Ein System zur Verarbeitung und Visualisierung von Voxeldaten

Inaugural-Dissertation

zur
Erlangung des Doktorgrades der
Mathematisch-Naturwissenschaftlichen Fakultät
der Heinrich-Heine-Universität Düsseldorf

vorgelegt von

Andreas Beck

aus Naila

Düsseldorf

2003

Gedruckt mit der Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Heinrich-Heine-Universität Düsseldorf

1. Referent: Prof. Dr. V. Aurich

2. Referent: Prof. Dr. E. Wanke

3. Referent: Priv. Doz. Dr. T. Hackländer

Tag der mündlichen Prüfung: 11.06.2003

Inhaltsverzeichnis

1 Einleitung **1**

2 Anforderungen an 3D-Visualisierungssysteme **3**

2.1 3D-Datenquellen 4

2.1.1 Oberflächenbasierte Verfahren 4

2.1.2 Destruktive Verfahren 4

2.1.3 Tomographieverfahren 5

2.2 Das herkömmliche Verfahren 6

2.2.1 Schichtweise Auswertung auf Film 6

2.2.2 Visualisierung der Schichten an Bildschirmarbeitsplätzen . . . 6

2.2.3 Darstellung anderer Schnittebenen 8

2.2.4 3D-Visualisierungssysteme 8

2.3 Anforderungen an ein neues System 9

2.3.1 Auflösung verfügbarer Bildschirme 9

2.3.2 Speicherbedarf 11

2.3.3 Gewünschte Darstellungsarten 12

2.3.4 Vorverarbeitungsschritte und ihr Aufwand 14

2.3.5 Bedienung 15

2.3.6 Orientierungshilfen für den Betrachter 17

2.3.7 Diagnostikmöglichkeit nach anerkannten Verfahren 18

2.3.8 Interaktive Bildverarbeitung 19

2.3.9 Weitgehende Automatisierung 21

2.3.10 Navigationshilfen 22

2.3.11 Virtuelle Bildgebung für neue An- und Einsichten 23

2.3.12 Computergestützte Diagnose (CAD) 24
2.3.13 Integration in bestehende Umgebungen 25

3 Konzeption des Gesamtsystems 29
3.1 Teilprogramme 29
3.1.1 VOXREN 30
3.1.2 VOXCONTROL 31
3.1.3 PLANEVIEW 32
3.1.4 BATCHVOX 34
3.1.5 Die graphische Shell 35
3.2 Speicherbedarf und Datenlayout 36
3.2.1 Voxelklassen 36
3.2.2 Geometrische Distanzabbildung 36
3.2.3 Oberflächennormale 37
3.2.4 Doppelnutzung des Grauwertbereiches 37
3.2.5 Grauwertklassen 38
3.3 Import von Daten 38
3.3.1 Medizinische Daten 39
3.3.2 Import proprietärer Formate 41
3.3.3 Transparente (De-)Kompression 41
3.4 Dokumentation 42
3.4.1 Aufzeichnen von Positionsdaten 42
3.4.2 Abspeichern von Einzelbildern 43
3.4.3 Aufzeichnen von Videosequenzen 44
3.5 Modularisierung 44
3.5.1 Exportmodule 44
3.5.2 Bedien- und Verarbeitungsmodule 46
3.6 Scripting 46
3.6.1 Beispiel 47
3.6.2 Umfang und Bedeutung der Scriptschnittstelle 47
3.6.3 Interner Einsatz der Scriptschnittstelle 48
3.7 Entwicklungsrichtlinien 49

3.7.1 Vermeidung von Code-Redundanz 49
3.7.2 Bedienung und Benutzeroberfläche 49
3.7.3 Wahl der Programmiersprache und Codierungsrichtlinien . . . 50
3.8 Vergleich mit anderen Visualisierungssystemen 51
3.8.1 Zielsetzung 52
3.8.2 Darstellungsqualität 52
3.8.3 Bedienbarkeit 53
3.8.4 Integration in das Arbeitsumfeld 57
3.8.5 Anwendungen 58

4 Visualisierungsalgorithmen 59
4.1 2D-Visualisierung 59
4.1.1 3-Ebenen-Darstellung 59
4.1.2 Konsistente Darstellung bei Benutzung des Zooms 61
4.1.3 Kopplung mit VOXREN 62
4.1.4 Darstellung von Overlays 62
4.1.5 Gamma-Korrektur 63
4.2 3D-Visualisierung 64
4.2.1 Raycasting 65
4.2.2 Oberflächendarstellung (Shading) 74
4.2.3 Raycasting mit Antialiasing in PLANEVIEW 84
4.2.4 Interpolation der Bilder 92
4.2.5 Erzeugung von stereographischen Ansichten 95
4.2.6 Alternative Tiefendarstellung mit Nebel-/Plasmaeffekten . . . 99

5 Vorverarbeitungsalgorithmen 101
5.1 Filterung 102
5.1.1 Speicherplatz 102
5.1.2 Einschränkung des zu filternden Volumens 103
5.2 Vorberechnung der Oberflächennormalen 103
5.2.1 Einfache integrierte Schätzung 103
5.2.2 Erweiterte Verfahren 104

5.3 Geometrische Distanzabbildung 105
5.3.1 Effiziente Berechnung 105
5.3.2 Parallele Betrachtung 106
5.3.3 Parallelisierung 107

6 Segmentierungsalgorithmen 109
6.1 Einfache Schwellwertentscheidung 109
6.2 Mehrstufen–Füllalgorithmus 110
6.2.1 Problemstellung 110
6.2.2 Algorithmus 111
6.2.3 Parameterwahl 114
6.2.4 Optimierung 115
6.2.5 Hintergrund zur Funktionsweise 115
6.2.6 Anwendungen 116
6.3 Dilation 117
6.3.1 Optimierte Implementation 117
6.4 Automatische Segmentierung des Darmlumens 119
6.4.1 Algorithmus 119
6.4.2 Bemerkungen 123
6.4.3 Nachbearbeitung 124
6.4.4 Bewertung und Verbesserungsmöglichkeiten 125
6.4.5 Entwicklung des Algorithmus 125
6.5 Normalenbasierte Segmentierung 125
6.6 Manuell unterstützte Segmentierung 127
6.6.1 Umfärben von Zusammenhangskomponenten 127
6.6.2 Finden von Verzweigungen 128
6.6.3 Der Hüllenoperator 134
6.7 3D-Werkzeuge und virtuelle Chirurgie 139
6.7.1 Mausgesteuertes Skalpell 139
6.7.2 Ebene, zusammenhängende Schnitte 142
6.7.3 Ummarkierung aller sichtbaren Voxel 145

7 CAD-Algorithmen **147**

7.1 Polypenfinder 147

7.1.1 Dosenwandverfahren 148

7.1.2 Inselverfahren 150

7.2 Erkennen von Artefakten 153

7.3 Marker 154

7.3.1 Anwendung 154

7.3.2 Implementation 154

7.4 Längenmessungen 155

7.5 Markierung des Abstandes „ab ano“ 156

7.6 2D-Suchlauf 157

8 Navigationsalgorithmen **159**

8.1 Autopilot 159

8.1.1 Voraussetzung 159

8.1.2 Problemstellung 159

8.1.3 Lösungsidee 160

8.1.4 Algorithmus 160

8.1.5 Bemerkungen 164

8.1.6 Vergleich mit anderen Verfahren 164

8.1.7 Typische Anwendungen 165

8.2 Pfadgenerierung durch Interpolation 166

8.2.1 Voraussetzungen 166

8.2.2 Problem 166

8.2.3 Erzeugung von kubischen Splines 167

8.2.4 Algorithmus 169

8.2.5 Bemerkungen 173

9 Anwendungen **175**

9.1 3D-Scanner zur Erzeugung von Reliefs 175

9.2 Anwendung in der Medizin 179

9.2.1 Relevanz 179

9.2.2 Einsatzorte . . . 180
9.2.3 Resonanz und Lernkurve . . . 181
9.2.4 Probleme . . . 181
9.3 Forschungsergebnisse . . . 182
9.3.1 CT-Colonographie . . . 182
9.3.2 Laufende Forschung . . . 184

10 Ausblick **187**
10.1 Technische Weiterentwicklung . . . 187
10.1.1 Verbesserte Rendergeschwindigkeit . . . 187
10.1.2 Integration von VOXREN und PLANEVIEW . . . 188
10.1.3 Modulare Renderer . . . 188
10.1.4 Neue Darstellungsmodi . . . 188
10.1.5 Mengenoperationen . . . 190
10.1.6 Entwicklung einer eigenen GUI-Komponente . . . 190
10.2 Verbesserung des Benutzerinterfaces . . . 191
10.2.1 Verbesserter Einstieg für neue Benutzer . . . 191
10.2.2 Erstellung von Spezialanwendungen . . . 191
10.2.3 Konsistente Bedienung . . . 191
10.3 Ausweitung der Anwendungsbereiche . . . 192

A Kommandoreferenz zu VOXREN **193**
A.1 Interne Kommandos . . . 193
A.1.1 HELP . . . 193
A.1.2 BCAST . . . 194
A.1.3 RECORD . . . 195
A.1.4 RECORDMODE . . . 196
A.1.5 REALTIMERECORD . . . 196
A.1.6 COLORIZE . . . 197
A.1.7 AUTO . . . 198
A.1.8 PLAY . . . 199
A.1.9 PLAYDIR . . . 200

A.1.10 LOADCLUT 201
A.1.11 MARK 202
A.1.12 MARKD 203
A.1.13 MARKZ 204
A.1.14 MARKZD 205
A.1.15 MARKE 205
A.1.16 DESTROYGREY 206
A.1.17 DENOISE 207
A.1.18 DILATE 208
A.1.19 CUTVOLUME 209
A.1.20 BFS 210
A.1.21 NORM 211
A.1.22 MAKEDIST 213
A.1.23 QUITPROGRAM 214
A.1.24 REPEAT 214
A.1.25 DISP 215
A.1.26 LOAD 216
A.1.27 MERGE 217
A.1.28 SAVE 218
A.1.29 SAVE16 219
A.1.30 ECHO 220
A.1.31 SLEEP 221
A.1.32 PLANETRANS 221
A.2 Modul: Multivolume 222
A.3 Modul: Marker 222
A.3.1 MARKERADD 222
A.3.2 MARKERDEL 223
A.3.3 MARKERSHOW 224
A.4 Modul: multiview 225
A.4.1 SETVIEW 225
A.4.2 ADDVIEW 226

A.4.3 ADDSLAVEVIEW 227
A.4.4 DELVIEW 228
A.5 Modul: surgery 229
A.5.1 FILLSURF 229
A.5.2 KILL_TOP 230
A.5.3 CUTSIZE 231
A.6 Modul: parms 231
A.6.1 JUMP 231
A.6.2 MAXDEPTH 232
A.6.3 ZOOM 233
A.6.4 LIGHT 234
A.6.5 LIGHTMODE 235
A.6.6 GLOBALOPACITY 237
A.6.7 RENDERMODE 238
A.6.8 FINE 239
A.6.9 ALIAS 240
A.6.10 FLAGS 241
A.6.11 BGCOLOR 243
A.6.12 SIZE 244
A.6.13 SIZEX 245
A.6.14 SIZEY 246
A.6.15 SCALE 247
A.6.16 ASPECT 248
A.6.17 LAYER 248
A.6.18 GCENTER 250
A.6.19 GWIDTH 251
A.6.20 GAMMA 252
A.6.21 GREYMARK 253
A.6.22 CUTPLANEGFC 254
A.6.23 CUTPLANESET 255
A.7 Modul: stereo 256

A.7.1 EYE . . . 256
A.7.2 EYEDEG . . . 257
A.8 Modul: nlg . . . 258
A.8.1 FILTER . . . 258
A.8.2 NLGS . . . 259
A.8.3 SAVEORI . . . 260
A.8.4 MIXORI . . . 261
A.8.5 DESTROYORI . . . 262
A.9 Modul: move . . . 263
A.9.1 STEP . . . 263
A.9.2 FWD . . . 264
A.9.3 REV . . . 265
A.9.4 RIGHT . . . 266
A.9.5 LEFT . . . 267
A.9.6 UP . . . 268
A.9.7 DOWN . . . 268
A.9.8 TMODE . . . 269
A.9.9 TPOINT . . . 270
A.9.10 TRIGHT . . . 271
A.9.11 TLEFT . . . 272
A.9.12 TUP . . . 273
A.9.13 TDOWN . . . 274
A.9.14 RLEFT . . . 275
A.9.15 RRIGHT . . . 276
A.9.16 UTURN . . . 276
A.9.17 CTRIGHT . . . 277
A.9.18 CTLEFT . . . 278
A.9.19 CTUP . . . 279
A.9.20 CTDOWN . . . 280
A.9.21 POS . . . 281
A.9.22 CPOS . . . 282

A.9.23 DIR 282
A.10 Modul: remark 283
A.10.1 FLIPSIDES 284
A.10.2 REMARK 285
A.10.3 HULL 285
A.10.4 MKTRANSP 286
A.10.5 FINDFORK 287
A.10.6 FINDFORK2 288
A.10.7 FINDFORK3 289
A.10.8 KILLSIDE 290
A.10.9 REMARKNORM 291
A.10.10CLASSMAX 292
A.11 Modul: polypen 293
A.11.1 MINDIST 293
A.11.2 MARKPOLY 294
A.11.3 OLDMARKPOLY 295

B Tastenreferenz zu VOXREN 297
B.1 Interne Kommandos 298
B.2 Modul: marker 299
B.3 Modul: multiview 299
B.4 Modul: surgery 299
B.5 Modul: parms 300
B.6 Modul: stereo 301
B.7 Modul: move 302
B.8 Modul: remark 303

C Mausreferenz zu VOXREN 305
C.1 Modul: marker 305
C.2 Modul: surgery 305
C.3 Modul: move 306

D Schnittstellen 307

D.1 Dateiformate 307
D.1.1 P5-Stapel 307
D.1.2 16HL-Stapel 308
D.1.3 3d32-Volumendateien 309
D.2 Relevante Datenstrukturen 311
D.2.1 Zugriff auf die Volumendaten 311
D.2.2 Zusatzdaten zu Volumina - volume_metadata.h 316
D.2.3 Virtuelle Kameras - view.h 317
D.3 Die Schnittstelle für Kommandomodule 322
D.3.1 Dateistruktur 322
D.3.2 Makefile 322
D.3.3 main.c 324
D.3.4 Implementierung eines Tastaturkommandos 327
D.3.5 Implementierung eines Scriptkommandos 328
D.3.6 Nützliche Headerdateien und Funktionen 333

Kapitel 1

Einleitung

Moderne bildgebende Verfahren wie z.B. die Computertomographie (CT) und Magnetresonanztomographie (MRT) ermöglichen heute, dreidimensionale Grauwertbilder der inneren Struktur von Objekten zu erstellen.

Die Tomographen erreichen dabei Auflösungen von $< 1\,\mathrm{mm}^3$ und erzeugen somit sehr große Datenmengen. So belegen die in dieser Arbeit häufig zu Illustrationszwecken herangezogenen CT-Aufnahmen des menschlichen Abdomens typischerweise 400 Megabyte (Stapel von typisch 800 Schichten mit je 512x512 Pixel Auflösung und je 2 Byte/Pixel zur Darstellung des großen Dynamikbereiches).

So große Datenmengen sind nicht mehr mit wirtschaftlichem Zeitaufwand durch schichtweise manuelle Beurteilung zu sichten und oft bleibt auch einem erfahrenen Betrachter mancher dreidimensionale Zusammenhang verborgen, der durch eine entsprechende Visualisierung sichtbar gemacht werden könnte.

Bis vor kurzem war es aufgrund der Speicher- und Rechenleistungsanforderungen jedoch nur auf aufwändiger Spezialhardware möglich, solche Visualisierungen durchzuführen. Und selbst wenn diese zur Verfügung stand, war die Verarbeitung in der Regel nicht echtzeitfähig und lieferte offline gerenderte Videosequenzen oder nur einzelne Bilder.

Das hier vorgestellte Programmpaket $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ wurde entwickelt, um unter Einsatz kostengünstiger PC Hardware, die inzwischen die notwendigen Voraussetzungen erfüllt, solche großen 3D-Datensätze aufzubereiten, in Echtzeit zu visualisieren und Hilfestellung bei der Interpretation zu geben.

Die Entwicklung eines eigenen Systems erschien sinnvoll, da kommerziell verfügbare Lösungen weder preislich noch von der gewünschten Funktionalität, Flexibilität und Geschwindigkeit in Frage kamen.

$\epsilon CC\epsilon T$ wurde zunächst als Visualisierungssystem entwickelt, um die Ergebnisse des an unserem Institut entwickelten Filterverfahrens [NLG] plastisch darstellbar zu machen. Später wurden in Zusammenarbeit mit der Uniklinik Düsseldorf und anderen Forschungseinrichtungen Filter- und Segmentierfunktionen sowie Detektionsalgorithmen für medizinisch interessante Strukturen in das System integriert, so dass sich insgesamt ein in sich abgeschlossenes System zur Vorverarbeitung, Segmentierung und Visualisierung dreidimensionaler Datensätze ergibt.

Ein modularer Aufbau ermöglicht dabei, leicht Erweiterungen in das System einzubauen sowie einzelne Teile auf mehrere Rechner zu verteilen, um mehr Information gleichzeitig darstellen zu können.

$\epsilon CC\epsilon T$ nutzt — soweit die eingesetzten Algorithmen dies erlauben — konsequent Multithreading, um auf Multiprozessor-Architekturen die volle Performance ausnutzen zu können. Eine Erweiterung dieser Fähigkeit auf Clustersysteme erschien uns aufgrund der großen Datenmengen, die dazu an die Knoten verteilt werden müssen, nicht als sinnvoll. Für extrem langlaufende Algorithmen stehen aber entsprechende Standalone-Programme zur Verfügung, die scriptgesteuert auf Clustern eingesetzt werden können.

Im Folgenden werde ich zunächst in Kapitel 2 die Anforderungen an ein 3D-Segmentierungs- und Visualisierungssystem betrachten, um dann in Kapitel 3 summarisch das Gesamtsystem zu beschreiben. Die Kapitel 4 bis 8 gehen dann im Detail auf die einzelnen Algorithmen ein.

Einige existente Anwendungen mit Beispielen finden sich in Kapitel 9, ein kurzer Ausblick auf die aktuellen Entwicklungen, die nicht mehr Einzug in diese Arbeit gefunden haben, in Kapitel 10.

Kapitel 2

Anforderungen an 3D-Visualisierungssysteme

Durch moderne bildgebende Verfahren stehen heute in vielen Bereichen (Konstruktion, Materialprüfung, Medizin etc.) dreidimensionale Datensätze von real existierenden oder noch zu bauenden Objekten zur Verfügung.

Einige wenige Verfahren liefern dabei primär triangulierte Oberflächendaten, so z.B. Laser-Range-Scanner, die durch Entfernungsmessung Reliefs abtasten können. In diese Kategorie fallen natürlich auch existierende CAD-Daten.

Die Mehrzahl jedoch liefert Volumendaten, die aus einzelnen Voxeln (Volume Element, dem dreidimensionalen Analogon zum Pixel, dem Picture Element) bestehen.

Während zur Visualisierung von triangulierten Oberflächen aufgrund ihrer Anwendung in Computerspielen leistungsfähige Hardwarelösungen zur Verfügung stehen, überforderte die Visualisierung von Volumendaten bis vor kurzem noch die Fähigkeiten allgemein verfügbarer PCs. Inzwischen sind allerdings ausreichend große Speicherbausteine und schnelle Prozessoren und Bussysteme im Massenmarkt entsprechend günstig erhältlich, so dass eine kostengünstige Realisierung von Volumen-Renderingsystemen möglich wird.

Plant man ein solches System, so sollte man sich zunächst die Frage stellen, was der Benutzer von einem solchen System erwartet und welche Vorteile sich gegenüber dem klassischen Verfahren ergeben.

Diese Fragestellung werde ich im Folgenden vorwiegend aus dem Blickwinkel der wichtigsten Anwendung, der medizinischen Diagnostik, betrachten.

2.1 3D-Datenquellen

Zur Erfassung dreidimensionaler Strukturen existieren heute eine Reihe von Möglichkeiten.

2.1.1 Oberflächenbasierte Verfahren

- Oberflächen können mit Digitalisierstiften, die mit einem Positionsbestimmungssystem versehen sind, abgetastet werden,
- Reliefs werden aus stereographischen Bildpaaren rekonstruiert.
- Der Einsatz von strukturiertem Licht ermöglicht die Triangulation von Oberflächen mit nur einer Kamera.
- Laser-Range-Scanner arbeiten ähnlich und können berührungslos ein Tiefenprofil erzeugen.
- Das Silhouettenschnittverfahren erlaubt die Rekonstruktion einfacher Körper aus einer Photoserie mit bekannten Kamerapositionen.
- Objekte können aufgrund von Messdaten in CAD-Systemen modelliert werden.

Der Nachteil dieser nichtinvasiven Verfahren ist, dass lediglich die Oberfläche des Objektes erfasst werden kann. Im Inneren des Objektes liegende Strukturen sind nicht oder nur unzureichend erfassbar.

Allerdings sind oft gerade diese Bereiche von besonderem Interesse, da sich dort Defekte verbergen können, die durch eine Sichtkontrolle nicht zu entdecken sind.

Eine — wenn auch „brachiale“ — Alternative bieten

2.1.2 Destruktive Verfahren

Bei diesen wird der Untersuchungsgegenstand mehr oder weniger sanft zerlegt und in seinen Einzelteilen erfasst.

Dabei stellen sich eine Reihe von Problemen bei an sich allen verwendeten Verfahren:

- Erhaltung der Form der Einzelteile bei Zerlegung (federnde Elemente, weiche oder flüssige Bereiche)

- Erhaltung des räumlichen Gefüges (bei Zerstörung/Entfernung von Halterungen)
- ggf. irreversible Zerstörung des Objektes (z.B. bei Feinschnitten).

Aufgrund dieser Nachteile wird auch diese Methode nur in bestimmten Bereichen angewandt, in denen diese sekundär sind.

Besonders die Schnittmethode liefert allerdings außerordentlich hochwertige (da hochaufgelöste und der natürlichen Färbung entsprechende) Datensätze, wie z.B. beim Visible-Human-Projekt [VH] ersichtlich.

2.1.3 Tomographieverfahren

Als Zwischenweg verwendet man zur Objektabtastung Strahlung, die in der Lage ist, hinreichend weit in das Untersuchungsobjekt einzudringen.

In seltenen Fällen kann das Licht sein (Streulicht-Tomographie), wenn die optischen Eigenschaften des Objektes es gestatten. Zumeist verwendet man jedoch Röntgenstrahlung (Computertomographie), die den Vorteil hat, hohe Eindringtiefen in sehr viele Materialien zu bieten, oder die Kombination aus einem starken Magnetfeld und elektromagnetischen Wellen im Radiofrequenzbereich (Kernspintomographie), die zwar nicht so universell einsetzbar ist, dafür aber nicht die Risiken von Röntgenstrahlung bei Einsatz an lebendem Gewebe hat.

Der große Vorteil dieser Methoden besteht darin, dass dort nicht nur Oberflächendaten gewonnen werden (wie es z.B. beim an sich voxelbasierten Silhouettenschnittverfahren zur 3D-Rekonstruktion aus 2D-Bildern der Fall ist), sondern die physikalischen Eigenschaften der verdeckten Bereiche rekonstruiert werden können.

Man erhält allerdings kein Realbild, da ja kein „Photo“ gemacht wird, sondern lediglich physikalische Eigenschaften (Röntgenabsorption, Wassergehalt, Resonanzverschiebung) gemessen werden, was allerdings je nach Situation auch deutlich aussagekräftiger sein kann als ein Bild.

Nach Anwendung geeigneter Rechenschritte (inverse Radon-Transformation, Fourier-Transformation etc.) ist es mit all den genannten Verfahren möglich, ein **Volumenmodell** des untersuchten Objektes zu erstellen.

Dabei wird der Gegenstand in sog. **Voxel** — kleine würfelförmige Bereiche, analog den **Pixeln** bei Bildern — zerlegt, denen gemäß den physikalischen Eigenschaften ein Zahlenwert zugeordnet wird. Sogenannte multimodale Verfahren erlauben theo-

retisch auch die Zuordnung mehrerer Werte, die unterschiedliche Eingenschaften repräsentieren; diese sind aber noch nicht weit verbreitet.

Die mit diesen Verfahren zugänglichen, unter der Oberfläche liegenden Punkte sind von besonderem Interesse, da sie sich einer unmittelbaren Sichtprüfung entziehen. Eine unmittelbare Reduktion des Volumens auf triangulierte Oberflächen macht somit ohne weitere Verarbeitung wenig Sinn.

