen, dass bei diesen Verfahren die Vorderseiten der Hüllgeometrie, welche das Volumen umschließt, zur Gewinnung der Startpositionen dient und die Innenseiten als Endpositionen.

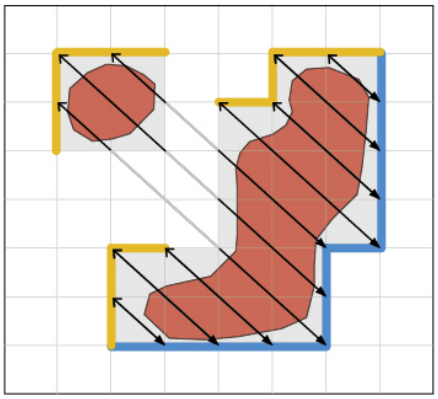

Abbildung 4.3: Das Volumen wird innerhalb der Ausdehnungen berechnet, das von Front- und Backfaces eingeschlossen wird. Quelle: [Scharsach, 2005]

Der Nachteil ist, dass die Verfahren in jedem Fall Fließkommapuffer benötigen, um für jeden Pixel die Positionen in ausreichender Genauigkeit zu speichern. Aus Leistungs- und Kompatibilitätsgründen wurde in SinusEndoscopy eine andere Technik genutzt. Ähnlich dem Prinzip der Vorder- und Innenseiten werden nur letztere gezeichnet. Die Geometrie dafür ist eine Box, welche die Dimensionen des Gesamtvolumens hat, und somit anisotropische Skalierungen mit einbezieht. Als Startposition dient der Schnitt einer Geraden, von der Kameraposition zu der Endposition, mit einer Startebene. Diese Ebene ist entweder die vordere Clip-Ebene der Kamera oder eine Tangentialebene an einer Kugel, die das Volumen umschließt. Je nachdem welche der beiden Ebenen näher am Zentrum des Volumens ist, wird diese als Startebene gewählt. Abbildung 4.4 veranschaulicht diesen Prozess, der als eine primitive Form des Überspringens von Leerräumen dient.

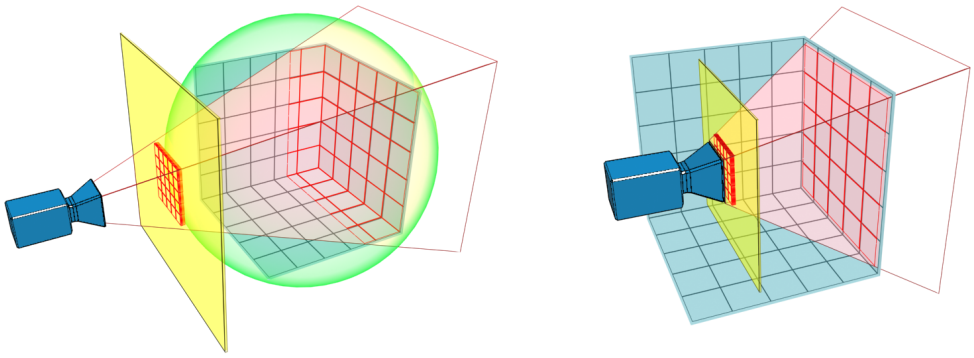

Abbildung 4.4: Überspringen von Leerräumen durch Starten der Strahlverfolgung an einer Tangentialebene (links), oder der vorderen Clip-ebene (rechts), ja nach Kameraposition

Die ersten Implementationen dieser Prozedur wiesen noch Fehler auf. Die Boxgeometrie muss ausreichend unterteilt werden, damit keine Interpolationsprobleme entstehen. Da die Schnittpunkte auf Vertexebene berechnet wurden, und deren Ergebnis danach linear über die Dreiecksfläche interpoliert werden, kann es zu Verzerrungen kommen, wie man in Abbildung 4.5 sehen kann. Im Versuch hat sich gezeigt, dass eine Unterteilung von 8×8 Quadraten je Seite ausreichte, um die Verzerrungen zu minimieren. Höhere Unterteilungen brachten keine wesentlichen Verbesserungen.

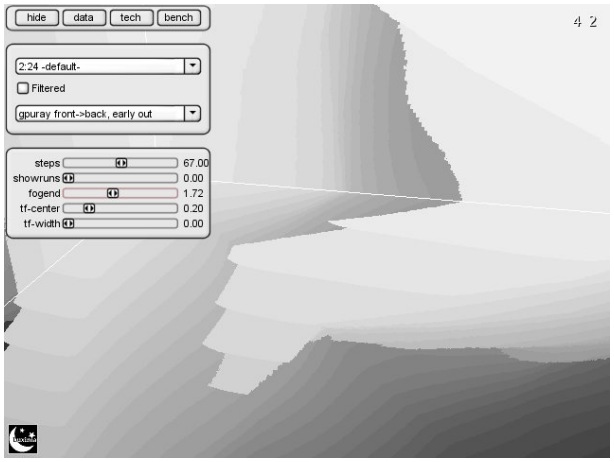

Abbildung 4.5: Verzerrung bei Hüllgeometrie ohne Unterteilung.

Ein anderes Hardware-abhängiges Problem bei der Berechnung der Schnittpunkte ist die Präzision, mit der der Vertexprozessor arbeitet. So wurde festgestellt, dass bei älteren Grafikkarten Schnittebenen, welche sehr nahe an der Kameraposition liegen, ignoriert werden. Die Startpunkte fallen dann auf die Kameraposition selbst, was jedoch nur bei der Komposition mit polygonalen Modellen Störungen verursacht.

Strahltraversierung

Nach der Erstellung von Start- und Endpositionen jedes Sichtstrahls innerhalb des Volumens wird dieser Strahl traversiert. Grundsätzlich können die Schrittweiten dabei variabel (Ebenen-

sampling) oder equidistant (Radialsampling) für alle Strahlen sein, wie in Kapitel 3.2.4 erwähnt. Beide Möglichkeiten wurden implementiert. Die Startposition muss beim Radialsampling angepasst werden, da die vorher bestimmte planare Startebene dafür nicht geeignet ist. Stattdessen muss eine gekrümmte Fläche benutzt werden, die durch Anwendung der folgenden Formel entsteht (S ist der Schnittpunkt mit der Ebene, C die Kameraposition und c deren Abstand zur Startebene):

$$\text{radial}(S) = \hat{\vec{CS}} \quad c \tag{4.3}$$

Die Traversierung erfolgt durch sukzessives Addieren eines Vektors auf die Samplingposition, welche zunächst die Startposition ist. Das System gibt eine maximale Sichtweite w und die maximale Anzahl an Schritten m innerhalb dieser vor. Dieser Additionsvektor wird wie folgt berechnet (S = Startpunkt, E = Endpunkt, $dist(x)$ = Distanz zur Startebene wie in Gleichung 4.6 später gezeigt):

$$\text{steps}(S,E) = \begin{cases} \text{dist}(E)/(w/m) & \text{Ebenensampling} \\ \|\vec{SE}\|/(w/m) & \text{Radialsampling} \end{cases} \tag{4.4}$$

$$\text{stepvec}(S,E) = \vec{SE}/(\text{steps}(S,E)\downarrow) \tag{4.5}$$

Eine Terminierung des Prozesses wird erreicht durch Erlangen der Endposition, nach einer maximalen Schrittanzahl oder wenn der Punkt innerhalb festen Gewebes ist. Vorher wird jedoch der erste Übergang von Leerraum nach Sekretschicht ermittelt. Diese zwei Übergangspunkte sind aber nur so genau wie die verwendete Schrittweite. Je geringer diese gewählt wird, desto genauer ist das Ergebnis, aber auch langsamer in der Berechnung. [Sigg et al., 2004] stellten ein Verfahren vor, wie mit konstantem Aufwand das Ergebnis signifikant verbessert werden kann. Nachdem der erste Punkt im neuen Medium gefunden wurde, wird die Schrittweite halbiert und die Richtung umgekehrt. Ist der nächste Punkt innerhalb des Mediums, wird die Richtung beibehalten und wieder halbiert, ist er außerhalb, wird die Richtung wieder gewechselt. Diese binäre Suche nähert sich dem eigentlichen Übergangspunkt. Anwendungstests haben gezeigt, dass bei der Standardsichtweite und Schrittzahl fünf Iterationen bei der Suche optimal für das Ergebnis sind. Sollten die Schrittweiten geringer werden, kann auch die Iterationsanzahl vermindert werden. Dies ist im Moment jedoch technisch noch nicht umgesetzt, so dass die Iterationsdurchläufe immer gleich oft geschehen. Da immer nur an gewissen Punkten die Klassifikation vorgenommen wird, kann es dennoch passieren, dass Strukturen übersprungen werden oder bei der Suche der Richtungswechsel falsch vorgenommen wird.

Der grundsätzliche Ablauf der Strahltraversierung ist in Abbildung 4.6 illustriert. Neben der Gewinnung der Übergangspunkte ist auch die Sekretdichte von Bedeutung. Zwei Verfahren wurden dafür getestet. Das erste geht von einer konstanten Dichte zwischen erstem Sekretpunkt und dem Gewebepunkt aus. Dies kann zu Fehleinschätzungen führen was die eigentliche Sekretdichte angeht, da Leerräume zwischen den beiden Punkten ignoriert werden. Für den Gesamteindruck liefert diese Vereinfachung jedoch die visuell besten Ergebnisse.

Eine genauere Variante ist die Sekretdichte zu akkumulieren. Der Dichtewerte wird mit Null initialisiert und um einen festen Betrag erhöht, wenn der Samplingpunkt nicht im Leerraum ist. Die Veränderungen, die sich durch die binäre Suche ergeben, fließen in diesen Wert ein, so dass er je nach Situation erhöht oder vermindert wird. Die Vorgehensweise dafür ist in Abbildung

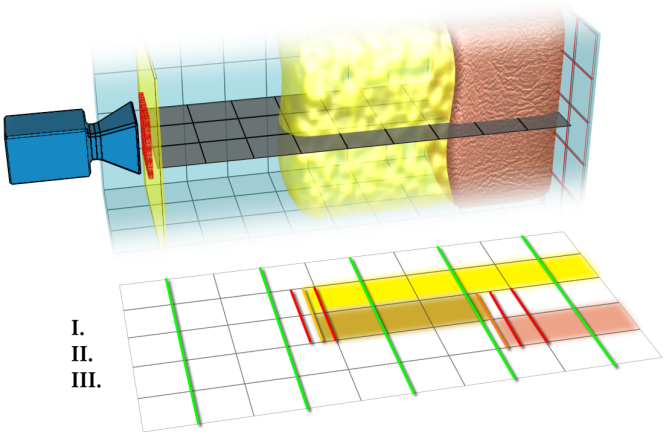

Abbildung 4.6: Ablauf der Strahltraversierung. Zunächst Suche nach der Sekretschicht (I), danach Ermittlung des Übergangs zum festen Gewebe (III). Die Sekretdichte (II) wird als konstant definiert. Die Schrittweite ist grün, die binäre Suche in rot dargestellt.

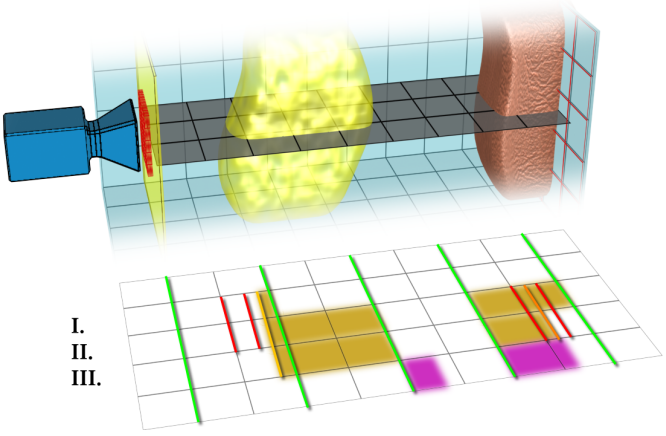

Abbildung 4.7: Strahltraversierung mit akkumulierter Sekretdichte. Die Sekretdichte (II) wird für alle Schritte, die außerhalb der Leerenraums, sind bis zum Treffen auf Gewebe (III) erhöht. Die purpurnen Regionen stellen die Fehler dar.

4.7 dargestellt. In der Illustration ist auch der Fehlerwert dieser Methode purpur unterlegt. Dieser Fehler ergibt sich, weil das Verlassen der Sekretschicht nicht genau ermittelt wird, genauso wenig wie die unmittelbare Dicke der Schicht über dem Gewebe. Allerdings können Wechsel zwischen Leerraum und Sekretschicht besser miteinbezogen werden als im vorherigen vereinfachten Verfahren und der Fehlerwert sinkt mit größerer Schrittzahl.

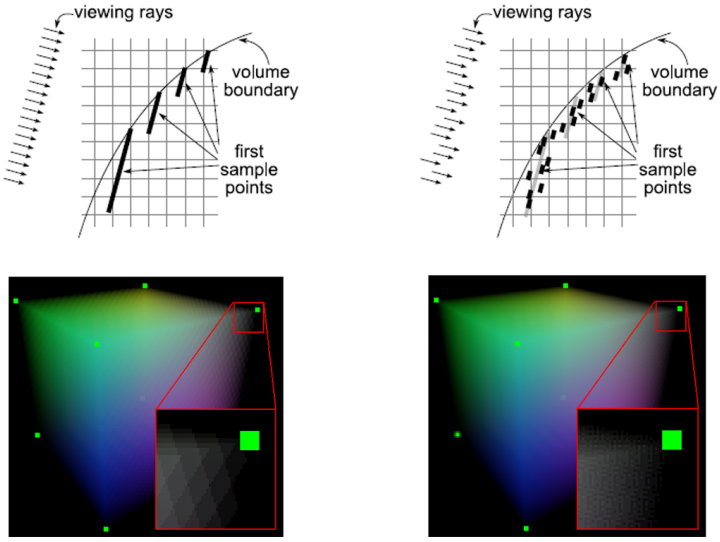

Abbildung 4.8: Interleaved Sampling. Die obere Illustration zeigt die Anwendung bei indirektem, während die unteren Bilder für direktes Volumenrendering gilt. Auf der rechten Seite ist zu sehen, dass Artefakte verringert werden. Quellen: [Termeer et al., 2006] (oben), [Keller and Heidrich, 2001] (unten)

Beide Verfahren profitieren vom *Interleaved Sampling*. Hier werden die Startpositionen des Strahls leicht entlang seiner Richtung verschoben. Der Betrag ist von der Pixelposition im finalen Ausgabebild abhängig, aber unterschiedlich für benachbarte Pixel. Die Vorteile dieses Verfahrens wurden in [Keller and Heidrich, 2001] ausgeführt und auch auf das Volumenrendering angewandt. Selbst bei hohen Schrittweiten sind die Artefakte der sichtbaren Ebenen, die durch die Schittpunkte aufgespannt werden, deutlich weniger auffällig, wie in Abbildung 4.8 zu sehen. [Scharsach, 2005] implementierten das Verfahren für GPU-Raycasting und ermittelten die Translationen mathematisch im Fragmentprozessor aus der Position des Pixels. Der Renderer dieser Arbeit benutzt eine 2D Textur, die über das gesamte Ausgabebild gelegt wird und für jeden Pixel einen zufälligen Grauwert liefert, welcher als Skalierungsfaktor für die Translation dient. Diese Skalierung kann zusätzlich mit einem Parameter von außen leicht gesteuert werden. In Abbildung 4.9 sind die Ergebnisse dieses Verfahrens zu sehen.

Ausgabe der Werte

Nach Terminierung des Strahls werden die Ausgabewerte geschrieben. Der Distanzwert eines Schnittpunkts h zur Startebene, welche die Normale n hat und auf dem Punkt e liegt, wird durch ein Skalarprodukt mit der Ebenengleichung gewonnen:

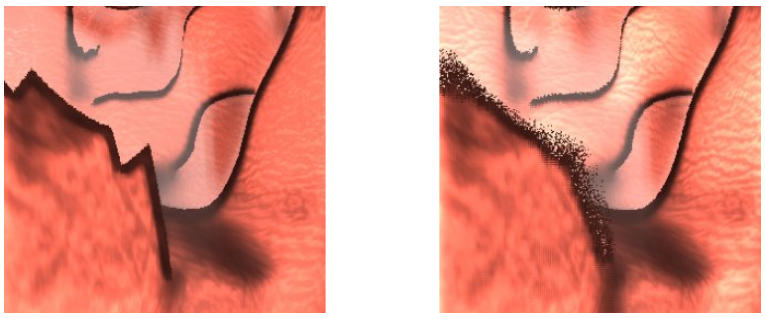

Abbildung 4.9: Anwendung von Interleaved Sampling. Das Shading wurde absichtlich verfremdet, um die Konturen zu betonen. Links waren noch gerade Kanten zu sehen, die rechts abgerundeter erscheinen. Die Kanten fallen bei Bewegung noch stärker negativ auf.

$$\text{dist}(h) = \begin{pmatrix} h_x \\ h_y \\ h_z \\ 1 \end{pmatrix} \cdot \begin{pmatrix} n_x \\ n_y \\ n_z \\ -(n \cdot e) \end{pmatrix} \tag{4.6}$$

Diese Methode stellte sich als präziser und effizienter heraus, als durch Addition je Samplingschritt die Distanzen zu ermitteln. Gespeichert in Texturen wird der Sekretdichtewert, die Distanzen der Übergangspunkte, sowie ein Maskenwert. Letzterer ist entweder null oder eins, je nachdem ob der Strahl ohne oder mit gefundenem Gewebeübergang terminiert wurde. Die Positionen der Punkte selbst werden nicht geschrieben. Dies liegt daran, dass wie bei den Start- und Endpunkten alle Methoden auch ohne Fließkommatexturen auskommen. Deren Verwendung dient ausschließlich zur Verbesserung der Ergebnisse, ist aber für die Speicherung keine notwendige Bedingung. Zwei Aufgaben können jedoch direkt im Extraktionsschritt mit Hilfe dieser Punktkoordinaten durchgeführt werden, zum einen das Sampling der Oberflächentexturen und zum anderen die Gewinnung der Oberflächennormalen. Andererseits lassen sich aber auch die Punktkoordinaten mit Hilfe einer Rückprojektion rekonstruieren. Für den Tiefenwert d, die Sichtrichtung v, der *ModelView*- und *Projection*-Matrix, sowie dem Offset o zwischen vorderer Clip-Ebene und Startebene, ist diese Umrechnung wie folgt definiert:

$$\text{invproj}(v, d) = \text{ModelView}^{-1}((\text{Projection}^{-1}v)(d + o)) \tag{4.7}$$

Diese Rückprojektion ist dann sinnvoll, wenn die Koordinaten von einer Glättung der Distanzwerte profitieren sollen. Beide Methoden finden in den verschiedenen umgesetzten Verfahrensvariationen Anwendung.

4.2.3 Generierung der Oberflächennormalen

Die Ausgabe der Oberflächennormalen noch im Extraktionsschritt hat sich als die beste Variante herausgestellt. Grundsätzlich ist es möglich, die Gradienten vorher durch Zentrale Differenz oder einem 3×3 Sobel-Operator zu gewinnen und diese in der Textur zu speichern. Dies würde jedoch die Speicherkosten erhöhen und liefert, wie in Abbildung 4.10 zu sehen, qualitativ schlechtere Ergebnisse als die Gewinnung der Normalen zur Laufzeit.

Jene Echtzeit-Methode wird primär mit Hilfe der Zentralen Differenz erreicht. Entlang der Hauptachsen werden die Werte von sechs umgebenden Punkten ermittelt. Die Schrittweite kann variiert werden. Experimentell wurde ermittelt, dass ein halber Voxelabstand qualitativ beste

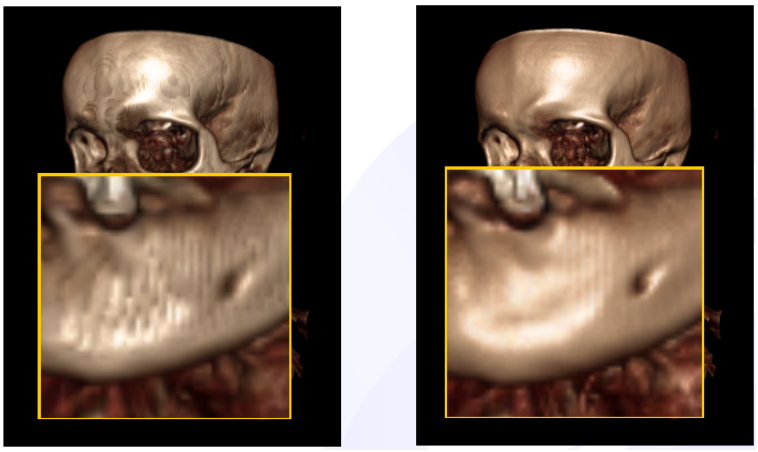

Abbildung 4.10: Vergleich zwischen vorher gespeicherten (links) und zur Laufzeit (rechts) erstellten Normalen. Quelle: [Engel et al., 2006]

Ergebnisse erzielte. Die Normale ergibt sich dann für die Stelle s und dem Volumenzugriff ϕ bei einer Schrittweite von 1:

$$n(s) = \begin{pmatrix} \phi(s_x - 1, s_y, s_z) \\ \phi(s_x, s_y - 1, s_z) \\ \phi(s_x, s_y, s_z - 1) \end{pmatrix} - \begin{pmatrix} \phi(s_x + 1, s_y, s_z) \\ \phi(s_x, s_y + 1, s_z) \\ \phi(s_x, s_y, s_z + 1) \end{pmatrix} \tag{4.8}$$

Sehr wichtig an dieser Stelle zu erwähnen ist, dass die Normalen skaliert werden müssen. In [Preim and Bartz, 2007, Kapitel 2.2.4] wird auf die Interpolationsartefakte hingewiesen, die bei anisotropen Datensätzen entstehen können. Die Normale muss, bevor sie normalisiert wird, mit den Größen des Volumens komponentenweise dividiert werden.

Neben der Gewinnung der Normale aus benachbarten Volumenpunkten wurden zwei weitere Verfahren implementiert und evaluiert. Diese würden kein weiteres Volumensampling benötigen und sind so für Grafikkarten geeignet, bei denen sich dies besonders negativ auf die Leistung auswirkt. Im einfachsten Fall wird der Gradient aus den Distanzwerten der Gewebeoberfläche gewonnen. Die Formel zur Erstellung der Normale für eine Pixelkoordinate p, dem Tiefenwertzugriff θ und einem benutzerdefinierten Wert u folgt:

$$\begin{aligned} b &= \theta(p_x, p_y) \\ n &= \begin{pmatrix} 0 \\ 0 \\ u \end{pmatrix} + \begin{pmatrix} b - \theta(p_x + 1, p_y) \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} \theta(p_x - 1, p_y) - b \\ 0 \\ 0 \end{pmatrix} + \\ &\quad \begin{pmatrix} 0 \\ b - \theta(p_x, p_y + 1) \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \theta(p_x, p_y - 1) - b \\ 0 \end{pmatrix} \\ \text{normal}(p) &= \hat{n} \end{aligned} \tag{4.9}$$

Dies ist eine sehr starke Vereinfachung, da die Methode eigentlich nur für orthogonale Projektion valid ist. Ein $u > 0$ skaliert die Normalen zur Kamera hin unterschiedlich stark

und soll ebenso sicherstellen, dass nach einer Normalisierung von gleichen Tiefenwerten der Vektor $(0, 0, 1)$ entsteht. Für einfachste Beleuchtungseffekte, welche die Rundungen rund um Silhouetten sichtbar machen, ist die Methode aber dennoch ausreichend. Eine genauere Variante ist die Rekonstruktion der Oberflächenpunkte im Kameraraum. Zwar könnten die Trefferpunkte direkt transformiert werden, stattdessen wurden aber die Punkte aus den Distanzwerten und einer invertierten Projektion rückgewonnen. Die Normalenberechnung erfolgt ähnlich wie im einfachen Fall aus den Differenzen der benachbarten Pixel, ist aber nun korrekt an die perspektivische Projektion angepasst. Das System lässt bei beiden Verfahren zu, dass Abstände zum Nachbarpixel oder diverse Skalierungswerte manuell geregelt werden. Dabei kann verhindert werden, dass Normalen über Unstetigkeiten der Tiefenwerte hinweg erstellt werden. So wird für jeden Nachbarpixel ermittelt, ob er entlang der Blickrichtung innerhalb einer vom Benutzer festgelegten Distanz liegt. Bei Überschreiten dieses Wertes wird von einer Silhouette ausgegangen. Diese Information kann genutzt werden, um die Normale so zu beeinflussen, dass sie in eine Ebene parallel zur Bildebene gedreht wird. Auf diese Weise können Silhouetten verstärkt werden, auch wenn die tatsächliche Normale an der Stelle nicht unbedingt diese Orientierung hat.

Abbildung 4.11: Ergebnisse der Normalenerstellung: Gradient aus Tiefenpuffer (links), Übergangspunkte im Kameraraum (mitte) und Zentraler Differenz im Volumen.

Der Normalenpuffer enthält die Normalen im Kameraraum, das heißt die Weltnormalen werden mit der Kameramatrix transformiert. Der Grund hierfür ist, dass die Methoden austauschbar sind und der Kameraraum erschien am sinnvollsten für die Umsetzung. Zum einen liegt dies daran, dass die Beleuchtung damit einfacher ausfällt, zum anderen lassen sich beim Glätten leichter Effekte erzielen, welche die Silhouetten verstärken. Abbildung 4.11 zeigt die unterschiedlichen Resultate der drei Verfahren. Um die Normalen als Farbwerte im Wertebereich [0,1] zu kodieren, wurde der ursprüngliche Wertebereich von [-1,1] zunächst halbiert und dann um +0.5 verschoben. Daraus ergibt sich, dass der Blauanteil im Normalenpuffer höher ist, weil sichtbare Oberflächen tendenziell zur Kamera zeigen, was der Z-Achse und somit dem blauen Farbkanal bei einer RGB Kodierung entspricht.

4.2.4 Glättung der Zwischenergebnisse

Es wurde bereits angedeutet, dass die Zwischenergebnisse aus Distanz-, Dichte- und Normalenwerten geglättet werden können. Die Programmierbarkeit der Grafikkarten kann für diverse Bildverarbeitungsschritte genutzt werden [Mitchell, 2002]. Mit Hilfe einer Glättung können Artefakte beseitigt werden, die aus zu großen Schrittweiten entstehen. Auch ergibt sich beim *Interleaved Sampling* zwangsläufig ein Rauschen durch die zufälligen Verschiebungen, welches man durch die Anwendung von Konvolutionsfiltern wieder abschwächen kann. Zur Glättung wurde eine separable Konvolution eingesetzt, so dass die Glättung sich aus einem horizontalen

und einem vertikalen Durchlauf zusammensetzt. Je nach Durchlauf werden entweder entlang der Breite oder Höhe des Bildes die benachbarten Pixel gewichtet summiert und ausgegeben. Zur Anwendung kam ein symmetrischer Tent-Filter von neun Wichtungsfaktoren, es ist aber auch möglich einen Gauss-Filter nachzubilden. In Abbildung 4.12 werden die Ausgaben ohne jegliche Verbesserungen denen mit qualitätssteigerenden Maßnahmen gegenübergestellt. Letztere setzen sich aus Interleaved Sampling, binäre Suche der Übergangspunkte sowie einiger Glättungsdurchläufe zusammen.