2.2 Das herkömmliche Verfahren

Die gewonnenen Daten existieren jetzt zwar, doch wie betrachtet man sie?

Analog zu einem zweidimensionalen Bild könnte man ein Modell bauen, aber damit wäre (abgesehen vom Aufwand) nicht viel gewonnen, da man dieses wieder zerlegen müsste, um die interessierenden inneren Strukturen betrachten zu können.

2.2.1 Schichtweise Auswertung auf Film

Klassisch werden daher diese Daten zumeist schichtweise ausgewertet. Dazu werden die digital rekonstruierten CT-Daten auf Röntgenfilme ausbelichtet und Bild für Bild ausgewertet. Damit wird quasi eine Anfertigung von Feinschnitten simuliert.

Dieses Verfahren wird aufgrund der immer feiner werdenden Schichtdicken (heute gängiger Schichtabstand $1,25\,\mathrm{mm}$) moderner Tomographen zunehmend aufwändiger, da z.B. für eine typische Abdomenaufnahme über 300 Schichten anfallen.

Eine Vergrößerung der Schichtdicke bedeutet automatisch eine Verschlechterung der Sensitivität für kleine Strukturen, so dass hier keine Möglichkeit zur Reduktion des Arbeitsaufwandes besteht.

2.2.2 Visualisierung der Schichten an Bildschirmarbeitsplätzen

Um den an sich unnötigen Schritt der Ausbelichtung zu sparen, wurden zunehmend Visualisierungsworkstations eingeführt, die ebenfalls eine schichtweise Beurteilung der Daten ermöglichen.

Dabei ergibt sich allerdings der Nachteil, dass der Dynamikumfang von üblichen Grafikkarten und Monitoren (typ. 256 Graustufen) nicht an den eines Röntgenfilmes (weit über 1000 Graustufen) heranreicht. Hier läßt sich allerdings aus der Not

eine Tugend machen: Durch Verwendung eines Grauwertfensters, das sich frei über das Helligkeitsintervall der zugrundeliegenden Daten verschieben lässt, kann man nachträglich die Aufnahme mit (im Rahmen der gegebenen Auflösung der Daten) beliebiger Helligkeits- und Kontrasteinstellung betrachten.

Dies erweist sich insbesondere deshalb als vorteilhaft, weil selten gleichzeitig der volle Dynamikumfang benötigt wird, sondern oft nur teilweise recht kleine Unterschiede zwischen verschiedenen Geweben betrachtet werden, die durch die einstellbare Kontrastüberhöhung besser erkennbar werden.

Gelegentlich kann durch Falschfarbendarstellung eine weitere Verdeutlichung erreicht werden. Dazu wird den einzelnen Grauwerten der Ursprungsdaten nicht wieder ein Grauwert, sondern über eine Farbtabelle (CLUT, Color Look Up Table) eine frei definierbare Farbe zugeordnet. Dies erleichtert oft die Erkennbarkeit kleiner Details mit feinen Abstufungen.

Ein weiterer Nachteil der Visualisierung am Bildschirm gegenüber der klassischen Betrachtung auf Röntgenfilm scheint die geringere Auflösung zu sein.
Bildschirme erreichen typischerweise eine Auflösung von 75-100 dpi gegenüber über 300 dpi bei Röntgenfilmen (z.B. Agfa Mamoray HD/HD-S Verstärkerfolien: ca. 6 Linienpaare/mm).

Allerdings werden — im Gegensatz zum klassischen Ausbelichten der Schichten auf Röntgenfilm — auf elektronischen Visualisierungsworkstations selten mehrere Bilder nebeneinander dargestellt, sondern vielmehr die einzelnen Schichten nacheinander dargestellt, wobei die anzuzeigende Schicht interaktiv vom Benutzer gewählt werden kann.

Auch hier erweist sich die Einschränkung eher als Vorteil, da sich aufeinanderfolgende Schichten besser vergleichen lassen, wenn sie passgenau übereinander (und größer) statt nebeneinander dargestellt werden. Bei Systemen mit guter Performance (die mehrere Bilder pro Sekunde anzeigen) ergibt sich oft ein zusätzlicher Vorteil, da die Abfolge von Bildern vom Auge als Pseudobewegung wahrgenommen wird und damit eine bessere Verfolgbarkeit von Strukturen gegeben ist.

Ein Nachteil der klassischen schichtgestützten Auswertung besteht darin, dass man jeweils nur die Schichten in Richtung einer parallelen Ebenenschar begutachten kann. Dadurch sind die zu sichtenden Strukturen je nach deren Lage oft nur unzureichend im Zusammenhang zu erkennen. Z.B. erkennt man Falten, die senkrecht zur Sichtebene verlaufen, als wellenförmige Strukturen, verlaufen sie parallel, so machen sie sich dagegen durch abwechselndes Verengen und Erweitern des betrachteten Objek-

tes bemerkbar.

Hier hilft zum Teil bereits der erwähnte Effekt der Darstellung aufeinanderfolgender Schichten am gleichen Ort mit kleiner zeitlicher Differenz.

2.2.3 Darstellung anderer Schnittebenen

Die Darstellung auf elektronischen Systemen erlaubt aber zusätzlich noch die Betrachtung aus prinzipiell beliebigen Richtungen. Zumeist sind aber aus drei Gründen nur die drei Hauptebenen als Sichtebenen realisiert:

1. Die Berechnung dieser Ansichten ist trivial - es muss keine Ebenengleichung (die allerdings ebenfalls recht einfach ist) gelöst werden.

2. Es entstehen keine Diskretisierungsartefakte durch den Sprung zwischen verschiedenen Ebenen.

3. Die Orientierung für den Betrachter wäre in beliebig liegenden Schnitten deutlich erschwert.

Moderne Systeme erlauben es problemlos, diese drei Ansichten gleichzeitig darzustellen (als Grund-, Auf- und Seitenriss, wie aus dem CAD-Bereich bekannt) und somit die Orientierung weiter zu verbessern.

Insbesondere wird es damit möglich, eine interessante Struktur gleichzeitig in drei Projektionen zu erfasssen und damit bedeutend mehr über deren Morphologie zu erfahren.

Dieses Verfahren ist heute weitgehend als Standard zu betrachten und großflächig akzeptiert.

Beim Design eines neuen Systems wollte ich daher unbedingt diese Funktionalität ebenfalls integrieren, um für die letztliche Diagnostik die Möglichkeit zu haben, auf ein anerkanntes Verfahren zurückzugreifen.

2.2.4 3D-Visualisierungssysteme

Der wesentliche Nachteil der Nutzung von 2D-Bildstapeln besteht darin, dass vom Betrachter ein hohes Maß an räumlichem Vostellungsvermögen gefordert ist, um aus den Schichtbildern den morphologischen Zusammenhang zu erschließen.

In jüngerer Zeit erschienen daher auf dem Markt 3D-Visualisierungssysteme, die aus dem Grauwertschichtstapel dreidimensionale Bilder berechneten. Allerdings waren diese Systeme bisher nicht wirklich echtzeitfähig, so dass sie vorwiegend zu Dokumentations- und Illustrationszwecken, nicht jedoch zur Diagnostik eingesetzt wurden.

2.3 Anforderungen an ein neues System

Das Anliegen war daher ein schnelles System zu entwickeln, das (nach eventuellen Vorberechnungen, die unbeaufsichtigt laufen können) dem Bediener mehrere dreidimensionale Bilder pro Sekunde liefert, um so den dreidimensionalen Strukturzusammenhang ggf. direkt im Bild erfassen zu können.

Diese Geschwindigkeit ist nicht nur wichtig, um nicht die Geduld des Bedieners zu strapazieren, sondern auch, um den dreidimensionalen Eindruck zu unterstreichen. Soweit keine echten 3D-Displays verfügbar sind (diese sind noch relativ teuer bzw. mit unpraktischen Restriktionen wie dem Tragen von speziellen Brillen verbunden), erhält der Bediener vor allem durch die unterschiedlichen Absolutgeschwindigkeiten verschieden weit entfernter Punkte bei Drehungen (konstante Winkelgeschwindigkeit) einen Eindruck des dreidimensionalen Zusammenhangs der Szene, der insbesondere bei komplexen Geometrien mit vielen Verdeckungen (Aderngeflechte z.B.) wichtig ist. Für diese Anforderung sind $\gg 3$ Bilder/Sekunde notwendig.

Hier stellen sich nun einige technische Probleme und Wünsche an ein solches System, die ich hier kurz anreißen möchte (sie erfahren zumeist in den Kapiteln über die einzelnen Algorithmen genauere Behandlung), um schon jetzt die Hintergründe für bestimmte Designentscheidungen klarzustellen:

2.3.1 Auflösung verfügbarer Bildschirme

Ein typischer Datensatz hat eine Auflösung von ca. 600 Schichten von 512x512 Pixeln großen Bildern. Schon für die traditionelle 3-Schichten-Ansicht benötigt man damit eine Auflösung von ca. 1024x1112, um alle Ansichten vollständig und gleichzeitig darstellen zu können. Da gute Bildschirme mit solchen Auflösungen immer noch sehr teuer sind, fiel die Entscheidung für ein System, das flexibel auch bei kleineren Auflösungen einsetzbar ist.

Eine Möglichkeit hierzu wäre eine Skalierung der Fenster gewesen. Allerdings hätte

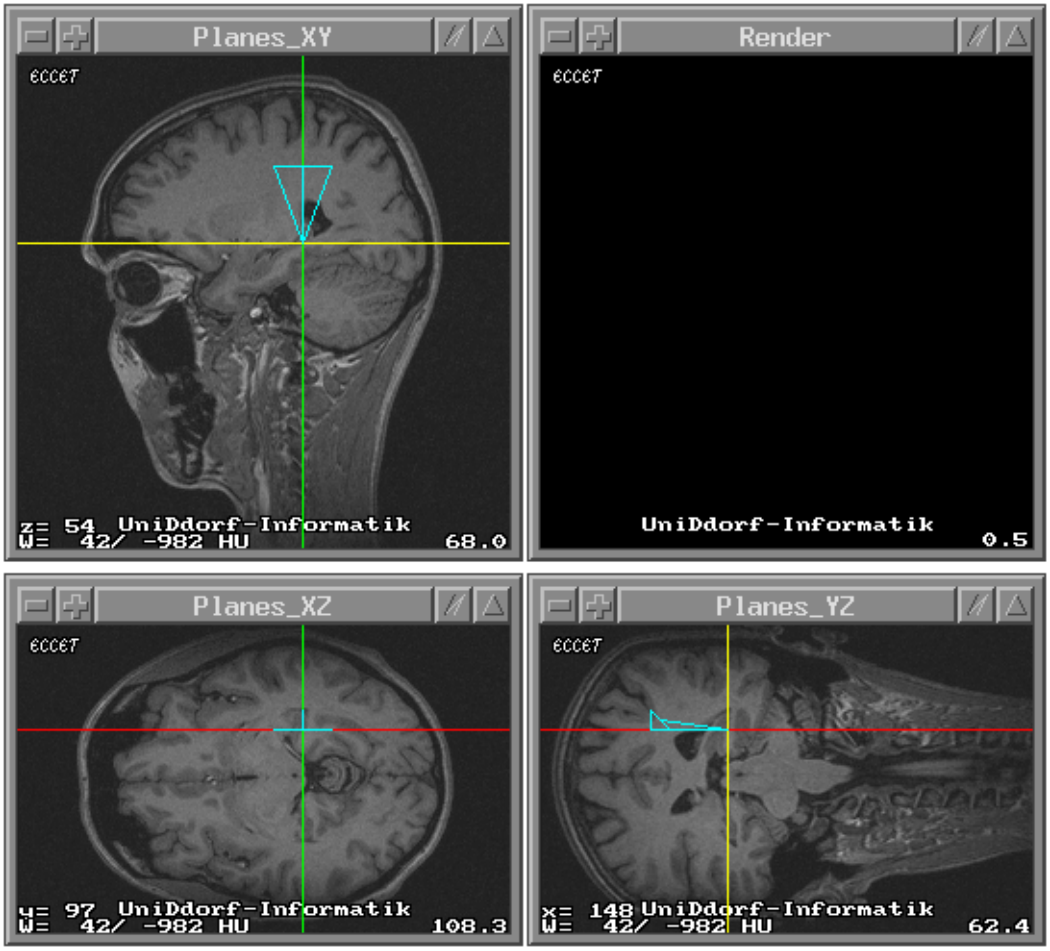


Abbildung 2.1: Anordnung der Ansichten bei 3-Schnitt-Darstellung

dies einen Verlust von darstellbaren Daten bedeutet und ggf. noch Interpolationsprobleme aufgeworfen und damit Artefakte erzeugt.

Daher wurde diese Variante verworfen und mit drei frei positionierbaren Fenstern mit den drei Einzelansichten (xy - 512x512, xz - 512x600, yz - 512x600) gearbeitet. Diese werden in der Grundeinstellung so angeordnet, dass sich (wie bei CAD-Arbeitsplätzen bewährt) ein durchgängiges Koordinatensystem zwischen je zwei Fenstern bildet (siehe Abb. 2.1). Dies führt allerdings dazu, dass auf gängigen Bildschirmauflösungen (1024x768 bzw. [empfohlen] 1280x1024) einige wenige Schichten unten über den Bildschirmrand ragen.

Dieses Problem kann aber leicht durch entsprechendes Hochschieben der beiden unteren Fenster umgangen werden, wenn Strukturen im unteren Bereich des Volumens betrachtet werden. Alternativ bieten viele Unix-Fenstermanager die Möglichkeit auf einem größeren „Virtuellen Bildschirm“ zu arbeiten, bei dem man schnell die nicht

sichtbaren Bereiche einsehen kann.

Bei dieser Anordnung bleibt in der rechten oberen Ecke Platz für ein weiteres Fenster von 512x512 Pixeln. Die erste Idee, diesen Platz für eine interaktive 3D-Visualisierung zu verwenden, wurde aus Gründen, auf die ich gleich näher eingehen werde, nicht realisiert.

Stattdessen wurde ein 2-Computer-2-Bildschirm-Ansatz verfolgt, da hiermit eine erheblich höhere Flexibilität erreicht werden konnte. Der zweite Bildschirm bietet mehr als genug Platz für eine 3D-Renderinganzeige sowie für die aufgrund der Vielfalt der möglichen Einstellungen recht umfangreiche zugehörige Steuerkonsole.

2.3.2 Speicherbedarf

Umfang der Rohdaten

Die von üblichen Computertomographen gelieferten Primärdaten besitzen einen typischen Dynamikumfang von 12 Bit (4096 unterscheidbare Werte), der üblicherweise in 2 Byte (16 Bit) pro Voxel codiert wird.

Andere Verfahren wie Magnetresonanztomographie haben zumeist einen kleineren Dynamikumfang in der Größenordnung von 8 Bit (256 Werte).

Damit wäre eine triviale Kompression um ca. 25%-50% je nach Datenquelle zwar theoretisch möglich, aber aufgrund der komplexeren Adressierung bzw. der nötigen Fallunterscheidungen nicht wirklich lohnend.

Insbesondere liegt der Bedarf für ein typisches Volumen mit 512x512x600 Voxeln (a 2 Byte) bei gerade mal 300 MB — eine Größenordnung, die heutzutage problemlos Platz im Hauptspeicher eines besseren PCs findet.

Abschätzung des Speicherbedarfs

Allerdings wäre es für die 3D-Visualisierung wünschenswert, mehr Platz belegen zu können. Hier hilft die bereits getroffene Entscheidung für zwei Maschinen.

Da es heute kein Problem darstellt, einen gängigen PC mit 1 GB Hauptspeicher auszurüsten (ab 2 GB wird die Behandlung auf günstig erhältlichen PCs mit 80x86 Prozessor in der Regel problematisch und 4 GB ist die absolute Obergrenze für den innerhalb eines einzelnen Prozesses adressierbaren Speicher auf 32-Bit-Maschinen), ergibt sich folgende Rechnung:

1024 MB/(512x512 Pixel/Schicht)=4096 Byte·Schicht/Pixel

Im Sinne einer effizienten Adressierung bietet es sich an, 1, 2, 4 oder 8 Byte/Voxel zu allozieren. Somit bleibt Speicher für 4096, 2048, 1024 oder 512 Schichten.

Bei dieser Auswahl erscheint die Wahl von 4 Byte/Voxel als optimal, da dann noch Volumina mit bis zu 1024 Schichten behandelt werden können. Allerdings verbrauchen natürlich auch das Visualisierungssystem sowie andere Komponenten (nicht zuletzt das Betriebssystem) Speicher, so dass in der Realität eher ca. maximal 800 Schichten erreicht werden können.

Diese Zahl genügt aber für zur Zeit übliche Untersuchungsmethoden vollauf und es sind schnellere Fortschritte im Bereich der Computertechnik zu erwarten als bei der Erhöhung der Auflösungen der Tomographen.

Innerhalb dieser 4 Byte/Voxel galt es nun, eine effiziente Darstellung zu finden, die es ermöglicht, das Volumen schnell zu visualisieren.

2.3.3 Gewünschte Darstellungsarten

Ältere Systeme bieten dem Benutzer als „3D-Darstellung“ lediglich die sogenannte Maximumsprojektion an, bei der jeweils das hellste Voxel entlang eines Sehstrahles dargestellt wird. Dieses Verfahren hat zwar den Vorteil, lediglich 2 Bytes/Voxel zu belegen und somit unsere eben angestellten Speicherplatzüberlegungen noch deutlich zu unterschreiten, aber auch den gravierenden Nachteil, relativ langsam zu sein, unübersichtliche Bilder zu liefern und somit maximal zur Darstellung von kontrastmittelgestützten Aufnahmen geeignet zu sein (vgl. Abbildung 4.23a).

Unser Ziel war aber die Darstellung beliebiger Objekte, die mit Hilfe von segmentierenden Vorverarbeitungsschritten im Volumen markiert werden sollten.

Begriffsbestimmungen

Unter **Objekt** verstehen wir im Folgenden eine Menge von Voxeln, die in der Regel in einem räumlichen und/oder logischen Zusammenhang stehen. Ein Beispiel dafür wäre die Menge aller Voxel in einem Volumen, die zur Leber gehören.

Im Allgemeinen erhalten wir jedoch von der Datenquelle keine Informationen über z.B. Organzugehörigkeiten, sondern nur über eine physikalische Eigenschaft (z.B. Röntgenabsorption). Aufgabe der **Segmentierung** ist es, mit Hilfe der vorliegenden Daten diejenigen Voxel zu bestimmen, die zu einem gesuchten Objekt gehören. Ein **Segmentierungsalgorithmus** versucht also anhand von ihm bekanntem Wissen über das gesuchte Objekt eine entsprechende Zuordnung zu treffen. Ein sehr

einfacher Algorithmus wäre z.B. die Verwendung eines **Grauwertbereiches**, d.h. der Zuordnung eines Intervalls aus dem Wertebereich der gemessenen physikalischen Eigenschaft zu einem Objekt.

Wie stellt man nun so ein Objekt dar?

Hätten wir das Objekt real vor uns liegen, so würden wir dessen **Oberfläche** betrachten. Ein Einblick ins Innere wird nur durch Aufschneiden des Objektes möglich.

Analog macht es auch in der rechnergestützten Darstellung Sinn, die **Oberfläche** eines solchen Objektes zu betrachten.

Für diese sind nur diejenigen Voxel interessant, die mindestens ein direktes Nachbarvoxel haben, das nicht zum Objekt gehört. Wir bezeichnen diese Voxelmenge im Folgenden als **Rand** oder **Oberfläche** eines Objektes.

Den Vorgang der Berechnung einer zweidimensionalen Ansicht einer dreidimensionalen Szene nennen wir **Rendern**.

Darstellung beliebiger Oberflächen

Wie eben gezeigt, ist für eine Darstellung eines Objektes im Wesentlichen dessen Oberfläche verantwortlich. Ist das Ziel die Darstellung **beliebiger** Objekte, so dürfen diese Oberflächen nicht explizit mit z.B. den Grauwerten des Volumens verbunden sein, sondern müssen frei vom Benutzer erzeugt werden können.

Eine explizite Bindung an die Grauwerte, wie sie von vielen Volume-Rendering-Techniken genutzt wird, birgt oftmals Probleme, wenn die Grauwerte eines zu betrachtenden Objektes nicht über das gesamte Volumen verteilt konstant sind, sondern noch z.B. mit Systemfunktionen der Messapparatur überlagert sind.

Realistische Oberflächendarstellung

Wünschenswert wäre eine Darstellung mit realistisch schattierten Oberflächen. Dazu wird aber die Oberflächennormale benötigt. Da diese in der Regel nicht als Messwert vorliegt, wird sie bei der Segmentierung aus der Morphologie des gewählten Objektes oder aus umliegenden Grauwerten approximiert.

Aus Geschwindigkeitsgründen sollte diese Schätzung aber vorberechnet vorliegen, um die zeitaufwändige Berechnung zur Laufzeit zu sparen.

Gleichzeitige Darstellung verschiedener Objekte

Das geplante System sollte auch in der Lage sein, verschiedene Objekte gleichzeitig und in Relation zueinander darstellen zu können (z.B. ein Organ und sein Gefäßsystem). Daher bot es sich an, ein Byte für eine „Voxelklasse“ zu reservieren.

Voxel der gleichen Klasse sollten ein einheitliches Erscheinungsbild (Farbe, Transparenzstatus etc.) erhalten und auch im Hinblick auf mehrstufige Segmentieralgorithmen erschien diese Entscheidung sinnvoll, da sie die Möglichkeit bot, verschiedene Zwischenergebnisse zu markieren und geeignet zu kombinieren.

Darstellungsgeschwindigkeit

Eine dreidimensionale Darstellung macht für den Praxiseinsatz nur Sinn, wenn sie in kurzer Zeit erzeugt werden kann. Lange Rechenzeiten stehen der Akzeptanz eines solchen Systems für die Praxis entgegen.

Angestrebt wurde Echtzeitfähigkeit, das heißt, es sollten so viele Bilder pro Sekunde erzeugt werden, dass Lageänderungen interaktiv vorgenommen und beurteilt werden können.

Des Weiteren wird es erst bei hinreichend hoher Bildfolgefrequenz für das Auge möglich, den dreidimensionalen Zusammenhang einer komplexen Szene aufgrund der unterschiedlichen Bewegungsgeschwindigkeiten zu rekonstruieren.

2.3.4 Vorverarbeitungsschritte und ihr Aufwand

Um ein ggf. verrauschtes Grauwertvolumen für eine automatische oder manuelle Segmentierung vorzubereiten, müssen oftmals relativ aufwändige Algorithmen (wie z.B. in [NLG] und [NLG2] beschrieben) eingesetzt werden. Dies soll aber weder die Benutzung erschweren, noch den Hardwareaufwand ansteigen lassen.

Rechenzeit

Da die Algorithmen oft lange Rechenzeiten haben, wurde versucht, bevorzugt Algorithmen zu wählen, die keinen manuellen Eingriff erfordern und somit unbeaufsichtigt im Hintergrund laufen können.

Des Weiteren wurden die Algorithmen optimiert und ggf. vorhandene Parallelisierbarbeit ausgenutzt, um akzeptable Rechenzeiten zu erhalten.

Speicherbedarf

Bestimmte Operationen wie die Vorfilterung erfordern Platz für ein Zielvolumen. Da diese Operationen aber noch alleine auf den Grauwerten arbeiten, steht dieser Platz auch zur Verfügung. Auf eine Auslagerung auf Festplatte wurde nach Möglichkeit aus Geschwindigkeitsgründen verzichtet.

2.3.5 Bedienung

Allein die Steuerung einer 3D-Ansicht erfordert bereits eine ganze Reihe von Bedienelementen, um die 3 translatorischen und 3 rotatorischen Freiheitsgrade abzubilden. Hinzu kommen weitere Kameraparameter wie der Öffnungswinkel (Brennweite) sowie Beleuchtungssteuerung (Helligkeit, Lichtquellenposition und -abstrahlcharakteristik) und einige aufgrund des virtuellen Charakters der Szenerie mögliche Spezialeffekte wie Schnittebenen und Semitransparenzeffekte.

Ein schnelles Visualisierungssystem benötigt natürlich auch eine entsprechend schnell und intuitiv zu bedienende Steuerung, insbesondere für die Kameraposition. Die anderen Parameter werden seltener verändert und bedürfen oft eher einer mehr exakten denn gefühlsmäßigen Steuerung.

Bedienung der virtuellen Kamera

Im Bereich der 3D-Actionspiele, die in dieser Hinsicht ein verwandtes Problem darstellen, haben sich für die Steuerung Kombinationen aus Maus und Tastatur bzw. aus Joystick und Tastatur bewährt. Diese Zusammenstellungen erlauben

- eine intuitive analoge Steuerung der Grundbewegungsformen (drehen, gehen, seitlich ausweichen),
- schnelle Auswahl von vielen Sonderfunktionen über die Tastatur und
- kombinierte Bewegungsformen (z.B. gleichzeitiges Drehen und Ausweichen) trotz geringer Achsenzahl des analogen Eingabesystems durch gleichzeitige Nutzung beider Eingabegeräte.

Da im Gegensatz zum 3D-Spiel des Öfteren auch komplexere Einstellungen bedient werden müssen, schien die Maus die geeignetere Alternative, da damit ein häufiger Wechsel des Eingabemediums (zur Nutzung von Menüs, die mit einem Joystick nur unzureichend bedienbar sind) vermieden wird.

Andere Geräte sollen selbstverständlich möglich bleiben, da insbesondere echte 6D-Eingabegeräte (z.B. sog. Spaceballs, s. [SPB]) eine Erleichterung bei der Bedienung bieten können. Leider sind diese Geräte recht teuer und damit auch wenig verbreitet. Somit musste die genutzte Eingabeschnittstelle flexibel gewählt werden.

Einstellung von Optionen und Parametern per GUI

Viele Parameter wie z.B. der Kameraöffnungswinkel oder Darstellungsoptionen erschienen hingegen weniger für eine direkte Bedienung per Tastatur oder gar Maus geeignet, sondern mehr ein Fall für eine graphische Benutzeroberfläche (Graphical User Interface - GUI), die es ermöglicht mit Hilfe von sog. Widgets, also z.B. Schiebern, Auswahlboxen, Druckknöpfen, Drehreglern etc. verschiedene Einstellungen zu treffen.

Leider sind viele der in dieser Hinsicht vorhandenen Tools (GTK, Qt, Motif etc.) nicht für Echtzeitapplikationen ausgelegt. Eine schnelle 3D-Darstellung ist nicht ohne größere Kunstgriffe möglich.

Andere Bibliotheken (wie SVGAlib), die in dieser Hinsicht geeigneter erschienen, hatten wiederum keine höheren Funktionen zur Widgetverwaltung und erwiesen sich darüber hinaus als zu fehleranfällig und waren auch zumeist mit bestimmter Grafikhardware verbunden.

Höhere 3D-Schnittstellen, wie OpenGL, sind zumeist auf das Rendern von Oberflächen spezialisiert und somit auch zum Volume-Rendering nur bedingt geeignet[1]. Auch hier ist in der Regel keine direkte Unterstützung von Widgets enthalten.

Um den Benutzer also nicht schon im Design an eine Bedienoberfläche zu binden, die eventuell gravierende Nachteile hat und später nur unter großer Mühe ausgetauscht werden kann, da sie die Kernschleife des Programmes bildet, bot es sich an, die einstellbaren Parameter über eine einfache und einheitliche Schnittstelle nach außen anzubieten und somit eine Trennung der Renderengine vom GUI zu ermöglichen.

Entkopplung von Darstellungssystem und GUI

Die übliche Ausführung einer solchen Schnittstelle unter Unix ist ein auf TCP/IP beruhendes Client-Server-System. Dieses bietet gegenüber den Optionen einer lokalen

[1] Durch Verwendung semitransparenter Texturen kann ggf. eine schnelle Volumendarstellung in Hardware erreicht werden. Allerdings sind gängige Grafikkarten mit einigen hundert MB Texturen dann doch überfordert.

Schnittstelle per dynamischem Linken, lokalen Sockets oder Interprozesskommunikation zwar eine etwas schlechtere Performance, ist aber deutlich portabler und kann für eine Interaktion über Systemgrenzen hinweg eingesetzt werden. Dies ist — wie wir gleich sehen werden — nützlicher als zunächst angenommen.

In Anlehnung an die existierenden Internetprotokolle wie FTP, HTTP, POP3 etc. wurde eine Kommandoschnittstelle mit Command/Response-Struktur implementiert. Über diese Schnittstelle können dann beliebige Einstellungen einfach und genau getroffen werden.

Damit wird es möglich, die Bedienoberfläche vom Programm zu entkoppeln, und es tun sich eine Reihe weiterer Vorteile auf, die wir im Folgenden noch besprechen werden.

2.3.6 Orientierungshilfen für den Betrachter

Bei der 3-Ebenen-Darstellung fällt die Orientierung noch relativ leicht - insbesondere wenn in die Darstellung der einzelnen Ebenen die jeweils anderen dargestellten Ebenen als Schnittlinie eingeblendet werden (siehe Abbildung 2.1).

Bei dreidimensionalen Ansichten erfolgt eine Groborientierung an charakteristischen Merkmalen der dargestellten Objekte (z.B. in der Medizin an anatomisch markanten Punkten). Diese Orientierung ist aber nicht sehr genau und zudem nicht immer anwendbar. Insbesondere im Inneren komplexer Strukturen — wo nicht mehr das Objekt als Ganzes erfasst werden kann — wird es sehr schwierig, die eigene Position und Lage korrekt zu bestimmen.

Integrierte qualitative Orientierungshilfen

Zur Darstellung der Lage wird oft ein Würfel eingeblendet, der die Lage der Primärachsen des Objektes visualisiert. Das ist zwar sehr hilfreich, zeigt aber nur die drei Lagewinkel, nicht jedoch die Position im Raum.

Um diese ebenfalls sichtbar zu machen, nutzen existierende Systeme oft eine Gesamtansicht des Objektes, in der die Kameraposition eingeblendet ist. Das liefert zwar auch nur 2 Achsen aufgrund der Projektion, erlaubt aber zumeist im Zusammenhang mit der von der Kamera gesehenen Morphologie eine recht gute Bestimmung der eigenen Position.

Wünschenswert ist dabei eine Bewegungsmöglichkeit für diese Kontrollkamera, so dass die in einem 2D-Bild nicht bestimmbare Tiefeninformation durch Drehen der

Szene rekonstruiert werden kann.

Im Sinne einer maximalen Symmetrie in der Implementierung erscheint es optimal, schlicht mehrere Kameras zuzulassen, die frei positioniert werden können. Diese Funktionalität kann vielen Zwecken dienen - wie z.B. dem gleichzeitigen Betrachten einer interessanten Struktur aus mehreren Blickwinkeln oder eben auch der eben erwähnten Anzeige eines oder mehrerer Übersichtsbilder. Blendet man nun noch die Position aller Kameras in die Bilder ein, so ergibt sich eine gute Orientierung.

Allerdings ist die Positionsbestimmung immer noch recht qualitativ und nicht mit den ursprünglichen Bilddaten unmittelbar verknüpft.

Quantitative Positions- und Lageinformation in den Schichtbildern

Durch das 2-Rechner-Modell ergibt sich hier zunächst scheinbar ein Nachteil: Die aktuelle Kameraposition kann nicht unmittelbar in der Schichtansicht dargestellt werden, wo sie einen absoluten Eindruck von der Kameraposition innerhalb der erfassten Daten erlauben würde.

Hier hilft allerdings die bereits getroffene Entscheidung für eine Netzwerkschnittstelle. Sie kann die Position der Kameras zum zweiten Rechner mit der Schichtansicht übertragen.

Führt man nun die angezeigten Schichten so mit, dass sie jeweils die durch die Kameraposition laufenden Schichten längs der Primärebenen zeigen, so ergibt sich eine gute Feinorientierung.

Des Weiteren wird es möglich, durch einen einfachen Klick auf ein Voxel in der 3D-Ansicht auf Originaldaten zuzugreifen.

Allerdings muss das bisher projektierte Command-Response-Protokoll hierzu um einen „Broadcast-Modus“ erweitert werden, um eine permanente (Rechenzeit kostende) Abfrage der einzelnen Kameraparameter zu vermeiden. Dieser Modus kann einfach durch ein Kommando eingeschaltet werden. Die Applikation muss dann allerdings fortan damit rechnen, asynchron mit neuen Positionsdaten versorgt zu werden.

Diese Vorgehensweise löst auch das folgende Problem:

2.3.7 Diagnostikmöglichkeit nach anerkannten Verfahren

Zusätzlich zu sich eventuell neu auftuenden Möglichkeiten ist es essentiell, sowohl für die Akzeptanz durch den Mediziner als auch für die Sicherheit des Patienten, dass alle bisher vorhandenen Diagnostikmöglichkeiten weiter zur Verfügung stehen.

Dies wird an sich schon durch die 3-Ebenen-Darstellung erreicht. Allerdings soll es ja insbesondere möglich sein, mit Hilfe der neuen, im Dreidimensionalen möglich werdenden Methoden ebenfalls Erkenntnisse zu gewinnen und diese dann ggf. über die altbekannten Techniken abzusichern. Die Kopplung der beiden Systeme ermöglicht eine effiziente Lösung dieser Problematik. Eine Anomalie kann so im 3D-Visualisierer gefunden und anschließend in den Schichtbildern weitergehend analysiert werden.

Ein Beispiel für dieses Vorgehen stellen in der Medizin Polypen dar. Erscheinen in der 3D-Rekonstruktion kugelige Objekte mit oder ohne Stiel auf Darmwänden, so kann oft ein Blick auf die 2D-Schnitte klären, ob es sich:

- um einen Multislice-CT-Artefakt (typische „Scheinwerferkegel“, die um den gefundenen Punkt zu drehen scheinen, wenn man die Einzelschichten in z-Richtung durchläuft) oder
- z.B. um Stuhlreste (manchmal erkennbar an Lufteinschlüssen, die in Polypen nicht vorkommen) oder
- um einen echten Polypen

handelt. In Fällen, in denen dieses Vorgehen keine Klärung bringt, liegt zumindest kein Nachteil für den Patienten vor, da ja eben auch die klassisch akzeptierte Methode der Betrachtung der Schnittebenen keine Klärung bringt.

Umgekehrt ist es damit auch möglich, einen in den zweidimensionalen Schnitten gefundenen Verdacht mit Hilfe der 3D-Darstellung schnell und einfach zu überprüfen.

2.3.8 Interaktive Bildverarbeitung

Bei neuen Aufgabenstellungen ist es sehr vorteilhaft, Änderungen in der Segmentierung oder in der Bildqualität, die durch die Anwendung eines Verarbeitungsschrittes entstehen, direkt sichtbar zu machen. Interaktiv können so neue Ansätze getestet, kann menschliches Expertenwissen eingebracht und können verschiedene Strategien verglichen werden.

Parameterübergabe an komplexe Algorithmen

Da viele übliche Verarbeitungsalgorithmen mehrere numerische Parameter bieten, die übersichtlich weder in einem GUI noch mit Hilfe von Tasten etc. gut darstellbar

sind, erweist sich die Bedienschnittstelle via TCP/IP-Kommandos erneut als vorteilhaft. Ein Befehl mit Parametern (numerisch oder auch alphanumerisch) bietet einen einfachen und dennoch exakt zu steuernden Zugang zu den Algorithmen und ihren Einstellungsmöglichkeiten.

Notwendigkeit hoher Verarbeitungsgeschwindigkeit

Da die Auswirkungen eines Verarbeitungsschrittes durch die Integration mit dem Renderingsystem direkt sichtbar gemacht werden können, wird die Entwicklung neuer Algorithmuskombinationen erleichtert. Allerdings ist es dazu essentiell, dass die Algorithmen möglichst schnell (d.h. maximal innerhalb weniger Minuten) abgearbeitet werden, damit ein effizientes Arbeiten mit dem System in der Entwicklungsphase möglich ist — der menschliche Experte soll unterstützt und nicht gebremst werden. Entsprechend soll das System möglichst alle Ressourcen des Rechnersystems nutzen.

Nutzbare Beschleunigungsmöglichkeiten

Durch die Festlegung auf herkömmliche PC-Systeme bietet sich hier allerdings nur eine sinnvolle Optimierung an: Multiprocessing. Multiprozessorsysteme sind für PCs relativ kostengünstig verfügbar und bieten auch aufgrund der lokalen Caches für jeden Prozessor die Möglichkeit die Leistung erheblich zu steigern. Insbesondere lassen sich viele einfache Algorithmen trivial parallelisieren und einige komplexere zumindest teilweise.

Entsprechend sollten möglichst solche parallelisierbaren Algorithmen präferiert werden, wenn für eine gegebene Aufgabe mehrere zur Auswahl stehen.

Eine Verbindung über das Netzwerk wie im Fall der Bedienoberfläche ist aber leider nicht möglich. Dies wäre zwar an sich extrem wünschenswert, da dann komplexe Berechnungen auf Clustersysteme verteilt werden könnten, aber aufgrund der enormen zu verarbeitenden Datenmengen ist dies mit handelsüblichen Netzwerken nicht darstellbar:

Eine Verteilung von 600 MB Volumendaten benötigt auf einem heute üblichen 100 MBit Netz ca. 50 Sekunden — für einen Scatter-Process-Gather-Zyklus also 100 Sekunden Kommunikationszeit plus Algorithmuslaufzeit. Das ist zu lange, um für die meisten Algorithmen vorteilhaft zu sein, insbesondere wenn man weiter von interaktiv sprechen will.

Lediglich eine starke kantenglättende Vorfilterung (mit Laufzeiten im Stundenbe-

reich) rechtfertigt ggf. einen solchen Schritt, der dann aber besser komplett offline läuft.

2.3.9 Weitgehende Automatisierung

Insbesondere die Vorfilterung eventuell verrauschter Daten und die Segmentierung erfordert oft rechenaufwändige Algorithmen, die vielfach komplex parametriert und kombiniert werden können.

Die Bedienung dieser Verfahren soll in der Anwendung auch für den Nicht-Bildverarbeitungsexperten problemlos möglich sein. Entsprechend ist es erforderlich, solche Algorithmenabfolgen inklusive der verwendeten Parameter abspeichern zu können, so dass sie vom Anwender nur noch als fertiges Programm abgerufen werden müssen.

Um dies zu erreichen, wird von einem Experten interaktiv nach einer stabilen Lösung für eine gegebene Problemklasse gesucht und die erforderlichen Schritte werden dokumentiert und in einem Script zusammengefasst. Der Anwender muss dann später lediglich dieses Script aufrufen, um ähnliche Probleme zu lösen.

Hier hilft das Konzept, dem Verarbeitungssystem Befehle in Form von Textkommandos über die Netzwerkschnittstelle erteilen zu können. Es erlaubt den trivialen Übergang von der Entwicklung einer Bildverarbeitungskette zu ihrem Einsatz in der Praxis: Man muss lediglich die verwendeten Befehle in ein Script kopieren, das dann vom Anwender aufgerufen wird.

Solche „Kochrezepte“ können durch die Netzwerkschnittstelle des Systems in jeder beliebigen Sprache, die in der Lage ist, TCP/IP- Verbindungen zu öffnen (z.B. Shellscripts, die netcat nutzen, bash2, Perl, C, Java etc.), realisiert werden und erlauben damit dem Experten volle Freiheit bei der Wahl seiner Entwicklungsumgebung.

Problematisch sind einzig Schritte, bei denen der Benutzer manuell interagieren muss. Dazu kann er aber leicht und mit einer genauen Anleitung von dem in der externen Sprache laufenden Script aufgefordert werden.

Sind keine interaktiven Schritte notwendig, kann man die Automatisierung noch einen Schritt weiter treiben und auf die graphische Kontrolloberfläche verzichten, so dass die Station (vom sonstigen Ressourcenverbrauch abgesehen) für einfache Aufgaben (Dokumentation, Betrachten von 2D-Bildmaterial) frei bleibt.

Entsprechend ist eine sinnvolle Einteilung des Gesamtsystems in Module anzustreben, die es ermöglicht, eine Version ohne Grafikoberfläche zu erstellen.

2.3.10 Navigationshilfen

Während der Entwicklung wurde schnell klar, dass die Navigation im dreidimensionalen Raum nicht immer trivial ist und neben Orientierungshilfen auch aktive Positionierungshilfen erfordert. Ein gutes Beispiel ist die Verfolgung eines gewundenen röhrenförmigen Gebildes, wie z.B. dem menschlichen Dickdarm bei der virtuellen Koloskopie.

Während beim realen Vorbild der Endoskopkopf vom Darmgewebe geführt wird und der Bediener das Endoskop lediglich vor- und zurückbewegen kann, muss in der virtuellen Variante sowohl die Lage als auch der Vorschub nachgeführt werden. Diese Aufgabe ist zwar aufgrund der interaktiven Performance nicht weiter schwierig, da eine Sichtkontrolle jederzeit gegeben ist, benötigt aber völlig unnötigerweise die Aufmerksamkeit des menschlichen Experten, die besser zur Erkennung von pathologischen Veränderungen als zur Erledigung von Routineaufgaben genutzt werden sollte.

Entsprechend soll der Benutzer so weit wie möglich von solchen Pflichten entbunden werden bzw. diese delegieren können.

Wegplanungssysteme

Bisherige Systeme beherrschten oft nur sehr rudimentäre Steuerungs- und Wegplanungsverfahren, wie z.B. die Angabe eines Fluchtpunktes, auf den dann mit wenigen grob aufgelösten Bildern pro Sekunde ohne Beachtung eventueller Durchdringungen zugegangen wurde.

Wichtig beim Design eines neuen Systems war, dass der interaktive Charakter des Gesamtsystems nicht gestört wurde. Eine feste a-priori-Wegplanung verträgt sich nicht gut mit Interaktivität. Sie mag zwar zu Demonstrationszwecken sinnvoll sein (und ist mit Hilfe einer Aufzeichnung der Kameradaten auch leicht zu realisieren), stört aber ansonsten eher, da man nicht ohne weiteres vom vorgeplanten Weg abweichen kann, wenn eine interessante Struktur auftaucht.

Dynamische Wegplanung - Autopilot

Daher erscheint eine Präferenz von dynamischen Systemen, die aus der aktuellen Situation heraus ihre Entscheidungen treffen, sinnvoll. Diese können jederzeit gestoppt werden, um gerade sichtbare Anomalien genauer zu untersuchen, und anschließend kann unverzüglich die automatische Navigation wieder aufgenommen werden.

Entsprechend soll der Einsatz einer solchen Hilfe die Echtzeitfähigkeit nicht merklich beeinflussen, so dass nur Algorithmen in Frage kommen, die relativ schnell und am besten auf Basis ohnehin vorhandener Daten laufen.

Aufzeichnung von Wegen

Zusätzlich kann es sinnvoll sein, einen menschlichen Experten (ggf. unterstützt durch die eben erwähnten Automatismen) den Weg finden und abspeichern zu lassen, um diesen einem anderen Experten als Hilfe an die Hand zu geben. Dennoch soll es natürlich jederzeit möglich sein, in den aufgezeichneten Weg einzugreifen.

Nur in Ausnahmefällen möchte man sich mit einem Film begnügen, der diese Möglichkeit nicht bietet. Dennoch ist es oft (z.B. aufgrund beim Empfänger nicht vorhandener Technologie) die einzige Möglichkeit, die eigene Ansicht zu transportieren, so dass dies ebenfalls möglich sein sollte.

2.3.11 Virtuelle Bildgebung für neue An- und Einsichten

Der virtuelle Charakter der 3D-Darstellung ermöglicht es, bisher unmögliche Sichten ohne Schaden für das zu betrachtende Objekt zu generieren. Während eine Innenansicht des Dickdarmes noch mit Hilfe der Koloskopie möglich ist, ist eine komplette Außenansicht in vivo nicht ohne weiteres möglich. Lediglich die Technik des Kontrasteinlaufes bietet eine Annäherung an diese Vorstellung.

In der virtuellen Ansicht ist dies jedoch problemlos möglich, und es kann auch diagnostisch genutzt werden. Während Divertikel (nach außen gehende Ausstülpungen im Darm) von innen oft schlecht zu sehen sind, können sie von außen trivial als hervorstehende Aussackungen erkannt werden.

Entsprechend kann die Möglichkeit genutzt werden, bereits gesehene Bereiche zu markieren und somit gezielt verdeckte Bereiche zu entdecken, die sonst übersehen würden.

Doch nicht nur aus solchen eher technischen Gründen bleiben manche eigentlich beachtungswürdigen Strukturen unentdeckt. Sicher kann sich jeder gut vorstellen, welche ungeheuere Konzentration es erfordert, über 400 Schichtbilder gewissenhaft nach zum Teil sehr kleinen Strukturen zu untersuchen.

Bereits die 3D-Rekonstruktion selbst bietet diesbezüglich oft schon einen großen Vorteil, da sie manche Auffälligkeiten, die man sich sonst schichtweise im Geist zusammenbauen müsste, auf einen Blick zeigt. Insbesondere vereinfacht sie die Ver-

folgung schräg und gekrümmt verlaufender Strukturen.

Doch nicht immer ist eine 3D-Ansicht klarer und aussagekräftiger, manchmal verdeckt sie mehr als sie enthüllt, so zum Beispiel bei relativ flachen Strukturen bei ungünstiger Beleuchtung. Dort ist es oft nützlich, ein anderes Beleuchtungsmodell oder eine andere Perspektive zu wählen oder schlicht einen Blick in die 2D-Schnitte zu werfen. Ein gutes Visualisierungssystem sollte diese Möglichkeiten bieten.

Doch woher soll man wissen, dass ein solcher Schritt sinnvoll wäre? Ihn bei jedem Bild zu gehen, wäre sicherlich kein Vorteil gegenüber einer klassischen Diagnose mit Hilfe der Einzelschichten.

Hier kommt ein neuer Bereich ins Spiel, die

2.3.12 Computergestützte Diagnose (CAD)

Bestimmte Anomalien können mit mehr oder minder hohem Aufwand mit Hilfe von Algorithmen entdeckt werden. Die Komplexität reicht von einfachen Schwellwertverfahren zur Entdeckung von Dichteänderungen bis hin zu komplexen geometrieabhängigen Klassifikatoren wie dem automatischen Auffinden von Polypen in Därmen.

Die Integration von Bildverarbeitungsalgorithmen in das Visualisierungskonzept bietet hier erneut die Möglichkeit, solche Verfahren in einer Weise einzubinden, die sowohl für die Entwicklung neuer Techniken als auch für den späteren Gebrauch geeignet ist.

Sind geeignete Klassifikatoren gefunden, kann deren Ergebnis aufgrund des schon vorhandenen Konzeptes der Voxelklassen problemlos in die Visualisierung integriert werden. Am einfachsten stellt man dazu die einzelnen Voxelklassen in verschiedenen Farben dar. Damit kann direkt das Ergebnis in die 3D-Darstellung einfließen.

Entsprechend wäre es auch schön, wenn auch die 2D-Ebenendarstellung von einer solchen automatischen Erkennung kritischer Punkte profitieren würde.

Um nun die Algorithmen nicht zweimal implementieren zu müssen und da manche Algorithmen zusätzliche Daten (insbesondere Oberflächennormalen) benötigen, erschien es hier auch im Sinne einer konsistenten Darstellung angebrachter, schlicht die Ergebnisse zwischen den beiden Teilsystemen auszutauschen.

Da hierzu nicht das komplette Volumen übertragen werden muss, sondern in der Regel lediglich eine Ja/Nein-Markierung pro Voxel, ist dies in sinnvoller Zeit (ca. 2-5 Sekunden) auf üblichen Netzen machbar.

Auch auf 2D-Ebene ist eine Farbkodierung derartig markierter Oberflächen die wohl sinnvollste Darstellungsmöglichkeit, wenngleich eine solche Einblendung natürlich abschaltbar sein sollte, um die Originaldaten auch unverfälscht darstellen zu können.

Diese Funktionalität ist insbesondere zur Überprüfung der Klassifikatorergebnisse beim CAD-Einsatz wichtig.

Da man für die meisten Anwendungen eher eine Übersensivität anstreben wird und somit lieber „false-positives“ in Kauf nimmt als ein Übersehen gesuchter Strukturen in unklaren Fällen, wird man in solchen Situationen oft die 2D-Darstellung zur Klärung heranziehen. Dort kann die Markierung genauso nützlich (schnelles Auffinden der betreffenden Stelle) wie ungünstig (Verdeckung/Verfälschung des betreffenden Bereiches) sein. Eine schnelle Möglichkeit, die Markierung ein- und auszuschalten, ist hier also ein großer Gewinn.

2.3.13 Integration in bestehende Umgebungen

Ein-/Ausgabe

Für die Akzeptanz des Systems ist es wichtig, dass 3D-Datensätze leicht in das System eingespeist werden können.

In der Medizin ist für den Bereich CT/MRT das DICOM-Format verbreitet, das zusätzlich zu den eigentlichen Bildern Metadaten (über Patient, Aufnahmemodalitäten, Arzt etc.) enthält. Ein Eingabemodul, das zumindest die Bilddaten, besser jedoch auch zumindest die für die Identifikation eines Falles nötigen Zusatzdaten, einlesen kann, ist daher in diesem Bereich notwendig.

Außerdem ist es wünschenswert, das System in vorhandene Datenbanken (PACS-System) zu integrieren.