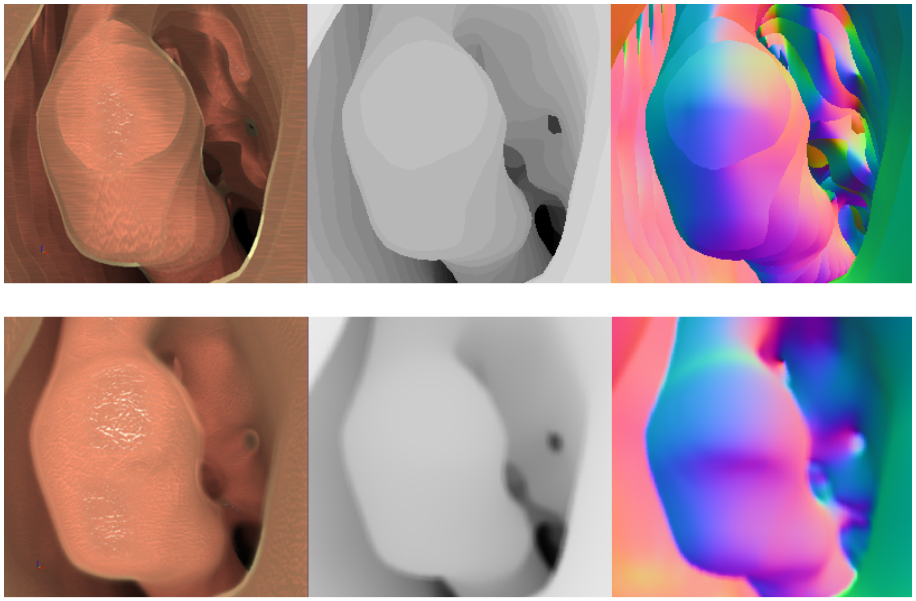

Abbildung 4.12: Oben: Resultate ohne Qualitätsverbesserungen. Auffällig sind die starken Ebenenartefakte.
Unten: Ergebnisse mit binärer Suche und Interleaved Sampling sowie einer einfachen Glättung.

Alle Glättungsvorgänge können auf drei Arten, welche in der Arbeit entwickelt wurden, durchgeführt werden. Sie brauchen in jedem Fall mehrere Renderingaufrufe und können daher nur von den *MultiPass* Verfahren verwendet werden. Die Basismethode benutzt die Kontributionen aller Nachbarpixel. Eine zweite Methode verwirft mit dem Maskenwert alle Pixel, die nicht Teil einer gefundenen Gewebeoberfläche sind. Die letzte und aufwändigste Methode filtert anisotrop, ähnlich wie bei der Normalengewinnung. Sie verhindert damit, dass über Silhouetten hinweg geglättet wird. Beim visuellen Vergleich der Methoden durch einen HNO-Chirurg wurde aber festgestellt, dass die anisotrope Weise nicht so gut bewertet wurde. Obwohl sie korrekter ist und Eigenüberlappungen trennt, wurden die Artefakte, die bei den anderen Verfahren entstehen, vorgezogen. Dies liegt daran, dass die fälschlichen Beleuchtungseffekte und Veränderungen der Distanzwerte an Silhouetten eher den Eindruck vermitteln, dass das Gewebe von einer Sekretschicht umgeben ist, wie auch in Abbildung 4.13 verdeutlicht wird.

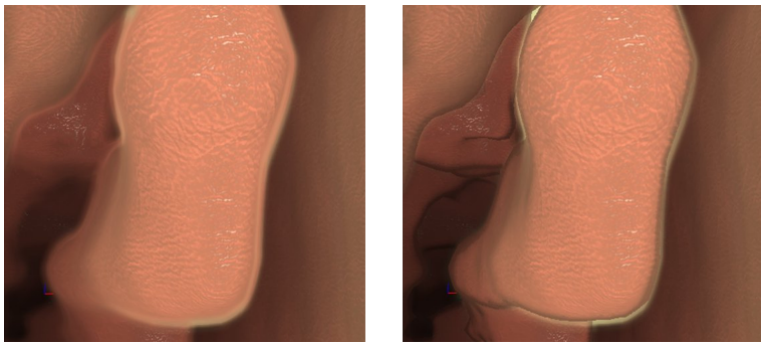

Abbildung 4.13: Unterschied von einfachem (links) und anisotropischem (rechts) Glätten.

4.3 Kompositionsschritte

Nachdem die Zwischenergebnisse geschaffen wurden, werden diese genutzt, um weitere Effekte zu erstellen. Alle Effekte werden abschließend zu einem Farbwert zusammengefasst. Da die Effekte weitestgehend autonom sind, müssen auch nicht alle angewendet werden, sondern können beliebig kombiniert werden. In Abbildung 4.14 ist eine Übersicht aller an der Komposition beteiligten Zwischenergebnisse sowie die Zusammensetzung der finalen Ausgabe dargestellt. In den nachfolgenden Abschnitten wird näher auf jeden einzelnen Vorgang eingegangen.

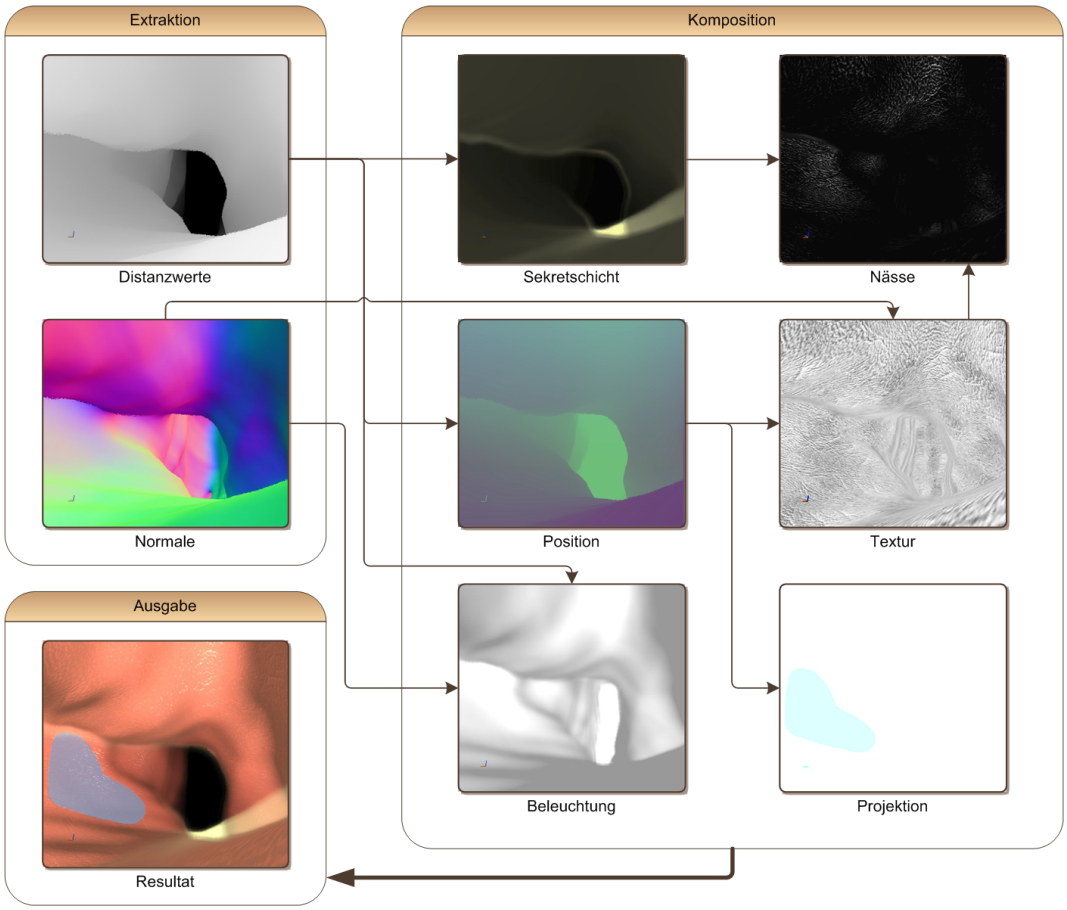

Abbildung 4.14: Zusammensetzung der verschiedenen Kompositionseffekte. In diesem Beispiel werden die Positionen der Schnittpunkte in der Komposition zurückgewonnen und die Normalen bei der Extraktion aus dem Volumen erstellt.

4.3.1 Beleuchtung

Um den Oberflächen mehr Ausdruck zu verleihen, werden sie durch eine Lichtquelle beleuchtet (siehe Abbildung 4.15). Die Bauweise des Endoskops führte zur Vereinfachung des Beleuchtungsmodells, da Blick- und Lichtrichtung identisch sind. In der Arbeit von [Kratz, 2006] wird die Verwendung von *Deep Shadowmaps* zur Schattenberechnung in Volumendaten vorgestellt. Schatten ist ein wichtiger visueller Hinweis für die Lagebeziehungen im Raum. Wie man in Abbildung 4.16 sehen kann, wirkt das Ergebnis mit Schatten plastischer.

Der relativ hohe Aufwand bei der Erstellung und Anwendung jener Schatten ließ jedoch

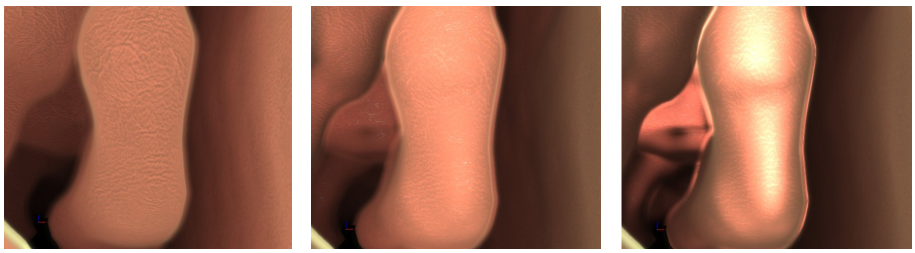

Abbildung 4.15: Auswirkung der Beleuchtung. Links wird nur linearer Lichtabfall benutzt, während mittig und rechts auch die Oberflächennormale hinzugezogen wird. Die beiden rechten Bilder unterschieden sich nur in Parametereinstellungen.

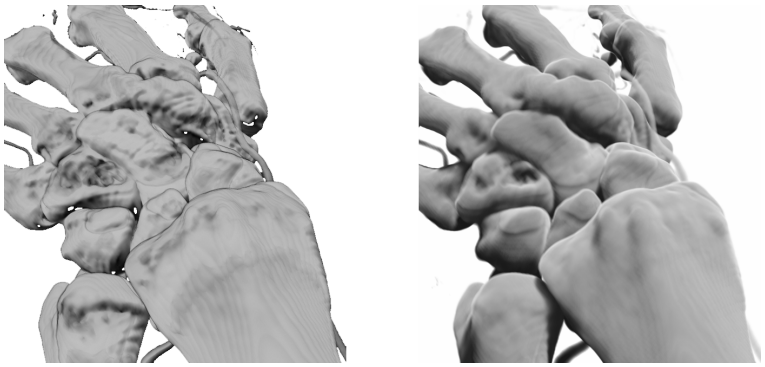

Abbildung 4.16: Komplexe Beleuchtung mit Deep Shadowmaps. Links werden keine Schatten dargestellt, rechts hingegen schon. Quelle: [Kratz, 2006]

kaum interaktive Bildraten zu. Für hochaufgelöste Ausgaben sind komplexere Schattierungsverfahren aber durchaus sinnvoll, doch selbst dann lohnt es sich beim Endoskop kaum. Blick- und Lichtrichtung sind nahezu parallel und verhindern so praktisch die Entstehung von Schatten. In dem entwickelten System wurde für die Beleuchtungsgleichung das Phongsche Modell benutzt und um eine Potenzierung erweitert, damit der Kontrast verändert werden kann (d ist die Distanz zur Startebene, w die maximale Sichtweite und n die Normale im Kameraraum):

$$\text{light}_{\text{parallel}}(n,d) = (1 - \text{saturate}(d/w)) \quad ((\text{saturate}(n \cdot \begin{pmatrix}0\\0\\1\end{pmatrix}) \, s)^p + a) \tag{4.10}$$

Die drei Beleuchtungswerte s, p und a können im System interaktiv vom Benutzer verändert, und so individuellen Wünschen angepasst werden. Bei der Lichtrichtung ist es möglich, entweder eine gerichtete parallele Lichtquelle oder ein Punktlicht zu verwenden. Der Lichtabfall wird über die Distanzwerte der Oberfläche simuliert und nimmt linear bis zur maximalen Sichtweite zu.

Während der Entwicklung wurde neben dem Phongschen Modell auch mit einem Bildbasiertem Verfahren experimentiert (*Image-based Lighting*). [Winnemöller and Bangay, 2002] stellten eine Technik vor, bei dem die Beleuchtungsstärke aus einer 2D Textur gewonnen wird. In Abhängigkeit vom Winkel der Oberfläche zur Blickrichtung und dem Winkel zwischen Oberfläche und Lichtstrahl wird dieser Textur der Farbwert entnommen, welcher Beleuchtungsfarbe und -stärke wiederspiegelt. Mit diesem Verfahren lassen sich stilisierte Visualisierungen erzielen wie

in Abbildung 4.17 gezeigt. Bei den Beleuchtungstexturen wurde der steigende Lichteinfallswinkel entlang der Höhe von oben nach unten kodiert und der Sichtwinkel der Breite nach.

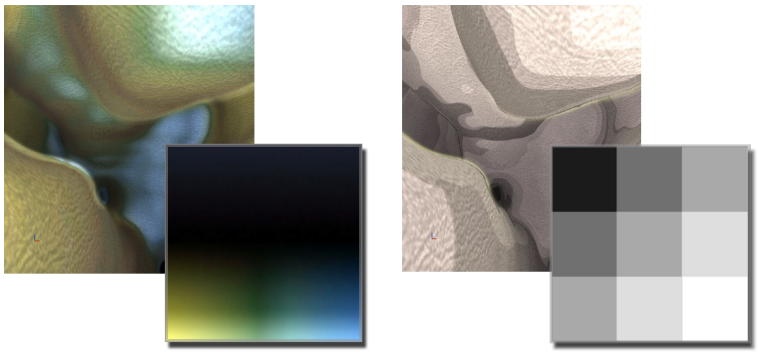

Abbildung 4.17: Image-based Lighting. Die Textur, die als Grundlage für die Lichtintensität dient, überlagert jeweils das Resultat.

4.3.2 Darstellung von Oberflächenstruktur

Die Beleuchtung hilft bereits, die Oberflächenformen besser wahrzunehmen, doch verbessern Oberflächendetails diese zusätzlich [Interrante, 1997]. In Abbildung 4.18 werden zwei Aufnahmen gezeigt, wobei zu erkennen ist, dass die Texturen deutlich den Verlauf von Krümmungen hervorheben. Dieser Effekt wurde durch triplanares Texturieren erreicht, der in Abbildung 4.19

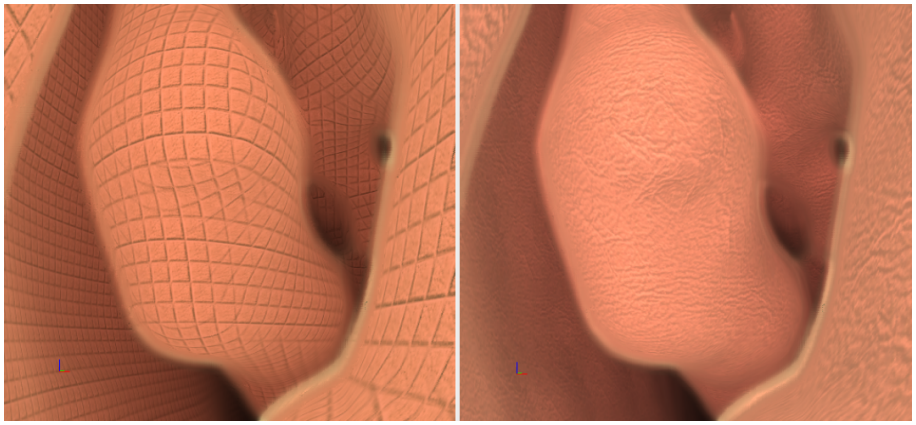

Abbildung 4.18: Anwendung verschiedener Detailtexturen. Reguläre Formen (links) verbessern die Wahrnehmung der Krümmungen, können aber zu Unschärfeartefakten führen, die bei organischen Texturen (rechts) nicht auffallen.

illustriert ist. Grundsätzlich wäre es zwar möglich gewesen, jedem Punkt im Raum einen Helligkeitswert zuzuweisen, zum Beispiel mit der Methode von [Perlin, 1985], doch erschien die Verwendung von 2D Texturen einfacher und performanter. Denn Perlin-Noise müsste entweder in einer weiteren 3D Textur gespeichert oder für jeden Punkt berechnet werden. Die Benutzung von 2D Texturen hat auch den Vorteil, dass man einen Eindruck davon bekommt wie das Resultat aussieht. Die Abbildung zeigt auch, dass Überlappungsartefakte der Projektionen der Texturen bei organischen Mustern kaum auffallen.

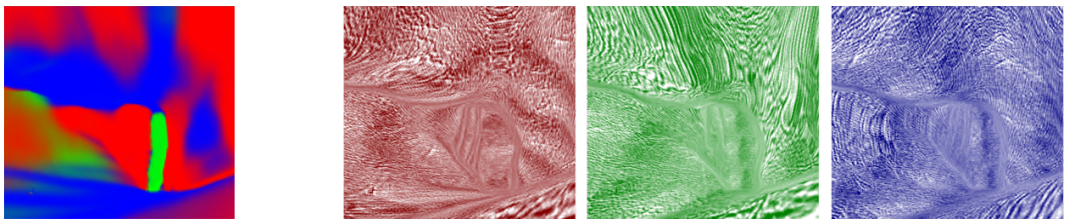

Abbildung 4.19: Triplanare Texturprojektion. Links werden den XYZ-Achsen die RGB-Farben zugeordnet, um die Wichtungen darzustellen. Rechts sind die einzelnen Projektionsrichtungen gezeigt, die abschließend gewichtet vermischt werden.

Das Aufbringen der Texturen erfolgt durch Sampling der Detailtextur mit Hilfe der Gewebekoordinate. Jene Koordinate wird entlang der Hauptachsen projiziert und dient dann als 2D Koordinate für den Zugriff auf die Textur. Dabei können die Zugriffskoordinaten vom Benutzer im System skaliert werden, was zur Folge hat, dass die Detailtextur unterschiedlich groß dargestellt werden kann. Anschließend werden die drei Farbwerte mit Hilfe der Oberflächennormale gewichtet vermischt. Dieses Verfahren wurde aus [Geiss and Thompson, 2006] entnommen und lässt sich wie in Abbildung 4.20 umsetzen.

```
1
2     // blend dependent on normal
3     scalar3 blend_weights= abs(normalworld.xyz) -0.2;
4     blend_weights*= 7;
5     blend_weights= pow(blend_weights, 3);
6     blend_weights= max(scalar3(0,0,0), blend_weights);
7
8     // and so they sum to 1.0:
9     blend_weights/= dot(blend_weights, 1);
10
11
12    scalar3 colx = tex2D(texdetail,scalar2((pos.yz)*texscale.x)).xyz;
13    scalar3 coly = tex2D(texdetail,scalar2((pos.xz)*texscale.y)).xyz;
14    scalar3 colz = tex2D(texdetail,scalar2((pos.xy)*texscale.z)).xyz;
15
16    scalar3 detailimg = (colx*blend_weights.x) +
17                        (coly*blend_weights.y) +
18                        (colz*blend_weights.z);
19
20    return saturate(detailimg);
21
```

Abbildung 4.20: Cg-Code für triplanare Texturierung.

Wichtig für die Oberflächenstruktur ist Framekoherenz, das heißt beim Berechnen neuer Bilder soll der gleiche räumliche Punkt unverändert bleiben. Sowohl Kameratranslation als auch -rotation sollten die Struktur nicht verändern, lediglich Beleuchtungseffekte würden den Punkt beeinflussen. Dies hängt jedoch stark von der Genauigkeit der Normalen ab. Sie werden aus dem Kameraraum rücktransformiert in den Weltraum. Je unpräziser die Gewinnung der Normalen, desto stärkere Fehler treten bei den Projektionsgewichtung auf. Insbesondere die einfachste Methode neigt zu Artefakten, die entstehen, wenn nur entlang der Blickrichtung projiziert werden würden, wie in Abbildung 4.21 zu sehen.

Wird das Sampling der Detailtexturen noch am Ende des Raycasting-Schritts durchgeführt, ergibt sich das Problem, dass insgesamt drei Farbwerte zwischengespeichert werden müssten, um später mit Hilfe der Normalen gewichtet zu werden. Um die Speicherkosten zu minimieren, wurden je drei Grauwerte auf die drei Farbkanäle verteilt und in der Komposition wieder aufgetrennt. Die besten Ergebnisse liefert jedoch das Sampling in der Komposition mit rekonstruierten Weltkoordinaten, da diese von den Glättungsschritten profitierten. Durch den Ein-

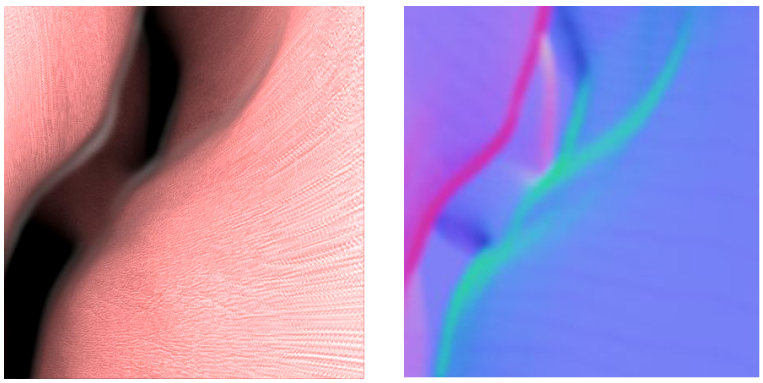

Abbildung 4.21: Artefakte der Texturierung (links), wenn die Normalen (rechts) nicht präzise genug sind.

satz von MipMapping kann die Leistung deutlich erhöht und Unschärfe bei größerer Entfernung erzielt werden. MipMapping wählt in Abhängigket von Entfernung und Kamerawinkel eine unterschiedliche Skalierungsstufe der gleichen Textur und greift dann auf diese zu. Je höher diese Stufe, desto geringer die Auflösung der Textur, und weniger hohe Frequenzen werden sichtbar. Auch wenn die Speicherkosten für eine Textur und allen ihren Verkleinerungen steigen, ist das Verfahren hardwareseitig sehr weit verbreitet und hilft der Leistung [1]. Der finale Farb- oder Helligkeitswert nach dem Zusammenmischen wird mit der Oberflächenfarbe für Gewebe und dem Beleuchtungswert multipliziert.

4.3.3 Eindruck von nasser Oberfläche

Der Helligkeitswert aus der Oberflächenstruktur wird in dieser Arbeit auch für die Darstellung von nassen Oberflächen genutzt. Dunkle Helligkeitswerte würden als Vertiefungen wahrgenommen, gleichzeitig würden die Ansammlungen von Flüssigkeiten in solchen Mulden dafür sorgen, dass die Wahrscheinlichkeit für eine intensivere Reflektion des Lichtes zurück zum Betrachter steigt. Dieser Glanzlichteffekt ist bei vielen nassen oder metallischen Oberflächen zu beobachten. Ebenfalls aus [Geiss and Thompson, 2006] stammt die Umsetzung, die sich aus einer Abdunklung und einem additiven Glanzlicht zusammensetzt. Während die Abdunklung überall dort geschieht, wo die Oberflächen zum Betrachter zeigen, wird das Glanzlicht nur dort besonders stark, wo auch der Helligkeitswert der Textur niedrig ist. Die Stärke des Effekts und wie der Oberflächenwinkel zum Betrachter gewichtet werden soll lässt sich interaktiv steuern. Ergebnisse davon sind in Abbildung 4.22 zu sehen.

4.3.4 Projektion von Bildern auf die Oberfläche

Nachdem nun alle Schritte für die Oberflächendarstellung des Gewebes erfolgt sind, können für die Bereitstellung des interaktiven Einzeichnens auf das Gewebe ein oder mehrere 2D Bilder projiziert werden. Dieser Kompositionsschritt erlaubt es, Zeichnungen von Chirurgen, wie in [Preim and Bartz, 2007, Kapitel 16.3.3] erläutert, auf das Volumen framekoherent zu projizieren. Jedes dieser Bilder setzt sich aus einem Farbwert, einer Transparenz und einem Distanzwert zusammen. Damit eine Projektion vorgenommen werden kann, sind eine Projektionsmatrix sowie die Ebeneninformation, die für die Distanzwerte gilt, notwendig. Die Projektion der Textur

[1] `http://download.nvidia.com/developer/presentations/GDC_2004/GDC2004_PracticalPerformanceAnalysis.pdf` (Stand 5.2005)

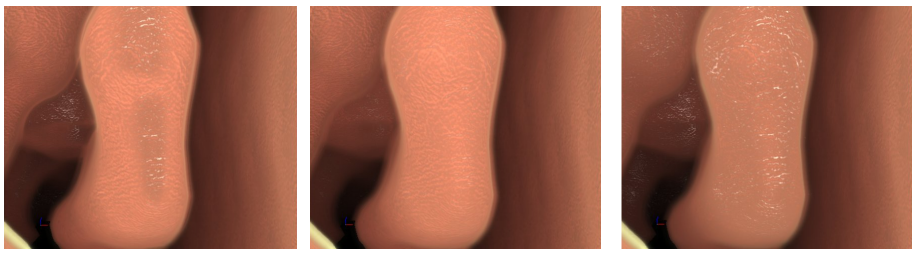

Abbildung 4.22: Verschiedene Einstellungen des Nässeeffekts, der das Glanzlicht erzeugt.

auf eine Oberfläche kann orthogonal oder perspektivisch geschehen und wird *Projective Texture Mapping* genannt [Everitt, 2001]. Die Gewebekoordinaten werden mit der Projektionsmatrix transformiert und überführen diese in den Bildraum des Projektors. Eine weitere Transformation liefert die finalen Texturkoordinaten, mit denen auf das Eingabebild zugegriffen wird. Je nach Transparenz wird danach der Farbwert der Gewebeoberfläche mit dem des Projektorbildes überdeckt. Allerdings würde dies dazu führen, dass alle Gewebepunkte entlang eines Projektorstrahls überdeckt werden. Um dies zu verhindern, wird überprüft, ob der aktuelle Distanzwert zur Ebene innerhalb eines gewissen Toleranzbereichs mit dem gespeicherten Distanzwert liegt. Wird der Distanzwerttest darauf reduziert, ob der gespeicherte Wert größer ist, ist die Methode auch als *Shadow Mapping* bekannt [Williams, 1978, Everitt et al., 2001].

4.3.5 Interaktion mit polygonaler Geometrie

Der Farbwert des Gewebes kann aber nicht nur durch Aufbringen von Bildern verdeckt werden, sondern auch wenn polygonale Geometrie über dem Gewebe liegt. Die *SinglePass* Renderingverfahren können nicht auf die notwendigen Daten für diesen Effekt zugreifen. Nur die *MultiPass* Verfahrensgruppe kann dies realisieren, weil sie über zusätzliche Eingabepuffer der polygonalen Farb- und Tiefenwerte pro Pixel verfügt. Erstere Gruppe kann den eigenen Transparenzwert der Ausgabe beeinflussen, so dass alle Gewebewände gleichermaßen transparenter werden. Dagegen werden die zusätzlichen Eingabepuffer genutzt, um selektiv pro Pixel zu entscheiden, ob er übermalt werden soll.

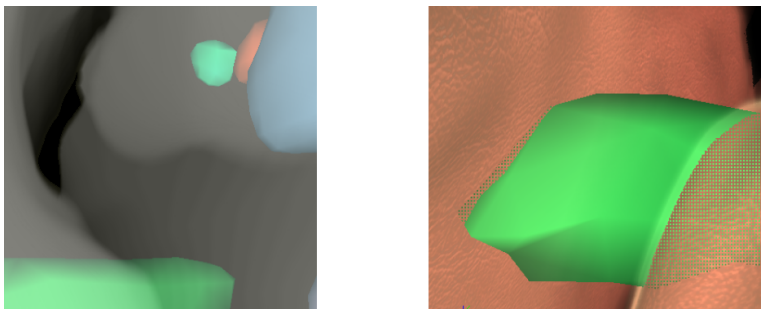

Abbildung 4.23: Interaktion mit polygonaler Geometrie. Links wurde eine einfache Transparenz auf die grauen Gewebewände angewendet (*SinglePass* Verfahren). Rechts wird die verdeckte Geometrie mit Stippling hervorgehoben und interagiert korrekt mit dem Volumen (*MultiPass* Verfahren).

Beide Ergebnisse im Vergleich sind in Abbildung 4.23 dargestellt. In dem Bild ist auch zu erkennen, dass solche Pixel, die hinter dem Gewebe liegen, mit einem Punktmuster versehen

wurden, um darauf hinzuweisen, dass sie „hinter“ dem Gewebe liegen. Das Punktmuster ist aus Implementationsgründen regulär, was aber auch den Vorteil hat, dass sich der Effekt deutlich als künstliche Gegebenheit abhebt. Mit dem Vergleich der Distanzwerte des Gewebes und der Tiefenwerte der Eingabe kann überprüft werden, welcher von beiden näher am Betrachter liegt. Beide müssen zuvor jedoch in das gleiche System übertragen werden. Dazu müssen erst die Tiefenwerte linearisiert werden[2], weil sie von der Hardware nicht linear gespeichert sind, um anschließend skaliert und verschoben zu werden. Für die Umrechnung wird der ZBuffer-Wert z, die Entfernung N der vorderen und F der hinteren Clip-Ebene, sowie das Offset o der Startebene zur vorderen Clip-Ebene benötigt:

$$z_{\text{local}} = ((NF/(N-F))/(z - F/(F-N))) - o \tag{4.11}$$

Das Resultat ist ebenfalls framekoherent, solange beim Erstellen der Startpunkte die Präzision ausreicht. Andernfalls ist die vordere Clip-Ebene nicht korrekt umgesetzt und die Tiefenwerte nicht synchron, was bei Kamerabewegung zu kleinen Überdeckungsinkonsistenzen führen kann.

Aufgrund der Rasterisierungspipeline ist es aber nicht möglich, mit transparenten Polygonen korrekt umzugehen. Denn im Tiefenpuffer ist immer nur ein Wert jedem Pixel zugeordnet, und transparente Flächen bräuchten mehrere. Durch das Verfahren des *Deferred Shading* ist es auch nicht möglich, Selbstverdeckung aufzulösen, da Gewebewände nicht völlig undurchsichtig definiert werden. So gibt es keine Möglichkeit, Gewebewände hinter anderen darzustellen. Es kann immer nur der erste Auftreffpunkt auf Gewebe dargestellt werden. In Absprache mit Medizinern ist dieser Effekt für das Anwendungsgebiet dieser Arbeit als nicht so wichtig eingeschätzt worden.

4.3.6 Visualiserung von weniger dichten Materie

Alle bisherigen Schritte dienten der Visualisierung von fester Materie, doch für den Anwendungsfall war es auch nötig, das Sekret zu repräsentieren, wie in Abbildung 4.24 zu sehen. Die

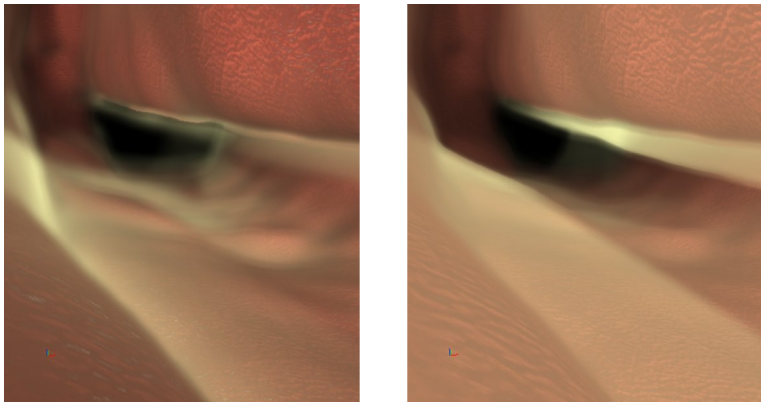

Abbildung 4.24: Gegenüberstellung der Verfahren zur Berechnung der Sekretdichte. Das Verfahren mit akkumulierter Sekretdichte (links) ist zwar genauer, aber benötigt eine geringere Schrittweite für Framekoherenz als das einfachere System (rechts).

[2] `http://www.sjbaker.org/steve/omniv/love_your_z_buffer.html` (Stand 11.2007)

Sekretdichte verstärkt den Nässe-Effekt, um den eher viskosen Charakter des Stoffes zu betonen. In erster Linie wird sie wie eine Art Nebel dargestellt. Eine Sekretfarbe überlagert je nach Sekretdichte die vorherigen Ergebnisse des Farbwerts. Die Sekretfarbe wird dunkler, je weiter weg die Distanz zur Sekretschicht ist, aber behält in jedem Fall eine minimale Helligkeit bei. Bei der Extraktion wurde bereits erwähnt (vgl. Abschnitt 4.2.2), dass es zwei verschiedene Modelle zur Bestimmung der Sekretdichte gibt, auch der Kompositionseffekt ist für beide unterschiedlich. Der Quelltext in Abbildung 4.25 ist für das einfachere Modell der beiden, das von einer konstanten Sekretdichte zwischen den Übergangspunkten ausgeht. Dieses Modell liefert mit den passenden Parametern einen besseren optischen Eindruck und ist bei geringerer Schrittanzahl dennoch framekoherent.

```
1  // All dValues are distances, normalized [0,1]
2  // and are "inverted", i.e. closest = 1
3  scalar getFogThickness( scalar dTissue,
4                          scalar dSlime,
5                          scalar dUsed,
6                          scalar fogdensity)
7  {
8      // dUsed might be > dTissue, when polygons overlap
9
10 #ifdef FOGACCUM
11     scalar curthick = (dSlime-dUsed);
12     scalar origthick = max(0.01,(dSlime-dTissue));
13     return saturate(fogdensity*1.2)*(curthick/origthick);
14 #else
15     return saturate(dSlime - dUsed) +  dUsed*0.15;
16 #endif
17 }
18
19
20 void doFog(inout scalar4 color, scalar dUsed,
21                                 scalar dSlime,
22                                 scalar fogthickness)
23 {
24     // dUsed normally is dTissue
25     scalar4 fogshading = scalar4(1,1,0.7,0);
26
27 #ifdef FOGACCUM
28     fogthickness = max(fogthickness,saturate(fogthickness * dUsed * 4));
29 #else
30     fogshading *= (pow(saturate(dSlime+0.3),8)+0.2);
31 #endif
32     color.xyz = lerp(color,fogshading,fogthickness).xyz;
33 }
```

Abbildung 4.25: Cg-Code für den Sekreteffekt. Intern wird das Sekret als „fog“ bezeichnet. `FOGACCUM` gibt jeweils den Codepfad an, der beim akkumulierten Modell genutzt werden soll.

Folglich ist hiermit die Komposition abgeschlossen. Der Farbwert wurde ermittelt und kann an den Ausgabepuffer übermittelt werden. Bei den *SinglePass* Verfahren muss zusätzlich eine Transparenz angegeben werden, mit der über das Originalbild gezeichnet wird. Diese ist die Summe aus Nebeldichte und dem Maskenwert, der angibt, ob überhaupt eine Sekret- oder Gewebeschicht vom Sichtstrahl getroffen wurde.

4.4 Leistung des Systems

Wie viel Rechenzeit Extraktion und Komposition verbrauchen hängt stark von deren Konfiguration und natürlich der verwendeten Hardware ab. Das System ist als skalierbare Lösung entworfen worden, das heißt es profitiert sehr von Fortschritten in der Hardwareleistung, aber auch von Einstellungsvariationen des Renderers. In Abbildung 4.26 wird die einfachste Einstellung mit der komplexesten verglichen, beide benutzen die gleiche Auflösung des Ausgabebilds.

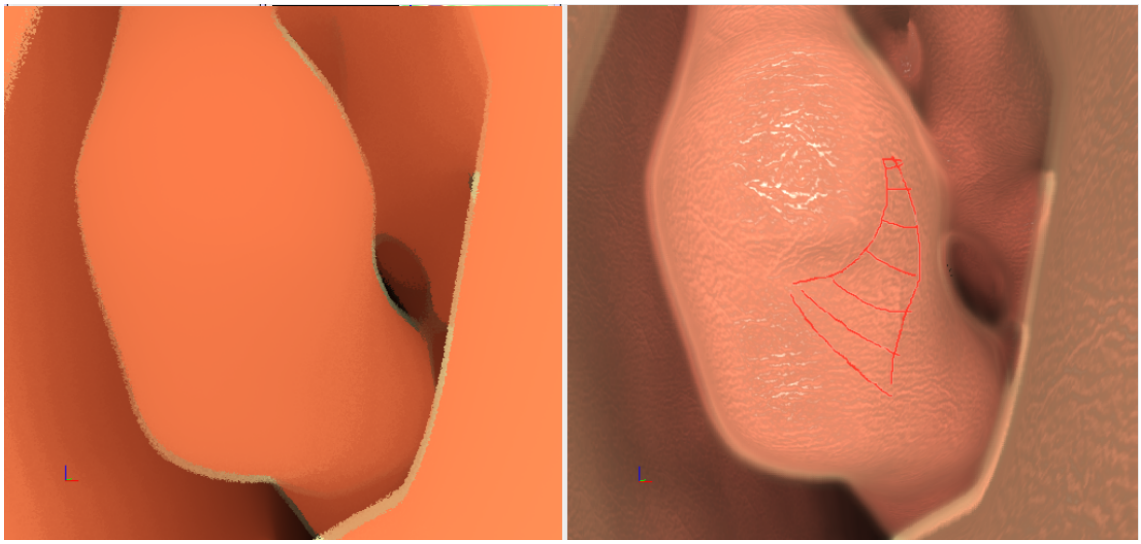

Abbildung 4.26: Vergleich der einfachsten Technik (links) mit der komplexesten (rechts).

Bei der Hardware zählt fast ausschließlich die Leistung der Grafikkarte, speziell die *Texture-Fillrate* ist ausschlaggebend. Sie gibt die Geschwindigkeit an, mit der Texturzugriffe erledigt werden können. Weil der Zugriff auf die 3D Texturen sehr häufig ist, und auch die anderen Schritte immer Komplexität der Bildgröße haben, lohnen sich Grafikkarten, die hier einen hohen Wert liefern. Die Programme für Vertex- und Fragmentprozessoren sind meist in ihrer mathematischen Komplexität eher einfach, so dass diese Art der Leistung eine nicht so entscheidende Rolle spielt. Die CPU-Belastung ist sehr gering und vernachlässigbar. Bis auf die Aktualisierung der Startebene, sowie der Projektionsmatrizen, muss die CPU eigentlich keine relevanten Berechnungen ausführen.

Was die Einstellungen angeht, so spielt die Bildgröße die wichtigste Rolle. Alle Schritte sind abhängig von der Anzahl der Pixel im finalen Bild, sowohl die Anzahl der Sichtstrahlen, die traversiert werden müssen, als auch die Glättungs- und Kompositionsschritte. Letztere machten keinen großen Unterschied aus, weil die Komposition nur einmal pro Pixel evaluiert wird, und auch nur eine begrenzte Zahl von Eingangsdaten benötigt. Im Gegensatz dazu sind die Konvolutionsfilter aufwändiger, weil sie viele Texturzugriffe und mindestens zwei Renderingdurchläufe pro Glättungsdurchlauf benötigen. Zwar werden mehrere Attribute auf einmal in eine Textur gespeichert, so dass insgesamt nur zwei Texturen zu glätten sind, doch ist dies im Verhältnis zur Komposition weit aufwändiger. Den größten Einfluss auf die Leistung hat der Renderingdurchlauf, der auf die Volumentextur zugreifen muss.

Beim Vergleichen der Ergebnisse bei einer Auflösung von 512×512 und dem Viertel davon, 256×256, ist aufgefallen, dass wegen der organischen runden Strukturen und der Sekretschicht der Qualitätsverlust nicht drastisch in Erscheinung tritt. Die Auflösung, mit der berechnet wird,

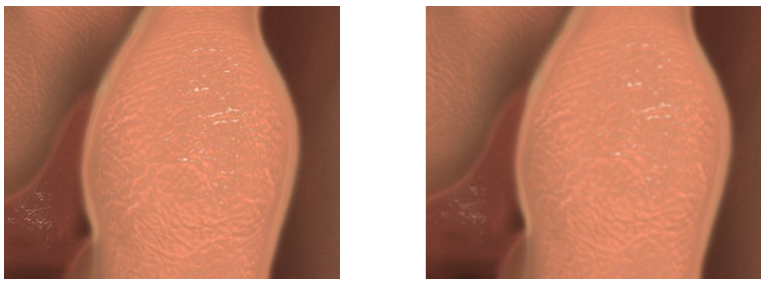

Abbildung 4.27: Auswirkung der Auflösungsverringerung. 512×512 (links) und 256×256 (rechts) mit bilinearem Skalierungsfilter. Beide Bilder sind Ausschnitte eines größeren.

muss nicht gleich der Auflösung sein, mit der das Resultat dargestellt wird. Zwar wird das Resultat umso unschärfer, je kleiner die Ausgangsauflösung, aber es ist trotzdem noch nutzbar, wie Abbildung 4.27 zeigt. Nur die Oberflächenstrukturen erscheinen als zu unscharf, aber auch sie können durch Anpassung der Detailtexturskalierung wieder an Schärfe gewinnen, was durch eine Benutzerinteraktion im System möglich ist. Eine andere Möglichkeit wäre, nur die Zwischenergebnisse in geringerer Auflösung zu berechnen, und dann die Komposition wieder mit höherer Auflösung durchzuführen, um die Schärfe der Detailtextur beizubehalten, was noch nicht umgesetzt wurde.

Kürzel	Grafikprozessor	Texture-Fillrate (MT/s)	Grafik-speicher	Fragment-prozessoren	CPU	RAM
GF6	GeForce 6600 GT	4000	128 MB	8	2.3 GHz P4	768 MB
GF7	GeForce 7800 GS	6000	512 MB	16	3 GHz P4	1 GB
GF8	GeForce 8600 GTS	10800	256 MB	32	2.3 GHz P4 DC	2 GB
GF8X	GeForce 8800 GTX	36800	768 MB	128	3 GHz P4	1 GB

Tabelle 4.1: Die Testsysteme im Überblick. Die Grafikkarte des ersten Systems ist aus dem Jahr 2004 und entspricht damit den minimalen Voraussetzungen. Die letzten beiden Systeme sind neuwertiger, wobei die letzte Grafikkarte aus dem High-End Bereich stammt und dementsprechend teuer ist.

Bei den anschließenden Leistungsmessungen kamen insgesamt vier Systeme zum Einsatz, deren Konfiguration in der Tabelle 4.1 zu sehen ist[3]. Alle Tests wurden am selben klinischen Datensatz der Auflösung 217×242×38 bei 8-Bit Präzision durchgeführt, mit identischer Einstellung von Transferfunktion, Blickfeld und Sichtweite. Die gemessenen Bilder pro Sekunde (BpS) wurden aus dem Durchschnittswert über fünf Sekunden ermittelt. Die maximale Schrittanzahl betrug 32 bei fünf iterativen Verbesserungen für die zwei Übergangspunkte. Ein Vergleich mit einem größeren Volumendatensatz 324×346×111 ergab nur minimale Leistungsunterschiede, die wegen unterschiedlicher Anatomie auf die Anzahl frühzeitig abgebrochener Traversierungen zurückgeführt werden kann.

[3]GPU Daten von `http://en.wikipedia.org/wiki/Comparison_of_NVIDIA_Graphics_Processing_Units` (Stand 10.2007)

4.4.1 Vergleich der Renderingvarianten

Es erfolgte eine differenzierte Betrachtung der einzelnen Einstellungsmöglichkeiten, die in Tabelle 4.2 aufgelistet sind und mit dem System GF8 ermittelt wurden. Die Ergebnisse des

Kürzel	Normalen-erstellung	Detailtextur-zugriff	Beleuchtung	Puffer-präzision	Multi / Single-Pass
NTW	Volumen	Komposition	voll	16-Bit float	MP
VTW	Kameraraum	Komposition	voll	16-Bit float	MP
VT	Kameraraum	Extraktion	voll	16-Bit float	MP
DT	Tiefenpuffer	Extraktion	voll	8-Bit	MP
DS	Tiefenpuffer	keine	keine Nässe	8-Bit	MP
SP	keine	keine	nur Distanzabfall	keiner nötig	SP

Tabelle 4.2: Die Varianten im Überblick. Bei späteren Ergänzungen von Zahlen an das Kürzel, gibt diese die Anzahl der Glättungsschritte an und der Buchstabe A steht für anisotrope Glättung. Ein Q bedeutet die Viertelung der Auflösung.

Verfahrensvergleichs sind in Abbildung 4.28 zu sehen. Die Auswertung ergibt klar, dass die Bildauflösung der entscheidende Faktor ist. Im Schnitt führt die Viertelauflösung zu einer Steigerung von 280%.

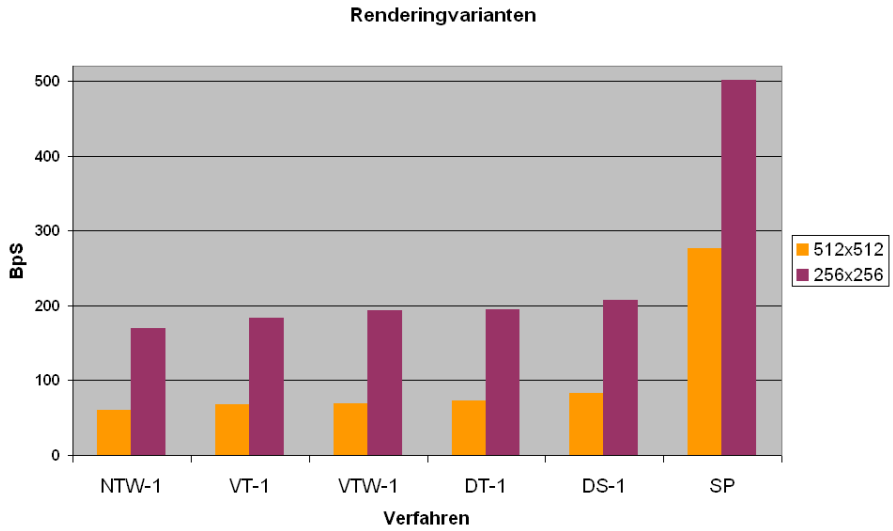

Abbildung 4.28: Verschiedene Renderingverfahren im Vergleich. Die *MultiPass* Varianten schneiden deutlich schlechter als die *SinglePass* Variante ab. Innerhalb ihrer Gruppe sind jedoch nur minimale Unterschiede.

Man kann ebenfalls erkennen, dass das *SinglePass* Verfahren am schnellsten ist. Dies hat primär Gründe, welche softwarebedingt sind. Durch den Wechsel der Renderingmodi und dem Hin- und Herschalten verschiedener Ausgabepuffer entsteht ein gewisser Leistungsverlust, der unabhängig von den eigentlichen Berechnungen ist. Innerhalb der *MultiPass* Verfahren sind die Unterschiede nur sehr gering. Die Erstellung der Normalen durch das Volumen ist geringfügig

langsamer als die anderen Gruppen. Beim direkten Vergleich von VTW und VT schneidet ersteres geringfügig besser ab, was Pipelinegründe haben könnte. Vermutlich ist es besser, am Anfang des Kompositionsdurchlaufs auf die Detailtextur zuzugreifen, als am Ende des GPU-Programms bei der Extraktion.

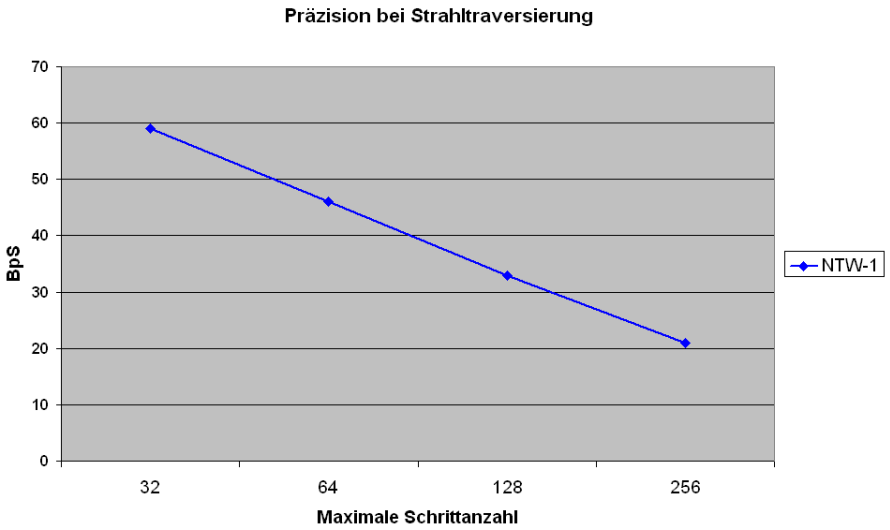

Abbildung 4.29: Performanceverlust bei Halbierung der Schrittweiten.

Als nächstes wurde die Auswirkung der Verringerung der Schrittweiten getestet. Dazu musste bei gleicher Sichtweite die maximale Schrittanzahl angehoben werde. Die Ergebnisse in Abbildung 4.29 zeigen, dass die Bildrate sich bei Halbierung linear verschlechtert, sich also insgesamt asymptotisch verhält.

Ein ähnliches Phänomen wurde auch beim Vergleich der Glättungstechniken festgestellt. In der Abbildung 4.30 ist zu erkennen, dass der Leistungsverlust im Verhältnis zur Anzahl von Glättungen abnimmt. Die Kurven flachen immer mehr ab. Wie zu erwarten war, ist das anisotrope Verfahren langsamer, je mehr Glättungen ausgeführt werden.

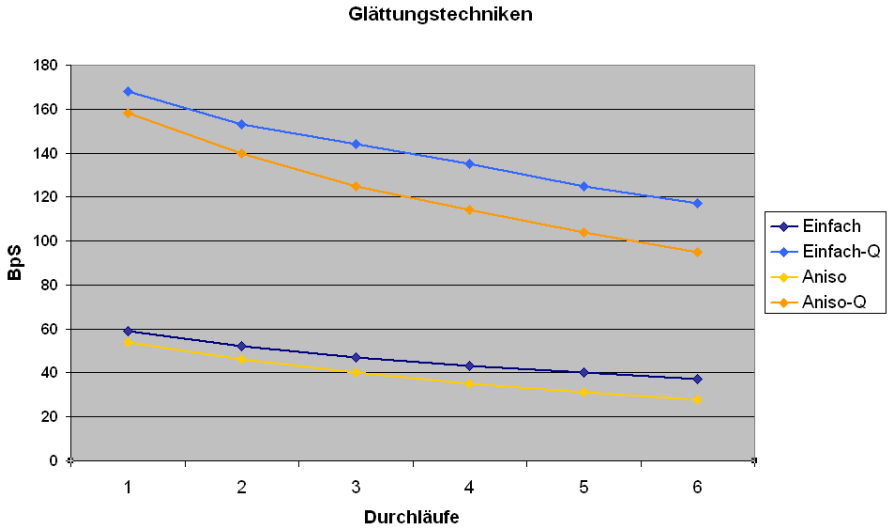

Abbildung 4.30: Auswirkungen der Auflösung und Anzahl der Durchläufe bei zwei verschiedenen Glättungsverfahren. Die oberen mit -Q markierten Linien entsprechen einem Viertel der Auflösung.

Auf dem GF7 System wurde getestet, welchen Einfluss die Reduktion der Berechnungspräzision hat. Ältere Systeme konnten intern noch mit 16-Bit float Berechnung ausführen, während neuere Systeme immer in 32-Bit float arbeiten. Der Test hat ergeben, dass der Leistungsgewinn sehr gering, der Verlust an Qualität aber enorm war und deshalb keinesfalls sinnvoll ist. Ein Vergleich der DT Technik mit verschiedenen Präzisionen der Puffer für die Zwischenergebnisse ergab, dass der Verlust bei Verdopplung der Präzision in etwa einem weiteren Glättungsdurchlauf entspricht. Von 16-Bit nach 32-Bit float gibt es aber keinen visuellen Qualitätsunterschied. Der Sprung nach 16-Bit lohnt jedoch und ist für einige Verfahren auch ein Minimum. Denn sonst könnten entlang der gesamten Sichtweite nur 256 Tiefen repräsentiert werden, was sowohl für die Normalenerstellung, als auch für die Rückprojektion der Gewebepunkte nicht ausreicht. Deswegen benötigt die DT-Technik bei 8-Bit eine relativ geringe Sichtweite, um Unterschiede aus dem Tiefenpuffer für die Gradienten zu erfassen.

4.4.2 Vergleich von Hardwaresystemen

Um verschiedene Hardwaresysteme zu vergleichen, wurden die Tests auf drei Varianten beschränkt. Die Ergebnisse sind in den Graphen 4.31 zu sehen. Das schwächste System GF6 schafft nur noch beim einfachsten Renderingverfahren Bildwiederholraten über 25 BpS, doch die Verringerung der Auflösung macht alle Verfahren interaktiv nutzbar.