Für andere Bereiche (Forschung, Industrie) ist hingegen ein möglichst einfaches Übergabeformat erwünscht, das leicht aus den dort oft verwendeten proprietären Datenformaten erzeugt werden kann.

Analog sollten interne Datenformate und Zwischenergebnisformate einfach aufgebaut sein, um eine leichte Integration in existierende proprietäre Lösungen zu ermöglichen. DICOM erscheint — trotz hoher Verbreitung — hierzu nur sehr begrenzt geeignet, da das Format relativ komplex zu parsen ist und nicht hundertprozentig vereinheitlicht.

Erweiterbarkeit

Kein System wird alle möglichen Anforderungen seiner Benutzer voraussehen und a priori mit fertigen Lösungen befriedigen können. Daher ist es insbesondere für den wissenschaftlichen Einsatz, bei dem man sich immer neuen Herausforderungen stellt, sehr sinnvoll, eine Möglichkeit zu schaffen, neue Ideen in das Produkt einfließen lassen zu können.

Einen Schritt dazu stellt die Verwendung einfacher und dokumentierter Dateischnittstellen dar. Allerdings ist es im Sinne einer nahtlosen Integration und der Erhaltung der interaktiven Verarbeitungsmöglichkeit wenig wünschenswert, zur Anwendung eines neuen Algorithmus die Volumendaten zunächst abspeichern zu müssen, sie in ein externes Programm einzulesen, damit zu bearbeiten und dann über einen erneuten Speichern/Laden-Zyklus die Ergebnisse wieder darstellbar zu machen.

Der logische nächste Schritt ist somit eine Modulschnittstelle, mit deren Hilfe extern entwickelte Programme direkt an das System angekoppelt werden können. Sie haben dann direkt Zugriff auf die Volumendaten sowie ggf. dazugehörige Metadaten (wie Voxelgröße, Patienteninformation etc.), das Steuerungssystem (Tastatur-, Maus- und Netzwerkkommandos) sowie die erzeugten Bilder.

Dokumentation

Aus der Nutzung des Systems resultierende Ergebnisse, Erkenntnisse und Diagnosen sollten leicht zu dokumentieren sein. Dazu ist es erforderlich, jederzeit Bilder und ggf. dazugehörige Metainformation (z.B. Patientendaten bei medizinischem Einsatz) abspeichern, ausdrucken, oder in andere Systeme importieren zu können.

Eine Exportschnittstelle für gängige Grafikformate ist daher ebenfalls sehr sinnvoll. Dabei kann in vielen Fällen die Metainformation als Kommentar innerhalb der Bilddatei erhalten werden, um auch nach Übertragungen auf andere Systeme den Zusammenhang erhalten zu können.

Aufgrund der Vielzahl möglicher Formate schien eine Modularisierung dieses Bereiches sinnvoll, so dass je nach Anwender geeignete Exportmodule zusammengestellt werden können, die den im jeweiligen Umfeld üblichen Konventionen Rechnung tragen.

Erzeugung von Filmen

Um den interaktiven Eindruck des Visualisierungssystems auch Anwendern zugänglich zu machen, die keinen Zugang zu den Originaldaten und/oder der entsprechenden Visualisierungstechnik haben, ist auch eine Exportmöglichkeit für Filmsequenzen notwendig. Allerdings fallen bei der Speicherung von Filmen große Datenmengen (typisch ca. 3.5 MB/Sekunde) an. Dies belastet das System in erheblichem Maße, worunter die interaktive Nutzung leidet.

Möchte man die Daten komprimiert (z.B. als MPEG) speichern, so werden dafür weitere Ressourcen benötigt und die Interaktivität leidet weiter (typisch sind noch ca. 2-3 Bilder/Sekunde möglich, wenn Videos mitgeschnitten werden).

Besser ist es daher, wenn zunächst die zu exportierende Sequenz in Form von Kameradaten mitgeschnitten wird. Dies kann ohne merklichen Einfluss auf die Darstellungsgeschwindigkeit geschehen. Später werden dann, ohne dass der Benutzer diesen Vorgang überwachen muss, die Videobilder berechnet, ggf. abgespeichert und in ein Filmformat komprimiert.

Da solche Filme oft für Präsentationen verwendet werden sollen, möchte man natürlich die Möglichkeit haben, eventuell misslungene Sequenzen neu aufzunehmen. Dies ist leicht möglich, indem man den geplanten Film zunächst in mehrere Teilsequenzen (Szenen) zerlegt und in je eine Positionsdatei abspeichert. Weniger gelungene Szenen werden einfach neu aufgenommen, wobei der Startpunkt einfach durch Abspielen der vorherigen Szene gefunden wird.

Eine weitere gängige Anforderung besteht darin, bestimmte Standardbewegungen (z.B. eine 360°-Drehung des gesamten Objektes) automatisch und damit auch „perfekt glatt“ ausführen zu lassen.

Dies ist mit Hilfe der integrierten Scriptfähigkeiten von VOXREN leicht möglich.

Dabei wird sowohl die Aufzeichnung des Kamerapfades als auch die letztliche Erzeugung der Filmdatei von den eben erwähnten Exportmodulen geleistet.

Kapitel 3

Konzeption des Gesamtsystems

3.1 Teilprogramme

$\epsilon CC\epsilon\mathcal{T}$ wurde modular ausgelegt, da einige Aufgabenstellungen doch sehr verschieden sind, aber dennoch stellenweise auf gemeinsame Daten und/oder Algorithmen zugegriffen wird.

Aufgrund der häufigsten Aufgabenstellungen zerfällt es in

- ein 3D-Visualisierungstool (VOXREN), das auch für viele andere Aufgaben (Segmentierung, automatische Geometrieerkennung), insbesondere in deren Entwicklungsphase, als Experimentier- und Arbeitsplattform dient, da es einen interaktiven Zugang zu mehrschrittigen Prozessen bietet.

- ein dazu komplementäres 2D-Visualisierungstool (PLANEVIEW), das die dreidimensionalen Daten in der klassischen 3-Schichtdarstellung aufbereitet, was für viele Anwendungen übersichtlicher und effizienter als eine volle 3D-Darstellung ist. PLANEVIEW ist allerdings auch in der Lage, unter bestimmten Voraussetzungen hochwertige 3D-Ansichten zu rendern - allerdings nicht immer mit der interaktiven Performance von VOXREN.

- ein Modul (BATCHVOX), das einmal gefundene Routineschritte zur Verarbeitung von Daten ohne interaktives Eingreifen eines Bedieners automatisiert, so dass zeitintensive Abläufe als Batchprozess ohne Aufsicht ausgeführt werden können.

- eine einfache graphische Oberfläche, die dem Benutzer den Aufruf der anderen Teilprogramme und die Navigation in den Daten erleichtert.

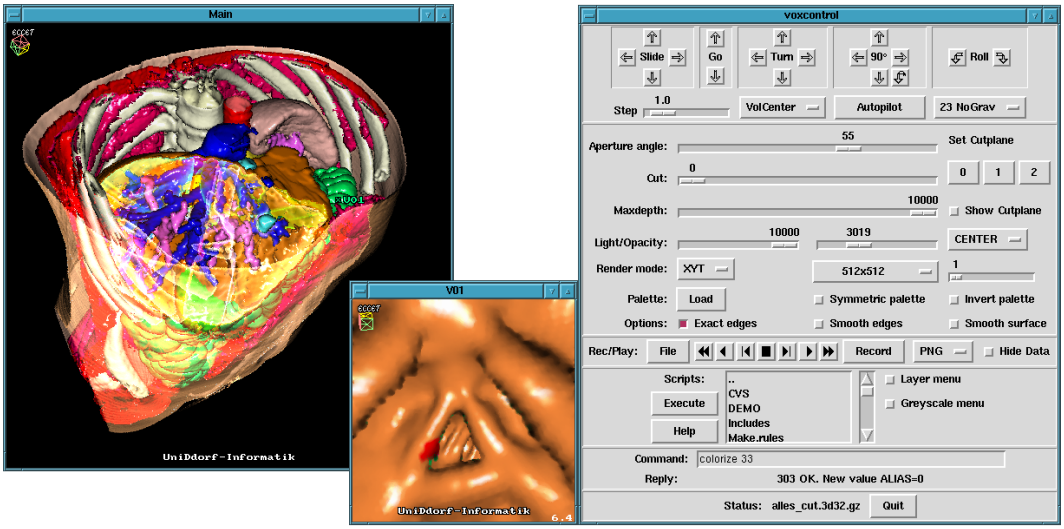


Abbildung 3.1: VOXREN im Einsatz

Diese Teilprogramme werden im Folgenden kurz vorgestellt, bevor auf wesentliche konzeptionelle Details des Gesamtsystems eingegangen wird.

3.1.1 VOXREN

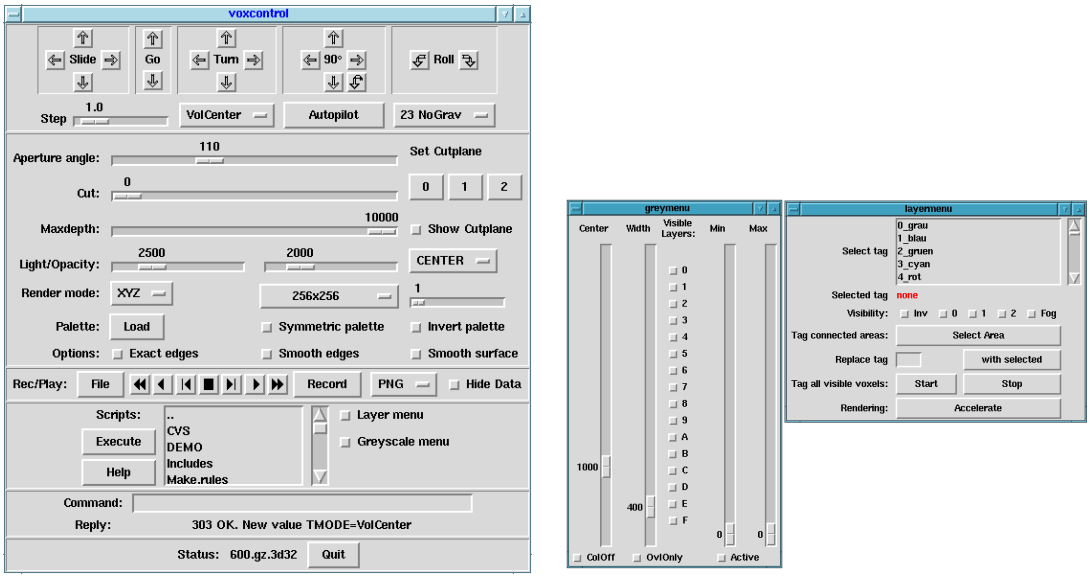
VOXREN bildet in vielerlei Hinsicht das „Herz“ von $\epsilon\mathcal{CC}\epsilon\mathcal{T}$, da es sowohl die Visualisierung dreidimensionaler Strukturen übernimmt, als auch den Zugriff auf eine Vielzahl interaktiver Manipulationsmöglichkeiten bietet, mit deren Hilfe die interessierenden Strukturen zunächst herauspräpariert werden.

Ziel bei der Entwicklung war, auf allgemein verfügbaren PCs eine dreidimensionale Darstellung von Voxeldaten **in Echtzeit** zu ermöglichen.

Während der Entwicklung wurde schnell klar, dass die Generierung der entsprechenden 3D-Datensätze aus den zugrundeliegenden CT-Schichtbildern eines interaktiven Tools bedarf, mit dessen Hilfe einzelne Bearbeitungsschritte kontrolliert und ggf. korrigiert werden können. Im Sinne der guten Bedienbarkeit erwies sich die Integration beider Komponenten in einem Programm als sinnvoll.

VOXREN bietet dem Benutzer die Möglichkeit,

- Grauwertvolumina in beliebigen Ansichten zu darzustellen,
- Segmentierungsalgorithmen interaktiv oder scriptgesteuert auf die Grauwertvolumina anzuwenden,



a) Hauptfenster b) Bei Bedarf ausklappbare Zusatzmenüs

Abbildung 3.2: Die graphische Benutzeroberfläche VOXCONTROL

- aus der Segmentierung gewonnene, dreidimensionale Volumendaten mit verschiedenen Darstellungsarten zu visualisieren,
- in diesen Darstellungen interaktiv zu navigieren,
- damit die Entscheidungen von Segmentierungsalgorithmen sofort zu kontrollieren und ggf. zu korrigieren,
- gleichzeitig mehrere verschiedene Ansichten zu betrachten,
- Objekte zu vermessen und
- die gewonnenen Erkenntnisse in Form von Bildern und Filmen zu dokumentieren.

Abbildung 3.1 zeigt VOXREN zusammen mit der Steuerkonsole VOXCONTROL bei der Anzeige eines komplexen Volumens, das mit $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ segmentiert wurde.

3.1.2 VOXCONTROL

Obwohl im Sinne der schnellen und effizienten Bedienbarkeit viele Funktionen von VOXREN direkt über die Tastatur oder Maus verfügbar sind, ist diese Bedienung für einige Bereiche ungeeignet (z.B. wenn Parameter übergeben werden müssen) oder

zumindest unpraktisch (z.B. das zyklische Durchschalten von Modi wie dem Drehpunkt). Wie bereits erwähnt, wurde daher der Weg beschritten, über die vorhandene TCP/IP-Schnittstelle das GUI VOXCONTROL anzubinden. Die Wahl der Programmiersprache fiel aufgrund der Möglichkeit der schnellen und einfachen Erstellung und Änderung auf Tcl/Tk.

Leider weist Tcl/Tk in einigen Bereichen Mängel auf, die nur durch unschöne Workarounds ohne großen Aufwand zu beheben sind. Daher stehen Überlegungen an, VOXCONTROL in nächster Zeit unter Verwendung einer geeigneteren Widgetbibliothek (z.B. einer hier gerade in Entwicklung befindlichen für LibGGI) neu zu schreiben.

3.1.3 PLANEVIEW

Nicht immer ist jedoch die dreidimensionale Betrachtung **allein** das geeignete Visualisierungsverfahren.

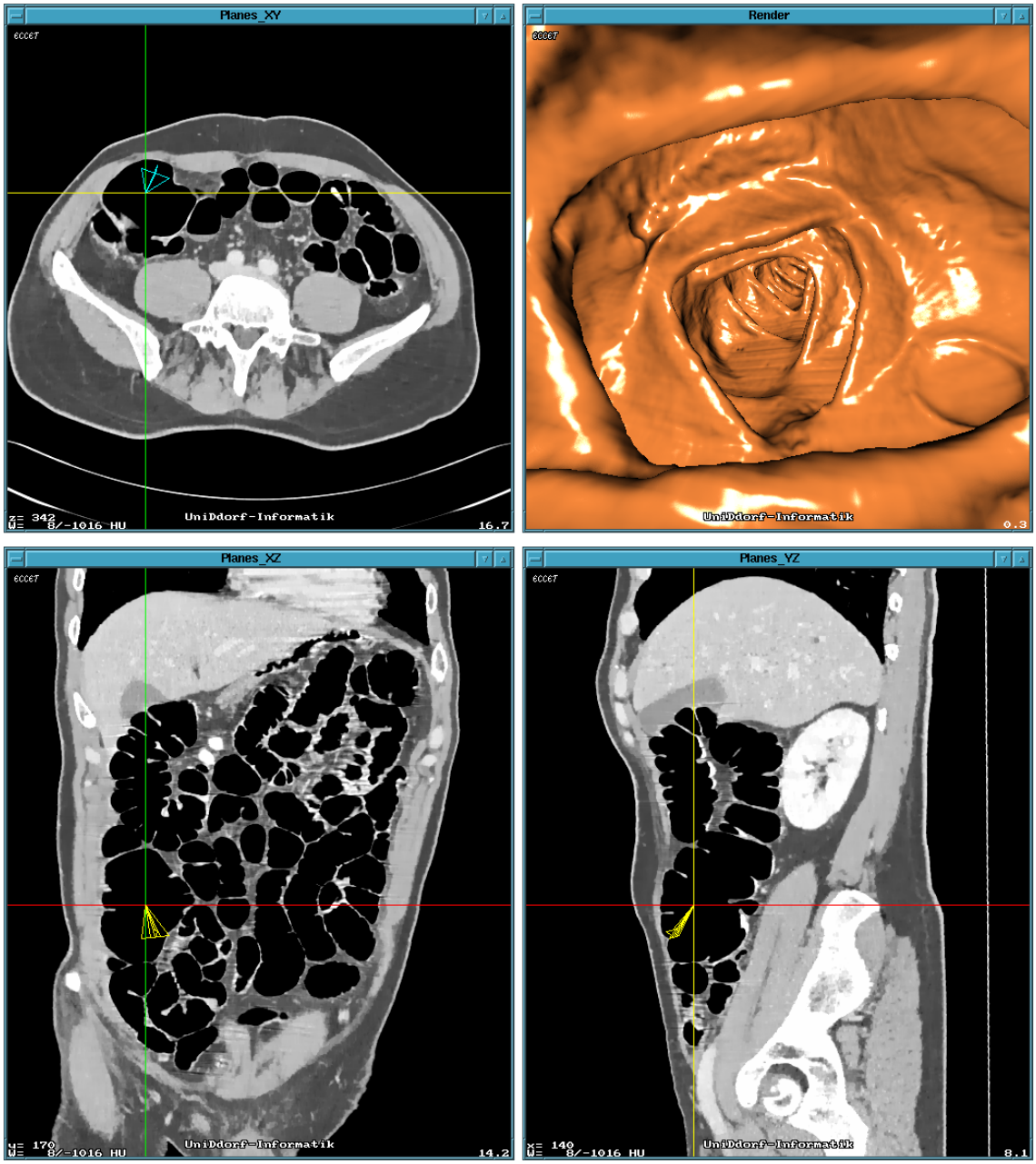
Besonders bei der initialen Beurteilung der Bildqualität (Rauschen, Ausschnittswahl, Dynamikumfang etc.) ist es oft sinnvoll, eine lediglich 2-dimensionale Ansicht der einzelnen Schichten eines Bildstapels zu verwenden.

Da die primäre Schnittebene nicht immer die geeignetste Ansicht darstellt, wurde nach dem Vorbild eines bereits früher am Institut entwickelten 3D-Viewers eine 3-Ebenen-Darstellung gewählt, die xy-, xz- und yz-Ebene gleichzeitig nebeneinander anzeigt. Diese Darstellung ist auch im Bereich 3D-Modeling (CAD) gebräuchlich, da sie durch die 3 verschiedenen Betrachtungsweisen auch Verdeckungen etc. besser sichtbar macht.

Gegenüber dem Vorgängerprogramm zeichnet sich PLANEVIEW insbesondere durch seine Fähigkeit aus, mit VOXREN zu kommunizieren und in Echtzeit seine Ansicht mit der von VOXREN abzugleichen.

Diese Möglichkeit ist insbesondere nützlich, um dem Betrachter die exakte Lage der virtuellen Kamera relativ zum Patienten zu visualisieren. Des Weiteren erlaubt sie es dem Radiologen, seine gewohnte Betrachtungsweise (Schichtbilder) beizubehalten und VOXREN lediglich als Werkzeug zu benutzen, um automatisch den zu befundenden Bereich vorzuselektieren.

Die Trennung der Programme VOXREN und PLANEVIEW resultiert aus deren unabhängiger Benutzbarkeit (nicht für jede Aufgabe benötigt man beide Programme) und dem Speicherbedarf.



Anmerkung: Die oben dargestellten Grauwertdaten wurden mittels NLG (s. Abschnitt 5.1) geglättet.

Abbildung 3.3: PLANEVIEW als virtuelles Koloskop

PLANEVIEW benötigt Speicher in der Größenordnung der Stapeldaten (typ. 512*512*800*2 Byte/Pixel = 400 MB), VOXREN sogar das Doppelte, wobei die ursprünglichen Grauwertdaten hier zumindest partiell nicht mehr vorhanden sind.

Die Gesamtgröße liegt somit oberhalb der mit den meisten 32 Bit Architekturen sinnvoll verwendbaren Speichergröße von 1 GB (das theoretische Limit beträgt 4 GB, meist sind aber durch eine logische Partitionierung des Speichers zur Einblendung von Betriebssystem, Bibliotheken und IO-Bereichen nur 1 oder 2 GB von Programmen aus nutzbar).

Daher wird empfohlen, bei großen Volumina VOXREN und PLANEVIEW auf zwei verschiedenen Maschinen laufen zu lassen. Durch die Kopplung per TCP/IP bleibt es dennoch möglich, das System zentral zu bedienen. Das löst gleichzeitig das Problem der unzureichenden Bildschirmauflösungen bzw. -größen. PLANEVIEW benötigt Platz für 3 Fenster von typisch 512x512 und 512x800 Pixel Größe. Selbst bei 1280x1024 Bildschirmauflösung bleibt nicht viel Platz für VOXREN und VOXCONTROL .

Des Weiteren beherrscht PLANEVIEW die Erzeugung hochwertiger 3D-Ansichten (s. Abbildung 3.3), wenngleich in der Regel nicht mit der interaktiven Geschwindigkeit von VOXREN. Dennoch kann damit für bestimmte Anwendungen auch PLANEVIEW alleine eingesetzt werden.

Zurzeit laufen Arbeiten an einem einfachen virtuellen Koloskop, das allein auf der Basis von PLANEVIEW arbeiten soll.

3.1.4 BATCHVOX

Mit der Realisierung von Scripten in VOXREN ergab sich der Wunsch, vollautomatische Vorverarbeitungsschritte, die keinerlei Aufsicht benötigen, in einer nichtgraphischen Applikation auszuführen.

Damit wird es möglich, die Vorverarbeitung auf externe Rechner ohne Bedienoberfläche auszulagern, so dass die Grafikworkstation für die Visualisierung frei bleibt.

BATCHVOX besteht im Wesentlichen aus den Modulen von VOXREN, allerdings ohne das Grafik- und Netzwerkmodul, und liest Kommandos aus einer Datei.

Im Praxiseinsatz zeigte sich jedoch, dass diese ohne optische Kontrolle auskommende Version nur selten genutzt wird - der Anwender verwendet lieber die (funktional gleichen) VOXREN-Scripte, deren Ergebnisse er direkt Schritt für Schritt am Bildschirm verfolgen kann.

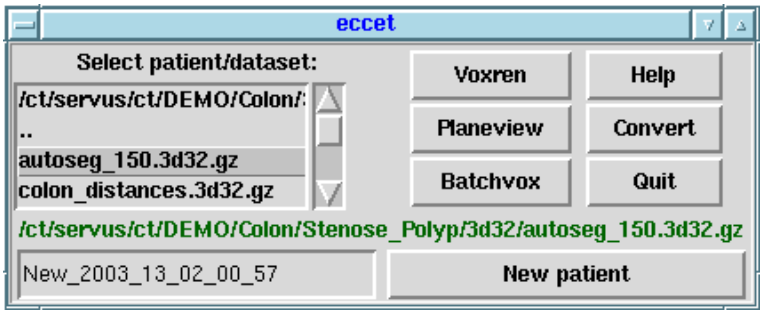


Abbildung 3.4: Die graphische Shell von $\epsilon\mathcal{CC}\epsilon\mathcal{T}$

Durch die Nutzung virtueller Fenstermanager kann das „Bildschirmplatzproblem“ leicht umgangen werden — man wechselt einfach, während das Script läuft, auf einen anderen virtuellen Bildschirm und arbeitet dort mit Applikationen, die keine hohen Anforderungen an das System stellen (z.B. Textverarbeitung), weiter und kann sich jederzeit per Mausklick über den aktuellen Bearbeitungsstand informieren.

3.1.5 Die graphische Shell

Normalerweise werden PLANEVIEW, VOXREN und BATCHVOX einfach von der Kommandozeile mit entsprechenden Parametern (zu ladendes Volumen, auszuführende Batchdatei etc.) gestartet. Dies ist zwar für den Experten sehr angenehm, für Benutzer, die den Umgang mit einer rein graphischen Oberfläche gewohnt sind, aber nicht zumutbar.

Für den klinischen Einsatz wurde daher mit Hilfe von Tcl/Tk eine einfache graphische Shell zum Start von VOXREN, PLANEVIEW und BATCHVOX geschaffen, die dieses Prolem löst.

Sie besteht aus einem einfachen Dateisystembrowser, mit dessen Hilfe der zu bearbeitende Patient ausgewählt wird (s. Abbildung 3.4). Hinzu kommen Startknöpfe für die Programme sowie für Konvertierungstools zur Umwandlung von Dicomstapeln, die bei einigen Tomographen leider in ungewöhnlichen Formaten und Namensgebungen exportiert werden.