Bei allen Systemen steigt die Leistung in etwa mit den Verhältnissen der maximalen *Texture-Fillrate* untereinander. Auffällig ist, dass beim stärksten System die CPU der limitierende Faktor ist und die Werte nicht weiter skalieren. Die Anzahl der maximalen Zeichenbefehle, die von der CPU an die Grafikkarte geschickt werden können, ist hier begrenzt. System GF8 besitzt eine

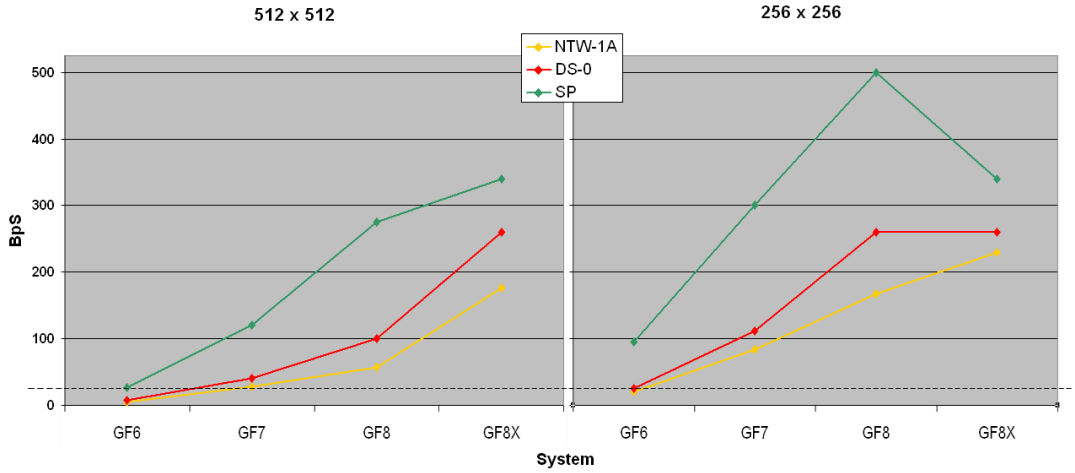

Abbildung 4.31: Vergleich dreier Verfahren auf unterschiedlichen Hardwaresystemen bei voller Auflösung (links) und verringerter Auflösung (rechts). Die 25 BpS Grenze für interaktives Arbeiten wurde hervorgehoben.

CPU, die der von System GF8X überlegen ist, weswegen die Kurve beim einfachen Rendering für GF8X abfällt. Auch die Tatsache, dass es zum Teil bei GF8X keinen Unterschied zwischen maximal und reduzierter Auflösung gibt, lässt sich daraus ableiten. System GF8 scheint für den Standardgebrauch ausreichende Reserven zu bieten, während System GF8X sehr viel Leistungsüberschuss erzielt. Dieser kann in geringere Schrittweiten und damit höherer Präzision des Ergebnisses investiert werden. Ferner ist die Anwendung von trikubischem Filtering der Volumentextur mit dem System möglich. Bei den zwei älteren Systemen zeigte sich bei Messungen, dass es ab einer gewissen Belastung für die unterschiedlichen *MultiPass* Verfahren fast keine messbaren Differenzen mehr gibt. Interessanterweise steigt bei den älteren Verfahren der Verbesserungsfaktor, der bei Auflösungsreduktion erzielt wird. Dies spricht für das asymptotische Leistungsverhalten.

Aus dieser Tatsache und den vorherigen Feststellungen lässt sich schließen, dass wenn ein System eine gewisse Leistung aufbringt, um die Interaktivität zu gewährleisten, innerhalb der Verfahrensvarianten die „rechenintensivste“, bzw. qualitativ hochwertigste Variante gewählt werden kann. Für leistungsstarke Systeme lohnt sich die Verminderung der Schrittweiten, weil sie bei konstantem Leistungsverlust die Genauigkeit verdoppeln. Zwar sind Glättungen nicht so rechenintensiv, dienen aber nur der optischen Vermeidung von Artefakten und erhöhen nicht die Genauigkeit der Schnittpunkte.

Aus softwaretechnischen Gründen, die im nächsten Kapitel der Implementierung ausgeführt werden, war es zur Zeit nur möglich, die Messungen mit NVIDIA Grafikkarten durchzuführen. Es ist aber anzunehmen, dass in ihren Leistungseigenschaften vergleichbare Grafikkarten von ATI/AMD ähnliche Ergebnisse liefern würden. Andere Hersteller sind wegen mangelnder Verbreitung, Leistung und Fähigkeit der Grafikprozessoren vernachlässigbar. Dies gilt insbesondere für viele On-Board Grafikchips, die zwar am meisten in Bürorechnern verbreitet, aber kaum für interaktive 3D Grafik geeignet sind. Separate Grafikkarten haben einen eigenen sehr schnellen Grafikspeicher und eine schnellere Systembusanbindung an die CPU und den normalen Hauptspeicher. Außerdem verfügen sie über komplexere GPUs und eigene Kühlmechanismen, die eine hohe Leistung zulassen. Unter dieser Art von Grafikkarten sind solche mit NVIDIA

GPUs zur Zeit die schnellsten und auch ihre Treiber sind für OpenGL Anwendungen robuster als die ihrer Konkurrenz. Das GF8 System zeigt jedoch, dass auch eine Grafikkarte aus der mittleren Preisklasse genügend Leistung für die in dieser Arbeit relevanten Anforderungen bietet.

Alle gestellten Anforderungen an das System wurden im Rahmen dieser Arbeit erreicht. Die gewählte Methode ist auch flexibel, genug um relativ einfach Verbesserungen einzuarbeiten, oder zusätzliche Effekte zu implementieren. Die Interaktivität ist weiterhin auf älteren Systemen gegeben. Qualitätsverbesserungen lassen sich mit Parameterveränderungen, wie Schrittweite der Strahlverfolgung oder komplexeres Filtering, erreichen und sind damit auch zukünftig den Fortschritten der Hardwareentwicklung von Grafikkarten angepasst.

Kapitel 5

Implementierung

Die konzeptionelle Entwicklung des Renderers war stark von den Fortschritten der Implementierung abhängig, weil die speziellen Eigenschaften der Hardware und deren Abläufe berücksichtigt werden mussten. Für die Implementierung der verschiedenen Ausführungen der Renderingtechniken war es notwendig, eine Testumgebung zu entwickeln. Diese Umgebung (SinusEndoscopy) sollte zum einen den leichten Vergleich zwischen Techniken zulassen, aber es auch ermöglichen, die Qualitätsunterschiede bei unterschiedlichen Volumendaten zu bewerten. Um die Entwicklungszeit zu verkürzen, wurde eine 3D Engine gewählt, die bereits viele notwendige Funktionen bereitstellte. Die Ersparnis liegt zum einen am Funktionsumfang, der nicht erst neu entwickelt werden muss, zum anderen aber auch in der Robustheit dieser Lösungen. Zum Beginn der Arbeit musste daher eine geeignete 3D Engine gefunden werden. Falls die Engine es zulassen sollte, welche API (Application Programming Interface) zur Ansteuerung der Grafikhardware verwendet werden soll, musste auch hier noch eine Wahl getroffen werden. Unter Microsoft Windows gibt es zwei Möglichkeiten 3D Grafikkarten zu programmieren, zum einen durch die von Microsoft selbst entwickelte DirectX Umgebung, insbesondere Direct3D als Teil von dieser, oder der von der Khronos Group verwalteten OpenGL[1] Grafikbibliothek.

Gerade im Bereich der Unterhaltungsindustrie ist der Einsatz modernster Grafiktechnik stark verbreitet, und so lassen sich viele Programmbibliotheken und derartige „Middleware“ in diesem Bereich der Softwareindustrie finden. Es gibt auch eine Reihe von qualitativ hochwertigen 3D Engines, deren Quellcode offen und damit kostenlos verfügbar ist. Zwar bieten viele dieser Programme Schnittstellen zu den Funktionen moderner Grafikkarten, doch erfolgt die Programmierung meist über Low-Level Programmiersprachen wie C/C++. Diese Sprachen ermöglichen besonders maschinennah effiziente Programme zu schreiben, und werden daher für den Großteil leistungsorientierter Software verwendet, wie zum Beispiel der Programmierung von Betriebssystemen. Diese Nähe an der Hardware verlangt aber dem Programmierer einen größeren Planungs-, und Implementierungsumfang ab, da er sich um Speicherverwaltung und dergleichen kümmern muss. Für die Entwicklung des Softwareprototypen sollte daher in erster Linie eine High-Level Sprache zur Verwendung kommen. Zwar sind diese zur Laufzeit etwas langsamer, dafür ist der Programmierkomfort größer, was letztendlich zu einer schnelleren Umsetzung der Software führt. Beim Einsatz von Middleware würde der performancekritische Teil der GPU-Anweisungen meist ohnehin in einer Low-Level Sprache geschrieben werden. Außerdem finden zur Laufzeit für die gewählten Techniken keine aufwändige Berechnungen statt, so dass der Leistungsverlust vernachlässigbar ist.

Folgende Eigenschaften waren für die Umsetzung daher wichtig:

[1] `http://www.opengl.org`

- Einfacher Zugriff auf moderne Renderingtechniken
- Programmiersprache für effizientes Arbeiten
- Hilfsfunktionen insbesondere zur Erstellung Grafischer Benutzeroberflächen (GUI)

Letztendlich fiel die Wahl auf die Luxinia Engine[2], welche OpenGL als Grafikschnittstelle benutzt und selbst in C geschrieben ist, Anwendungen mit ihr aber durch die High-Level Sprache Lua programmiert werden. Luxinia wurde vor allem für Rapid-Prototyping von 3D Spielen entwickelt, fand aber auch im universitären Umfeld erfolgreich Anwendung [Kubisch, 2007]. Die Volumendaten wurden mit Hilfe der Software MeVisLab[3] von MeVis Research aus den klinischen DICOM Daten konvertiert. Die Segmentierung von polygonalen Modellen fand in dieser medizinischen Bildverarbeitungs- und Visualisierungssoftware statt.
In den nächsten zwei Abschnitten wird näher auf OpenGL und die für den Prototyp relevanten Teile von Luxinia eingegangen.

5.1 OpenGL als Schnittstelle zur graphischen Darstellung

OpenGL ist ein Standard, der seit 1992 existiert und aus IrisGL der Firma SGI hervorging. Er dient seither als API zur Kommunikation mit der Grafikkarte und ist vor allem in Anwendungsbereichen wie CAD weit verbreitet. Es ist der einzige platformunabhängige Standard, der sowohl unter den großen drei Betriebssystemen Windows, Mac OS und Linux, als auch einer Reihe anderer Platformen verfügbar ist. Mit der Opensource Implementation des Standards, genannt MESA[4], ist außerdem eine Hardwareemulation auf der CPU zu erreichen. Jene Emulation erlaubt es, bei stark verminderter Leistung auch ohne die notwendigen Hardwaremöglichkeiten ein Programm darzustellen. Bei der Entwicklung kam jedoch primär eine hardwarebeschleunigte Version von OpenGL zum Einsatz.

OpenGL ist eine Zustandsmaschine, das heißt der Programmierer definiert durch Befehle einen aktuellen Zustand. Teile dieses Zustands werden bei gewissen Funktionsaufrufen benutzt. Der Vorteil ist, dass beim Zeichnen von Objekten mit einen ähnlichen Zustand, nur auf die Unterschiede zwischen dem Zeichnen der Objekte eingegangen werden muss. Der Nachteil hingegen ist, dass bei großen Unterschieden immer wieder viele Befehle notwendig sind. OpenGL 3.0, das sich zur Zeit in der Entwicklung befindet, wird keine Zustandsmaschine mehr sein, sondern objektorientiert arbeiten [5], ähnlich wie es bereits bei Microsofts Direct3D der Fall ist. Luxinia benutzt allerdings noch den OpenGL 1.3 Standard mit diversen Erweiterungen.

Jene Erweiterungen sind ein großer Unterschied zu Microsofts Direct3D. Den Herstellern wird bei OpenGL die Möglichkeit gegeben, besondere Funktionen durch sogenannte *Extensions* bereitzustellen. Dies macht OpenGL zu der API, welche meistens als erste Zugriff auf neue technische Besonderheiten der Hardware erlaubt. Der Programmierer benötigt lediglich einen aktuellen OpenGL Treiber, der die Funktionen bereitstellt, und kann diese dann sofort benutzen. Meist einigen sich die Produzenten auf Extensions, die herstellerunabhängig eingesetzt werden können, doch ist dies nicht zwingend notwendig. Dieser Umstand macht es dem Entwickler nicht einfach. Zwar gibt es immer wieder Aktualisierungen des OpenGL Standards, doch wird dies nicht so strikt umgesetzt wie das von Microsoft kontrollierte Direct3D.

[2] `http://www.luxinia.de`
[3] `http://www.mevislab.de`
[4] `http://www.mesa3d.org`
[5] `http://www.opengl.org/pipeline/article/vol003_1/` (Stand 11.2007)

5.1.1 Programmierung der GPU

Zunächst gab es für beide APIs einen fest definierten Ablaufplan beim Zeichnen, der nur bedingt konfigurierbar war, dafür aber standardisiert. Dieser, als *Fixed Function Pipeline* bezeichnete Ablauf, regelte Beleuchtung, Texturkombination usw. in einer unveränderbaren Weise und es konnten lediglich gewisse Parameter verändert werden. Wie bereits bei den Verfahren in Kapitel 3.2.4 vorgestellt, ist die moderne Rendering Pipeline um Flexibilität erweitert worden. Für die Umsetzung der Renderingverfahren waren die Programmierung auf Vertex und Pixel(Fragment)-Ebene wichtig. OpenGL bieten mit den Extensions `ARB_vertex_program` und `ARB_fragment_program` seit 2002 Standards zur Programmierung an. Die Programmierung erfolgte hier noch in Maschinensprache und wurde 2003 mit der High-Level Shading Sprache GLSL 2003 durch `ARB_fragment_shader`,`ARB_vertex_shader` erleichtert. Der GLSL Code, der von der Semantik an die Programmiersprache C angelehnt ist, wird vom Treiber herstellerabhängig für die Grafikkarte kompiliert. Eine genauere chronologische Auflistung dieser Entwicklung folgt:

1. Generation (1998)

- Die Hauptvorteile waren die Beschleunigung des Rasterisierungsprozesses, sowie Multi-Texturierung.

2. Generation (1999)

- Die Transformierung der Punkte erfolgte nun mit Hardwarebeschleunigung.

3. Generation (2001)

- Eine Programmierbarkeit des Vertexprozessors, sowie erste einfache Programmierung des Fragmentprozessors waren möglich.
- `ARB_vertex_program`, `NV_register_combiners`, `NV_texture_shaders`, `ATI_fragment_shader`

4. Generation (2003)

- Steigerungen der Programmierbarkeit wurden vor allem durch mehr Instruktionen in den Programmen und Schleifen, sowie bedingte Sprünge in Vertexprogrammen, erreicht.
- `ARB_fragment_program`, `NV_vertex_program2`
- Cg ist für OpenGL verfügbar.

5. Generation (2004)

- Der Zugriff auf Texturen in Vertexprogrammen und bedingte Sprünge und Schleifen auch in Fragmentprogrammen sind implementiert worden.
- `NV_fragment_program2`, `NV_vertex_program3`
- GLSL ist verfügbar: `ARB_fragment_shader`, `ARB_vertex_shader`

6. Generation (2006)

- Ein programmierbarer Geometrieprozessor erlaubt nun zum Beispiel das Erstellen von zusätzlichen Dreiecken.
- `NV_gpu_program4`, `NV_fragment_program4`, `NV_vertex_program4`, `NV_geometry_program4`

```
float4 simpletransfer_f(vfconn_fixed IN,
    uniform sampler3D tex0 : TEXUNIT0,
    uniform sampler1D tex1 : TEXUNIT1,
    uniform float4 transfer,
    uniform float4 campos,
    uniform float4 axisconvx,
    uniform float4 axisconvy) : COLOR
{

    float4 sampled = tex3D(tex0,IN.tex0);
    float4 color = tex1D(tex1,sampled.w);

    // transfer function
    sampled.w = (sampled.w - transfer.z)*transfer.w;
    sampled.w = saturate(sampled.w);

    // dither pattern
    float2 intpart = step((float2)1,fmod(IN.tex1.xy,2));
    float stipple = dot(intpart,float2(0.5,0.5));
    stipple *= step(0.01,sampled.w)*0.5;

    // apply dither for "none-solid"
    sampled.w = lerp(stipple,1,step(0.99,sampled.w));

    // find how close pixel is to major axis
    float3 dist = float3(IN.tex1.xy - campos.xy ,0);
    float thresh = 1.0;
    dist = abs(dist);
    dist = 1.0-step((float3)thresh,dist);
    float distinf = saturate(dot(dist,float3(1,1,0)));
    dist = (axisconvx.xyz*dist.y) + (axisconvy.xyz*dist.x);

    // change colors a bit for more intensity
    float3 intensity = float3(0.32f, 0.40f, 0.30f);
    color.xyz = lerp(    (float3)dot(color.xyz,intensity)*2,
                         color.xyz,0.35);
    color *= sampled.w;

    // overwrite with major axis color
    return lerp(color,float4(dist.xyz,1),distinf);
}
```

```
PARAM c[7] = { program.local[0..3],
        { 0.63999999, 0.80000001, 0.60000002, 0 },
        { 0.34999999, 0.0099999998, 0.5, 2 },
        { 1, 0.99000001, 0 } };
TEMP R0;
TEMP R1;
TEMP R2;
TEX R0.w, fragment.texcoord[0], texture[0], 3D;
TEX R1, R0.w, texture[1], 1D;
DP3 R0.z, R1, c[4];
ADD R1.xyz, R1, -R0.z;
MAD R1.xyz, R1, c[5].x, R0.z;
MUL R0.xy, fragment.texcoord[1], c[5].z;
ABS R0.xy, R0;
FRC R0.xy, R0;
MUL R0.xy, R0, c[5].w;
CMP R2.xy, fragment.texcoord[1], -R0, R0;
SGE R2.xy, R2, c[6].x;
MUL R2.xy, R2, c[5].z;
ADD R0.w, R0, -c[0].z;
MUL_SAT R0.w, R0, c[0];
ADD R2.y, R2.x, R2;
SGE R2.x, R0.w, c[5].y;
MUL R2.x, R2, R2.y;
MUL R2.w, R2.x, c[5].z;
MOV R0.z, c[4].w;
ADD R0.xy, fragment.texcoord[1], -c[1];
ABS R0.xyz, R0;
SGE R0.xyz, R0, c[6].x;
ADD R2.xyz, -R0, c[6].x;
SGE R0.y, R0.w, c[6];
ADD R0.x, -R2.w, c[6];
MAD R0.w, R0.y, R0.x, R2;
MUL R0.xyz, R2.x, c[3];
MUL R1, R1, R0.w;
MAD R0.xyz, R2.y, c[2], R0;
MOV R0.w, c[6].x;
ADD R0, R0, -R1;
DP3_SAT R2.x, R2, c[6].xxzw;
MAD result.color, R2.x, R0, R1;
END
```

Abbildung 5.1: Cg-Kompilierung. Der Cg-Fragmentshader (links) für die orthogonalen Ansichten (5.3.2) wird nach `ARB_fragment_program` (rechts) kompiliert.

Da zum Zeitpunkt der Entwicklung von Luxinia, die GLSL Verbreitung noch recht gering war, und die Maschinencode-Extensions robuster waren, wurde Cg als High-Level Shading Sprache für Luxinia gewählt. NVIDIA entwickelte Cg[6] und es wurde in Kooperation mit Microsoft mit leichten Veränderungen zur High-Level Sprache HLSL für Microsofts Direct3D. Cg kann in mehrere Sprachen mit unterschiedlichem Befehlsumfang kompiliert werden. Hierbei sind sowohl die Direct3D Maschinencodes als auch OpenGL Extension spezifische Ausgaben möglich. Cg wird auch für die Programmierung der PlayStation 3 verwendet und unterstützt damit die meisten verfügbaren Rendering APIs. Ein Nachteil der derzeitigen Version von Cg ist allerdings, dass die modernen Fähigkeiten in der Shaderprogrammierung unter OpenGL nur für NVIDIA Extensions zur Verfügung stehen. Zwar kann Cg den Code auch in GLSL überführen, doch ist der Übersetzer noch zu fehlerhaft. Alternativ kann man aber selbst diese Übersetzung durchführen, was im Rahmen der Entwicklung von SinusEndoscopy auch teilweise geschehen ist. Solange der

[6] `http://developer.nvidia.com/object/cg_toolkit.html` (Stand 5.2007)

Cg-Compiler diese Probleme noch besitzt, ist es besser, GLSL direkt zu verwenden. Abbildung 5.1 zeigt die Überführung des Cg-Fragmentprogramms für die orthognalen Schnittansichten in `ARB_fragment_program`.

Die Vertexshader für SinusEndoscopy sind relativ unkompliziert und kommen mit einfachen Befehlssätzen aus und haben nur einen geringen Anteil an der Gesamtleistung. Die Hauptlast wird von den Fragmentshadern getragen, welche auch modernere Befehlssätze benötigen. Für leistungsorientiertes Programmieren ist zu beachten, dass der OpenGL Grafikkartentreiber die Programme für die darunterliegende Hardwarearchitektur optimiert. Details über diese Optimierungen bleiben dem Anwender verborgen. Die Hersteller optimieren softwarespezifisch ihre Treiber, vor allem bei sehr populären Spielen wird die Leistung für die Funktionen des Spiels gesteigert. Anwender außerhalb der Spieleindustrie oder Entwickler weniger bekannter Titel können nur bedingt davon profitieren. Es ist für den OpenGL Treiber legitim, Funktionen, die nicht hardwareseitig unterstützt werden, per CPU zu emulieren, was drastische Leistungseinbrüche nach sich zieht. Diese Bedingungen mindern zum Teil die Aussagekraft der gemessenen Zeitwerte und erschweren die Entwicklung.

5.1.2 Relevante Textur Funktionen

Neben der Shaderprogrammierung ist der Umgang mit Bildwerten zur Ein- und Ausgabe für diese Arbeit von besonderer Bedeutung. OpenGL definiert Texturen hinsichtlich ihrer Dimension, Anzahl der Farbkanäle und dem Präzisionsgrad. Mit `ATI_texture_float` wurde es möglich, vorzeichenbehaftete 16 und 32-Bit Fließkommawerte in Texturen zu speichern. Allerdings ist der Zugriff, das Blenden und das Filtering nicht für alle Grafikkarten effizient. Ähnlich wie bei der bereits erwähnten Softwareemulation durch den Treiber, ist auch hier den Herstellern Spielraum gegeben, dass sich das tatsächlich verwendete Format von dem Angefragten unterscheidet. Sowohl NVIDIA als auch ATI/AMD bieten allerdings Tabellen[7] an, denen man entnehmen kann, welche Formatkonvertierungen vorgenommen werden.

Eine weitere wichtige OpenGL Funktion ist das Rendern in Texturen. Normalerweise werden die finalen Pixel in den Backbuffer geschrieben, von dem sie nicht innerhalb eines Shaders gelesen werden können. Über das *Render-To-Texture* Verfahren ist es aber möglich, entweder eine Kopie des Backbuffers in eine Textur zu machen, oder direkt in diese die Ausgaben umzuleiten. Ersteres Verfahren hat den Nachteil, dass es langsamer ist und die Größe des Applikationsfensters ein Limit darstellt. Bei letzterem Weg ist es auf Grund des Parallelismus nicht möglich, in eine Textur zu schreiben, von der auch gelesen wird. Dieser Umstand führt zu der Technik des *Ping-Pong Rendering*, bei der wechselweise Ein- und Ausgabetextur getauscht werden. Nach diesem Prinzip wurde hier das Glätten der Tiefen- und Normalmap realisiert. Für das Rendern in Texturen wird die `EXT_framebuffer_object` (FBO) Extension benützt, diese gehört noch nicht zum OpenGL Standard, ist aber bereits weit verbreitet und erleichtert die direkte Ausgabe in Texturen.

Da Texturen nur über maximal vier Farbkanäle verfügen, und es zum Teil notwendig war, mehrere Informationen pro Ausgabepixel zurückzugeben, kam die *Multiple Render Targets* Technik zur Verwendung. Mit der `ARB_drawbuffers` Extension ist es in OpenGL möglich, mehr als nur einen 4-Komponenten Vektor zurückzugeben. Moderne Hardware ermöglicht das Schreiben in bis zu vier Ausgabetexturen, allerdings nicht ohne Leistungsverlust. Eine wichtige Einschränkung ist derzeit noch, dass die Ausgabetexturen ein Format haben müssen, das die gleiche Anzahl an Bits pro Pixel besitzt. Das heißt man kann zum Beispiel eine 4-Kanal 8-Bit Textur und eine 1-Kanal 32-Bit Textur gleichzeitig verwenden.

[7] `http://developer.nvidia.com/object/nv_ogl_texture_formats.html` (Stand 6.2007)

SinusEndoscopy zeichnet die 3D Ansicht primär in eine 4-Kanal 8-Bit Textur. Aufgrund des dadurch eingeschränkten Wertebereichs können die Volumenrendertechniken angepasste Setups verwenden. Das Finale Ergebnis, sowie die Benutzeroberfläche wird direkt in den Backbuffer des Fensters gezeichnet.

5.2 Luxinia als Entwicklungsplatform

Luxinia wird seit 2004 entwickelt und ist als platformunabhängige 3D Engine konzipiert, wenn gleich bisher nur eine Version für Windows zur Verfügung steht. Ziel war es in erster Linie, Computerspiele mit geringem Programmieraufwand schreiben zu können. Um dies zu erreichen, wurde eine API entwickelt, die wichtige Eigenschaften und Funktionen bereitstellt. Dazu gehört primär das Verwalten und Darstellen von Texturen, 3D Modellen, Animationen und dergleichen, aber auch das Schaffen von räumlichen Abhängigkeiten, Kollisionserkennung und physikalische Simulation. Um anderen Entwicklern ein effizientes Arbeiten zu ermöglichen, sollte es nicht notwendig sein, Programmcodes kompilieren zu müssen oder die internen Strukturen der Engine zu kennen. Deswegen lässt sich Luxinia über die Skriptsprache Lua programmieren, welche an der Pontifical Catholic University of Rio de Janeiro in Brazil seit 1992 entwickelt wird[8]. Lua gehört zu den schnellsten interpretierten Sprachen [9] und wird vermehrt in der Spieleindustrie eingesetzt[10]. Der Entwickler kann jederzeit dynamisch Code nachladen, der auch automatisch erzeugt werden kann, oder Zustände zur Laufzeit über eine Konsole abfragen und verändern. Lua stellt weiterhin einen DLL-Lademechanismus bereit, mit dem Dynamische Code Bibliotheken geladen und angesprochen werden. Jene Bibliotheken können in anderen Programmiersprachen geschrieben sein und leistungskritische Probleme effizienter lösen, womit Luxinia indirekt durch Lua beliebig erweiterbar ist.

5.2.1 Graphical User Interface (GUI)

Eine der ersten Lua-seitigen Erweiterungen Luxinias entstand aus dem universitären Umfeld. Es ergaben sich andere Anwendungsgebiete, die nicht aus dem Unterhaltungssektor stammen, für die es notwendig war, Benutzeroberflächen ohne großen Programmieraufwand erstellen zu können. Dieses GUI System war für die Entwicklung von SinusEndoscopy ebenfalls besonders

Abbildung 5.2: Der Wert kann entweder durch Schieben verändert werden (links), oder durch manuelle Eingabe. Diese ist per Klicken auf den Wert möglich, sofern dieser sich bei Berührung durch die Maus verfärbt (mittig). Der Benutzer hat nun über ein Pop-Up Menü die Möglichkeit einer direkten Werteeingabe (rechts).

nützlich, weil schnell Steuerelemente zur Veränderung von Werten erzeugt werden konnten. Gerade beim Arbeiten an grafischen Problemen erschien dies besonders wichtig. Direkte Manipulationsmöglichkeiten helfen, Auswirkungen von Parameterwerten des Renderers zu verdeutlichen. Hierbei kamen oft Schieberegler als Elemente zum Einsatz. Da für die medizinische Anwendung

[8] `http://www.lua.org` (Stand 7.2007)
[9] `http://shootout.alioth.debian.org/gp4/benchmark.php?test=all&lang=all` (Stand 11.2007)
[10] `http://www.lua.org/uses.html` (Stand 7.2007)

jedoch exakte Reproduzierbarkeit von Einstellungen wichtig ist, wurde eine numerische Eingabe von Werten ermöglicht. Bild 5.2 erläutert die Funktionsweise der Werteregler.

5.2.2 Material- und Shadersystem

Diese Regler steuern in erster Linie Parameter des Renderingsetups, welches sich primär aus der Kombination von Zeichenbefehlen, Eingangstexturen und benutzten GPU-Programmen zusammensetzt. Letztere werden vom Material- und Shadersystem gesteuert. Diese werden in speziellen Resourcenscriptdateien definiert und können durch die Luxinia API im Nachhinein zur Laufzeit manipuliert werden. Ein Material definiert eine Reihe von Eingabeparametern wie Texturen und

```
luxinia_Material_v100
Shader{
    SHD "ortho_transfer_sm2.shd";
    control "campos" 0 3;       # control object instance
    param "campos" (1,1,0,0);   # parameter in shader
    control "axisconvx" 0 3;
    param "axisconvx" (0,0,0,1);
    control "axisconvy" 0 3;
    param "axisconvy" (0,0,0,1);
}
Texture:0{
    TEX "USER_TEX(3DDATA)";     # application controlled
}
```

```
luxinia_Shader_v300
Technique:VID_ARB_VF{
    GpuProgram{
        BASE 0 "ortho_transfer.cg.vp" "main_v";
        VPROG;
    }
    GpuProgram{
        BASE 0 "ortho_transfer.cg.fp" "simpletransfer_f";
        param "transfer" 0 VID_VALUE (0,0,0,0);
        param "campos" 1 VID_VALUE (0,0,0,0);
        param "axisconvx" 2 VID_VALUE (0,0,0,0);
        param "axisconvy" 3 VID_VALUE (0,0,0,0);
        FPROG;
    }
    Texture{
        TEX "Texture:0";
    }
    Texture{
        TEX "transferid.tga";
        texcoord -1; #no texcoords needed, computed by VS
    }
}
```

Abbildung 5.3: Luxinia Material und Shader. Das Material (links) definiert eine Eingabetextur und den zu verwendenden Shader (rechts). Beide entstammen der Entwicklung dieser Arbeit und werden für die orthogonalen Ansichten (5.3.2) benutzt.

deren Transformationsmatrizen, während der Shader definiert, wie diese benutzt werden sollen. Im Anwendungsgebiet für virtuelle Umgebungen unterscheiden sich Oberflächen oft nur durch ihre Texturen, welche zum Beispiel diffuse und Glanzfarben kontrollieren. Die Operationen, die beschreiben, wie diese Texturen kombiniert werden, sind dagegen oft gleich. Um Redundanz zu vermeiden und auch Leistung zu gewinnen, ergab sich daher die Trennung in Material und Shader.

Ein Material besteht folglich meist aus mehreren Texturzuweisungen und einem Shader, der festlegt, wie diese Texturen verarbeitet werden sollen. Im Shader wird auch geregelt, wie mit verschiedenen Beleuchtungssituationen umgegangen werden soll. Ein Shader ist aus mehreren Techniken (*Techniques*) aufgebaut, welche verschiedenen Hardwarefähigkeiten entsprechen. So kann der gleiche Effekt je nach Systemleistung unterschiedlich komplex realisiert werden. Die für die derzeitige Hardware geeignetste Technik wird angewendet, während alle anderen ignoriert werden. Es muss mindestens eine Technik gespeichert sein, welche sich aus mehreren Durchläufen zusammensetzt. Jeder Durchlauf (*Pass*) bedeutet dabei einen Zeichenaufruf mit dem aktuellen Dreiecksobjekt. Alle Techniken, die in dieser Arbeit entwickelt wurden, kamen mit einem Durchlauf aus. Jeder Durchlauf speichert die Renderingzustände, die beim Zeichenaufruf verwendet werden sollen. Dazu gehören etwa Textureingaben, GPU-Programme und deren Parameter. Texturen können entweder vom Material stammen, das den Shader verwendet, oder vom Shader selbst geladen worden sein. Pro Durchlauf kann maximal ein Vertex und ein Fragment-GPU-Programm aktiviert sein. Eine derartige Beschreibung von Effekten ist Standard in vielen Spielen

und liegt dem DirectX .FX Dateiformat zu Grunde [11]. Fast alle Shader in SinusEndoscopy bestehen aus einem Duchlauf mit je einem Vertex- und einem Fragmentprogramm. Die Anzahl der Eingabetexturen variiert je nach Volumenrenderingtechnik. Ausführlichere Informationen über das Material- und Shadersystem sind in [Kubisch, 2007] zu finden.

5.2.3 Aufbau des Renderers

Damit jene Materialeffekte sichtbar werden, müssen Dreiecksobjekte gezeichnet werden. Der Renderer von Luxinia kann eine Reihe unterschiedlicher Visueller Szenenknoten (`l3dnode`) darstellen.

l3dmodel Statische oder per Skelettanimation deformierte 3D Modelle.

l3dprimitive Einige Grundkörper wie Rechteck, Kugel, Box und Zylinder. Es besteht die Möglichkeit, eigene Primitive zu erstellen.

l3dtext 2D Text im Raum.

l3dlight Punkt- oder parallele Lichtquellen. Eine kann als Sonne definiert werden, andere sind Effektlichter mit begrenzter Reichweite.

l3dcamera Eine perspektivische oder orthographische Kamera, mit der Ansichten gerendert werden.

l3dprojector Projektor von Texturen auf Geometrie.

l3dtrail Zum Zeichnen von Linien oder Spuren.

l3dpemitter,l3dpgroup Partikelemitter oder Partikelgruppen die kontrolliert werden.

Diese visuellen Beschreibungen werden an eine Lagebeschreibung gekoppelt, welche entweder statisch oder dynamisch in ihrer Bewegung ist. Dynamische Knoten können in Luxinia von einer physikalischen Simulation kontrolliert werden. Die Szenenknoten sind in verschiedenen Bühnen (`l3dsets`) organisiert. Jede Bühne fungiert als eigenständige visuelle Welt. In dieser Welt können mehrere Ansichten (`l3dview`) erzeugt werden; jede Ansicht benutzt eine Kamera (`l3dcamera`) für die Projektion. Ein dynamischer *OcTree* übernimmt vor dem Rendern das Entfernen aller Objekte außerhalb des Sichtfeldes. Die Szenenknoten besitzen zum Teil Zeichenknoten, die alle Informationen zum Zeichnen der in OpenGL zulässigen Primitive enthalten. Zusätzlich speichern sie die Materialzuweisungen pro Objekt. Die Zeichenknoten lassen sich einzelnen Ebenen (`l3dlayers`) zuweisen, so dass der Benutzer eine gewisse Kontrolle über die Reihenfolge der gezeichneten Objekte hat. Innerhalb einer Ebene kann nach Distanz zur Kamera oder Effizienz sortiert werden. Die Szenenbeschreibung für SinusEndoscopy spielte nur eine untergeordnete Rolle. Es gibt eine perspektivische Kamera, die zunächst die segmentierten Modelle oder Werkzeugmodelle darstellt, und danach die Geometrie, die zum Zeichnen des Volumens benötigt wird. Für den in der Arbeit vorgestellten Renderer ist dies ein geschlossenes Boxmodell mit je 8×8 Rechtecken pro Seite und Flächen, die nach Innen zeigen.

Während der Arbeit wurde Luxinia um einen `RenderCommand` Mechanismus erweitert, so dass innerhalb jeder `l3dview` noch genauer die Renderingabläufe kontrollierbarer sind. Vor allem Funktionen der FBO-OpenGL Extension wurden bereitgestellt, sowie das Zeichnen von

[11] `http://www.toymaker.info/Games/html/effects_files.html` (Stand 2.2007)

ansichtsfüllenden Rechtecken. Für die Renderingtechniken, die viel im Bildraum arbeiteten, war dies notwendig. Der Ablaufplan ist in Abbildung 5.4 gezeigt. Je nachdem welches Prinzip angewendet wird, ob *SinglePass* oder *MultiPass* Verfahren wie in Kapitel 4 erläutert, wird vor dem Zeichnen des Volumens das Zeichnen in andere Texturen umgeleitet. Diese können, wie in der rechten Seite des Schaubilds gezeigt, wiederholt geglättet werden. Abschließend wird im Kompositionsverfahren ein Rechteck als finale Ausgabe gezeichnet, welches das Kompositionsmaterial der aktuellen Volumenrenderingtechnik benutzt.

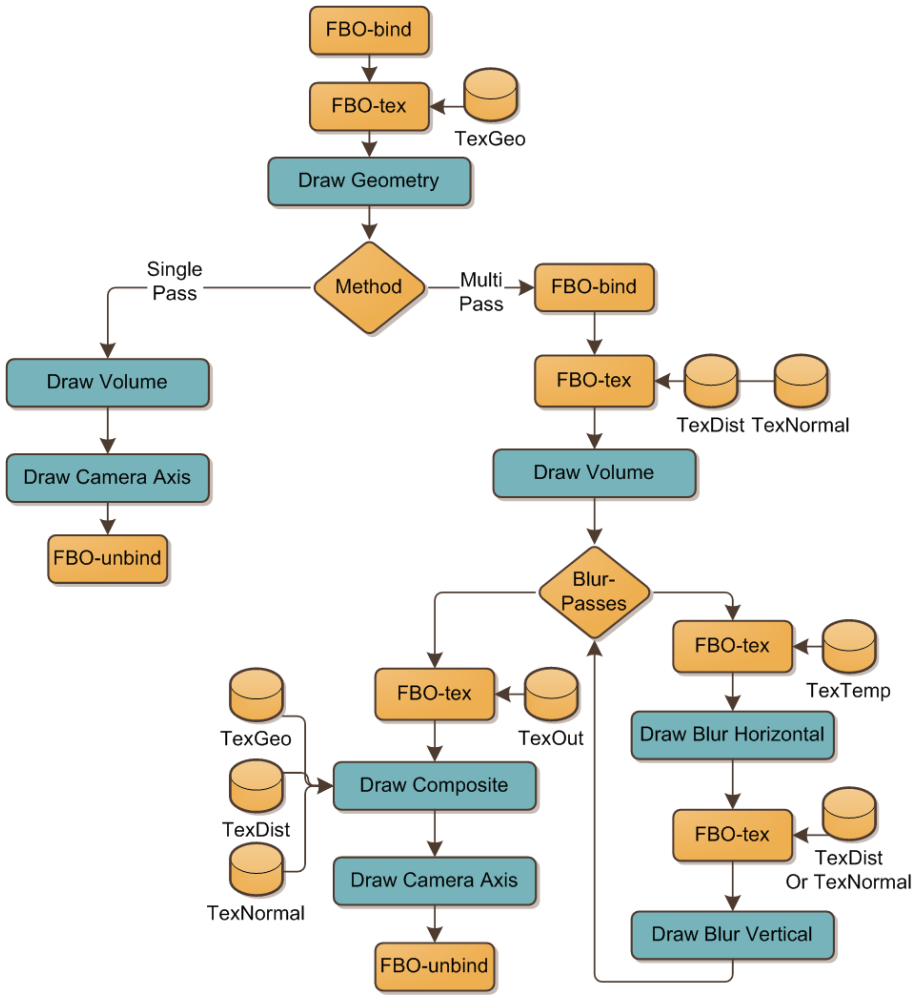

Abbildung 5.4: `RenderCommand` Ablaufplan für *SinglePass* (links) und *MultiPass* (rechts). *SinglePass* gibt TexGeo und *MultiPass* gibt TexOut zum späteren Zeichnen auf der Benutzeroberfläche zurück.

5.3 Strukturierung von SinusEndoscopy

Die Entwicklung dieser Renderingabläufe und der gesamten Applikation SinusEndoscopy verlief in zwei Schritten. Das erste Testframework, was in Abbildung 5.5 zu sehen ist, hatte nur eine einfache Steuerung und besaß lediglich die perspektivische Endoskopsicht. Hauptziel dieses Frameworks war das Experimentieren mit verschiedenen Techniken, insbesondere auch mit

den älteren Standardverfahren, die polygonale Schnittebenen benutzen (vgl. 3.2.3). Trotz der Einfachheit des Frameworks waren folgende Basisfunktionen enthalten:

Navigation Mausgesteuerte Navigation aus der Ich-Perspektive.

Datensätze Dynamischer Wechsel zwischen verschiedenen 3D Volumentexturen.

Rendertechniken Austausch des Renderingmechanismus zur Darstellung der 3D Textur.

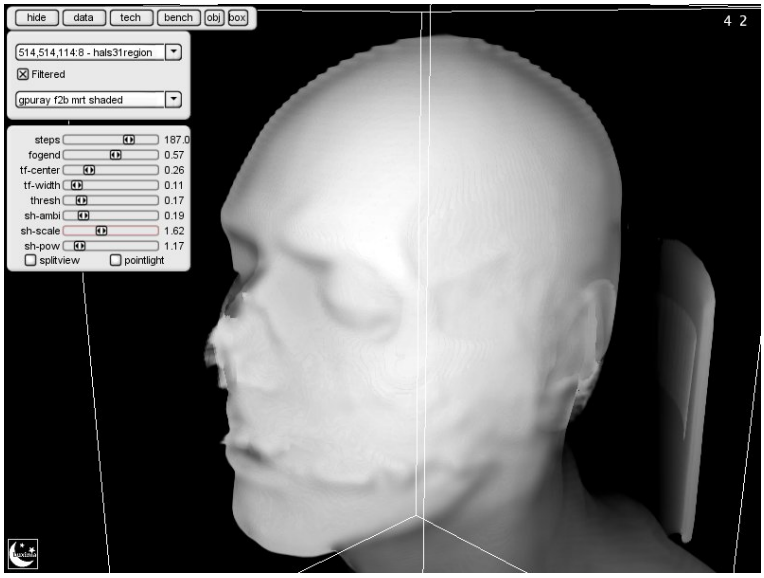

Abbildung 5.5: Erstes Testframework. Es besaß noch keine orthogonalen Schnittansicht, aber die wichtigsten Funktionen zum erneuten Laden von Daten oder Rendertechniken, sowie Parametermanipulationen.

Nachdem die Tests und Vergleiche mit den unterschiedlichen Basistechniken abgeschlossen waren, wurde das Framework überarbeitet, um die Benutzbarkeit zu verbessern. Hierbei wurden Gemeinsamkeiten unter den Techniken identifiziert und Redundanzen entfernt. Die Navigation wurde durch das Hinzufügen orthogonaler Ansichten erleichtert. Eine weitere Neuerung war das Laden von bereits segmentierten Strukturen. Die interne Programmarchitektur orientiert sich an dem Mehrschichtmodell, das von [Vollrath et al., 2005] vorgestellt wurde:

Anwendung Verantwortlich für die Applikationslogik und Reaktion auf Benutzereingaben.

Daten Speicherung von Volumendaten und abgeleiteter Daten, wie Histogramme und Transferfunktionen.

Renderingalgorithmus Gruppierung aller Komponenten, die für die Darstellung der Volumendaten notwendig sind.

Dieser Aufbau erscheint durch die Wahl der Basisfunktionen als sinnvoll. Es gibt jedoch einige Unterschiede zum Framework, das in [Vollrath et al., 2005] definiert ist. In SinusEndoscopy sind die Hilfsfunktionen nicht so zahlreich, da nach der Testphase bereits feststand, in welcher Form das GPU-Raytracing benutzt werden würde. Ihr Framework bietet hier wesentlich mehr Möglichkeiten, um eine größere Menge unterschiedlicher Herangehensweisen einfacher zu testen.

Dies kann darauf zurückgeführt werden, dass ihr Anliegen das generelle Volumenrendering war. Das heißt große Datenmengen und Betrachtung von außerhalb sind dort wichtiger als in SinusEndoscopy. In SinusEndoscopy können diese Verfahren ebenfalls umgesetzt werden, allerdings mit mehr Programmieraufwand. Diesen Vereinfachungen in SinusEndoscopy stehen aber einige Ergänzungen gegenüber, wie dem Zeichnen von segmentierten Modellen und dem Zugriff auf deren Ausgabedaten, wie Farb- und Tiefenwerte in Form von Texturen. SinusEndoscopy besteht

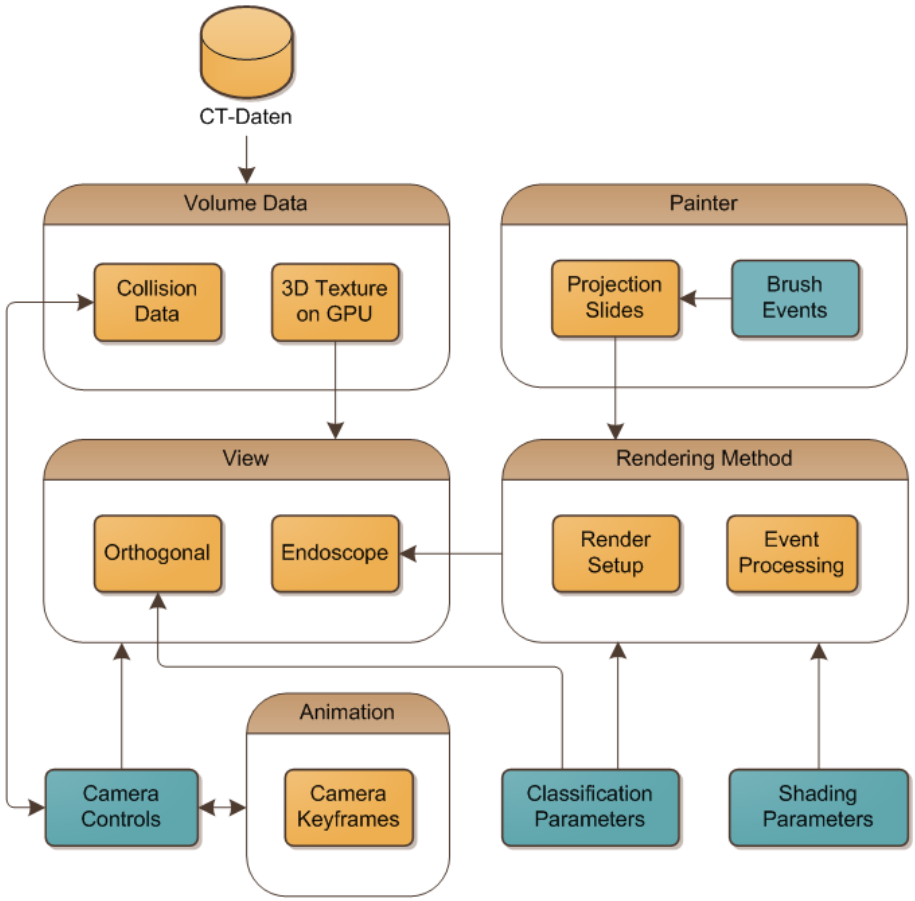

Abbildung 5.6: Programmstruktur von SinusEndoscopy. Benutzerinteraktionen wurden mit blau hervorgehoben.

aus folgenden Strukturen, die in Abbildung 5.6 schematisiert sind:

Application Erstellt das Hauptmenü und kümmert sich um das Laden der Volumendaten, Dreiecksmodelle sowie der Renderingtechniken. Es verwaltet die Einstellungen zur Sichtweite und Transferfunktion.

View Zur Betrachtung der Volumendaten stehen verschiedene Ansichtstypen und Layouts zur Verfügung.

Endoscope Die Perspektivische Endoskopsicht, die mit der ausgewählten Renderingtechnik die Volumendaten darstellt. Vor dem Rendern des Volumens werden die Dreiecksmodelle gemalt. Über Ziehen der Maus wird je nach Tastendruck die Kameraposition oder Rotation verändert.

OrthoView Drei orthogonale Ansichten entlang der Hauptachsen zeigen die Schnittebene mit der aktuellen Kameraposition. Per Maus kann der Benutzer die Kameraposition ändern. Es stehen Zoom- und Paninteraktionen zur Verfügung.

Volume Data Speicherung von Volumendaten und abgeleiteter Daten, wie Histogramme und Transferfunktionen.

Rendering Method Gruppierung aller Komponenten, die für die Darstellung der Volumendaten notwendig sind.

Animation Mit diesem Modul können die Kamerazustände zu verschiedenen Zeiten gespeichert werden, um eine Animation zu erstellen und wiederzugeben.

Painter Erlaubt dem Benutzer über alle Fenster hinweg zu zeichnen. Es können vier verschiedene Folien angelegt werden. Je nach Renderingtechnik ist es auch möglich, den Ausschnitt über der Endoskopsicht permanent auf die Gewebefläche zu projizieren.

Die verschiedenen Strukturen kommunizieren über ein Nachrichtensystem untereinander. Gibt es Veränderungen, werden virtuelle Funktionen der Benutzer aufgerufen und ihnen die Zustandsänderung mitgeteilt. Folgende Nachrichtentypen sind definiert:

- Wechsel des Datensatzes
- Wechsel der Renderingtechnik
- Laden eines segmentierten Modells
- Änderung der Kameraposition oder -rotation
- Änderung der Sichtweite
- Änderung der Transferfunktion
- Änderung der Fenstergröße
- Änderung der Ausgabeauflösung
- Projektion einer Zeichnungsfolie

Im Folgenden wird näher auf einige dieser Strukturen der finalen Benutzeroberfläche von SinusEndoscopy eingegangen, wie sie in Abbildung 5.7 zu sehen ist.

5.3.1 Steuermodul

Neben der Erstellung der Benutzeroberfläche und den damit verbundenen Kontrollfunktionen, sind hier vor allem die Ladefunktionen verankert. Die Applikation kann Volumendaten im 16-Bit RAW Format einlesen und daraus 8,16 oder 32-Bit 1 Kanal OpenGL 3D Texturen erstellen oder direkt Volumentexturen im DDS Format lesen. Für die Dreiecksmodelle kommen entweder Luxinias proprietäres F3D Format in Frage oder ein einfacher OpenInventor Dateilader, der an die Ausgaben segmentierter Modelle von MeVisLab angepasst ist, und im Rahmen dieser Arbeit entwickelt wurde.

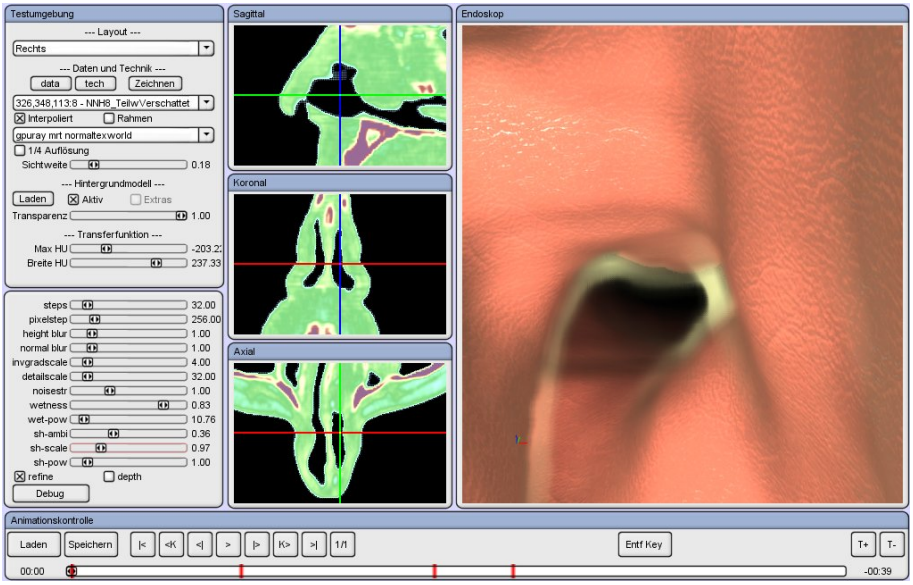

Abbildung 5.7: Die Benutzeroberfläche von SinusEndoscopy. Links oben sind die wichtigsten Steuerfunktionen, darunter befinden sich je nach Renderingmethode unterschiedliche Manipulations- oder Debugmöglichkeiten. Im unteren Bereich ist die Zeit und Animationssteuerung. Den größten Teil der Oberfläche machen die Ansichten zur Betrachtung der Volumendaten aus. Alle Bereiche passen sich automatisch neu an, wenn die Fenstergröße der Applikation verändert wird, um den Platz optimal auszunutzen.

5.3.2 Ansichten

Die Ansichten setzen sich zusammen aus orthogonalen Ansichten und einer perspektivischen. Die orthogonalen Ansichten für axiale, sagittale und koronare Schnittbilder sind in der medizinischen Informatik weit verbreitet [Lehmann et al., 2005]. Je nach Bedarf sind verschiedene Layouts möglich, im Moment ist die klassische Viertelansicht, wie sie auch in [Neubauer et al., 2004] verwendet wird, und eine Ansicht mit vergrößerter Endoskopiesicht möglich, wie sie in 5.7 zu sehen ist. Kamerabewegungen können entweder durch die orthogonale Ansicht oder die perspektivische ausgelöst werden. In jedem Fall wird die neue Position der Kamera überprüft, so dass sie nicht innerhalb des soliden Bereichs liegen kann. Dafür wird eine Testanfrage an den aktuellen Datensatz gestellt. Bei gültiger Bewegung werden die Ansichten aktualisiert und das Animationssystem benachrichtigt.

In der orthogonalen Sicht wird die 3D Textur mit einem Shader dargestellt, der die Hauptachsen visualisiert. Gleichzeitig entspricht die Schnittposition dieser Achsen denen der Kamera in der aktuellen Ebene im Volumen. Zur Darstellung der Skalarwerte wird eine 1D Farbskala verwendet, deren Ergebnis je nach dynamischer Transferfunktion abgedunkelt und mit Stippling versehen wird. Diese Darstellung ist in Abbildung 5.8 zu sehen. Alle schwarzen Bereiche repräsentieren Leerräume und die gepunkteten weisen auf Sekretschichten hin, innerhalb der die Kamera bewegt werden kann. Die Implementierung war bereits in den Abbildungen 5.1 und 5.3 zu sehen.
Die Endoskopsicht rendert zunächst die normalen Dreiecksdaten in eine Farb- und Tiefentextur. Dabei werden durch Verwendung des Stencilbuffers jene Stellen im Alphakanal des Bildes markiert, in denen sich die Kamera innerhalb eines Dreiecksmodells befindet. Dies setzt jedoch voraus, dass alle Modelle geschlossene Oberflächen haben. Anschließend wird die Volumentechnik

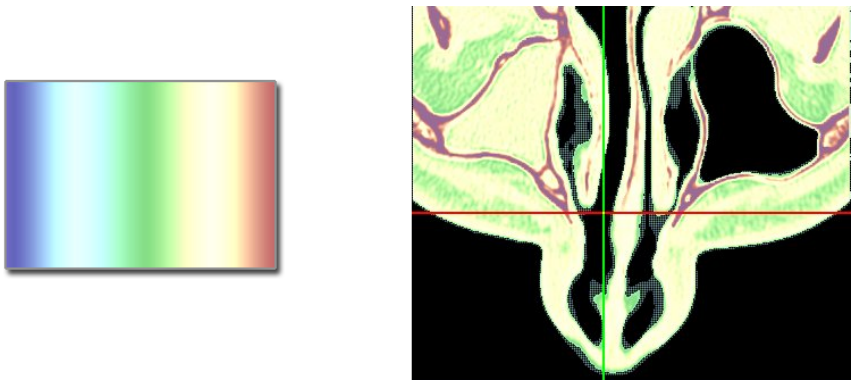

Abbildung 5.8: Darstellung der Dichtewerte des Volumens in den Schnittbildern. Der Wertebereich wird mit einer Farbskala (links) visualisiert. Rechts ist die finale Ausgabe der Ansicht zu sehen, bei der Leerräume, Sekretschichten und die Position der Kamera eingezeichnet sind.

mit der gleichen Kamera gezeichnet. Die Technik kann selbst bestimmen, ob sie in die gleichen Ausgabetexturen rendert wie das Dreiecksmodell (*SinglePass*), oder in eine eigene und das Ausgabebild per Komposition erstellt (*MultiPass*. Zuletzt wird zur Orientierung ein Achsenkreuz gezeichnet, das die Orientierung der Kamera visualisiert.