Alternativ können bei Einsatz entsprechender Desktopoberflächen (Gnome/KDE) natürlich geeignete Bindungen in den oberflächeneigenen Dateisystembrowsern angelegt werden.

Für die Designkriterien dieser Komponente gelten im Wesentlichen die gleichen Bemerkungen wie für VOXCONTROL. Insbesondere ist auch hier eine Neuprogrammierung mit Hilfe einer mächtigeren Widgetbibliothek angedacht.

3.2 Speicherbedarf und Datenlayout

Wie bereits in Abschnitt 2.3.2 angerissen, benötigen Volumendaten sehr große Speichermengen. Entsprechend war es für das Design von $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ von grundlegender Bedeutung, die zur Verfügung stehenden Speicherressourcen optimal zu nutzen.

Wie bereits erwähnt, ergibt sich nach überschlägiger Berechnung ein möglicher Speicherplatz von 4 Byte/Voxel. Dieser soll nun möglichst effizient genutzt werden, um eine schnelle Visualisierung zu ermöglichen.

2 Byte/Voxel werden bereits von den Grauwerten verbraucht, so dass lediglich weitere 2 Byte/Voxel für Hilfswerte zur Verfügung stehen.

3.2.1 Voxelklassen

In 2.3.3 wird eine Unterscheidbarkeit verschiedener Objekte gefordert. Dabei kann ein Volumen auch nach der Segmentierung durchaus 20 und mehr verschiedene Objekte enthalten, und während der Segmentierung kann es sehr nützlich sein, Teilergebnisse verschiedenen Klassen zuzuordnen.

Daher wurde entschieden für die **Voxelklasse**, die für verschiedene Objekte jeweils verschieden sein soll, 1 Byte zu reservieren. Damit wird es möglich, bis zu 256 Voxelklassen zu unterscheiden.

3.2.2 Geometrische Distanzabbildung

Eine weitere dringliche Aufgabe, deren Lösung den Einsatz von Speicherressourcen rechtfertigt, ist die Darstellungsgeschwindigkeit. Ohne zusätzliche Information müsste in ungünstigen Fällen jedes einzelne Voxel des Volumens durchsucht werden, um diejenigen (in der Regel wenigen) Voxel zu finden, die wirklich dargestellt werden sollen.

Ein bekanntes Verfahren zur Beschleunigung dieses Vorgangs besteht in der Nutzung der geometrischen Distanzabbildung (GDA). Dazu wird für jedes Voxel die Entfernung zum nächsten darzustellenden Voxel (in der Regel sind das nur die Oberflächenvoxel, also eine relativ dünne Teilmenge aller Voxel) gespeichert. Dadurch wird weniger Zeit für die Durchsuchung leerer Bereiche verbraucht.

Diese Information lässt sich in ausreichender Auflösung in ein Byte codieren. In einem 512x512x600 Volumen können zwar Entfernungen von bis zu 600 (in der Maximums-Norm) auftreten, aber so wenig dicht besetzte Volumina treten in der

Praxis nicht auf und können unter marginalem Performanceverlust trotzdem korrekt dargestellt werden, indem der Wertebereich der Distanzfunktion einfach nach oben auf 255 begrenzt wird. Dadurch ergeben sich maximal 2 zusätzliche Sprünge. Diese fallen, verglichen mit den Verhältnissen in den typischen zerklüfteten Strukturen bei medizinischer Anwendung, kaum ins Gewicht.

Mit Hilfe der GDA alleine kann nun (siehe 4.2.1) bereits ein Schattenriss der markierten (GDA(x)=0) Voxel schnell dargestellt werden.

Damit wäre der vorhandene Speicher aber bereits vollständig belegt.

3.2.3 Oberflächennormale

Leider gibt es eine weitere Operation, die recht zeitaufwändig ist: die Berechnung der Oberflächennormalen.

Mit Voxelklasse, GDA und Oberflächennormale könnte man eine schnelle, realistische Oberflächendarstellung (vgl. 2.3.3) erreichen. Wird deren Ergebnis noch durch eine Farbtabelle (CLUT) ergänzt, so wird es möglich zusätzlichen Realismus durch optische Effekte wie Glanzlichter zu erzielen. Der Einsatz mehrerer CLUTs, die durch die aktuelle Voxelklasse ausgewählt werden, führt letztlich zu mehrfarbigen, realistisch schattierten Oberflächen.

Leider steht für die Oberflächennormale kein Speicher mehr zur Verfügung. Eine Möglichkeit bestünde darin, sie „on-the-fly“, also zur Laufzeit des Renderingalgorithmus für jeden darzustellenden Punkt zu berechnen. Dies wird bei der Darstellung von 3D-Szenen durch PLANEVIEW genutzt (s. 4.2.3), ist aber nur in bestimmten Fällen schnell möglich. Für die allgemeine Situation eines beliebigen Volumens ist es zu zeitaufwändig. Die Normale muss für diese Fälle vorberechnet werden, um die Szene schnell darstellen zu können.

3.2.4 Doppelnutzung des Grauwertbereiches

Es wurde entschieden, im Bereich der Oberflächen die Grauwertdaten durch die Oberflächennormale zu ersetzen.

Dies ist kein großer Verlust, da oft im Sinne einer robusteren Segmentierung und korrekteren Darstellung sowieso nicht die Originaldaten, sondern eine geglättete Version (siehe 5.1) zur Erzeugung der Oberflächen verwendet wird und durch den 2-Rechner-Ansatz die Original-Grauwertdaten dennoch jederzeit verfügbar sind.

Die Oberflächennormale wird üblicherweise durch einen Vektor dargestellt, von dem an sich nur die Orientierung benötigt wird. Diese sollte allerdings auf wenige Grad genau sein, da sich sonst deutliche Abstufungen in ansonsten glatten Oberflächen ergeben.

Der erste Ansatz, den Vektor mit drei fixkomma-codierten Bytes darzustellen, ist allerdings nicht brauchbar, da auch bei Doppelnutzung des Grauwertbereiches lediglich 2 Byte zur Verfügung stehen. Prinzipiell ist dieser Ansatz auch zu verschwenderisch, da die Länge des Vektors ja nicht interessiert.

Stattdessen wurde eine quantisierte Darstellung der Normalen (s. 4.2.2) verwendet, die mit 2 Byte auskommt und dennoch eine Auflösung von mindestens 1° bietet.

3.2.5 Grauwertklassen

Während der Segmentierung ist es wichtig, sowohl Grauwerte als auch Oberflächen darstellen zu können, da diese ja Quelle und Ziel des Segmentierungsprozesses darstellen.

Zur Unterscheidung von Oberflächen und Grauwerten, wurden daher die Voxelklassen 0-31 für Grauwerte reserviert. Innerhalb der Grauwertklassen bestimmt die Klasse dann eine zusätzliche Einfärbung des Grauwertvoxels durch „Aufprägen“ einer Farbe (s. Abbildung 6.8f). Diese Eigenschaft wird zur 2D-Darstellung von Bereichsgrenzen und anderen Segmentierungsergebnissen verwendet.

Die Verwendung der Klassen 0-31 wurde im Wesentlichen nur in Form einer logischen Reservierung realisiert (d.h. sie wird vom System nicht erzwungen), um möglichst wenig zusätzliche Entscheidungen während des Renderings treffen zu müssen.

Die Entscheidung, ob ein Voxel als Grauwert oder als Oberfläche (mit der Normalen im Grauwertfeld) dargestellt wird, wird stattdessen mit Hilfe der geometrischen Distanzabbildung getroffen: Oberflächenvoxel haben eine GDA von 0, für Grauwertvoxel ist der Wert der GDA bedeutungslos, er muss lediglich ungleich 0 sein.

Der Vorteil dieses Kriteriums ist, dass es während des Renderprozesses sowieso geprüft werden muss und daher keine zusätzliche Zeit benötigt.

3.3 Import von Daten

Bevor eine dreidimensionale Visualisierung erfolgen kann, müssen natürlich zunächst einmal entsprechende Volumendaten erfasst werden.

Dieser Vorgang liegt in den meisten Fällen nicht innerhalb des intendierten Aufgabenfeldes von $\epsilon\mathcal{CC}\epsilon\mathcal{T}$, sondern wird von externen Systemen übernommen und soll daher hier nur kurz gestreift werden.

Das Einlesen der vom externen System erzeugten Daten soll aber so einfach und reibungslos wie möglich geschehen.

3.3.1 Medizinische Daten

In der Medizin gebräuchliche Tomographen besitzen in der Regel eine eigene rechnergestützte Bedienoberfläche mit integrierten einfachen Visualisierungsfunktionen, die es ermöglichen, Schichten zu betrachten und interessante Bereiche zu selektieren.

Die Programme unterstützen durchweg einen Export in das gängige Medizindatenformat DICOM sowie in der Regel eine Möglichkeit zur Anbindung an ein PACS (Picture Archiving and Communication System).

Import der Daten

$\epsilon\mathcal{CC}\epsilon\mathcal{T}$ arbeitet intern mit einfachen, an die gängigen Unix-PNM-Bildformate angelehnten Dateiformaten.

Zur nahtlosen Integration in eine medizinische Umgebung wurde ein Eingabekonverter für gängige DICOM-Dateien erstellt. Dieser wird von $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ automatisch benutzt, wenn DICOM-Daten als Eingabe präsentiert werden.

Dabei extrahiert der Konverter nicht nur die eigentlichen Bilddaten, sondern auch zusätzliche Informationen über den betreffenden Patienten, die Bildgeometrie etc.

Da es sich bei DICOM um einen gewachsenen Standard handelt, der leider eine erhebliche Menge an Interpretations- oder Optionsspielräumen enthält, ist es allerdings möglich, dass dieser einfache Konverter nicht jede beliebige Datenquelle verarbeiten kann. In der Praxis werden jedoch die meisten Dateien korrekt gelesen und nicht korrekt gelesene Dateien können vom menschlichen Betrachter sofort als solche erkannt werden.

Sollte einmal der eingebaute Konverter versagen, kann leicht ein externer Konverter benutzt werden, da die PNM-Formate sehr gut dokumentiert und extrem einfach strukturiert sind.

Einen solchen Konverter hat dankenswerterweise Herr Hackländer mit seinem auf der freien DICOM-Bibliothek dcm4che ([D4C]) basierenden Java-Programm „dcm2pgm“ zur Verfügung gestellt. Es hat gegenüber anderen Programmen den

Vorteil, dass es ebenso wie der interne Konverter in den Bildkommentaren die wichtigsten DICOM-Daten (Geometrie, Patientendaten etc.) in einem für $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ lesbaren Fomat zur Verfügung stellt.

Transfer der Daten

Ein kompletter CT-Schichtstapel benötigt erheblichen Speicherplatz (typisch ca. 300MB). Der Transport der Daten kann daher im Allgemeinen nicht auf herkömmlichen Wechselmedien (Disketten) erfolgen.

Als mögliche Wege wurden bisher beschritten:

1. Speichern der Daten auf MOD-Platten:
   Mit MO-Laufwerken stehen Wechselmedien von ausreichender Kapazität (einige GB) zur Verfügung. Leider ergaben sich in diesem Bereich gewisse Interoperabilitätsprobleme (Medien mussten von Unix aus formatiert werden, um sowohl von Unix als auch von der windowsbasierten Oberfläche des verwendeten CTs verwendbar zu sein).

2. Transport der Daten mit CD(-RW):
   Während sich hier aufgrund des sauber spezifizierten ISO 9660 Dateisystems in der Regel keine Probleme ergaben, ist doch der zusätzliche Arbeitsgang des Erstellens und Brennens einer CD hinderlich. Die Methode eignet sich jedoch ausgezeichnet für Backups.

3. Mounten eines von der Linuxmaschine freigegebenen SMB-Shares:
   Dieser Weg erwies sich als sehr praktisch. Dazu wird einfach auf der Linuxmaschine, auf der $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ läuft, ein Verzeichnis per samba [SMB] freigegeben und auf der CT-Konsole als Laufwerk eingebunden. Danach können die üblichen Exportfunktionen genutzt werden. Natürlich muss die entsprechende CT-Konsole diese Operation unterstützen.

4. Verwendung von PACS:
   Dieses Verfahren ist netzwerkgestützt und in Klinikumgebungen verbreitet. Es ist sicherlich die komfortabelste Lösung und erfordert lediglich die Installation eines PACS-Servers auf dem $\epsilon\mathcal{CC}\epsilon\mathcal{T}$-Rechner und eine entsprechende Konfiguration der Datenquellen. Der nächste Abschnitt beschäftigt sich etwas genauer mit dieser Methode.

Anbindung an PACS

In Klinikumgebungen dient oft PACS zur Verteilung von Volumendaten an Visualisierungsstationen.

Mit Hilfe der JDicom-Bibliothek von TIANI ([JDC]) ist es möglich, auf einem $\epsilon\mathcal{CC}\epsilon\mathcal{T}$-Rechner einen einfachen PACS-Server einzurichten, der seine empfangenen Datensätze in einem Directory für $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ einfach zur Verfügung stellt.

Dazu muss das im JDicom Paket enthaltene ImageServer-Modul gestartet und parametriert werden. Eine gute Anleitung hierzu findet man in [JDA]. Zur Integration mit $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ legt man einfach unterhalb des $\epsilon\mathcal{CC}\epsilon\mathcal{T}$-Patientenverzeichnisses einen Pfad an, und trägt diesen in die Option `Fileset.Path` im ImageServer ein.

Damit wird eine gute Integration in den gewohnten Ablauf erreicht.

3.3.2 Import proprietärer Formate

Im Forschungsbereich, für den $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ aufgrund seiner Vielseitigkeit besonders interessant ist, ist es in der Regel erforderlich, mit bereits vorhandenen proprietären Datenformaten zurechtzukommen.

Im Anhang D.1 wird detailliert beschrieben, wie die von $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ lesbaren Datenformate aussehen.

Für viele Fälle dürfte die Möglichkeit genügen, P5-Dateien mit 8 bzw. 16 Bit Tiefe zu lesen. Dank der netpbm-Tools [PBM] stehen Konverter zu und von den wichtigsten graphischen Datenformaten zur Verfügung.

Allerdings können so nur die Bilddaten, nicht jedoch Metadaten übernommen werden. Diese werden aber als Kommentare im PBM-Dateikopf gespeichert, so dass sie leicht manuell mit einem Editor oder automatisiert mit Tools wie `sed` eingefügt werden können.

Für ganze Volumendatensätze (statt Schichtstapeln) lagen mir leider keine vergleichbaren dokumentierten Standardformate vor. Daher wurde ein eigenes Format (3D32) entwickelt, das in D.1 dokumentiert ist, so dass leicht entsprechende Konverter erstellt werden können.

3.3.3 Transparente (De-)Kompression

Da Volumendaten zumeist sehr große Speichermengen belegen, unterstützt $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ beim Laden von Daten in allen Formaten eine transparente Dekompression von

mit `gzip` komprimierten Daten. Dies kann insbesondere bei schnellen Prozessoren und verhältnismäßig langsamen Datenquellen (z.B. Netzwerkdateisystemen) auch der Ladegeschwindigkeit zuträglich sein.

Analog kann beim Speichern auch transparent mit `gzip` komprimiert werden.

3.4 Dokumentation

Insbesondere im medizinischen Umfeld ist die Dokumentation der aus der Anwendung des Systems gewonnenen Erkenntnisse von großer Bedeutung.

VOXREN ist in der Lage, ganze Sitzungen oder Einzelbilder in Dateien aufzuzeichnen und Dokumentationssystemen zu übergeben. Dabei werden verschiedene Formate unterstützt.

3.4.1 Aufzeichnen von Positionsdaten

Zeichnet man alle relevanten Kameradaten[1] während der Benutzung von VOXREN auf, was ohne nennenswerten Verlust bei der Darstellungsgeschwindigkeit möglich ist, so kann man mit geringem Aufwand später dieselben Ansichten noch einmal betrachten.

Außerdem können diese Daten dazu genutzt werden, offline Bilder mit hoher Auflösung zu berechnen und sie zu einem Film zusammenzufügen.

Durch die Kopplung mit PLANEVIEW ist eine synchrone Aufzeichung von dessen Positionsanzeigen und/oder der sehr hochaufgelösten Bilder möglich.

Während des Abspielens kann jederzeit angehalten und die Kamera frei bewegt werden. Ebenso ist eine Wiedergabe mit erhöhter bzw. variabler Geschwindigkeit per Maussteuerung möglich.

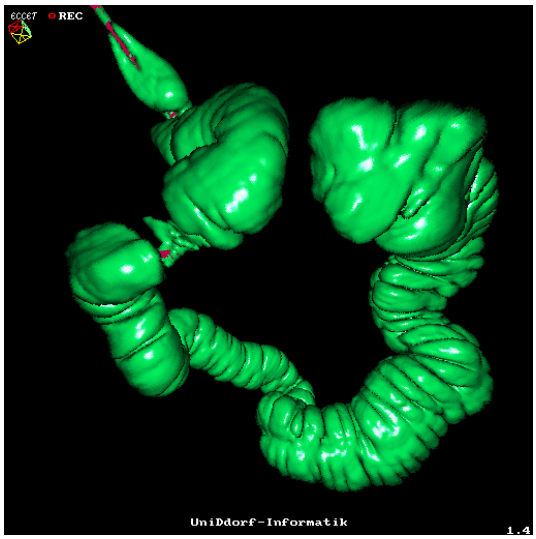
Dieses Verfahren eignet sich sehr gut zur internen Dokumentation, insbesondere da es eine sehr kompakte Möglichkeit bietet, ganze Sitzungen abzuspeichern, ohne dabei die Geschwindigkeit des Systems zu beeinträchtigen.

Will man hingegen die gewonnenen Einsichten an Dritte übermitteln, so sind auf der Gegenseite zwei Voraussetzungen erforderlich, die in der Regel nicht gegeben sind:

- Es werden die zugehörigen Volumendaten benötigt.

[1] Kameraposition, Blickrichtung und Orientierung, Öffnungswinkel des Objektives sowie vordere und hintere Schnittebene

a) Aufgezeichneter Pfad. Die Daten in der Positionsdatei enthalten zusätzlich noch die Kameraorientierung und weitere Parameter, die in einem Plot aber nicht übersichtlich dargestellt werden können.



b) Realbild des durchflogenen Darmes von außen in einer zur Projektion der Pfaddatei vergleichbaren Lage

Abbildung 3.5: Aufzeichnen von Kamerapfaden

Diese könnten zwar mitgeschickt werden, sind aber in der Regel sehr voluminös (einige hundert MB). Innerhalb einer Organisation ist dieses Problem durch den Einsatz schneller Vernetzung lösbar, zur Kommunikation mit außerhalb der eigenen Organisation gelegenen Einrichtungen bliebe nur die CD-ROM.

- Zur Darstellung ist die Verwendung von $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ erforderlich.
  Aufgrund der noch geringen Verbreitung von Linux und $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ wird dies in den seltensten Fällen gewährleistet sein.[2]

3.4.2 Abspeichern von Einzelbildern

Zur Übermittlung an Dritte geeigneter erscheinen Einzelbilder in gängigen Formaten wie PNG, JPEG, PNM und DICOM, die von handelsüblichen Programmen gelesen und dargestellt werden können. $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ bietet daher ebenfalls die Möglichkeit, einzelne (z.B. für die Diagnose entscheidende) Ansichten als Bild abzuspeichern. Dabei

[2] Zur Zeit arbeiten wir an einer Möglichkeit, $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ auf einer bootfähigen CD (basierend auf Knoppix [KPX]) unterzubringen. Damit wäre es möglich, Volumen und Visualisierungssystem auf einer einzigen CD unterzubringen, die man einfach in einen beliebigen PC einlegen und von ihr starten kann.

können optional auch Metadaten, wie z.B. Patientennamen etc. mit ausgegeben werden können, sofern dies vom Zielformat unterstützt wird.

Die abgespeicherten Bilder können dann in elektronischer oder ausgedruckter Form weiter verarbeitet, mit Kommentaren etc. versehen werden und so auch eine Grundlage für die Dokumentation schaffen.

Die Mehrzahl der Abbildungen in der vorliegenden Arbeit wurden mit Hilfe dieser Funktion erzeugt.

3.4.3 Aufzeichnen von Videosequenzen

Zusätzlich gibt es die Möglichkeit, ganze Filmsequenzen im MPEG-Format aufzuzeichnen, die mit üblicher Multimedia-Software abgespielt werden können oder auch auf (Super-)Video-CD gebrannt werden können, die mit handelsüblichen DVD-Playern abgespielt werden können.

Da die Codierung von MPEG-Video erhebliche Ressourcen beansprucht, ist zu empfehlen, zunächst eine Positionsdatei aufzuzeichnen.

Danach wählt man die gewünschte Filmauflösung und Qualität (z.B. Glättungsfilter), spielt die Positionsdatei ab und zeichnet dabei MPEG-Video auf.

Manuell erzeugte Positionsdateien eignen sich allerdings nicht für alle Anwendungen. Zu Präsentationszwecken möchte man in der Regel ruckelig wirkende Bewegungen und Korrekturen vermeiden.

Eine Möglichkeit dazu besteht in der Zerlegung des Films in mehrere Abschnitte, so dass misslungene Sequenzen leicht neu aufgenommen werden können.

Des Weiteren stehen für gängige Bewegungen (z.B. für Drehungen um 90, 180 und 360 Grad) Scripte zur Verfügung, die eine glatte Drehung erzeugen.

Eine weitere Möglichkeit besteht darin, nur einzelne Punkte des Filmes (sog. Keyframes) vorzugeben. Ein externes Programm (s. 8.2) berechnet dann einen geeigneten Kamerapfad, der in einer Positionsdatei abgespeichert wird und anschließend wie üblich weiterverarbeitet wird.

3.5 Modularisierung

3.5.1 Exportmodule

VOXREN und PLANEVIEW benutzen ein gemeinsames Modulinterface zur Ansteue-

rung von Speicherroutinen. Abgesehen von der damit verbundenen Vereinfachung für den Programmierer (ein Ausgabefilter muss nur einmal geschrieben werden und ist anschließend in beiden Applikationen verfügbar), ergibt sich auch für den Anwender ein Vorteil:

Er kann relativ leicht durch Ersetzen einer einzigen Datei das Exportsystem seinen Bedürfnissen anpassen. Verschiedene Umfelder setzen verschiedene Dateiformate ein (unixgeprägte Forschungsumfelder gerne PNM, PNG, Postscript, Anwender mit Windows-Umgebungen eher JPG, GIF, BMP, HTML oder PDF, im Medizinbereich zusätzlich noch DICOM), so dass hier eine leichte Anpassung sehr praktisch erscheint.

Realisiert wurde dieser Bereich mit Hilfe von dlopen()/dlsym()/dlclose(), einem auf Linux und Solaris weit verbreiteten Standard zum dynamischen Nachladen externer Programmteile.

Die Formatwandlungen werden großenteils von externen Programmen, die via `popen()` angesteuert werden, übernommen. Dies minimiert die externen Abhängigkeiten von Libraries bzw. vermeidet die aufwändige Neuimplementation von bekannten Formatkonvertern wie cjpeg, ppmtopng usw..

Die Geschwindigkeit leidet nicht signifikant unter der Nutzung eines externen Prozesses. Für Einzelbilder ist die Zeit für das Abspeichern ohnehin im Subsekundenbereich und daher nicht relevant und zum Speichern von Sequenzen wird der externe Prozess in der Regel nur einmal gestartet und dann kontinuierlich über die Pipe mit Daten versorgt.

Auch von Nutzern bereitgestellte Formatwandlungsprogramme können leicht integriert werden. So entstand in Zusammenarbeit mit Herrn Dr. Hackländer vom Klinikum Wuppertal ein Exportmodul für DICOM (secondary capture). Dieses Format eignet sich besonders für den klinischen Einsatz, da es dort sehr verbreitet ist und auch wichtige Metadaten (Patientendaten etc.) aufnimmt.

Das Modulinterface ist dabei flexibel genug ausgelegt, um sowohl die Bedürfnisse von einzelbildorientierten Formaten wie JPG zu befriedigen (diese speichern dann in nummerierten Einzelbildern mit zwischen verschiedenen Ansichten synchronisierten Nummern), als auch die von Filmformaten wie MPEG zu erfüllen, die als Sequenz in einer einzigen Datei landen.

Ja es ist sogar möglich, das Speichersubsystem zu nutzen, um die aktuelle Renderansicht über ein schnelles Netzwerk in Echtzeit an einen weiteren Arbeitsplatz zu übertragen (Telediagnostik/Teleradiologie).

3.5.2 Bedien- und Verarbeitungsmodule

Des Weiteren erschien es sinnvoll, die Zahl der verwendbaren Algorithmen nicht a priori durch den Programmumfang zu begrenzen, sondern auch hier eine Modulfunktionalität vorzusehen.