5.3.3 Renderingtechnik

Bei Aktivierung der Technik wird eine `l3dview`, in der die `RenderCommands` zum Zeichnen des Volumens definiert werden, dem `l3dset` der Endoskopsicht hinzugefügt. Zusätzlich kann die Technik noch Steuerelemente in der Benutzeroberfläche erstellen, die das Verändern von Parametern zulassen. In den meisten Fällen sind solche Parameter direkt mit Shaderparametern verbunden und über Schieberegler repräsentiert. Zwar ist die Genauigkeit bei der Eingabe der Werte im Vergleich zum numerischen Eingabefeld nicht so hoch, doch reicht dies meist aus, um Auswirkungstrends zu erkennen. Speziell für die Kompositionsverfahren wurde zusätzlich eine *Factory* Klasse geschaffen, die Techniken automatisch über einige Anfrageangaben erstellen kann. Wegen der Variationsvielfalt dieser Gruppe aus Renderingtechniken konnte somit Redundanz vermieden werden. Alle anderen Techniken müssen eine Reihe von Funktionen selbst implementieren, in denen sie auf die Nachrichten reagieren und Aktivierungsfunktionen bereitstellen.

5.3.4 Volumendatensatz

Ein Datensatz beinhaltet eine OpenGL 3D Textur des Volumens und Angaben zur Ausgabegröße. Die meisten Testdaten waren entlang der Höhenachse niedriger aufgelöst und müssen deshalb in dieser Richtung skaliert werden. Dazu werden nicht die Volumendaten selbst verändert, sondern lediglich die Hüllgeometrie, die für die Strahltraversierung genutzt wird. Es kann ein beliebiger Skalierungsvektor für das Volumen angegeben werden, der mit den Texturdimensionen verrechnet wird. Das Gesamtvolumen wird auf eine Maximalausdehnung von 128 Einheiten normalisiert. Um den Wechsel zwischen Datensätzen zu erleichtern, erschienen einheitliche Größen sinnvoll, sonst wäre es wahrscheinlich, dass man mit der Kamera nach dem Wechsel schnell außerhalb liegt. Auch war es für die alten Techniken einfacher, von einem Volumen fester Maximaldimension auszugehen. Da nicht alle Grafikkarten die `ARB_texture_non_power_of_two`

Extension unterstützen, müssen die 3D Daten auf Dimensionen gebracht werden, die zur Potenzreihe von zwei gehören. Im Moment geschieht dies durch Abschneiden oder Leerauffüllen der 3D Textur, was entweder zu Informationsverlust oder erhöhtem Speicherverbrauch führt. Die damit veränderte Texturdimension muss intern zusätzlich kompensiert werden, damit gespeicherte Kamerapositionen von unterschiedlichen Systemen dennoch zu gleichen Ergebnissen führen. Die Daten für die Volumentextur müssen nicht unbedingt aus Bilddaten stammen, sondern können zu Testzwecken auch prozedural erstellt werden. Die Kollisionserkennung erfolgt durch die Evaluierung einer Kopie der Bilddaten im Arbeitsspeicher und der aktuellen Transferfunktion. Um die Interpolation zwischen den Bildpunkten der Grafikkarte anzugleichen, wurde der Quelltext von MESA herangezogen.

5.3.5 Animation

Das Animationsmodul implementiert die Interpolation zwischen Schlüsselzuständen der Kamera. Bei jeder Lageveränderung der Kamera wird ein Schlüsselbild (Keyframe) zum aktuellen Zeitpunkt erstellt oder ein vorhandener Schlüssel in zeitnaher Umgebung aktualisiert. Diese Schlüssel können auch entfernt werden. Die Darstellung und Funktionalität wurde der TrackView von

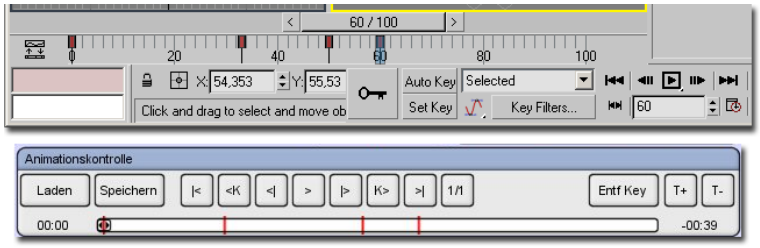

Abbildung 5.9: Oben: Autodesk 3dsmax Trackview. Unten: Animationsleiste in SinusEndoscopy.

Autodesks 3dsMax entnommen (siehe Abbildung 5.9). Die Interpolation erfolgt für die Position kubisch nach *Catmull-Rom-Splines* und für die Rotation über Quaternion sphärisch. Während des Abspielens wird die Kollisionserkennung mit dem Volumen aufgehoben. Animationsdaten können geladen und gespeichert werden.

5.3.6 Zeichenmodul

Mit Hilfe des Zeichenstifts kann über die gesamte Oberfläche gezeichnet werden. Ist der Modus aktiviert, wird ein Rahmen eingeblendet, der verdeutlicht, dass alle Mausaktionen auf dem Bildschirm als Zeichenaktionen gewertet werden. Lediglich das Zeichenmenü selbst kann noch bedient werden. Mit diesem Menü ist es möglich, zwischen vier Folien zu wechseln. Der Anwender kann zwei verschiedene Stifte definieren, die sich in Farbe und Größe unterscheiden können. Das Zeichenelement ist ein Kreis mit einer maximalen Größe von 64 Pixeln. Ziehen mit der linken Maustaste löst die Zeichenfunktion aus, während bei gedrückter rechten Maustaste ein Rechteck mit doppelter Stiftgröße gelöscht wird. Durch einen variablen Transparenzwert können Regionen farbig gekennzeichnet werden, ohne vollständig überdeckt zu sein. Die Daten der Textur werden direkt auf Pixelebene manipuliert und an die Grafikkarte geschickt, so dass ein Vollbild Quadrat zum Zeichnen der Resultate ausreicht.

Sollte es die Volumenrendertechnik erlauben, kann ein Schnappschuss des Ausschnitts über der Endoskopsicht auf die Gewebewand projiziert werden. Die Rechtecksregion wird in einen

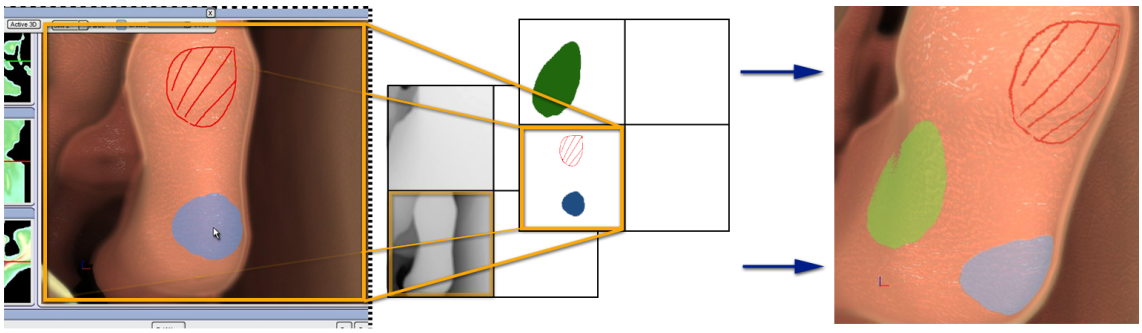

Abbildung 5.10: Zeichnen auf der Oberfläche. Zunächst wird die Zeichnung (links), die über der Endoskopsicht liegt, in eine Atlastextur übertragen (mittig). Diese kann dann auf das Gewebe projiziert werden (rechts). Der Atlas erlaubt zur Zeit maximal vier Projektionen zur gleichen Zeit.

Texturatlas kopiert, der alle Folienausschnitte speichern kann, dies spart die Anzahl der benötigten Texturen im Shader. Darüber hinaus werden pro Folie die Projektionsmatrix, Sichtweite und Kameraebene zum Zeitpunkt der Übertragung in den Atlas gespeichert. Jene Parameter ermöglichen es zusammen mit einem zweiten Texturatlas, der die Tiefeninformationen speichert (vgl. Abschnitt 4.3.4), das gezeichnete auf dem Volumen anzubringen und im Nachhinein die Kamera zu bewegen. Mit diesem Werkzeug ist es dem Mediziner möglich, Eingriffe zu dokumentieren oder Strukturen bei der Patientenaufklärung hervorzuheben. Der Prozess wird in Abbildung 5.10 illustriert.

Die Implementation verlief erfolgreich und ohne größere Schwierigkeiten, bis auf die Problematik der Verwendung von Cg auf Nicht-NVIDIA Hardware. Durch den Einsatz von High-Level Sprachen und die Benutzung der 3D Engine Luxinia konnten die notwendigen Quelltextzeilen minimiert werden. Dieser Zeitgewinn konnte in die praktischen Vergleiche vieler Verfahrensvariationen und dem Einbetten neuer Techniken investiert werden. Allerdings bleibt immer noch Raum für Verbesserungen, auf die im nächsten Kapitel eingegangen wird.

Kapitel 6

Abschließende Bewertung

Das erfolgreiche Abschließen der Implementierung ermöglicht den Vergleich mit anderen Volumenrenderern und Systemen zur virtuellen Endoskopie. Dieser Vergleich geschieht jedoch mehr in den Punkten Funktionsumfang und Qualität der Bilddarstellung, als im Bereich der Leistung. Zum einen, weil die in Kapitel 5.1.1 angesprochenen Treiberoptimierungen starke Fluktuationen beim Messen verursachen, zum anderen, weil die am Anfang erwähnte enorme Leistungsentwicklung im Bereich der Grafikkarten ständig für Verbesserung sorgt. Beim Leistungskriterium sollte daher mehr darauf eingegangen werden, inwiefern es von dieser Entwicklung profitieren kann. Die Gesamtleistung des Systems wird zusätzlich durch den Funktionsumfang der Applikation beeinflusst. Man kann leider einigen Leistungsangaben in anderen Arbeiten nicht entnehmen, bei welcher Auflösung die Werte gemessen wurden, bzw. ob die Messung isoliert oder eingebettet in einer größeren Applikation vorgenommen wurde. So waren die Leistungswerte des vorher erwähnten ersten Testframeworks höher als die der finalen Applikation, selbst bei gleichen Verfahren. Da der Renderer aber selten isoliert Verwendung finden wird, erscheint die isolierte Betrachtung weniger relevant. Aus diesem Grund wird sich die Bewertung mehr auf die Qualität der Bilddarstellung und des Umgangs mit der Software konzentrieren.

6.1 Vergleich mit anderen Systemen

SinusEndoscopy wurde für ein spezielles medizinisches Anwendungsgebiet entwickelt, für das noch keine optimierten VE-Systeme vorlagen. Deshalb werden zum Vergleich ältere Systeme oder solche mit ähnlichen anatomischen Gegebenheiten hinzugezogen. Bei dem älteren Endoskopiesystem z.B von [Vilanova et al., 2000] war der Freiheitsgrad noch stark eingeschränkt. Jene Arbeit musste aus Performancegründen noch auf vorher berechnetes Bildmaterial zurückgreifen, was lediglich eine geführte Navigation entlang des Arbeitsbereichs zuließ. Nichtsdestominder konnte durch die Vorberechnung bereits recht gute Qualität bei der Darstellung erzielt werden. Vor allem war ein Vorteil dieser Technik, dass das Gerät zur Darstellung kaum beansprucht wurde, und damit ein großer Nutzerkreis zur Verfügung stand.

Wie man in Abbildung 6.1 sehen kann, ist die Oberfläche einfach gehalten, ermöglicht aber die wichtigsten Grundfunktionen, wie eine Übersicht über die Strukturen und die Endoskopansicht, in der freie Drehung möglich ist. Die Resultate von SinusEndoscopy sind im Punkt der Qualität mit [Vilanova et al., 2000] vergleichbar, insgesamt aber besser für den Anwendungsfall optimiert und erlauben durch die Generierung der Bilder zur Laufzeit eine viel höhere Flexibilität bei der Interaktion. Diese Flexibilität hat natürlich den Preis, dass es keine Ersparnis durch Vorberechnungen gibt, und damit die Hardwareanforderungen viel höher sind.

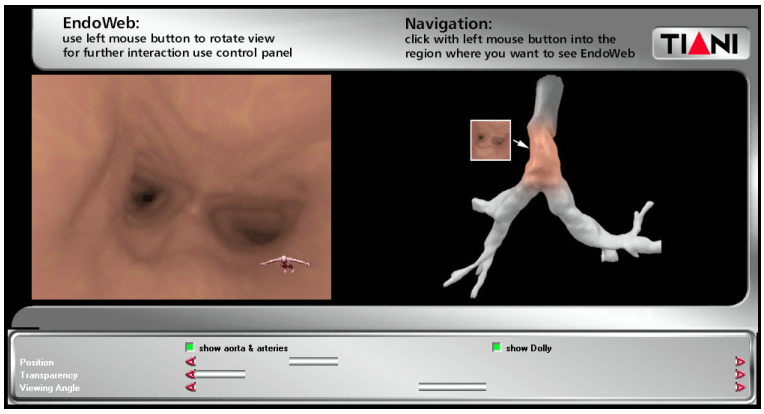

Abbildung 6.1: CubeMap-basiertes Endoskopiesystem. Der Benutzer kann sich nur entlang eines Pfades bewegen, die freie Drehung der Kamera wird über CubeMaps realisiert, die vorher berechnete Bilder anzeigen. Quelle: [Vilanova et al., 2000]

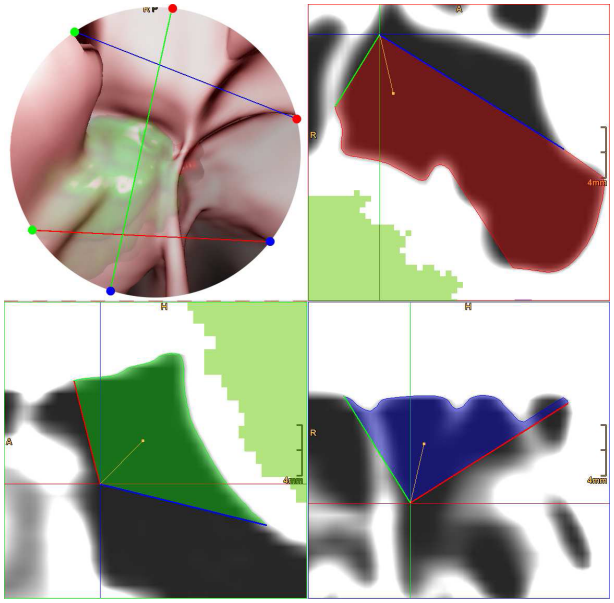

Abbildung 6.2: STEPS Hauptansichten. Sowohl segmentierte Daten, als auch die Sichtkegel werden in den orthogonalen Schnittansichten dargestellt. Quelle: [Neubauer, 2005]

Ein aktuelleres System, genannt STEPS (Simulation of Transsphenoidal Endonasal Pituitary Surgery), zur virtuellen Endoskopie in den Nasennebenhöhlen ist den Ergebnissen dieser Arbeit näher. Zwar ist die anatomische Umgebung die gleiche, doch dient STEPS für die Planung und Simulation von Eingriffen, bei denen Gehirntumore entfernt werden und die Nasennebenhöhlen nur ein Zugangskanal sind. Der Renderer von STEPS benutzte Zellen-basiertes Ray Tracing, das eine reine CPU Lösung ist und in eine Java-basierte PACS J-Vision Umgebung integriert ist. Interaktive Bildraten werden ab einem Pentium 4 3Ghz bei etwa der Hälfte der Gesamtanzahl der Pixel von SinusEndoscopys Standardauflösung erzielt. In [Neubauer et al., 2004] kann man Funktionsweise und Aufbau des Systems entnehmen. Die Benutzeroberfläche ist, wie man in Abbildung 6.2 sehen kann, diesem System ähnlich, mit den orthogonalen Ansichten und der Endoskopsicht. Bei den orthogonalen Sichten fällt auf, dass der Sichtkegel hervorgehoben ist, was für SinusEndoscopy übernommen werden sollte. In den Ansichten sind auch segmentierte Modelle zu erkennen. Da STEPS die segmentierten Modelle auch aus den Originalvolumendaten gewinnt, ist hier keine Trennung des Renderingverfahrens notwending. Bei SinusEndoscopy muss dies mit Hilfe von polygonbasierten Modellen geschehen. Die Zusammenführung der Ergebnisse der Hintergrundmodelle und der Zellwände erfolgt aber in beiden Applikationen in einem ähnlichen Kompositionsschritt. SinusEndoscopy benutzt zusätzlich Stippling, um die Transparenz noch deutlicher zu machen, während STEPS weiter entfernte Objekte langsam ausblendet. STEPS ist folglich auf ein Segmentierungsmodul angewiesen, in welchem der Nutzer einzelne Strukturen manuell oder halbautomatisch extrahieren kann. SinusEndoscopy verzichtet auf derartige Module, um die Anwendungskomplexität geringer zu halten, dafür ist das Shading der Zellwände durch Texturierung detailierter gestaltet. Im Bereich der Interaktion bietet STEPS Vorteile, die in SinusEndoscopy ergänzt werden sollten. Mit einem *Force-Feedback* Joystick kann die Einschränkung der Bewegungsfreiheit durch Kraft-Rückkopplung dem Benutzer spürbar übermittelt werden. Für die Simulation ist es auch wichtig, andere Optiken, die seitlich angebracht sind und unterschiedliche Öffnungswinkel haben, verwenden zu können. Schließlich kann auch das Gewebe verändert werden, so dass Schnitte simuliert werden können. Ein anderer Modus unterstützt die Exploration des Volumens durch eine geführte Navigation zu einem Zielpunkt im Volumen. Die in dieser Arbeit entwickelte Applikation verfügt noch nicht über derartige Bewegungsmodi, mit dem Animationsmodul kann jedoch in wenigen Schritten ebenfalls ein geführter Kameraflug erstellt werden.

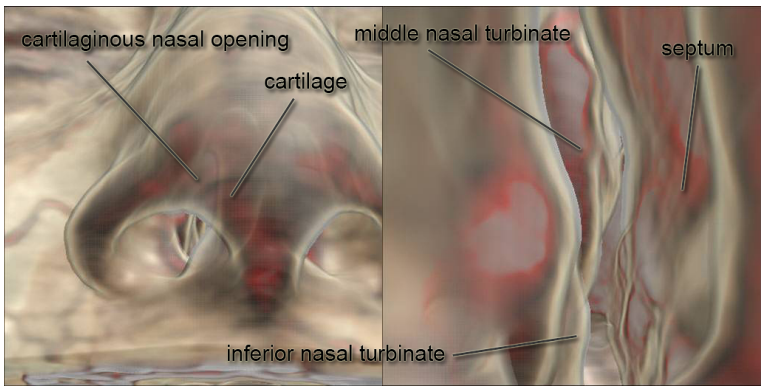

Abbildung 6.3: GPU Renderer für STEPS. Die Semitransparente Gewebeoberfläche erlaubt einen Blick auf verborgene Strukturen. Quelle: [Scharsach et al., 2006]

Später wurde die hohe Belastung der CPU durch STEPS vermindert, indem ein GPU-basierter Raycaster ergänzt wurde. Doch ist das Betrachtungsmodul immer noch Teil eines

größeren Systems und keine „Stand-Alone“ Lösung wie SinusEndoscopy. [Scharsach et al., 2006] stellen den Renderer vor, der vom Prinzip ähnlich dem hier entwickelten System ist und dessen Ergebnis in Abbildung 6.3 zu sehen ist. Der primäre Unterschied des Renderers ist jedoch eine komplexere Transferfunktion, welche die Strukturen hinter den Gewebewänden noch differenzierter darstellen soll. Ihr Renderer findet zunächst die feste Gewebeschicht als Isooberfläche und führt die Beleuchtung mit Gradienten aus der 3D Textur durch. Danach kann der Strahl weiter verfolgt werden, sofern der Benutzer dies wünscht und die Oberfläche als transparent definiert. In diesem Schritt erfolgt die Evaluierung dann praktisch über direktes Volumenrendering ohne Beleuchtungseffekte. Dies ermöglicht einen Blick hinter die Gewebewände und die Hervorhebung von anatomischen Details, die sonst verborgen sind. SinusEndoscopy erlaubt dies nicht, weil es in einem anderen Wertebereich arbeitet. Es bezieht die Werte vor der Gewebeschicht mit ein und stoppt dann an dieser endgültig. STEPS wurde nicht dafür entwickelt, um auf Entzündungskrankheiten oder Sekretverstopfungen einzugehen. Man geht von einem gesunden Gewebe aus bis zur Stelle des Tumors, weswegen auch der Blick hinter das Gewebe viel wichtiger ist. Der medizinische Fokus bei SinusEndoscopy ist anders, die Visualiserung von verschleimten Regionen und eine realitätsnahe Darstellung der Oberfläche standen im Vordergrund. Was hinter den Wänden liegt, könnte dennoch mit Hilfe von Dreiecksmodellen visualisiert werden. Abbildung 6.4 zeigt die Vorteile von SinusEndoscopy bei der Darstellung von Sekretverstopfungen, indem die Transferfunktion flexibel angepasst werden kann.

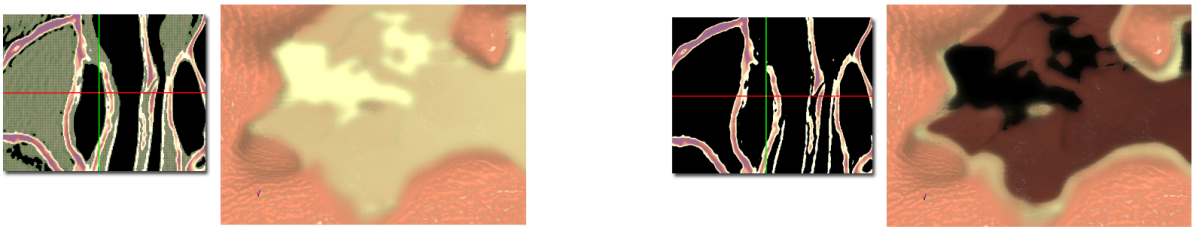

Abbildung 6.4: Durch die flexible Anpassung der Transferfunktion ist es in SinusEndoscopy möglich, auch stark verschleimte Regionen darzustellen. Im rechten Bild werden die Sekretdichten hervorgehoben, während diese im linken vernachlässigt wurden. Rechts ist jeweils der zugehörige Axialschnitt zu sehen.

6.2 Zukünftige Entwicklungen

Aus dem Vergleich mit STEPS und anderen Renderern ergeben sich eine Reihe von Verbesserungsmöglichkeiten. Die Diskussion geht hierbei zunächst auf die visuelle Verbesserung und danach auf Interaktionsmöglichkeiten des Benutzers ein.

6.2.1 Renderer

Die erste Verbesserung des Renderers sollte die Steigerung der Bildqualität sein, wenn die Kamerabewegung ausbleibt. Das starre Blickfeld erlaubt weit höhere Berechnungszeiten, womit viele Vereinfachungen wegfallen können. Die Samplingschrittweite des Renderers würde auch im statischen Fall erhöht werden, da die Renderingleistung moderner Grafikkarten selbst bei Volllast immer noch Bilder innerhalb 1-2 Sekunden generieren würde, in den meisten Fällen sogar

schneller. Diese adaptive Qualitätssteigerung wird bereits erfolgreich beim Volumenrendering benutzt [Link et al., 2006].

Der Renderingalgorithmus lässt sehr viele Parametereinstellungen zu. Einige beeinflussen die Qualität und Leistung unmittelbar, andere vor allem die Shading- und Farbwerte. Diese sind schwerer in ihren Auswirkungen einzugliedern. Insgesamt stehen jedoch im Moment dem Nutzer zu viele Parameter zur Verfügung, die zu unterschiedlichsten Ergebnissen führen können. Im Bereich der Leistung kann hier durch Qualitätsabstriche und Leistungsevaluierung der einzelnen Variationen des Renderers auf dem System eine entsprechende Kombination automatisch ermittelt werden. Das System sollte durch Zeitmessung feststellen, welche Verfahren im Durchschnitt mehr als 25 Bilder pro Sekunde erzeugen können, und diese dann bei Kamerainteraktion benutzen. Ruht die Kamera. könnte ein Bild mit der aufwändigsten Technik berechnet und eingefroren werden. Hierbei wäre es sinnvoll, die Zwischenergebnisse, wie die Tiefentextur, geglättete Normalen usw. extra aufzuheben, den Kompositionsschritt aber dynamisch durchzuführen. Das hätte den Vorteil, dass das Rendern der Dreiecksmodelle, welches ohnehin getrennt vorher geschieht, unabhängig veränderbar bleibt. So wäre das Neuladen von Dreiecksmodellen, sowie deren Transparenz bei hohen Bildraten, möglich, ohne dass das Volumen neu berechnet werden müsste.
Neben der Auswahl einer passenden Technik könnte auch einfach die Ausgabetextur dynamisch verkleinert werden, um eine interaktive Bildrate zu gewährleisten. Das Prinzip von *graceful degradation* ist in der Softwaretechnik auch in anderen Bereichen bekannt. Oft werden bei Bewegung einfachere Geometrien oder geringere Auflösungen benutzt. Letzteres lohnt sich bei füllratenabhängigen Verfahren wie diesem Renderingalgorithmus.

Dem automatischen Permutieren der Renderingverfahren sollte aber die Entwicklung eines Moduls vorausgehen, das den Programmcode, wie Cg- und Luxinia-Shader, automatisch aus Bausteinen zusammensetzt. Die Applikation besitzt zwar schon eine Programmklasse, die zur Erstellung der Techniken genutzt wird, aber diese bedarf noch einiger manueller Zuarbeit. Denn die Programmklasse kümmert sich in erster Linie um die Erstellung der `l3dview RenderCommands` und darum, die Shader mit notwendigen Parametern zu versorgen. Die Shadererstellung in ihren Variationen geschieht noch manuell. Es ist mittlerweile aber üblich geworden, Shader aus Programmbausteinen automatisch zu erstellen. In [Mittring, 2007, Sousa, 2007] wird kurz auf die Handhabung einer solchen Lösung eingegangen. Zwar entstammt dieses Beispiel der Spieleindustrie, doch ist die Problematik die gleiche. Für den selben Effekt gibt es je nach Hardwareleistung, und zum Teil auch Hersteller, unterschiedliche Lösungen. Außerdem können Effekte kombiniert sein, z.B. Variation der Anzahl von Lichtquellen, Art des Schattens usw.. Ein weiterer Vorteil wäre, dass Redundanz, die durch ähnliche oder fast gleiche Shader entsteht, beseitigt werden kann. Außerdem ließen sich einfacher automatische Evaluierungen der Leistungsauswirkungen verschiedener Einstellungen durchführen. Diese hätten bei einer repräsentativen Auswahl an Hardware auch den Vorteil, dass die Teile des Renderers, die nachweislich nirgends Verbesserung bringen, entfernt werden können.

Die Qualität des Renderingalgorithmus hängt bei der Benutzung von 8-Bit Tiefentexturen stark von der Sichtweite ab. Obendrein profitieren die präziseren Einstellungen von einer Helligkeitsverteilung, welche die Tiefenunterschiede besser zur Geltung bringt. Diese maximale Sichtweite automatisch anzupassen, wäre deswegen wünschenswert. Je weniger Parameter der Nutzer zur Laufzeit verändern muss, desto einfacher zugänglich wird die Software. Eine Sichtweitenanpassung wäre durch die Auswertung des Histogramms der Tiefentextur denkbar. Sind die dunklen Werte in der Überzahl, das heißt die meisten Pixel am Ende oder außerhalb der Sichtweite, sollte diese erhöht werden. Im Gegenzug kann bei zu hellem Durchschnitt versucht

werden, die Sichtweite zu verringern. In jedem Fall wäre wichtig, dass die Anpassung graduell über einige Bilder verteilt passiert, und nicht sofort, damit der Benutzer sich an die Veränderung gewöhnt. Es ist auch denkbar, die Region für die Erstellung des Histogramms auf die Bildmitte einzuschränken. Mit dieser Methode kann der Benutzer wie mit einer Art automatischen Fokus das Bild beeinflussen, nur dass statt der Bildschärfe die Helligkeit verändert wird. In [Mitchell et al., 2006] wurde ein Algorithmus erläutert, mit dem das Histogramm auf der GPU effizient berechnet werden kann. In jenem Fall wurden die Histogrammdaten zur Simulation der dynamischen Anpassung von Belichtungszeiten einer Kamera verwendet.

Der Vorteil des GPU-basierten Ansatzes ist, dass die CPU Belastung gering bleibt und die Parallelität von CPU und GPU erhalten bleibt. Da in unserem Anwendungsfall jedoch kaum CPU Ausnutzung stattfindet, könnte die Histogramm Auswertung auf der CPU stattfinden. Das Herunterladen der Tiefendaten aus dem Grafikkartenspeicher in den Hauptspeicher zur Weiterverarbeitung durch die CPU, ist jedoch eine sehr teure Operation, die eine Synchronisation von CPU und GPU erfordert, und man kann damit die Bildrate stark negativ beeinflussen. Es gibt jedoch die Möglichkeit, in OpenGL durch die `ARB_pixel_buffer_object` Extension Daten asynchron zu lesen oder zu schreiben. Vermutlich müsste auch das Tiefenbild nicht in voller Größe heruntergeladen werden, sondern könnte durch die GPU erst verkleinert werden. Die resultierenden Histogrammdaten sollten es dann ermöglichen, die Sichtweite dynamisch anzupassen.

Die Verzerrung des Bildes durch die Optik des Endoskops ist auch ein Effekt, um den der Renderer noch erweitert werden sollte. Eine einfache Variante, diesen Effekt umzusetzen, wäre bei der Komposition die Position zu verändern, mit der auf die Zwischenpuffer zugegriffen wird. In [Ansari, 2004] lassen sich auch Programmbeispiele finden, wie mit dem Fragementprozessor Bildverzerrungen realisiert werden können. Denkbar wäre auch, die Sichtstrahlen im Extraktionsschritt zu verändern. Allerdings wäre es dann nicht mehr möglich, polygonale Geometrie darzustellen, weil diese nicht mit Raycasting berechnet wird.

Schließlich bleiben damit die Shadingparameter neben der Transferfunktion die letzte Möglichkeit, das Ausgabebild zu verändern. Letztere sollte nicht eingeschränkt oder automatisiert werden, da der Vorteil des gesamten Systems eben jene Flexibilität ist. Einige festgelegte Einstellungen zur Auswahl wären jedoch praktisch, weil anzunehmen ist, dass meist ähnliche Daten betrachtet werden. Gleiches wäre bei den Shadingparametern der Fall, der Intensität der Beleuchtung, den Farben des Gewebes oder der Oberflächentextur, sowie der Darstellung der Nässe, möglich. Welche Werte hier sinnvoll erscheinen, muss eine Evaluierung durch Chirurgen ergeben. Anzunehmen ist, dass einerseits eine möglichst realistische Darstellung angestrebt wird, andererseits eine Darstellung, die räumliche Beziehungen deutlicher zum Ausdruck bringt. Durch den Einsatz von *Deferred Shading* ist es möglich, eine Vielzahl zusätzlicher Informationen, wie z.B. Krümmung, in die Komposition einzubringen, was auch in [Sigg et al., 2004] gezeigt wurde. Im Bereich der medizinischen Visualisierung wird die verfremdete Darstellung, *Non-Photorealistic Rendering*, oft zum Aufmerksamkeitslenken eingesetzt. Da Photographien von Organen und Eingriffen entweder nicht möglich sind oder eine zu hohe Selbstähnlichkeit auf Grund von umschließendem Gewebe oder Blutungen während der Operation aufweisen, ist eine reduzierte Darstellungsform nützlicher zur Erkennung von Formen und Beziehungen.

Neben dem Shading des Gewebes könnte auch das Shading der Dreiecksmodelle angepasst werden wie in [Preim et al., 2005] (vgl Abbildung 6.5 links).

Abbildung 6.5 zeigt darüber hinaus, wie kontextsensitiv verdeckte Objekte freigelegt werden können [Krüger et al., 2005]. SinusEndoscopy könnte auf ähnliche Weise segmentierte Modelle darstellen und auf Mauseingaben des Benutzers reagieren. Im Planungs- oder Lehreinsatz würden

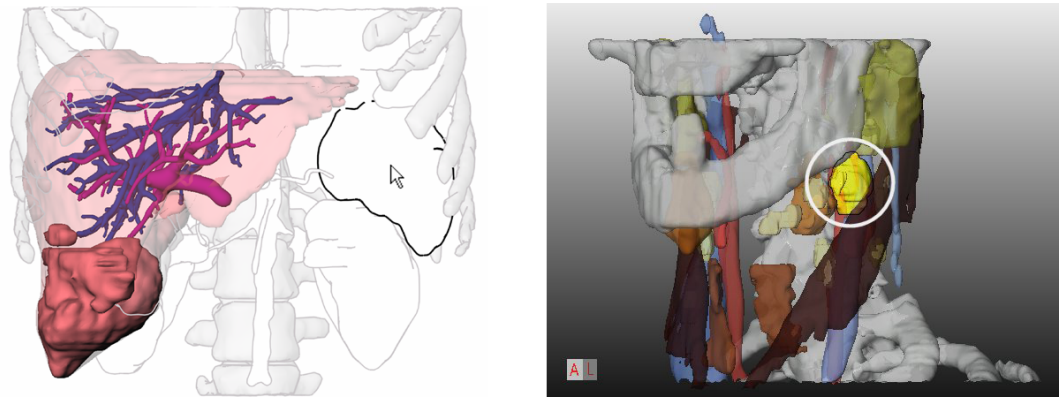

Abbildung 6.5: Visualisierungstechniken für medizinische Anwendungen. Der Einsatz verschiedener *Non-Photorealistic Rendering* Techniken (links) erlaubt die Lenkung von Aufmerksamkeit und Fokus auf das Wesentliche. Auch kontextsensitive Darstellungen, wie das Zeichnen von Objektsilhouetten (links) oder Aufhebung von Verdeckungen (rechts), sind möglich. Quellen: [Preim et al., 2005, Krüger et al., 2005]

zusätzliche Informationen, wie Größe eines Tumors oder Name einer ausgewählten Struktur, angegeben werden.
Die präferierten Renderstile und Farben müssen zunächst durch Tests ermittelt werden. Der Spielraum ist hier sehr groß, so dass es ohne direkte Diskussion mit den Endanwendern wenig sinnvoll erscheint, im Vorhinein Einschränkungen vorzunehmen.

6.2.2 Interaktion

Die Kamerahandhabung zu verbessern, ist ein Ziel von praktischen Anwendungstests. Eine geführte Navigation wie in STEPS oder anderen Systemen ist hier wahrscheinlich. Es gibt eine Reihe von Algorithmen [Bartz et al., 1999, Wan et al., 2001, Telea and Vilanova, 2003], die eine Skelettierung der Volumendaten vornehmen, um Führungslinien zu extrahieren. [Bartz et al., 1999] benutzen dabei nicht nur diese Führungslinien, sondern auch das Distanzfeld zum umgebenden Gewebe. [Stampe, 2006] benutzte diese Distanzfelder und ein 3D Eingabegerät mit Kraftrückkopplung, um neben dem visuellen Reiz auch eine haptische Erfahrung zu ermöglichen. Das Distanzfeld diente der Berechnung von Abstoßungskräften zum Gewebe und erzielte eine realistischere Kamerasteuerung. Die interaktive Transferfunktion von SinusEndoscopy erschwert eine Vorausberechnung des notwendigen Distanzfeldes, für einen noch leichteren Umgang mit der Kamera müsste aber ein derartiges System implementiert werden. Auch wäre es sinnvoll, Positionsangaben in Maßeinheiten einzuführen, da im Moment die Navigation noch komplett einheitenlos geschieht. Für die Planung von Eingriffen ist es aber wichtig, sich leicht Informationen über die Dimensionen der Anatomie aus den Daten beschaffen zu können. Ein spezielles Bemaßungswerkzeug wie in [Preim and Bartz, 2007, Kapitel 13.2] ausgeführt, wäre daher eine sinnvolle Erweiterung. [Preim et al., 2002] weisen auf die Benutzung von polygonalen Hilfsobjekten (vgl. Abbildung 6.6) in der 3D Szene hin, welche das Messen erleichtern.

Eine weitere wichtige Interaktion beim Arbeiten mit Volumendaten ist das Clipping. Da meist nicht der gesamte Volumendatensatz von Interesse ist, sondern nur Teilbereiche, kann der Benutzer die dargestellten Daten einschränken. SinusEndoscopy unterstützt im Moment die Daten beim Laden einzugrenzen, sowohl im Wertebereich, als auch im Volumen. Der Nutzer hat damit zur Zeit keine Handhabe, erst die Gesamtdaten zu sichten und dann bequem Regionen

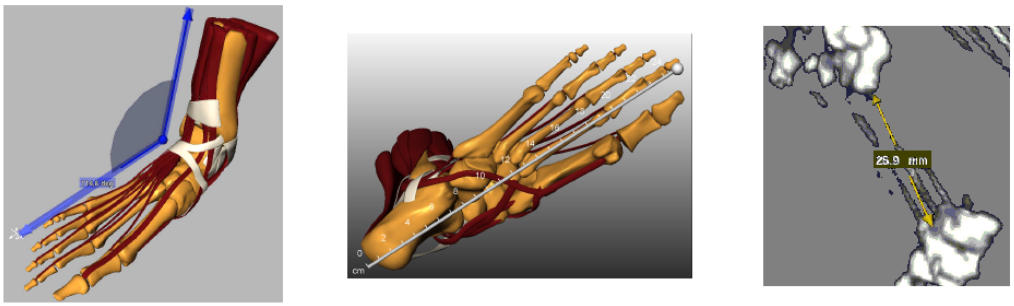

Abbildung 6.6: Verschiedene polygonale Bemaßungswerkzeuge im Einsatz. Quelle: [Preim et al., 2002]

auszuwählen, die von Interesse sind. Um Ressourcen zu sparen, könnte man zunächst bei Grafikkarten mit wenig Speicher, eine Verkleinerung vornehmen und nach einer manuellen Bestimmung des Interessensgebietes dieses nachladen.
Eine flexiblere Methode, die jedoch das komplette Volumen in den Speicher lädt, ist das interaktive Clipping. [Engel et al., 2006, Kapitel 15] widmet sich den Problemen, auf die hier zu achten ist. Meist kann der Benutzer über Ebenen das Volumen in sicht- oder unsichtbare Halbräume teilen. Durch die Evaluierung des Skalarprodukts eines Punktes im Volumen mit der Ebene in Normalenform kann mit einer GPU Instruktion evaluiert werden, ob der Punkt verworfen werden soll. Da die Anzahl dieser Ebenen durchaus variabel sein könnte, würde sich das vorher beschriebe System zur automatischen Quelltextgenerierung der Shader auch hier lohnen. Denn so könnten wirklich nur die Instruktionen eingebracht werden, die alle benutzten Ebenen repräsentieren. Hier ergibt sich aber eine weitere Problematik. Die Dreiecksmodelle können im Moment in Luxinia nur an einer Ebene geschnitten werden. Der Grund liegt in dem bereits angesprochenen Mangel von Standards bei OpenGL, was zu unterschiedlichem Clippingverhalten von ATI/AMD und NVIDIA Hardware führt. Alternativ kann dieses Clipping manuell durch GPU Programme auf Pixelebene erfolgen.

Wegen des geringen Leistungsabfalls großer Volumendaten, wie in Kapitel 4.4 erwiesen, und wegen der zunehmenden Speichergröße der Grafikkarten genügt das derzeitige System für die gängigen CT-Datengrößen. Der Arzt befindet sich zur virtuellen Endoskopie ohnehin innerhalb des Volumens, und die Einschränkung der Sichtweite, sowie das schnelle Auftreffen auf Gewebe, verhindern einen merklichen Leistungsabfall.

Die erwähnten Verbesserungen greifen auf bereits robuste Lösungen zurück, weswegen es weniger aufwändig ist, diese zu implementieren. Um optimale Bedienungsfreundlichkeit zu erreichen, liegt hier die Herausforderung mehr auf den genutzten Interaktionsfeinheiten, als auf der Entwicklung neuer Algorithmen und Technologien.

6.3 Weiterführende Techniken

Die fortschreitende Entwicklung der Computertechnik bringt mit neuen Hardwaregenerationen oft neue Lösungswege hervor. Im Bereich der Grafikkarten ist seit 2006 eine neue Generation von GPUs auf dem Markt, die einige Pipelineveränderungen mit sich bringen (siehe Abbildung 6.7). Durch den Geometrieshader ist es möglich, dynamisch neue Geometrie aus der vorhandenen zu erstellen. Daneben verfügen die Karten die Fähigkeit, in 3D Texturen zu zeichnen. Bisher ging dies nur für 2D und `CUBEMAP` Texturen. Daraus ergeben sich weiter Ideen, die auch für das Anwendungsgebiet dieser Arbeit interessant sind.

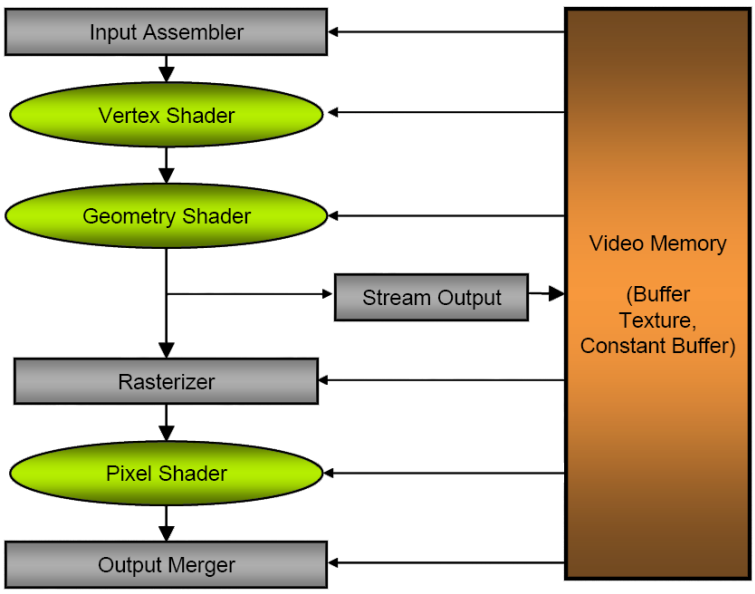

Abbildung 6.7: Erweiterung der Rendering Pipeline um den Geometrieshader. Quelle: [Tariq, 2006]

6.3.1 Realtime Deformation

Durch das Rendern in eine 3D Textur kann diese prozedural verändert werden. In [Tariq and Llamas, 2007] wurde diese Funktion genutzt, um physikalische Simulation mit der GPU durchzuführen. Hierbei kommt wiederum die hochgradig parallele Leistungsfähigkeit der GPU zum Tragen. Durch die Rasterung von Polygonalen Objekten in eine zusätzliche 3D Textur können die simulierten Stoffe mit Kollisionseffekten auf die Objekte reagieren.

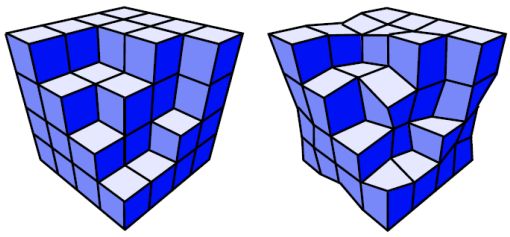

Abbildung 6.8: Freiformdeformation durch Translationsfelder. Quelle: [Engel et al., 2006]

Diese Technik könnte genutzt werden, um die Gewebeoberfläche pyhsikalisch zu simulieren. Durch eine zweite 3D Textur können Offsets für jeden Volumenpunkt erzeugt werden. Damit lässt sich eine Freiformdeformation, wie in [Engel et al., 2006, Seite 335] beschrieben und in Abbildung 6.8 angedeutet, erreichen. Neben dem Verschiebungsvektor, der in den RGB Werten kodiert ist, könnte der Alphawert zur Löschung genutzt werden. Damit ließe sich das Gewebe durch virtuelles Operationswerkzeug verändern. Allerdings ist hierfür die Rasterung der Kollisionsgeometrie problematisch und vielleicht durch die analytische Beschreibung einer Kapsel präziser und schneller.

6.3.2 Realtime Opening

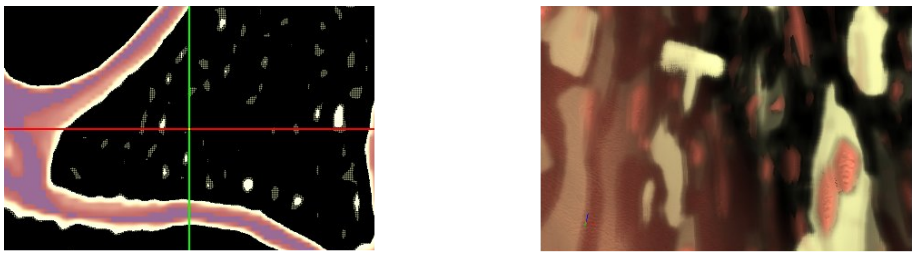

Abbildung 6.9: Freischwebende solide Objekte im Sekret.

Während der Entwicklung von SinusEndoscopy fiel beim Betrachten der Volumendaten vor allem auf, dass es bei gewissen Transfereinstellungen viele kleine freischwebende Objekte im Volumen gibt (siehe Abbildung 6.9). Meshbasierte Verfahren, die am Anfang der Arbeit erwähnt wurden, können die Daten komplexer aufbereiten und mit aufwändigerem Filtering auf solche Werte reagieren und sie gegebenenfalls entfernen. Es liegt im Ermessen des Mediziners, inwiefern diese Strukturen wichtig oder eher störend sind.

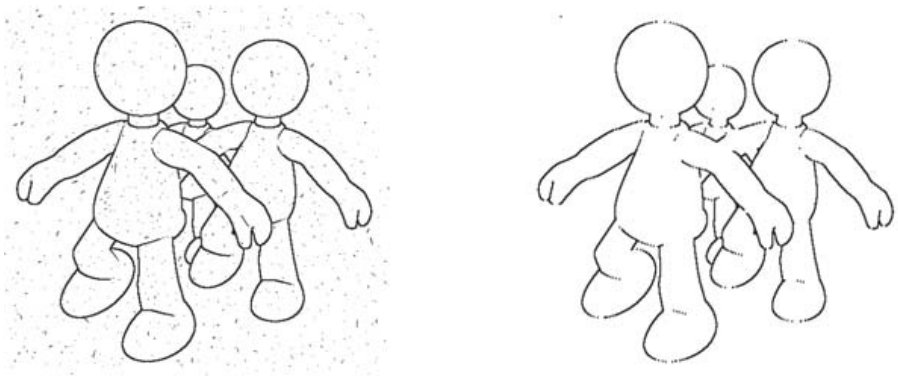

Abbildung 6.10: Morphologisches Opening zur Entfernung von *salt and pepper noise*. Quelle: [Mitchell, 2002]

In letzterem Fall kommen in der medizinischen Bildtechnik morphologische Operatoren zum Einsatz. Durch ihre Anwendung können Bilddaten bereinigt werden. Abbildung 6.10 zeigt die Daten vor und nach der Anwendung der *Opening* Operation. Wie zu sehen ist, lassen sich damit kleinere isolierte Strukturen sukzessiv entfernen. In der Bildverarbeitung spricht man von der Entfernung von *salt and pepper noise* durch die Kombination von *Erosion* und *Dilation*. Mit der Fähigkeit, in 3D Texturen zu zeichnen, könnte eine zweite Textur angelegt werden, die als Löschmaske für die Hauptdaten fungiert. Diese Maske müsste zunächst mit der Transferfunktion aus den Originaldaten erstellt und danach durch Morphologie verändert werden. Da nach wie vor nicht gleichzeitig von einer Texture gelesen und in sie geschrieben werden darf, bedarf es einer zweiten Löschungstextur, die im erwähnten *Ping-Pong Rendering* Verfahren zum Einsatz kommt. Der Morphologiekernel kann in eine 2D Textur kodiert und dann in der *uv* Ebene der Textur, sowie der *vw* Ebene angewendet werden. Eine technische Umsetzung für je eine Ebene ist in [Mitchell, 2002] zu finden. Die Operation ist im Vergleich zur Bildglättung nicht separabel und benötigt daher pro Pass zwei Schleifen entlang der betrachteten Achsen. Trotz der Leistungsfortschritte der Grafikkarten ist anzunehmen, dass dieser Prozess auf Grund der

vielen Schreib- und Leseoperationen von 3D Texturen nicht echtzeitfähig ist. Deswegen ist es sinnvoll, ihn erst dann auszuführen, wenn der Benutzer schon über längere Zeit nicht mehr die Transferfunktion geändert hat. Während der Änderungen würden man die Löschungstextur ignorieren, sonst könnte man sie entweder direkt zum Rendering solider Teile verwenden oder mit den Originaldaten verrechnen.

6.3.3 Realtime Marching Cube

Eine der ältesten Techniken zur Oberflächenerstellung aus Volumendaten kann auf neue Weise mit Hilfe des Geometrieshaders umgesetzt werden. Mit ihm lässt sich die bekannte Mesh Erstellung mittels *Marching Cube* [Lorensen and Cline, 1987] in Echtzeit durchführen. Eines der Probleme, die beim *Marching Cube* Algorithmus erwähnt wurde, war die große Menge an Dreiecken, die entsteht, was meist aufwändigere Reduktionsverfahren benötigt. Dieses Problem kann durch den Geometrieshader umgangen werden. Der Ablaufprozess hierfür ist in Abbildung 6.11

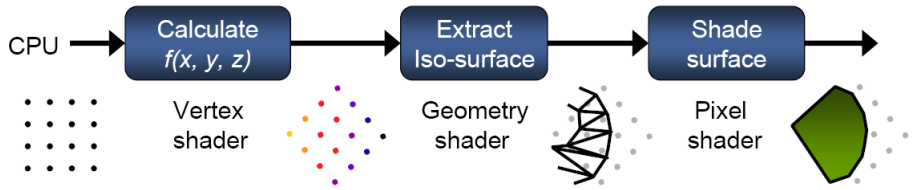

Abbildung 6.11: Ablaufprozess für Realtime Marching Cube. Quelle: [Tariq, 2006]

zu sehen.

In [Uralsky, 2006] wird der Geometrieshader dazu genutzt, um pro Punkt im Raum zu entscheiden, welcher der 256 Fälle des *Marching Cube* vorliegt, um die passende Ausgabegeometrie zu erstellen. Da der Geometrieshader auch auf 3D Texturen zugreifen kann, bleibt das Verfahren ähnlich flexibel wie das in dieser Arbeit vorgestellte, da die Transferfunktion jederzeit veränderbar ist. Der Aufwand des Verfahrens ist konstant, da immer das gleiche Netz im Sichtfeld mit der gleichen Anzahl an Volumenpunkten evaluiert wird. Über die Verteilung dieser Punkte im Sichtfeld kann zusätzlich noch eine sichtabhängige Dreiecksunterteilung realisiert werden, wenn mehr Punkte nahe der Kamera platziert sind. So erreicht man in der Nähe des Betrachters feinere Strukturen, als im Hintergrund. Damit der Algorithmus funktioniert, muss natürlich nach wie vor der gesamte Sichtbereich in ein uniformes Gitter zerlegt werden. Abbildung 6.12 zeigt die beschriebene Raumaufteilung.

Die Vorteile des Verfahrens sind hohe Flexibilität, keine Artefakte wie aus dem hier beschriebenen bildbasierten Ansatz, und durch die Ausgabe normaler Geometrie interagiert diese auch wie gewohnt durch den Z-Buffer auf bereits gezeichnete Geometrie.

Sollte die Aufteilung des Gitters so erfolgen, dass im Hintergrund die Zelldichte sehr viel geringer ist, so ist zu beachten, dass im Volumen nun ein Samplingpunkt ein großes Gebiet repräsentiert. Dies kann bei Bewegung zu starken Fluktuationen führen und damit sehr störend sein. Der Fragmentshader müsste abschließend auf die Dichtwerte vor und nach der Isooberfläche eingehen oder kann diese auch aus Leistungsgründen ignorieren. Aufbau und Auflösung des Eingabegitters beeinflussen die Leistung und Qualität der Ausgabe, weswegen erst konkrete Tests Rückschlüsse für dieses Anwendungsgebiet zulassen. Allerdings ist zu vermuten, dass das Verfahren für zukünftige „schwache Systeme“ bessere Ergebnisse liefert als füllratenintensive

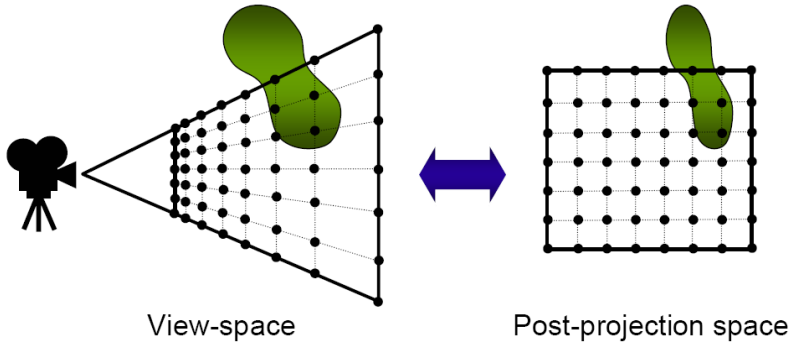

Abbildung 6.12: Gitteraufteilung für Marching Cube im Kameraraum. Nahe des Betrachters werden kleinere Abstände zwischen den Zellen erreicht, was einer höheren Auflösung entspricht. Quelle: [Tariq, 2006]

Verfahren, weil die Samplingpunkte vorher als Eingangspunkte in den Geometrieshader definiert wurden und keinerlei Schleifen im Fragmentshader notwendig sind. Diese Technik sollte aber wegen der Ähnlichkeiten, wie minimaler CPU Belastung und Verzicht von Aufbereitung der Volumendaten, in das vorhandene Framework integriert werden. Die Ausgabewerte wie Texturdetails, Tiefe und Normale könnten am Ende genauso durch die Kompositionsschritte des aktuellen Algorithmus laufen. Dies zeigt auch den Vorteil der Übertragbarkeit des hier entwickelten Systems, mit unterschiedlich gewonnenen Eingabedaten umzugehen.