Der Anwender hat dadurch die Möglichkeit, sein System durch schlichtes Hinzukopieren extern geschriebener Module zu erweitern oder auch nicht benötigte Module im Sinne der Übersichtlichkeit zu entfernen.

Solche Kommandomodule haben Zugriff auf die Tastatur- und Mauseingaben sowie die TCP/IP-Verbindungen. Zusätzlich sind alle Variablen und Felder des Hauptprogrammes voll zugänglich.

Letzteres ist zwar aus Sicht der objektorientierten Programmierung nicht unbedingt erstrebenswert, da es den Grundsatz der Kapselung verletzt, aber es ist zur Erreichung einer hohen Verarbeitungsgeschwindigkeit notwendig. Wenn man für jeden Zugriff auf ein Voxel erst eine Abstraktions-/Interfacefunktion aufrufen müsste, würde die Geschwindigkeit zu stark leiden.

Damit ist es sowohl möglich, neue Bedienelemente zu schaffen, als auch neue Kommandos zu definieren, die im Hauptprogramm nicht enthaltene Verarbeitungsschritte ausführen.

Diese Erweiterungen müssen natürlich innerhalb der Renderingengine darstellbar bleiben - eine Erweiterung der Darstellungsmodi ist nicht vorgesehen, da solche Anknüpfpunkte selbst bei Nichtnutzung Geschwindigkeitseinbußen bedeuten würden.

Die Anknüpfung neuer Kommandos und Befehle, die das Volumen verändern, ist aber unkritisch, da diese nur 1x/Bild überprüft werden müssen, und das auch nur, wenn Kommandos anliegen.

Für spezielle Anwendungen können somit leicht und ohne Umwege Funktionen nachgerüstet werden, die so nicht vorgesehen waren.

Auch viele der oben vorgestellten Grundfunktionalitäten sind bereits als Module (zzt. 12 Module mit insgesamt 127 Kommandos) verwirklicht.

3.6 Scripting

Alle Kommandos von VOXREN sind auch über TCP/IP verfügbar, so dass fast alle manuellen Vorgänge, die kein Wissen voraussetzen, automatisiert werden können. Das Kommandointerface benutzt wie in den Internetstandards (RFCs) allgemein

üblich reine Textkommandos mit numerisch/textuellen Rückgabewerten, was die Bedienung sowohl per Hand mit einem beliebigen Telnet-Client ermöglicht, als auch dem automatischen Client entgegenkommt.

Ein solcher Client kann in einer beliebigen, dem jeweiligen Anwender genehmen Programmiersprache verfasst werden und aufgrund der Anbindung über Netzwerk sogar auf einem entfernten System laufen.

3.6.1 Beispiel

Ein einfaches Beipiel für ein solches Script sei hier anhand eines bash-Scriptes, das im Standardumfang von VOXREN enthalten ist, kurz erklärt. Das folgende Script führt eine einfache automatische Segmentierung aus, indem es die Oberfläche der Objekte generiert, deren Grauwert unter 200 liegt:

```
#!/bin/bash
```

[... unwichtige Setup-Teile entfernt ...]

```
netcat -w 1 $HOST $PORT >/dev/null <<EOF
mark 0 65535 0
mark 0 200 2
norm 32
layer 0 1
layer 1 1
layer 2 1
makedist 1
rendermode XYZ
EOF
```

Dieses Script stellt zunächst sicher, dass alle Voxel der Voxelklasse 0 zugeordnet werden. Dann markiert es den Grauwertbereich 0-200 (`mark 0 200 2`) mit der Voxelklasse 2 und lässt die Oberfläche dieser Grauwertmarkierung berechnen und mit Normalen versehen (`norm 32`). Anschließend werden die Grauwertklassen ausgeblendet (`layer`-Kommandos) und die GDA neu berechnet (`makedist 1`).

3.6.2 Umfang und Bedeutung der Scriptschnittstelle

Die Scriptsprache umfasst zur Zeit 127 Kommandos, die im Anhang A aufgelistet sind.

Aus diesem Kommandoumfang bedienen sich ca. 90 im Grundumfang von VOXREN enthaltene Scripte, die diverse einfache Segmentierungsaufgaben übernehmen, Bedienhilfen geben, Filterketten ausführen etc..

Diese Scripte können direkt von der Steuerkonsole VOXCONTROL aus gestartet werden. Sie sind dort nach Einsatzgebiet und Funktion in Dateiordnern kategorisiert.

Solche Scripte können vom fortgeschrittenen Benutzer leicht angepasst oder selbst entwickelt werden, so dass eine Anpassung von VOXREN an individuelle Anforderungen oft durch den Benutzer selbst möglich ist.

Eine noch weitergehende Anpassungsmöglichkeit bietet die Modulschnittstelle (s. 3.5), die aber nur in der Programmierung versierten Anwendern vorbehalten bleiben wird, da sie über ein Grundwissen hinausgehende Kenntnisse in der Programmierung in C verlangt.

Diese Schnittstelle ist in Anhang D dokumentiert.

3.6.3 Interner Einsatz der Scriptschnittstelle

Wie bereits erwähnt, ist die TCP/IP-Schnittstelle eine der zentralen Designideen von $\epsilon\mathcal{CC}\epsilon\mathcal{T}$. Sie ermöglicht die reibungslose Zusammenarbeit streng getrennter Komponenten.

So kommunizieren mit dem in VOXREN implementierten Serverprozess

- VOXCONTROL — zur Steuerung aller wesentlichen Parameter der virtuellen Kamera und der Segmentierungsalgorithmen,
- Scripte (die von VOXCONTROL aus gestartet werden können) — zur Ausführung komplexerer Kommandosequenzen wie z.B. der in 6.4 beschriebenen vollautomatischen Vorverarbeitung und Polypendetektion,
- PLANEVIEW — zum Austausch von Kameraposition, Renderingparametern (Schnittebenen z.B.) und Klassenmarkierungen und
- andere VOXREN-Prozesse im Slave-Modus — die z.B. zur Ferndemonstration oder zur Erzeugung anderer Ansichten (z.B. als „Rückspiegel“) genutzt werden können.

Darüber hinaus können vom erfahrenen Benutzer Kommandosequenzen über VOXCONTROL oder eine Telnet-Sitzung an das System übergeben werden, was besonders bei der Entwicklung neuer Scripte sehr praktisch ist.

3.7 Entwicklungsrichtlinien

Bei der Entwicklung von $\epsilon CC\epsilon\mathcal{T}$ wurde Wert darauf gelegt, das System nach Möglichkeit so auszulegen, dass es einerseits auf der Anwenderseite funktionsreich, bedienbar, schnell und stabil ist und andererseits auf der Entwicklerseite möglichst flexibel, übersichtlich und wartbar ist.

Die folgenden Leitgedanken wurden dabei verfolgt, um diese Ziele zu erreichen:

3.7.1 Vermeidung von Code-Redundanz

Trotz der sehr unterschiedlichen Aufgabenstellungen haben insbesondere die Teilprogramme VOXREN, PLANEVIEW und BATCHVOX erhebliche Gemeinsamkeiten. Daher wurden soweit wie möglich dieselben Datenstrukturen in allen Programmen verwendet. Ebenso wurden mehrfach verwendbare Codeabschnitte in einzelne Module aufgeteilt, deren Interfaces so angelegt sind, dass sie problemlos von allen Teilprogrammen genutzt werden können. Dieses Prinzip gilt insbesondere für aufwändige und ggf. fehlerträchtige Routinen wie das Laden und Speichern von Daten in verschiedenen Formaten.

In einigen Fällen (z.B. beim DICOM-Konverter) wurde solche Funktionalität auch in eigene Hilfsprogramme verpackt, die dann von den Anwendungen einfach per `popen()` verwendet werden.

3.7.2 Bedienung und Benutzeroberfläche

$\epsilon CC\epsilon\mathcal{T}$ sollte sowohl für einen Neunutzer leicht intuitiv bedienbar sein als auch dem erfahrenen Benutzer eine schnelle und effiziente Bedienung ermöglichen. Darüber hinaus sollte es möglich sein, auch komplexe Operationen durchzuführen und diese ggf. später zu automatisieren.

Daher besitzt insbesondere der komplexe Anteil VOXREN die Möglichkeit, schnell und effizient per Tastatur und Maus bedient zu werden. Dies befriedigt zwar die Bedürfnisse eines erfahrenen Benutzers, eignet sich aber nicht für neue Benutzer, da eine solche Bedienung nur sehr bedingt intuitiv ist.

Anwender, denen die Bedienung eines Programmes noch nicht vertraut ist, kommen in der Anfangszeit zumeist mit einer graphischen Benutzerschnittstelle besser zurecht.

Aufgrund des verhältnismäßig hohen Aufwandes, der zur Integration eines GUI-

Toolkits mit der für den Renderinganteil verwendeten schnellen Grafiklibrary LibGGI [GGI] notwendig geworden wäre, wurde diese allerdings nicht in VOXREN selbst realisiert.

Da für VOXREN jedoch auch aus anderen Gründen bereits eine Netzwerkschnittstelle vorgesehen war, lag es nahe, diese zu nutzen, um eine graphische Benutzerschnittstelle anzubinden.
Die in Tcl/Tk geschriebene Steuerapplikation VOXCONTROL (s. Abbildung 3.2) verwendet diesen Mechanismus und realisiert damit ein GUI für VOXREN, das für den neuen Benutzer eine simple Bedienung ermöglicht.

Darüber hinaus wird der Netzwerkzugang zu allen wichtigen Kommandos von VOXREN genutzt, um mit Hilfe von Skripten wiederkehrende Aufgaben zu automatisieren und natürlich um spezielle Aufgaben zu bearbeiten, indem ein menschlicher Experte interaktiv mit diesen Kommandos operiert.

Dem Benutzer wird also sowohl eine Expertenschnittstelle per Tastatur- und Mauskommandos zur Verfügung gestellt als auch eine für den Einsteiger besser geeignete intuitive graphische Oberfläche. Die Trennung von Hauptprogramm und GUI ermöglicht dabei zusätzlich eine flexible Anpassung dieser oft systemabhängigen Komponente.

Des Weiteren wird es so in Zukunft leicht möglich sein, spezielle Anwendungen zu erstellen, die mit einem eingeschränkten GUI auskommen, so dass sich der Lernaufwand für den Benutzer verringert.

3.7.3 Wahl der Programmiersprache und Codierungsrichtlinien

$\epsilon\mathcal{CC}\epsilon\mathcal{T}$ sollte im Rahmen geeigneter Architekturen portabel sein, Multiprozessorsysteme nutzen, in Module zerlegbar sein, deren Code in anderen Teilbereichen einfach wiederzuverwenden ist, und insbesondere in den Visualisierungskomponenten sehr gute Geschwindigkeit bieten.

Daher fiel die Wahl der Sprache für die zeitkritischen Bereiche auf C, da hiermit sowohl eine ausreichende Maschinennähe zur Optimierung möglich ist, aber dennoch die Portabilität gewährleistet bleibt.

Auf eine Optimierung auf niedrigerer Ebene (Einsatz von Assembler oder Nutzung von Hardwarefeatures wie z.B. 3D-Effekten von Grafikkarten) wurde hingegen bewusst verzichtet, da der zu erwartende Gewinn (der durchaus beträchtlich sein kann)

in keinem Verhältnis zum resultierenden Aufwand bei der Portierung auf neue Hardware zu stehen scheint.

Die Unterstützung von parallelen Architekturen erfolgt mittels POSIX-Threads (pthreads), die auf vielen Systemen vorhanden und bewährt sind. Clusterorientierte Lösungen wie PVM wurden nicht in Betracht gezogen, da das zwischen den Maschinen auszutauschende Datenvolumen zu groß wäre.

Die Netzwerkfunktionalität, die zur Kommunikation der Einzelkomponenten sowie zur Anbindung graphischer Oberflächen verwendet wird, wird mit TCP/IP abgewickelt und über das allgemein übliche Unix-socket-Konzept angesteuert.

Zur Grafikausgabe wird LibGGI [GGI] verwendet, eine schnelle, für viele Plattformen verfügbare Grafikbibliothek, die den komplexen Anforderungen von $\epsilon CC\epsilon\mathcal{T}$ standhielt bzw. leicht entsprechend angepasst werden konnte. Vorteilhaft erwies sich hier insbesondere, dass die Library im Source zur Verfügung steht, so dass gefundene Fehler leicht korrigiert werden konnten (die Änderungen sind in der aktuellen Version von LibGGI bereits enthalten).

Einige Subsysteme wie die graphische Shell sowie die Oberfläche VOXCONTROL wurden in Tcl/Tk implementiert.

Diese Interpretersprache ist ebenfalls auf sehr vielen Plattformen verfügbar und erlaubt die leichte und schnelle Erstellung einfacher Graphikoberflächen.

Die Anbindung dieser Teile an das $\epsilon CC\epsilon\mathcal{T}$-System wurde absichtlich relativ „lose“ (via Netzwerkverbindung bzw. durch Starten der anderen Programme mit geeigneten Parametern) realisiert. Damit ist es leicht möglich, später aufwändigere Lösungen unter Verwendung von Widgetbibliotheken zu implementieren und so die Beschränkungen von Tcl/Tk aufzuheben.

Zur Zeit wird an einer LibGGI-basierten Widgetbibliothek gearbeitet, mit deren Hilfe viele Beschränkungen des derzeitigen GUIs aufgehoben werden können.

3.8 Vergleich mit anderen Visualisierungssystemen

Im Rahmen dieser Arbeit hatte ich verschiedentlich Gelegenheit, professionelle Visualisierungssysteme verschiedener Hersteller demonstriert zu bekommen.

Dabei konnte folgender genereller Trend bezüglich der Unterschiede zu $\epsilon CC\epsilon\mathcal{T}$ gefunden werden.

3.8.1 Zielsetzung

$\epsilon\mathcal{CC}\epsilon\mathcal{T}$ wurde als Forschungswerkzeug geschaffen, dessen primäre Zielsetzung die Entwicklung voll- oder halbautomatischer Segmentierungs- und Diagnosetechniken ist. Die Visualisierung ist dabei ein zur Erreichung dieses Zieles hilfreicher Nebenaspekt.

Bei den kommerziellen Systemen steht bisher in vielen Fällen zunächst die Visualisierung im Vordergrund. Funktionen zur Diagnoseunterstützung werden in einigen Fällen als Option angeboten.

3.8.2 Darstellungsqualität

- Systeme der älteren Generation, wie sie zu Beginn der Arbeit üblich waren, lieferten trotz Einsatz spezieller Hardware (typisch SGI oder Sun Grafikworkstations) in der Regel qualitativ schlechtere Bilder als VOXREN und PLANEVIEW.

- Aktuelle Systeme laufen inzwischen oft auch auf leistungsfähigen PC-Systemen (z.B. die Anwendungen von [TNI]), teilweise allerdings auch weiterhin nur auf Spezialhardware [GE] und liefern mit VOXREN und PLANEVIEW vergleichbare, teilweise auch bessere Qualität.

- Die meisten dieser Systeme setzen Volumerenderingtechniken mit Transparenzkurven ein. Diese Technik liefert aufgrund der weichen Entscheidungen für den Menschen angenehm wirkende Darstellungen. Sie erlaubt aber nicht ohne weiteres eine gute Visualisierung eines explizit angegebenen Voxelvolumens.

- $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ liefert mit VOXREN eine für die Routinearbeit akzeptable Darstellungsqualität mit Voxelauflösung. Da eine zuverlässige Auswertung mit Subvoxelgenauigkeit in der Regel ohnehin nicht möglich ist, ergibt sich kein Nachteil.

- PLANEVIEW liefert bei Bedarf Darstellungen mit aus den Grauwertbildern rekonstruierter Subvoxel-Auflösung. Diese Bilder können zur Dokumentation eingesetzt werden bzw. zu Präsentationszwecken. Dabei sollte allerdings bedacht werden, dass hierbei starke Annahmen über das Volumen und das Bildgebungsverfahren einfließen, die im Regelfall nicht nachgewiesen werden können. Diese Problematik betrifft alle Systeme mit subvoxelgenauem Rendering.

Zum Vergleich der Bildqualitäten sind in den Abbildungen 3.6, 3.7 und 3.8 noch einmal einige typische von PLANEVIEW bzw. VOXREN erzeugte Darstellungen zusammengefasst. Abbildungen der Resultate kommerzieller Systeme finden Sie auf folgenden Websites:

GE	http://www.gemedicalsystems.com/it_solutions/rad_pacs/products/aw/efficiency_gallery.html
Siemens	http://www.siemensmedical.com/webapp/wcs/stores/servlet/ProductDisplay?storeId=10001&langId=-11&catalogId=-11&catTree=100001%2C12781%2C12752&level=0&productId=16732&x=9&y=7
Philips	http://www.medical.philips.com/de/products/ct/applications/mxview/clin_img_overview.html
Tiani	http://www.tiani.com/research/re_galleries_images.htm

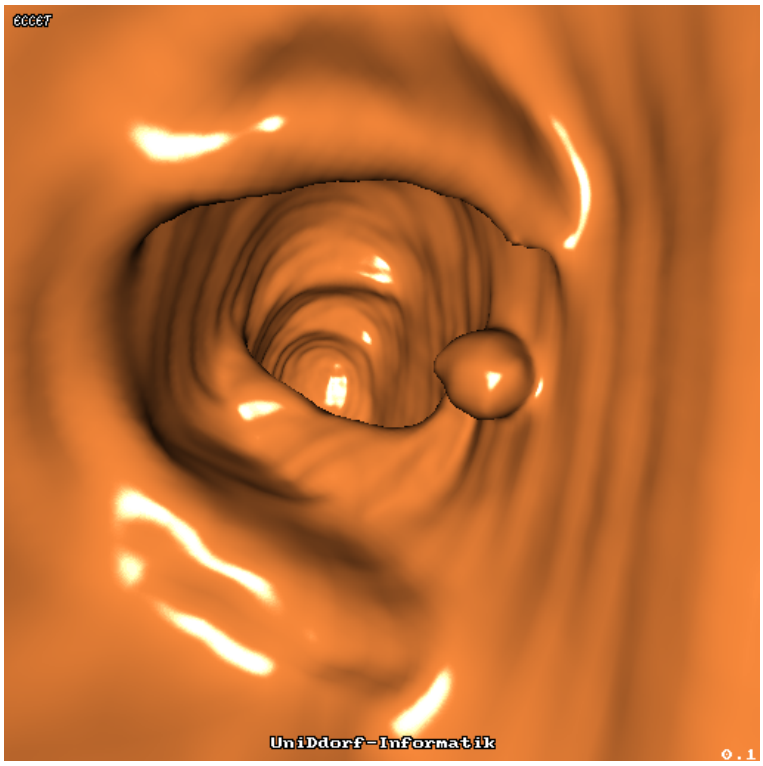
3.8.3 Bedienbarkeit

- Aufgrund der Verwendung aufwändiger Widgetbibliotheken und der Integration von Standardeinstellungen — die VOXREN in seiner Eigenschaft als Forschungswerkzeug nicht in dem Maße nutzen kann — sind die meisten kommerziellen Produkte für einen ungeschulten Anwender leichter zu bedienen als $\epsilon CC\epsilon\mathcal{T}$.

- Nach geeigneter Schulung erlaubt $\epsilon CC\epsilon\mathcal{T}$ allerdings, mit Hilfe der Kommandoschnittstelle eigene, komplexe, neue Verarbeitungsschritte zu entwickeln und damit die Einsatzmöglichkeiten selbst zu verbessern. Im Zuge dieser Entwicklung entstehen jetzt auch mehr und mehr Standardscripte, die die Bedienung für unerfahrene Anwender erleichtern.

- Durch den von den üblichen Systemen verschiedenen Ansatz, nicht die Visualisierung in den Vordergrund zu stellen, sondern primär als Werkzeug zur Entwicklung und Anwendung automatischer Segmentierungs- und Diagnoseverfahren zu dienen, ergibt sich nach der Entwicklungsphase zuweilen ein Vorteil, da die Interaktion des Benutzers mit dem System minimiert wird und er lediglich die Entscheidungen des Systems validieren und ggf. korrigieren muss.

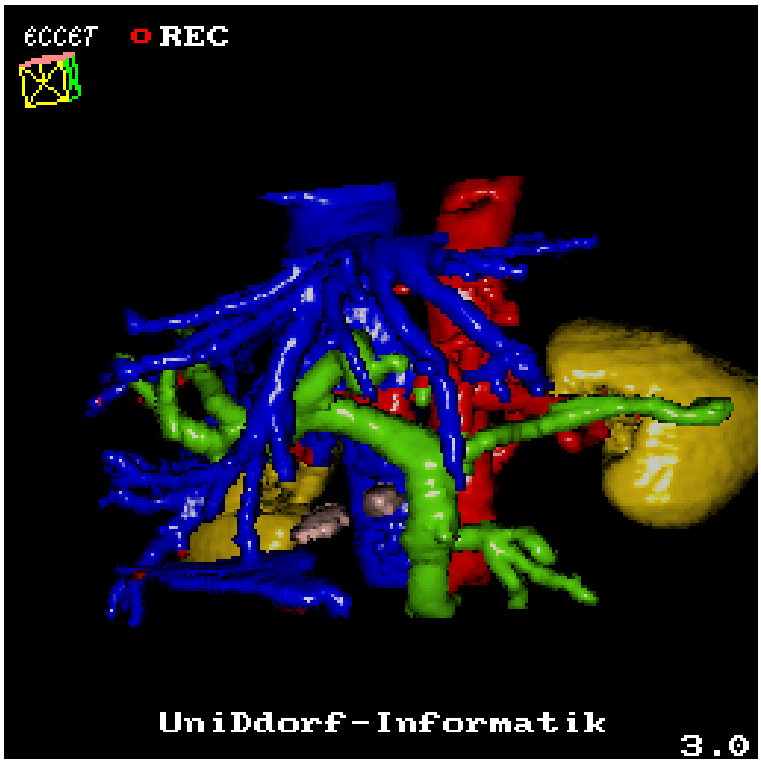
- Insbesondere ist es mit $\epsilon CC\epsilon\mathcal{T}$ möglich, auch bisher in der Software nicht vorgesehene Anwendungen auszutesten und ggf. spezifische Lösungen zu ent-



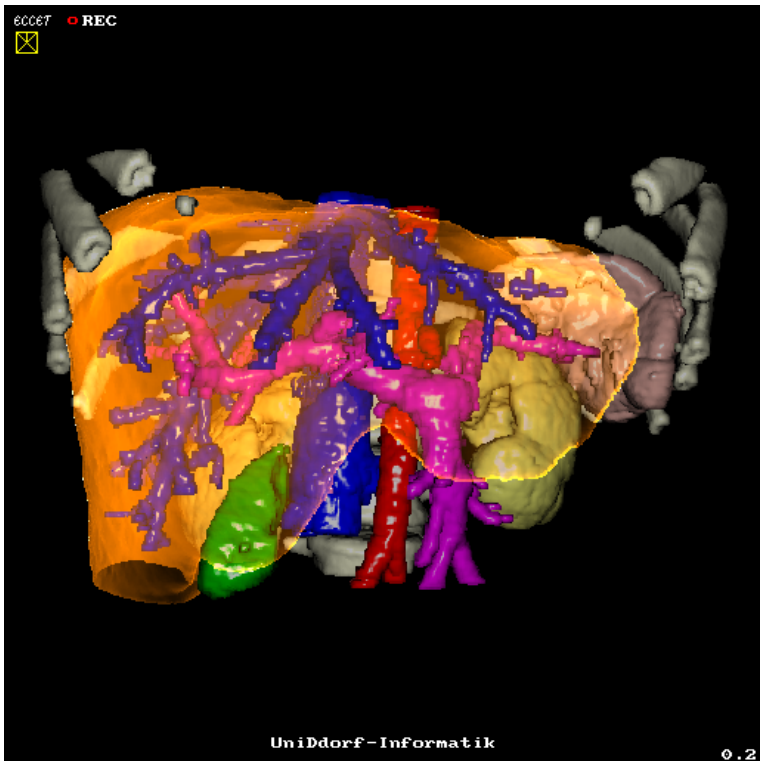
a) Innenansicht Colon mit großem Polyp

b) Außenansicht Colon unter Nutzung einer in VOXREN segmentierten Maske

Abbildung 3.6: Mit PLANEVIEW erreichbare Renderingqualität



a) Gefäßsystem der Leber



b) Nutzung von Depth-Fogging zur Verbesserung des dreidimensionalen Eindruckes

Abbildung 3.7: Mit VOXREN erreichbare Renderingqualität

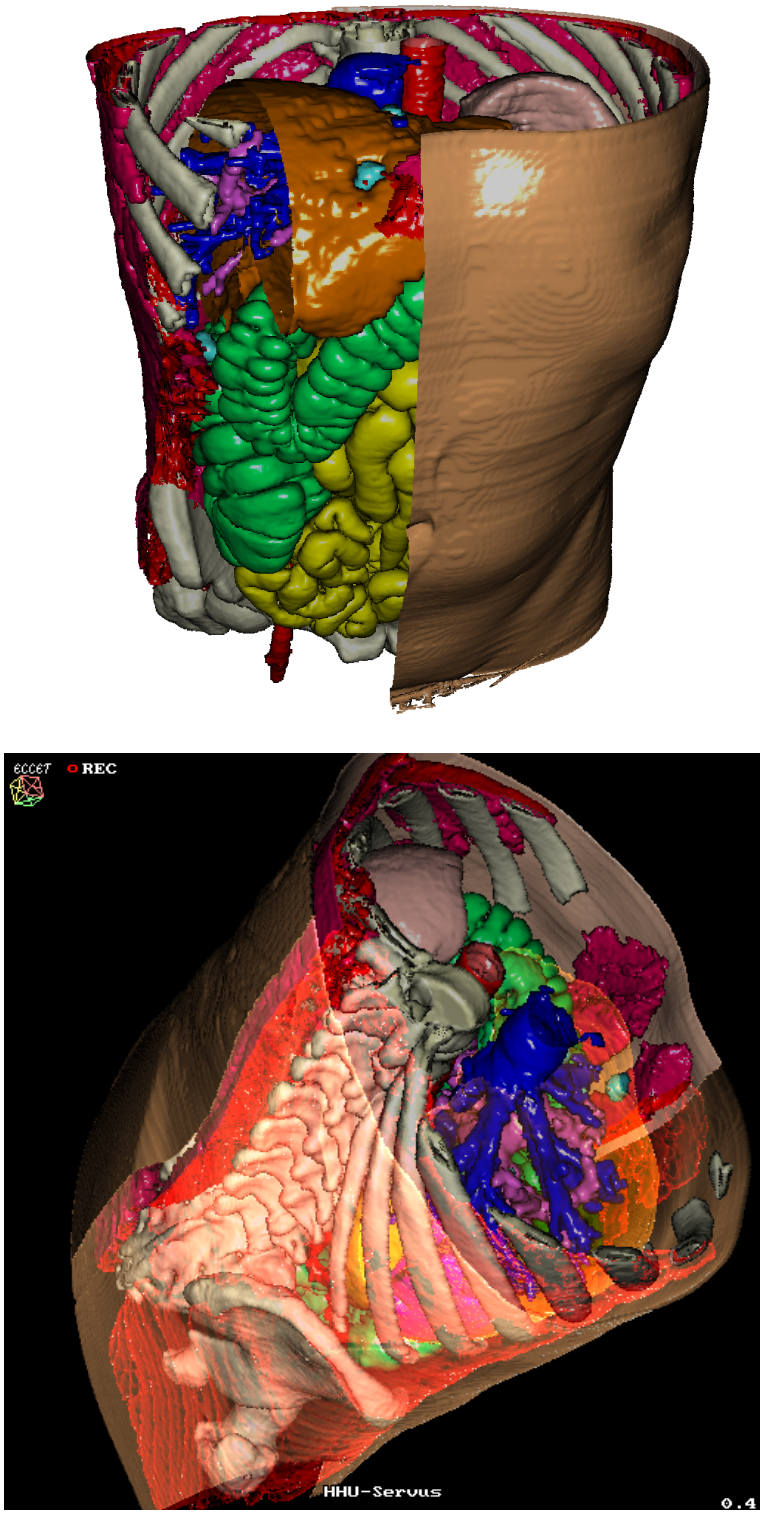


Abbildung 3.8: Komplexe Szenen in VOXREN

wickeln. Die offenen Schnittstellen erlauben es, auch komplexe Lösungen effizient zu integrieren.