Inwiefern diese neuen technologischen Möglichkeiten sich tatsächlich für die Anwendung eignen, und ob sie die gewünschten Vorteile erzielen würden, bedarf weiterer Forschung. Allerdings werden zunächst jene Verbesserungen bevorzugt, die ohne größeren Aufwand das System optimieren. Das System erfüllt bereits viele Ansprüche, was noch fehlt, sind Standardwerte von Parametereinstellungen und Anpassungen der Programmoberfläche für den alltäglichen klinischen Einsatz. Diese Verbesserungen werden zur Zeit wichtiger eingeschätzt, als weiterführende Technologien. Denn jene sind noch nicht so weit ausgereift wie die Basistechniken, welche bei SinusEndoscopy zum Einsatz kommen.

6.4 Klinische Anwendung

Während der Entwicklung fanden bereits Anwendungstests durch HNO-Chirurgen des Universitätsklinikums Leipzig statt, so dass auf diverse Wünsche und Anregungen eingegangen werden konnte. Zu Beginn der Arbeit wurde der erste Prototyp vorgeführt, der noch keine Textur- oder Beleuchtungseffekte besaß, und auch keine Schnittbildansichten. Dennoch fiel positiv auf, dass das System auf einem handelsüblichen Laptop genutzt werden konnte und die Steuerung der Kamera als intuitiv eingeschätzt wurde. War die Kamera allerdings weiter außerhalb des Volumens, so gab es Probleme bei der Steuerung. Die Mediziner waren beim Betrachten von Objekten, wie dem Schädel, eine objektzentrische Steuerung gewöhnt. Sie gingen davon aus, dass das Objekt gedreht und verschoben wird. Innerhalb der Volumendaten war die egozentrische Steuerung, welche Kamerarotation und -translation unmittelbar verändert, die bevorzugte Variante. Die derzeitige Version von SinusEndoscopy umgeht das Problem, in dem der Benutzer immer sofort im Volumen startet, und so die umgesetzte egozentrische Steuerung nie als störend

empfindet. Auch hat er jetzt die Möglichkeit, mit Hilfe der Schnittbilder zu navigieren. Auffällig war hierbei, dass die Maussensitivität je nach System unterschiedlich ausfiel, was für die Kamerabewegung oft hinderlich ist. Ein manueller Regelmechanismus, die Bewegung individuellen Wünschen anzupassen, ist demzufolge erforderlich.

Positiv dagegen wurden die vielen Einstellungsmöglichkeiten wahrgenommen. Die Chirurgen sind es bereits gewohnt, Darstellungen je nach persönlichen Vorlieben einzustellen. Bei der Transferfunktion musste es aber möglich sein, die Hounsfieldwerte präzise einzustellen. Woraufhin manuelle Eingaben für Werte im System ergänzt wurden, wie in Abschnitt 5.2.1 gezeigt. Ebenso wurde die Transferfunktion leicht verändert. Die erste Implementation bestand aus einem „Breite"- und „Zentrum"-Wert für den abgebildeten Wertebereich. Dies war aber nicht intuitiv nutzbar, denn wann immer einer der beiden Werte verändert wurde, veränderte sich auch der Wert, der für die Gewebeoberfläche verantwortlich war. Der Steuerungsmechanisums wurde so abgeändert, dass nicht das „Zentrum", sondern das Maximum, also eben genau jener Übergang von Sekret nach Gewebe, gesteuert wird. Auf diese Weise beeinflusst die Veränderung des Wertebereichs, der als Sekret klassifiziert wird, nicht mehr die Gewebeoberfläche und macht die Transferfunktion konsistent nutzbar.

Ein Problem, das die klinische Benutzung noch nicht direkt möglich macht, ist die fehlende Unterstützung DICOM-Dateien direkt zu laden. Auch fehlen in der Benutzeroberfläche noch Hinweise auf die Mess- und Patientendaten. Jene Daten über Volumengrößen könnten dazu genutzt werden, die Skalierung der Detailtextur einheitlich zu gestalten. Die Größe der Oberflächendetails und die Sichtweite sind im Moment noch abhängig von der Volumenauflösung und nicht der metrischen Ausdehnung. Dies kann zu Inkonsistenzen in der Darstellung verschiedener CT-Daten führen. Wie in Abschnitt 4.2.1 zu sehen war, variieren auch die Ergebnisse je nach Interpolationsverfahren beim Filtering der Volumendaten. Gerade wenn die Daten sehr anisotrop sind, ist das ein gewisser Unsicherheitsfaktor. Daher wäre es wünschenswert, diese Unsicherheit, die durch den starken Interpolationseinfluss entsteht, in Form einer Farb- oder Werteskala anzuzeigen. Denn nur bei relativ hochaufgelösten Volumendaten wird der Spielraum für durch Interpolation entstehende Werte geringer. Diesen Toleranzbereich zu finden, ab welchen Auflösungen und Ausdehnungsverhältnissen das System der Realität ausreichend nahe kommt, muss bei einer größeren Evaluierung ermittelt werden. Eine Vorführung mit anschließender Diskussion am ICCAS (Innovation center computer assisted surgery)[1] in Leipzig führte zu einer Verbesserung der Normalenerstellung durch den Hinweis auf die Interpolationsproblematik.

Neben diesen eher datenbezogenen Problemen war der Gesamteindruck bezüglich des Systems aber sehr gut. Bei der Darstellung wurde die erreichte Darstellungsqualität der Anatomie und die Interaktivität gelobt. Die vielen interaktiven Einstellungsmöglichkeiten der Renderingparameter fielen positiv auf, wenngleich eine Option vermisst wurde, diese Parameter zu speichern. Die Detailtextur, welche aus einem Photo von Lederstrukturen abgeleitet wurde, ist ebenfalls als realistisch eingeschätzt worden. Dazu gehören auch die Effekte wie Nässe und Sekretdarstellung. Das Zeichenwerkzeug, dessen grundsätzliche Idee aus Forschungsergebnissen am ICCAS hervorging, wurde für die Patientenaufklärung und Dokumentation als sehr nützlich bewertet. Im Vergleich zu anderen vom Universitätsklinikum getesteten Endoskopie-Systemen, ist auch die einfache und vor allem schnelle Benutzbarkeit von SinusEndoscopy ein Vorteil. Die Ladezeiten sind sehr gering und binnen Sekunden können die gemessenen CT-Daten ohne jegliche Vorverarbeitung oder Zwischenspeicherungen genutzt werden. Dies ist deshalb von hoher Bedeutung, weil dem Chirurg so weniger Zeit für störende technische Aufgaben oder Wartezeiten auferlegt werden, die nicht zu seinem eigentlichen Aufgabenfeld gehören.

[1] `http://www.iccas.de`

Die gesamte Planungsdauer dürfte nach Angaben der Chirurgen nicht mehr wie etwa zwei Minuten in Anspruch nehmen. Für den Patienten ist es vorteilhaft, dass ihm im Idealfall eine zusätzliche Endoskopie zur Diagnostik erspart werden kann. Ob dies so zu erreichen ist, werden erst klinische Studien zeigen. In jedem Fall ist das System ohne große Aufwendungen für Spezialhardware zur Exploration der Daten und zur Patientenaufklärung nutzbar.

Weitere Anwendungsszenarien im klinischen Einsatz oder die Übertragbarkeit auf andere anatomische Umgebungen, bei denen keine reale Endoskopie möglich ist (z.B. im Ohrbereich), müssen erst Evaluierungen im größeren Umfeld ergeben. Die Mediziner schätzten jedoch bereits im jetzigen Zustand der Software ihren praktischen Nutzen als gegeben ein. Daraus lässt sich schließen, dass die Ziele dieser Arbeit erreicht wurden und die entwickelte Software als Basis für Weiterentwicklungen dienen wird.

Literaturverzeichnis

[Ansari, 2004] Ansari, M. Y. (2004). Image Effects with DirectX 9 Pixel Shaders. In Engel, W. F., editor, *ShaderX squared*, chapter 4, pages 481–518. Wordware Publishing, Inc. Plano, Texas.

[Bartz, 2005] Bartz, D. (2005). Virtual Endoscopy in Research and Clinical Practice. *Computer Graphics Forum*, 24(1):1–17.

[Bartz et al., 1999] Bartz, D., Strasser, W., Skalej, M., and Welte, D. (1999). Interactive exploration of extra-and intracranial blood vessels (case study). In *VIS '99: Proceedings of the conference on Visualization '99*, pages 389–392, Los Alamitos, CA, USA. IEEE Computer Society Press.

[Becker, 1983] Becker, W. (1983). *Atlas der Hals-Nasen-Ohren-Krankheiten*. Thieme, Stuttgart, 1. Aufl.

[Cabral et al., 1994] Cabral, B., Cam, N., and Foran, J. (1994). Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *VVS '94: Proceedings of the 1994 symposium on Volume visualization*, pages 91–98, New York, NY, USA. ACM Press.

[Cullip and Neumann, 1994] Cullip, T. J. and Neumann, U. (1994). Accelerating Volume Reconstruction With 3D Texture Hardware. Technical report, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA.

[DiGioia et al., 1998] DiGioia, A. M., Colgan, B. D., and Koerbel, N. (1998). Computer-Aided Surgery. In Satava, R. M., editor, *Cybersurgery: Advanced Technologies for Surgical Practice*, volume Cybersurgery Advanced Technologies for Surgical Practice of *Protocols in General Surgery*, chapter 7, pages 121–139. Wiley-Liss.

[Engel, 2003] Engel, K. (2003). Advanced Volume Rendering Techniques. In *IEEE Visualization 2003 Tutorial: Interactive Visualization of Volumetric Data on Consumer PC Hardware*.

[Engel et al., 2006] Engel, K., Hadwiger, M., Kiss, J. M., Rezk-Salama, C., and Weiskopf, D. (2006). *Real-Time Volume Graphics*. A K Peters, Ltd., Wellesley, Massachusettes.

[Everitt, 2001] Everitt, C. (2001). Projective Texture Mapping. Technical report, NVIDIA Corporation. `http://www.nps.navy.mil/cs/sullivan/MV4470/resources/projective_texture_mapping.pdf` (Stand 11.2007).

[Everitt et al., 2001] Everitt, C., Rege, A., and Cebenoyan, C. (2001). Hardware Shadow Mapping. Technical report, NVIDIA Corporation. `http://www1.cs.columbia.edu/~ravir/6160/papers/shadow_mapping.pdf` (Stand 11.2007).

[Faller et al., 2004] Faller, A., Schünke, M., and Schünke, G. (2004). *Der Körper des Menschen. Einführung in Bau und Funktion.* Thieme, Stuttgart, 14. Aufl.

[Geiss and Thompson, 2006] Geiss, R. and Thompson, M. (2006). Cascades. Technical report, NVIDIA Corporation. `http://developer.download.nvidia.com/presentations/2007/gdc/CascadesDemoSecrets.zip` (Stand 11.2007).

[Hadwiger, 2004] Hadwiger, M. (2004). *High-Quality Visualization and Filtering of Textures and Segmented Volume Data on Consumer Graphics Hardware.* PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria.

[Hadwiger et al., 2005] Hadwiger, M., Sigg, C., Scharsach, H., Bühler, K., and Gross, M. (2005). Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces. In *Proceedings of Eurographics 2005*, pages 303–312. The Eurographics Association and Blackwell Publishing.

[Harris, 2004] Harris, M. (2004). Deferred Shading. In *6800 Leagues under the Sea*. NVIDIA. `http://download.nvidia.com/developer/presentations/2004/6800_Leagues/6800_Leagues_Deferred_Shading.pdf` (Stand 5.2007).

[Harris, 2006] Harris, M. (2006). GPGPU Lessons Learned. In *Game Developer Conference 2006*. NVIDIA. `http://developer.nvidia.com/object/opengl-gpgpu-gdc-2006.html` (Stand 5.2007).

[Interrante, 1997] Interrante, V. L. (1997). Illustrating Surface Shape in Volume Data via Principal Direction-Driven 3D Line Integral Convolution. *Computer Graphics*, 31(Annual Conference Series):109–116.

[Kabayim, 2007] Kabayim (2007). `http://kabayim.com/virt_labs_radiology.htm` (Stand 12.2007).

[Keller and Heidrich, 2001] Keller, A. and Heidrich, W. (2001). Interleaved Sampling. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 269–276. Springer-Verlag.

[Kniss, 2003] Kniss, J. (2003). Transfer Functions: Classification. In *IEEE Visualization 2003 Tutorial: Interactive Visualization of Volumetric Data on Consumer PC Hardware.*

[Koloszár and Jae-Young, 2003] Koloszár, J. and Jae-Young, Y. (2003). Accelerating Virtual Endoscopy. In *Winter School of Computer Graphics, University of West Bohemia*, volume 11.

[Kratz, 2006] Kratz, A. (2006). Advanced Illumination Techniques for GPU-Based Direct Volume Rendering. Master's thesis, Institute for Computer Science, University of Koblenz - Landau.

[Krüger et al., 2007] Krüger, A., Stampe, K., Hertel, I., Strauss, G., and Preim, B. (2007). Virtuelle Endoskopie mit Force Feedback-Unterstützung für die Operationsplanung an Nasennebenhöhlen. In Schulze, T., Preim, B., and Schumann, H., editors, *Simulation und Visualisierung 2007*, pages 341–345, Magdeburg. SCS Publishing House e.V.

[Krüger et al., 2005] Krüger, A., Tietjen, C., Hintze, J., Preim, B., Hertel, I., and Strauß, G. (2005). Interactive Visualization for Neck-Dissection Planning. In *EuroVis*, pages 295–302.

[Krüger and Westermann, 2003] Krüger, J. and Westermann, R. (2003). Acceleration Techniques for GPU-based Volume Rendering. In *Proceedings IEEE Visualization 2003*.

[Kubisch, 2007] Kubisch, C. (2007). Effekte für die Echtzeitgrafik mit Luxinia. Bachelor's thesis, Otto-von-Guericke University of Magdeburg.

[LaMar et al., 1999] LaMar, E., Hamann, B., and Joy, K. I. (1999). Multiresolution techniques for interactive texture-based volume visualization. In *VIS '99: Proceedings of the conference on Visualization '99*.

[Lehmann et al., 2005] Lehmann, T. M., Hiltner, J., and Handels, H. (2005). Medizinische Bildverarbeitung. In Lehmann, T. M., editor, *Handbuch der Medizinischen Informatik*, chapter 10, pages 361–423. Carl Hanser Verlag München.

[Li et al., 2003] Li, W., Mueller, K., and Kaufman, A. (2003). Empty Space Skipping and Occlusion Clipping for Texture-based Volume Rendering. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 42, Washington, DC, USA. IEEE Computer Society.

[Link et al., 2006] Link, F., Koenig, M., and Peitgen, H.-O. (2006). Multi-Resolution Volume Rendering with per Objects Shading. In *Vision Modelling and Visualization*, pages 185–191. Aka GmbH Berlin.

[Lorensen and Cline, 1987] Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, New York, NY, USA. ACM Press.

[Manegold and Groitl, 1998] Manegold, B.-C. and Groitl, H. (1998). Chirurgisch Endoskopie - Diagnostik und Therapie. In Lippert, H., editor, *Praxis der Chirurgie Allgemein- und Viszeralchirurgie*, chapter 3, pages 38–70. Georg Thieme Verlag.

[Mantke, 1998] Mantke, R. (1998). Sonographie. In Lippert, H., editor, *Praxis der Chirurgie Allgemein- und Viszeralchirurgie*, chapter 2, pages 29–37. Georg Thieme Verlag.

[Mitchell et al., 2006] Mitchell, J., McTaggart, G., and Green, C. (2006). Shading in Valve's Source Engine. In *SIGGRAPH '06: Advanced Real-Time Renderin ing 3D graphics and games courses*, pages 129–142, New York, NY, USA. ACM.

[Mitchell, 2002] Mitchell, J. L. (2002). Image Processing with 1.4 Pixel Shaders in Direct3D. In Engel, W. F., editor, *Direct3D ShaderX Vertex and Pixel Shader Tips and Tricks*, chapter 3, pages 258–269. Wordware Publishing, Inc.

[Mittring, 2007] Mittring, M. (2007). Finding next gen: CryEngine 2. In *SIGGRAPH '07: Advanced Real-Time Renderin ing 3D graphics and games courses*, pages 97–121, New York, NY, USA. ACM.

[Neubauer, 2005] Neubauer, A. (2005). *Virtual Endoscopy for Preoperative Planning and Training of Endonasal Transsphenoidal Pituitary Surgery*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology.

[Neubauer et al., 2002a] Neubauer, A., Mroz, L., Hauser, H., and Wegenkittl, R. (2002a). Cell-based first-hit ray casting. In *VISSYM '02: Proceedings of the symposium on Data Visualisation 2002*, pages 77–86, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

[Neubauer et al., 2002b] Neubauer, A., Mroz, L., Hauser, H., and Wegenkittl, R. (2002b). Fast and Flexible Iso-Surfacing for Virtual Endoscopy. In *Proceedings CESCG 2002*.

[Neubauer et al., 2004] Neubauer, A., Wolfsberger, S., Forster, M.-T., Mroz, L., Wegenkittl, R., and Bühler, K. (2004). STEPS - An Application for Simulation of Transsphenoidal Endonasal Pituitary Surgery. In *IEEE Visualization 2004*, pages 513–520.

[Oeken et al., 1993] Oeken, F. W., Plath, P., and Federspil, P. (1993). *Hals-Nasen-Ohren-Heilkunde*. Ullstein Mosby, 7. Aufl.

[Parker et al., 1999] Parker, S., Parker, M., Livnat, Y., Sloan, P.-P., Hansen, C., and Shirley, P. (1999). Interactive Ray Tracing for Volume Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):238–250.

[Perlin, 1985] Perlin, K. (1985). An image synthesizer. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 287–296, New York, NY, USA. ACM.

[Preim and Bartz, 2007] Preim, B. and Bartz, D. (2007). *Visualization in Medicine Theory, Algorithms, and Applications*. Morgan Kaufmann.

[Preim et al., 2005] Preim, B., Tietjen, C., and Doerge, C. (2005). NPR, Focussing and Emphasis in Medical Visualizations. In *Simulation und Visualisierung 2005*, pages 139–152. SCS.

[Preim et al., 2002] Preim, B., Tietjen, C., Spindler, W., and Peitgen, H.-O. (2002). Integration of Measurement Tools in Medical Visualizations. In *IEEE Visualization (27. Oktober - 2. November 2002, Boston*, pages 21–28.

[Raab and Hau, 1998] Raab, K. and Hau, T. (1998). Roentgen, Computertomographie, Szintigraphie, MRT. In Lippert, H., editor, *Praxis der Chirurgie Allgemein- und Viszeralchirurgie*, chapter 2, pages 10–29. Georg Thieme Verlag.

[Rezk-Salama, 2002] Rezk-Salama, C. (2002). *Volume Rendering Techniques for General Purpose Graphics Hardware*. PhD thesis, Technical Faculty, University of Nuremberg.

[Rogalla, 2001] Rogalla, P. (2001). Virtual Endoscopy of the Nose and Paranasal Sinuses. In Baert, A., Sartor, K., and Yourker, J., editors, *Virtual Endoscopy and Related 3D Techniques*, chapter 2, pages 17–37. Springer.

[Saito and Takahashi, 1990] Saito, T. and Takahashi, T. (1990). Comprehensible rendering of 3-D shapes. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 197–206. ACM.

[Satava and Jones, 1998] Satava, R. M. and Jones, C. S. B. (1998). Virtual Reality. In Satava, R. M., editor, *Cybersurgery: Advanced Technologies for Surgical Practice*, chapter 5, pages 75–96. Wiley-Liss.

[Scharsach, 2005] Scharsach, H. (2005). Advanced GPU Raycasting. In *Central European Seminar on Computer Graphics 2005*, pages 69–76.

[Scharsach et al., 2006] Scharsach, H., Hadwiger, M., Neubauer, A., Wolfsberger, S., and Bühler, K. (2006). Perspective Isosurface and Direct Volume Rendering for Virtual Endoscopy Applications. In *Proceedings of Eurovis/IEEE-VGTC Symposium on Visualization*.

[Sigg et al., 2004] Sigg, C., Hadwiger, M., Gross, M., and Bühler, K. (2004). Real-Time High-Quality Rendering of Isosurfaces. Technical report, VRVis-Wien.

[Simmen and Jones, 2005] Simmen, D. and Jones, N. (2005). *Chirurgie der Nasennebenhöhlen und der vorderen Schädelbasis*. Thieme, Stuttgart [u.a.], 1. Aufl.

[Sousa, 2007] Sousa, T. (2007). Vegetation Procedural Animation and Shading in Crysis. In Nguyen, H., editor, *GPU Gems 3*, chapter 16, pages 373–385. Addison-Wesley.

[Stampe, 2006] Stampe, K. (2006). Haptische Interaktion zur Planung von Nasennebenhöhlen-Operationen. Master's thesis, Otto-von-Guericke-Universität Magdeburg.

[Stegmaier et al., 2005] Stegmaier, S., Strengert, M., Klein, T., and Ertl, T. (2005). A Simple and Flexible Volume Rendering Framework for Graphics-Hardware based Raycasting. In *InternationalWorkshop on Volume Graphics '05*, pages 187–195.

[Strengert et al., 2006] Strengert, M., Klein, T., Botchen, R., Stegmaier, S., Chen, M., and Ertl, T. (2006). Spectral Volume Rendering using GPU-based Raycasting. *The Visual Computer*, 22(8).

[Tariq, 2006] Tariq, S. (2006). DirectX 10 Effects. In *SIGGRAPH 06*. NVIDIA.

[Tariq and Llamas, 2007] Tariq, S. and Llamas, I. (2007). Real-Time Volumetric Smoke using D3D10. In *Game Developers Conference*. NVIDIA. `http://developer.download.nvidia.com/presentations/2007/gdc/RealTimeFluids.pdf` (Stand 8.2007).

[Telea and Vilanova, 2003] Telea, A. and Vilanova, A. (2003). A robust level-set algorithm for centerline extraction. In *VISSYM '03: Proceedings of the symposium on Data visualisation 2003*, pages 185–194, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

[Termeer et al., 2006] Termeer, M., Bescós, J. O., and Telea, A. (2006). Preserving Sharp Edges with Volume Clipping. In *Proceedings Vision, Modeling and Visualization 2006*.

[Thibieroz, 2004] Thibieroz, N. (2004). Deferred Shading with Multiple Render Targets. In Engel, W. F., editor, *ShaderX squared*, chapter 2, pages 251–269. Wordware Publishing, Inc. Plano, Texas.

[Uralsky, 2006] Uralsky, Y. (2006). Practical Metaballsand Implicit Surfaces. In *Game Developer Conference*. NVIDIA. `http://download.nvidia.com/developer/presentations/2006/gdc/2006-GDC-DX10-Practical-Metaballs.pdf` (Stand 8.2007).

[Valient, 2007] Valient, M. (2007). Deferred Rendering in Killzone 2. In *Develop Conference, Brighton*. GUERRILLA. `http://www.develop-conference.com/developconference/downloads/vwsection/Deferred%20Rendering%20in%20Killzone.pdf` (Stand 8.2007).

[Vilanova et al., 2000] Vilanova, A., Hegedüs, B., Gröller, E. M., Wagner, D., Wegenkittl, R., and Freund, M. C. (2000). Mastering interactive virtual Bronchioscopy on a Low—end PC. In *VIS '00: Proceedings of the conference on Visualization '00*, pages 461–464, Los Alamitos, CA, USA. IEEE Computer Society Press.

[Vollrath et al., 2005] Vollrath, J. E., Weiskopf, D., and Ertl, T. (2005). A Generic Software Framework for the GPU Volume Rendering Pipeline. Technical report, Visualization and Interactive Systems Group (VIS), University of Stuttgart.

[Wald et al., 2004] Wald, I., Dietrich, A., and Slusallek, P. (2004). An Interactive Out-of-Core Rendering Framework for Visualizing Massively Complex Models. In *Rendering Techniques 2004, Proceedings of the Eurographics Symposium on Rendering*.

[Wald et al., 2005] Wald, I., Friedrich, H., Marmitt, G., Slusallek, P., and Seidel, H.-P. (2005). Faster Isosurface Ray Tracing using Implicit KD-Trees. In *IEEE Transactions on Visualization and Computer Graphics*.

[Wan et al., 2001] Wan, M., Dachille, F., and Kaufman, A. (2001). Distance-field based skeletons for virtual navigation. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 239–246, Washington, DC, USA. IEEE Computer Society.

[Williams, 1978] Williams, L. (1978). Casting curved shadows on curved surfaces. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 270–274, New York, NY, USA. ACM.

[Winnemöller and Bangay, 2002] Winnemöller, H. and Bangay, S. (2002). Geometric Approximations Towards Free Specular Comic Shading. In *EUROGRAPHICS 2002*, volume 21.

[Wolfsberger et al., 2004] Wolfsberger, S., Forster, M.-T., Donat, M., Neubauer, A., Bühler, K., Wegenkittl, R., Hainfellner, J., and Knosp, E. (2004). Virtual Endoscopy is a Useful Device for Training and Preoperative Planning of Transsphenoidal Endoscopic Pituitary Surgery. *Minimally Invasive Neurosurgery*, 47:214–220.

[You et al., 1997] You, S., Hong, L., Wan, M., Junyaprasert, K., Kaufman, A., Muraki, S., Zhou, Y., Wax, M., and Liang, Z. (1997). Interactive volume rendering for virtual colonoscopy. In *VIS '97: Proceedings of the 8th conference on Visualization '97*, pages 433–ff., Los Alamitos, CA, USA. IEEE Computer Society Press.

[Zylka-Menhorn and Koch, 1996] Zylka-Menhorn, V. and Koch, K. (1996). *Endoskopische Operationen. Weniger Schmerzen, schneller gesund.* Springer, Berlin [u.a.], 1. Aufl.

Selbstständigkeitserklärung

Hiermit versichere ich, Christoph Kubisch (Matrikel-Nr. 165040), die vorliegende Arbeit allein und nur unter Verwendung der angegebenen Quellen angefertigt zu haben.

Christoph Kubisch, Dezember 2007