3.8.4 Integration in das Arbeitsumfeld

Medizin

- Da die existenten Lösungen zumeist von Anbietern der entsprechenden Tomographiegeräte oder von PACS-Lösungen entwickelt wurden, sind die Schnittstellen zu den Datenquellen dort im Allgemeinen sehr ausgereift und leicht bedienbar.

- Durch die Möglichkeit des DICOM-Importes und der direkten Anbindung an das PACS-System mit Hilfe von JDicom kann $\epsilon CC\epsilon\mathcal{T}$ hier zwar nicht völlig gleichziehen, aber eine für die Praxis ausreichende Anbindung an die Datenquellen erreichen.

- Beim Export erstellter Ansichten in üblichen Dateiformaten kann $\epsilon CC\epsilon\mathcal{T}$ zumeist konkurrieren. Es stehen eine Reihe von Formaten zur Verfügung, unter anderem die gängigen Multimediaformate MPEG und JPG sowie mit Hilfe des nahtlos integrierten ppm2dicom-Konverters von Herrn Hackländer auch das im Medizinumfeld bedeutsame DICOM-Format.

- Die offene Systemplattform Linux erlaubt eine gute Anbindung an externe Stationen zur Verfassung von Berichten bzw. erlaubt auch eine direkte Erstellung selbiger mit gängiger Officesoftware (z.B. Staroffice, OpenOffice oder Applixware).

- Mit Hilfe der zumeist ohnehin vorhandenen TCP/IP-Vernetzung ist ohne Mehraufwand auch eine zusätzliche Darstellung der Anzeige auf einer entfernten Station (Stichwort Teleradiologie) möglich, wobei letztere lediglich einen X-Server benötigt.

Forschung

- Im Bereich der Forschung erlauben die dokumentierten und einfach strukturierten Datenformate von $\epsilon CC\epsilon\mathcal{T}$ eine einfache und schnelle Anbindung an proprietäre Formate, wie sie oft von experimentellen Anlagen erzeugt werden.

- Der geringe Hardwareaufwand (üblicher PC) ermöglicht in vielen Fällen, ohne teure zusätzliche Anschaffungen auszukommen. Lediglich eine RAM-Aufrüstung muss ggf. erfolgen, wenn große Volumina bearbeitet werden sollen. Kleine Volumina (z.B. 256x256x200), wie sie z.B. von MRTs geliefert werden, können mit üblichen Speichergrößen (128MB) problemlos verarbeitet werden.

- Die vielfältigen Darstellungsmöglichkeiten erlauben, eine große Zahl von 3D-Visualisierungsproblemen mit einer einzigen Applikation zu lösen.

  So löste der Einsatz von PLANEVIEW und VOXREN die meisten Visualisierungsprobleme für funktionale MEG-Daten im MEG-Labor der Neurologie der Universitätsklinik Düsseldorf.

3.8.5 Anwendungen

$\epsilon CC\epsilon\mathcal{T}$ eignet sich, nach den bisherigen Erfahrungen, für eine ganze Reihe von Anwendungen — insbesondere in der Forschung und im medizinischen Bereich.

Bevor ich diese jedoch in Kapitel 9 genauer beleuchte, sollen in den folgenden Kapiteln die Algorithmen besprochen werden, die $\epsilon CC\epsilon\mathcal{T}$ zur Visualisierung, Vorverarbeitung, Segmentierung, Navigation und Computerunterstützten Diagnose verwendet, um zunächst die Möglichkeiten von $\epsilon CC\epsilon\mathcal{T}$ an konkreten Beispielen aufzuzeigen.

Kapitel 4

Visualisierungsalgorithmen

Dieses und die folgenden Kapitel stellen die in $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ verwendeten Verfahren und Algorithmen gegliedert nach deren Zielsetzung detailliert vor und zeigen die verschiedenen angewandten Optimierungstechniken auf.

4.1 2D-Visualisierung

„Klassisch“ werden CT-Aufnahmen mit Hilfe von Schichtdarstellungen ausgewertet - entweder auf einem entsprechenden Röntgenfilm oder auf einer Grafikworkstation.

Diese gewohnte Ansicht ermöglich oft eine schnelle Diagnose und liefert manchmal auch mehr Information als eine rekonstruierte 3D-Oberfläche. So kann z.B. nicht nur erkannt werden, **dass** eine Verengung im Darm vorliegt, sondern ggf. auch, **woran** das liegt.

Um den Arzt optimal zu unterstützen, bietet $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ mit PLANEVIEW zur Darstellung von 2D-Schnitten und VOXREN zur Visualisierung von Volumina beide möglichen Ansichten gleichzeitig an.

Allerdings haben klassische Schichtaufnahmen den Nachteil, dass oft die orthogonalen Schichten interessanter wären. Besonders bei stark gerichteten, diese Richtung aber verändernden Strukturen — wie dem Darm — wäre dies wünschenswert.

4.1.1 3-Ebenen-Darstellung

PLANEVIEW bietet dem Betrachter daher die Schnitte in xy-, xz- und yz-Ebene. Andere Ebenenlagen sind mit VOXREN möglich (s. 4.2.1), aber natürlich mit ei-

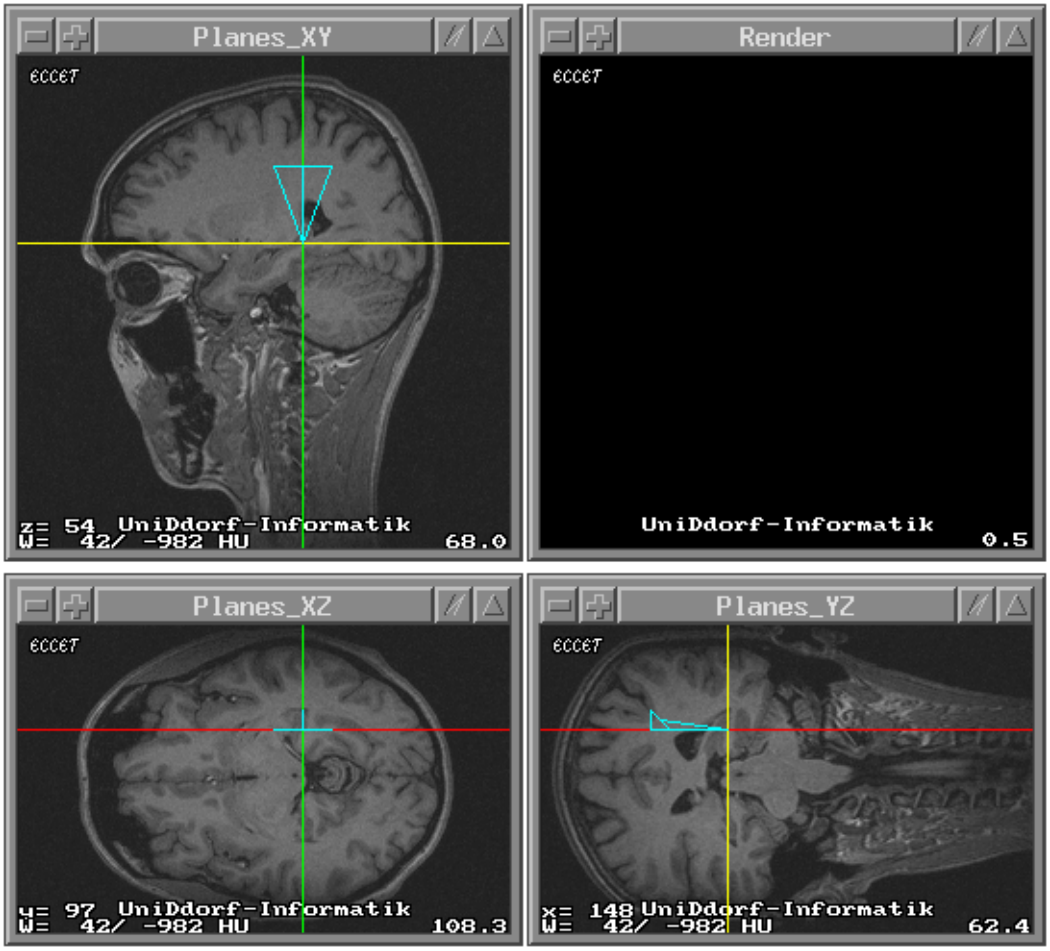


Abbildung 4.1: Die 2D-Darstellung in Planeview

nem Qualitätsverlust (Treppeneffekte bei Schichtwechsel) verbunden und meist auch schwierig zu interpretieren.

Die Anzeige einer solchen Ebene ist programmtechnisch trivial. Man durchläuft einfach eine Ebene des dreidimensionalen Datenfeldes längs zweier Koordinaten und hält die dritte fest.

Im Sinne einer einfachen und logischen Bedienung werden die Ebenen durch einen gemeinsamen Punkt im Datenvolumen gelegt und die Lage der einzelnen Ebenen in den jeweils anderen angezeigt (vgl. Abb 4.1).

Dazu wird den Ebenen jeweils eine Farbe zugeordnet (xy rot, xz gelb, yz grün) und die Ebene in dieser Farbe jeweils auch in den anderen Ebenen als Schnittlinie eingeblendet (abschaltbar, damit die Linie bei der Betrachtung feiner Details nicht stört).

Durch die Anordnung der einzelnen Ebenen in der Form

xy	
xz	yz

sind die beiden jeweils adjazenten Achsen identisch, so dass sich eine durchgehende Linie ergibt.

4.1.2 Konsistente Darstellung bei Benutzung des Zooms

PLANEVIEW erlaubt, die einzelnen Ansichten in ganzzahligen Verhältnissen zu vergrößern.

Die beiden „naheliegenden“ und bei anderen Programmen üblichen Methoden der Implementation eines Zoom-Effektes schienen aber für PLANEVIEW wenig geeignet:

1. Vergrößern des Darstellungsfensters
   Diese Lösung scheidet bei den üblichen Volumengrößen von vorneherein aus. Schon bei einer Vergrößerung um den Faktor 2 wären die Bilder typischerweise 1024x1024 Pixel groß, und somit nicht mehr sinnvoll auf üblichen Bildschirmen anzuordnen.
   Außerdem bestand der Wunsch, *einzelne* Ansichten zoomen zu können, was die Vorteile der in 4.1 gezeigten Anordnung zunichte gemacht hätte. Weder können die Fenster bei unterschiedlichen Vergrößerungen sinnvoll angeordnet werden, noch können die Ebeneneinblendungen bei Bewegung als durchgängige Linie erhalten werden.

2. Vergrößern des Inhaltes mit Fokuspunkt im Zentrum
   Meist möchte man die Umgebung des aktuell markierten Punktes genauer sehen. Es macht daher Sinn, eine vergrößerte Darstellung anzuzeigen, die diesen Punkt im Zentrum hat. Dadurch kann es sich ergeben, dass Punkte jenseits des Randes dargestellt werden müssten. Dieses Problem lässt sich entweder durch Auffüllen mit Dummywerten umgehen oder durch entsprechendes Abschneiden (Clippen) der Position, so dass der Fall nicht auftritt.
   Beide Varianten haben wieder den Nachteil, dass die angenehme Eigenschaft der durchgängigen Achsen zerstört wird.

Für PLANEVIEW wurde daher das in anderen Bereichen ungebräuchliche Verfahren implementiert, als Streckzentrum den aktuell betrachteten Punkt zu wählen.

Da dieser dann Fixpunkt der Abbildung ist, bleiben die Verbindungsgeraden an gleicher Stelle. Bewegt man nun den betrachteten Punkt, so „scrollt“ das Bild unter dem Cursor durch, so dass immer dieser Punkt das Streckzentrum bleibt.

Angenommen die aktuelle Vergrößerung ist n, und der Cursor soll um $\binom{x}{y}$ bezüglich der unvergrößerten Bildpixel bewegt werden: Dazu bewegt man den Cursor wie im unvergrößerten Fall um $\binom{x}{y}$ und gleichzeitig das Bild um $(n-1)\binom{x}{y}$ in die entgegengesetzte Richtung. Dies resultiert in der erwünschten Gesamtbewegung von $n\binom{x}{y}$ auf dem vergrößerten Bild.

4.1.3 Kopplung mit VOXREN

In der zweidimensionalen Darstellung ist es oft schwierig, verschlungene Strukturen (wie den Darm) zu verfolgen, während es in der dreidimensionalen Ansicht oft problematisch ist, geometrische Auffälligkeiten (wie z.B. polypoide Ausbuchtungen) einer Ursache (z.B. Polyp oder Reststuhlklümpchen) zuzuordnen. Daher bietet sich ein kombinierter Einsatz beider Techniken an.

PLANEVIEW kann dazu per TCP/IP mit VOXREN gekoppelt werden und liefert dann die zur aktuellen Kameraposition gehörigen Schnittbilder.

Man sucht nun z.B. in der 3D-Darstellung nach geometrischen Auffälligkeiten (der Darm kann dabei vom Autopiloten durchflogen werden) und betrachtet diese und deren Umgebung ggf. in den von PLANEVIEW präsentierten Schnitten genauer.

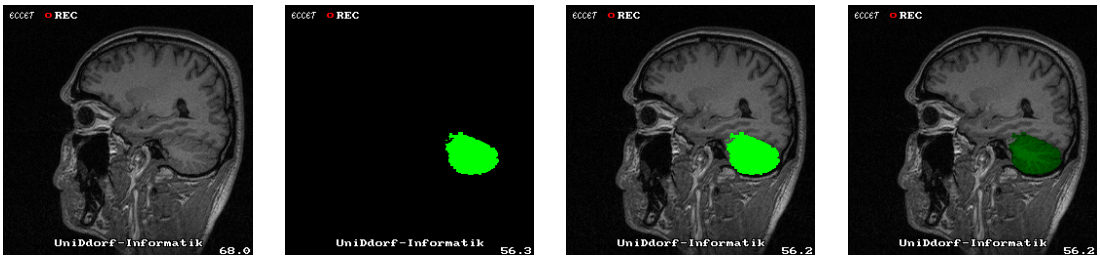
Dazu kann per Hotkey von VOXREN aus der Cursor von PLANEVIEW auf ein beliebiges gerade im Blickfeld befindliches Voxel gesetzt werden. Die wesentlichen Steuerfunktionen (Drehungen, Bewegungen, Autopilot) lassen sich im gekoppelten Zustand wechselseitig bedienen, so dass ein dauernder Wechsel der Eingabegeräte entfällt.

4.1.4 Darstellung von Overlays

Gängige Röntgenbilder nutzen nicht den vollen möglichen Dynamikumfang, den die 16-Bit-Daten in PLANEVIEW bieten. Typischerweise werden ca. 11-12 Bit genutzt.

In die verbleibenden Bits können drei „Overlay-Kanäle“ eingeblendet werden (ein Bit bleibt reserviert), die z.B. wichtige oder verdächtige Strukturen markieren.

Diese Zusatzkanäle können entweder opak angezeigt werden (als feste rote, grüne und blaue Farben - sind mehrere Markierungen aktiv, ergibt sich die entsprechende Mischfarbe), was sehr gut zu finden ist, oder transparent (die Grauwerte bleiben



ohne nur Overlay opaque transparent

Abbildung 4.2: Einblendung von Overlays

erhalten, sie werden nur entsprechend eingefärbt), was den gleichzeitigen Blick auf die eigentlichen Daten erlaubt.

Des Weiteren kann die Anzeige natürlich unterdrückt werden (beste Beurteilbarkeit der Originaldaten) bzw. die Overlays können alleine dargestellt werden (erleichtert das Auffinden kleiner Markierungen). Siehe dazu auch Abbildung 4.2.

Übernahme von Kanälen von VOXREN

VOXREN besitzt eine Reihe von Verarbeitungsmechanismen, wie den Polypendetektor, die es wünschenswert machen, diese Daten auch in der 2D-Ansicht, die zur letztlichen Befundung immer konsultiert werden sollte, sichtbar zu machen.

Daher besitzt PLANEVIEW die Möglichkeit, eine beliebige Voxelklasse aus einem angeschlossenen (s. 4.1.3) VOXREN in eine der drei Overlay-Ebenen zu übertragen.

Zu den Anwendungen dieser Funktion vgl. auch 7.6.

4.1.5 Gamma-Korrektur

Übliche Monitorsysteme besitzen kein lineares Ansprechverhalten auf den anliegenden Spannungswert.

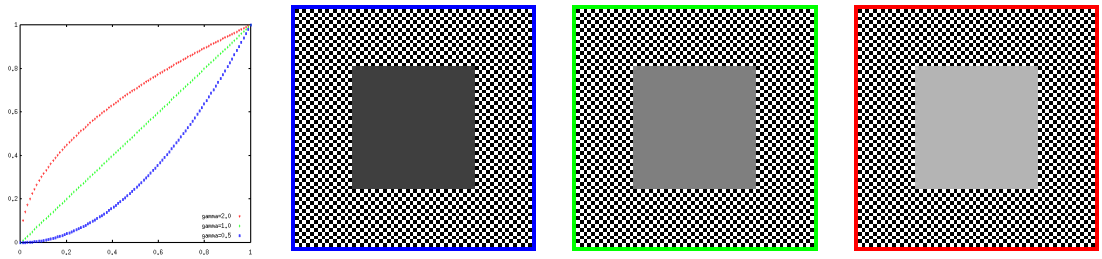
Besonders in der 2D-Darstellung ist es aber wichtig, die Röntgendaten so wiederzugeben, wie sie erhoben wurden.

Besonders auffällig wird der Fehler z.B. im direkten Vergleich von Originaldaten mit gefilterten Daten. Beispiel:

Ein Schachbrettmuster mit den Grauwerten 0 (schwarz) und 1 (weiß) wird mit einem 2x2 großen Rechteckfilter (alle Koeffizienten $\frac{1}{4}$) gefiltert.

Offensichtlich ergibt sich eine konstante Graufläche mit Wert 0,5.

Ist die Darstellung am Monitor nicht linear, wird das gesamte Bild nach der Filterung



Korrekturfunktion $\gamma = 0,5$ $\gamma = 1,0$ $\gamma = 2,0$

Das Bild zeigt ein Schachbrettmuster aus weißen und schwarzen Flächen und darin eine 50%-Graufläche. Bei korrekter Gammakorrektur des Ausgabesystems sollten beide Bereiche aus größerer Entfernung gleich hell erscheinen.

Abbildung 4.3: Gammakorrektur

heller bzw. dunkler erscheinen, was den fälschlichen Eindruck erweckt, das Filter erhielte die durchschnittliche Helligkeit nicht.

Üblicherweise werden die Nichtlinearitäten näherungsweise durch eine sog. Gammakorrektur (s. Abbildung 4.3) ausgeglichen.

Diese Korrektur erfolgt mit Hilfe der Funktion

$$U_{out} := U_{in}^{\frac{1}{\gamma}} , \tag{4.1}$$

wobei $U_{in,out} \in [0..1]$, so dass die Abbildung stetig ist und schwarz (0) und weiß (1) als Fixpunkte erhält.

Dieser Gammafaktor kann — gemäß obiger Überlegung mit dem Rechteckfilter — leicht bestimmt werden. Man erzeugt dazu ein Testbild (für PLANEVIEW verfügbar), das aus einem solchen Schachbrettmuster und einem mittelgrauen Feld in der Mitte besteht.

Nun justiert man den Gammafaktor in PLANEVIEW so lange, bis beide Bildanteile gleich hell erscheinen.

Streng genommen müssten natürlich noch weitere Stützpunkte für die Ausgleichskurve ermittelt werden, was aber dann über eine Gammakorrektur hinausginge. In der Praxis genügt diese recht einfache Anpassung in der Regel den Anforderungen.

4.2 3D-Visualisierung

Dieses Kapitel fasst die Methoden zusammen, mit denen VOXREN und PLANEVIEW Projektionen dreidimensionaler Objekte erzeugen.

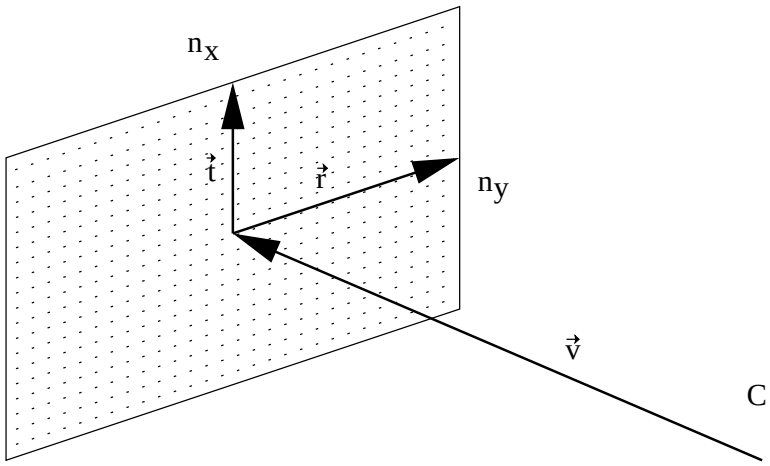


Abbildung 4.4: Kameramodell beim Raycasting

4.2.1 Raycasting

VOXREN und PLANEVIEW verwenden eine Zentralprojektion und Raycasting zur Visualisierung dreidimensionaler Voxelmengen.
Abbildung 4.4 verdeutlicht die Funktionsweise dieses Verfahrens:

1. eine virtuelle Kamera C wird im Raum positioniert.

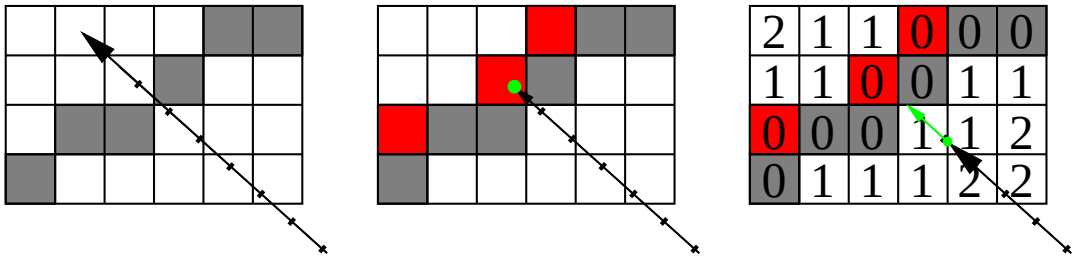
2. Ein Sichtvektor $\vec{v}$ mit $|\vec{v}| = 1$ wird gewählt. Dieser bestimmt die „Blickrichtung“ der virtuellen Kamera.

3. Der Blick der Kamera steht nun bis auf Rotation um die von $\vec{v}$ bestimmte Achse und den Öffnungswinkel fest. Diese beiden Parameter legt der Vektor $\vec{r}$ mit $\vec{r} \perp \vec{v}$ fest.

4. Senkrecht zu $\vec{r}$ und $\vec{v}$ wird nun noch $\vec{t}$ bestimmt, wobei $\vec{t} \,||\, \vec{v} \times \vec{r}$ und der Quotient von $|\vec{r}|$ und $|\vec{t}|$ gerade dem gewünschten Aspektverhältnis des Bildes $\frac{n_x}{n_y}$ entspricht.

Die von $\vec{r}$ und $\vec{t}$ aufgespannte Ebene wird nun der gewünschten Auflösung (n_x, n_y) des zu erzeugenden Bildes entsprechend mit einem diskreten Raster $\vec{R}(z_x, z_y)$ belegt:

$$\vec{R}(z_x, z_y) = \left[\left(\frac{z_x}{n_x - 1} - \frac{1}{2} \right) \vec{r} + \left(\frac{z_y}{n_y - 1} - \frac{1}{2} \right) \vec{t} \right] \quad \text{mit } z_x \in \{0, 1, ..., n_x - 1\} \text{ und } z_y \in \{0, 1, ..., n_y - 1\} \tag{4.2}$$



a) mit dünnen Wandungen

b) mit geeignet verdickten Wandungen

c) mit GDA und exakter Schnittpunktsberechnung

Abbildung 4.5: Mögliche Bildfehler durch diskrete Schrittweite

Berechnung der sichtbaren Voxel

Geometrisch gesehen besteht nun die Aufgabe darin, für jeden Vektor $\vec{v}_{xy}$ in $\vec{R}$ den ersten Schnitt (s minimal) der vom Standpunkt der Kamera ($\vec{c}$) ausgehenden Halbgeraden

$$\vec{S} = \vec{c} + s\vec{v}_{xy} \ , \ s \geq 0, \tag{4.3}$$

mit der gegebenen Voxelmenge zu bestimmen. Wir werden diese Halbgerade im Folgenden „Sehstrahl“ nennen. Weitere Schnitte (mit größerem s) sind zunächst nicht von Interesse, da unter der Annahme opaker Voxel diese Bereiche verdeckt sind.

Im klassischen Raytracing-Verfahren werden hierzu Methoden der linearen Algebra verwendet, um die Lösungen der entsprechenden Gleichungen zu bestimmen, da hier die Szenenbeschreibung als Parameterdarstellung der zu rendernden Oberflächen vorliegt.

Im vorliegenden Fall besteht die Szenenbeschreibung jedoch aus auf einem regulären Gitter vorliegenden Messwerten, die als Zentrum eines entsprechenden Quaders (Voxel) aufgefasst werden.

Eine Schnittberechnung mit allen vorliegenden Quadern ist hier zu aufwändig. Bei einer typischen Anwendung mit 800x512x512 Voxeln á 6 Seitenflächen ergäben sich ca. 1,3 Milliarden Schnitttests Gerade/Quadrat, und das nur für den Test eines einzigen Sehstrahls. Für ein 256x256 Pixel großes Bild ergäben sich insgesamt ca. $8,2 \cdot 10^{13}$ Tests.
Selbst wenn man annimmt, dass die meisten dieser Tests sehr früh abgebrochen werden könnten, bleiben noch deutlich zu viele Möglichkeiten übrig.

Stattdessen „springt“ man in hinreichend kleinen Schritten ϵ_s an der durch Gleichung 4.3 gegebenen Halbgeraden entlang, indem man $s(n) = \epsilon_s \cdot n \ , n \in \{0, 1, ...\}$ dort

für s einsetzt. Dabei prüft man an jedem Punkt $\vec{S}(n)$ (Samplepunkt), ob er sich innerhalb eines „gefüllten“ Voxels befindet.

Abbildung 4.5 a) illustriert, dass dieses Vorgehen eine fehlerhafte Abbildung verursachen kann, wenn die betrachtete Voxelmenge nicht eine ausreichend dicke Schicht bildet. Im dargestellten Fall ist der Abstand zwischen den Punkten, an denen die Überprüfung erfolgt, zu groß, so dass eine Durchdringung nicht bemerkt wird. Dieser Fehler kann aber durch geeignete Generierung der Voxelmenge für Wandungen vermieden werden, indem man dafür sorgt, dass es keine diagonalen Durchtrittspunkte gibt (s. Abbildung 4.5 b).

Allerdings bleibt ein gewisser kleiner Abbildungsfehler, da ggf. bei kleinen Bewegungen einmal die „vordere“ und einmal die „hintere“ Schicht getroffen wird. Besonders auffällig ist dieser Effekt an vorspringenden Kanten wie Darmfalten. Er wird allerdings bei interaktiver Bewegung des Objektes kaum wahrgenommen.

Bei der Generierung von Filmen mit geringer Schrittweite entsteht jedoch ggf. ein unschöner „Welleneffekt“. Dieser kann durch exakte Schnittpunktsberechnung (s. unten) — die allerdings auf Kosten der Rendergeschwindigkeit geht — unterdrückt werden.

Nutzung der Bounding-Box

Befindet sich die virtuelle Kamera weit außerhalb des die Voxelmenge enthaltenden Quaders (der sogenannten Bounding-Box), so würde dieses Verfahren sehr viel Zeit damit verschwenden, sich dem Gebiet, in dem überhaupt Voxel vorkommen können, zu nähern. Dieser Fall kann leicht durch einfache Koordinatentests erkannt werden, und mit Hilfe von maximal 3 Schnittberechnungen zwischen aktuellem Sehstrahl und den entsprechenden Begrenzungsflächen des Voxelquaders kann der uninteressante Bereich schnell übersprungen werden.

Allerdings ist auch dann die resultierende Anzahl von Tests (die zumeist sehr einfach sind, da man in der Regel nur eine einfache Eigenschaft der Voxel aus dem Voxelfeld liest) noch sehr hoch: $n_x \cdot n_y \cdot \overline{n_s}$. Dabei ist $\overline{n_s}$ die durchschnittliche Schrittzahl bis zur ersten Oberfläche. $\overline{n_s}$ hängt stark von der gewählten Szene ab.

Nutzung der geometrischen Distanzabbildung

Diese Schrittzahl kann in typischen Szenen, die nur sehr dünn mit darzustellenden Voxeln belegt sind (Oberflächen) drastisch durch Einsatz der geometrischen

a) Nutzung der Bounding-Box (blau dargestellt)

b) Anzahl Schritte ohne Verwendung der GDA (weiß=375)

c) Anzahl Schritte mit GDA (5fach verstärkt gegenüber Abb. b), weiß=75)

Abbildung 4.6: Optimierungen mit Bounding Box und GDA

Distanzabbildung (GDA) reduziert werden.

Ist die Verteilung der zu visualisierenden Voxel a priori bekannt, so kann für alle **anderen** Voxel der Abstand d zum nächstgelegenen darzustellenden Voxel vorberechnet werden.

Da innerhalb dieses Abstandes per Definition kein darzustellendes Voxel liegen kann, kann man sicher um $d\frac{\vec{v}_{xy}}{|\vec{v}_{xy}|}$ fortschreiten, ohne ein darzustellendes Voxel zu übersehen.

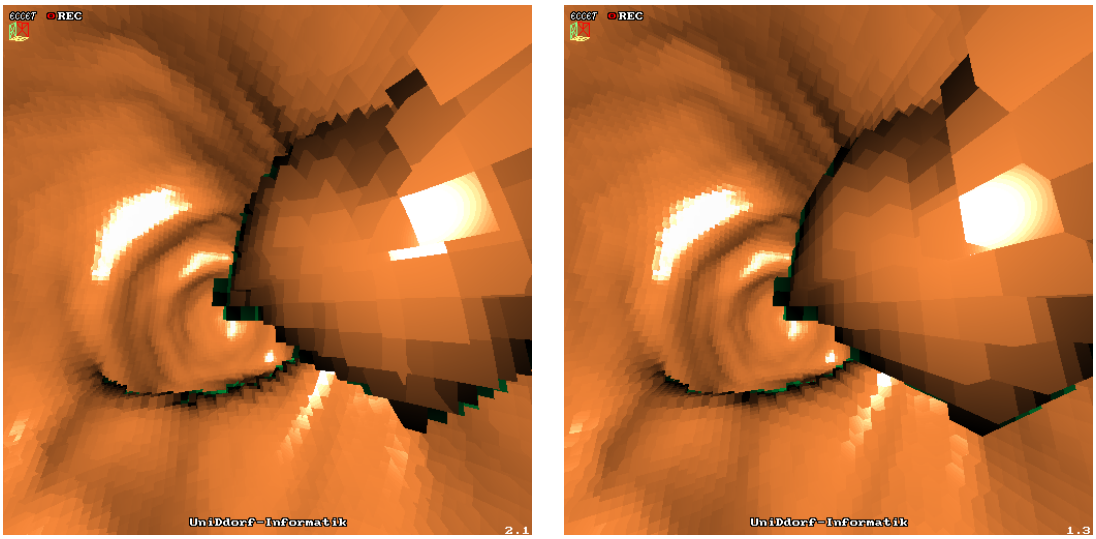
Das erfordert allerdings zusätzlichen Speicheraufwand, der bei VOXREN mit einem Byte/Voxel dimensioniert ist. Darin lassen sich zwar nur Abstände bis 255 kodieren, was aber kein praktisches Problem darstellt, da man einfach bei 255 abschneiden kann, ohne signifikante Geschwindigkeitseinbußen hinnehmen zu müssen.

Leider ist eine **exakte** Berechnung der GDA in der euklidischen Metrik sehr rechenaufwändig — nicht jedoch in den von der 1- bzw. ∞-Norm induzierten Metriken, für die jeweils effiziente Verfahren existieren, die noch in 5.3 genauere Beachtung finden werden.

Für das gegebene Problem ist eine Berechnung in der euklidischen Norm auch gar nicht nötig, bzw. bei geeigneter Wahl des Fortschrittsvektors $s_1 \cdot \vec{v}_{xy}$ (man wählt ihn gerade so, dass die größte Komponente im Koordinatensystem der Voxelmenge gerade 1 wird) eignet sich die Maximumsnorm sogar besser.

Die Verwendung dieser Optimierung senkt die Anzahl der nötigen Schritte erheblich (s. Abbildung 4.6 b und c) und bringt bei typischen Szenen eine ca. fünffach höhere Framerate. Sie stellt damit eine der wichtigsten algorithmischen Optimierungen in den Renderingroutinen von VOXREN dar.

Als Nachteil soll nicht unerwähnt bleiben, dass natürlich Änderungen an der Szenerie



a) Aliasing zwischen Voxelgitter und Abtastgitter der Kamera erzeugt „krumme“ Voxelkonturen

b) Mit exakter Schnittpunktsberechnung erscheinen die Voxel korrekt als kleine Würfel.

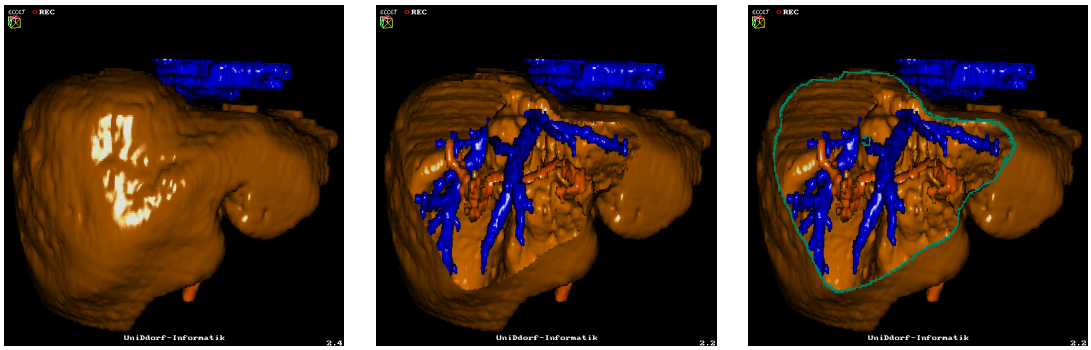
Abbildung 4.7: Exakte Schnittpunktsberechnung

im Allgemeinen eine Neuberechnung der GDA erfordern.

Exakte Schnittpunktsberechnung

Wie bereits ausgeführt, kann es durch das konstante Vorwärtsschreiten mit endlicher Schrittweite zu Fehlern beim Rendering kommen. Diese äußern sich durch „krumme“ Übergänge zwischen Einzelvoxeln und verzerrte Ränder an Außenkanten von Objekten (s. Abbildung 4.7a). Aufgrund der diskreten Schrittweite werden Voxel insbesondere am Rand von Objekten bei streifendem Einfall abhängig von der genauen Lage des Startpunktes des Sehstrahls getroffen oder nicht. Die Lösung gemäß Abbildung 4.5 durch Verdicken der Wandung funktioniert nicht, wenn das Objekt nur gestreift wird. Besonders auffallend ist dieser Effekt, wenn man zwei Bilder mit minimal veränderter Kameraposition kurz hintereinander präsentiert bekommt, wie es z.B. bei langsamer Vorwärtsbewegung der Kamera in Filmen der Fall ist. Durch die Interferenz zwischen dem von der Voxelmenge induzierten regulären Gitter und dem kugelförmig gebogenen Abtastgitter kommt es dabei zu wellenförmigen Artefakten.

Dieser sehr störende Effekt kann durch ein geeignetes Fortschreiten längs des Sehstrahles weitgehend ausgeschaltet werden. Mit Hilfe der GDA wird nun nicht **direkt** bis zum Objekt gesprungen, sondern jeweils nur bis die Distanz dazu 1 ist ($\text{gda}(\vec{x}) = 1$). Ist man dort angelangt, so wird der nächste Schnittpunkt mit einer



a) Der Blick auf das Gefäßsystem ist durch die Leberoberfläche verdeckt.

b) Einführen einer Schnittebene macht es sichtbar.

c) Bessere Anschaulichkeit bei markiertem Schnittrand

Abbildung 4.8: Nutzung einer Schnittebene zur Entfernung unerwünschter Vordergrundstrukturen

Voxel-Begrenzungsfläche **berechnet**.

Der nächste Schritt erfolgt nun mit der berechneten Entfernung bis zum Schnitt plus einer kleinen Konstante, um den Punkt sicher **innerhalb** des Voxels zu platzieren und Rechenungenauigkeiten bei der Bestimmung des getroffenen Voxels zu vermeiden. Dadurch wird erreicht, dass das sich dort ggf. befindliche Voxel gerade auf einer Seitenfläche getroffen wird (s. Abbildung 4.5 c). Man beachte, dass sich dort nicht unbedingt ein darzustellendes Voxel befinden muss, z.B. wenn der Sehstrahl gerade parallel zu einer Voxelfläche ausgerichtet ist.

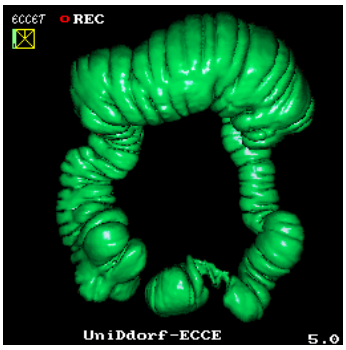
Durch diese Modifikation werden die Voxel als exakte Würfel dargestellt und der Interferenzeffekt zwischen Abtastgitter und Voxelgitter verschwindet (s. Abbildung 4.7b).

Schnittebene zur Ausblendung unerwünschter Strukturen

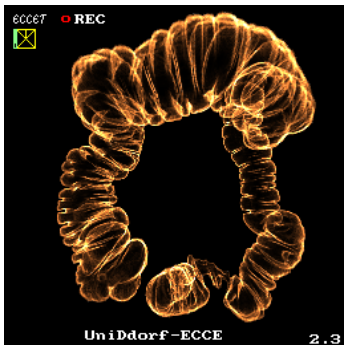
Beginnt man mit der Suche nach Schnittpunkten erst in einer gewissen Entfernung von der Kamera bzw. beendet man die Suche in einer gewissen Entfernung, so ist es möglich, eine vordere und hintere Schnittebene einzubauen, die einen Blick hinter störende Strukturen im Vordergrund erlaubt.[1]

Allerdings erweist es sich oft als verwirrend, wenn die dadurch entstehenden

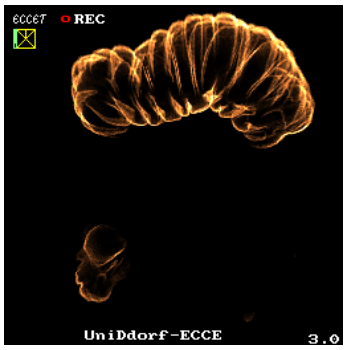
[1] Damit sich eine Schnitt**ebene** ergibt, ist hier mit „Entfernung“ die Entfernung von der Kameraebene und nicht von der Position der Kamera gemeint. Letzteres würde Kugelflächen als Schnittfiguren erzeugen. Die Entfernung eines Punktes von der Kameraebene kann leicht über das Skalarprodukt des Differenzvektors Kamera→Punkt mit einem normierten Vektor in Blickrichtung der Kamera berechnet werden.



a) Außenansicht eines Dickdarms



b) Simulierter Kontrasteinlauf. Hintereinanderliegende Schlingen erzeugen unübersichtliche Stellen (rechts oben im Bild).



c) Eine hintere Schnittebene hilft hier die Bereiche zu trennen.

Abbildung 4.9: Nutzung der hinteren Schnittebene

Schnittkanten nicht gesondert markiert werden (s. Abbildung 4.8 b). Daher bietet VOXREN, wie dies auch bei technischen Zeichnungen üblich ist, die Option, Schnittkanten mit einer Sonderfarbe hervorzuheben.
Sie wird technisch realisiert, indem beim **ersten** Kontakt mit dem Voxelvolumen (also direkt nach dem initialen Sprung) aufgefundene Voxel mit der Sonderfarbe markiert werden (vgl. Abbildung 4.8c).

Analog kann eine **hintere** Schnittebene eingeführt werden, mit deren Hilfe auch störende Objekte im Hintergrund entfernt werden können.

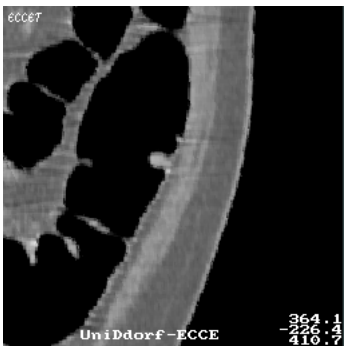
Bei opaken (undurchsichtigen) Darstellungen stellen jene Objekte selten ein Problem dar (außer gelegentlich bezüglich der Tiefenzuordnung bei Gleichfarbigkeit), aber bei semitransparenten Darstellungen (siehe Abbildung 4.9) kann eine Struktur im Hintergrund sehr wohl stören.

Darstellung beliebig orientierter Schnitte in Grauwertvolumina

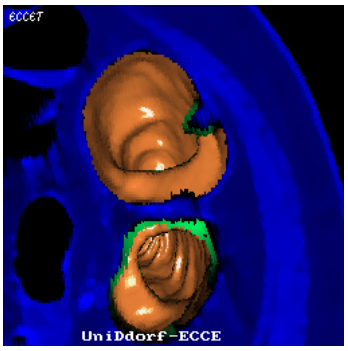
Gelegentlich ist es nützlich, beliebige Schnitte in Grauwertvolumina betrachten zu können, z.B. einen Schnitt quer zu einer physiologischen Struktur, z.B. einem Stück Darm, oder ein Schnitt durch einen Polypen.

Abbildung 4.10 zeigt einen solchen Schnitt mitten durch einen Darmpolypen.

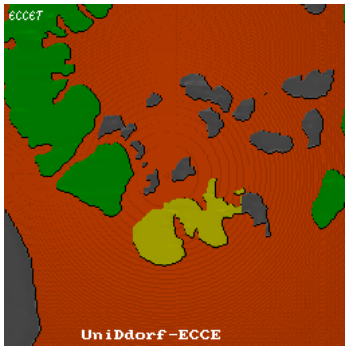
Dazu wird derselbe Mechanismus benutzt wie beim 3D-Rendering mit Schnittebene. Zusätzlich wird beim ersten Schnitt-Test geprüft, ob ein ggf. angetroffenes Grauwertvoxel (GDA $\neq 0$) dargestellt werden soll. Falls ja, wird keine Oberflächendar-



a) Schnitt durch einen Polypen im Dickdarm



b) Derselbe Polyp, anderer Blickwinkel, Innenvolumen des Darms ausgeblendet



c) Nutzung der Schnittebene bei der Segmentierung

Abbildung 4.10: Beliebig orientierte Schnittebene in Grauwertvolumina

stellungsfunktion aufgerufen, sondern direkt der gefundene Grauwert eingezeichnet.

Dabei ist pro Voxelklasse (die Klassen 0-31 sind für Grauwertvoxel reserviert) steuerbar, ob sie dargestellt werden soll. Damit ist es möglich, z.B. selektiv nur das Darm**äußere** zusätzlich einzublenden und im Innenraum weiterhin die Darmwand in 3D darzustellen (siehe Abbildung 4.10b).

Diese Funktion wird auch während der interaktiven Segmentierung genutzt. Sie erlaubt z.B. geeignete Saatpunkte für Füllalgorithmen zu setzen, Grauwerte zu vermessen etc. (s. Abbildung 4.10c). Durch die farbige Darstellung der verschiedenen Voxelklassen wird die Arbeit mit mehreren Bereichen, Ausschlusszonen etc. übersichtlicher.

Partielle Ausblendung von Strukturen

Zuweilen genügen Schnittebenen senkrecht zur Kameraachse nicht, um den gewünschten Visualisierungseffekt zu erzielen.

Es wäre wünschenswert, **einzelne Objekte** durchschneiden zu können, ohne andere Objekte dabei zu stören. Dies entspricht eher der Analogie der Explosionszeichnung.

Des Weiteren wäre eine **beliebige Lage** der Schnittebenen wünschenswert sowie eine Kombinierbarkeit mehrerer Ebenen zu einem komplexeren Schnitt.

VOXREN bietet die Möglichkeit bis zu drei beliebig orientierte Schnittebenen zu definieren, die einzeln beliebigen Objekten zugeordnet werden können. Werden einem Objekt mehrere Ebenen zugeordnet, so wird das logische UND der beiden von den Ebenen definierten Halbräume dargestellt, alternativ kann auch das logische ODER dargestellt werden.

Zwei frei orientierte Schnittebenen gewähren Einblick in das Innere der Leber.

Abbildung 4.11: Frei definierbare Schnittebenen

Damit wird es möglich, verdeckte Strukturen sichtbar zu machen und gleichzeitig andere Bereiche des störenden Objektes zur Orientierung sichtbar zu halten (s. Abbildung 4.11).

Technisch realisiert wird diese Möglichkeit folgendermaßen:

- Für jedes potentiell zu zeichnende Voxel stelle fest, welcher Klasse es angehört.
- Falls für diese Klasse Schnittebenen aktiv sind, prüfe für jede aktive Schnittebene, ob sich das Voxel im von der Schnittebene und ihrer Orientierung definierten Halbraum befindet.
- Treffen alle Schnittbedingungen zu, so zeichne das Voxel ein.
- Ansonsten ignoriere das Voxel und suche weiter.

Da diese Tests nur stattfinden, wenn auch Schnittebenen aktiv sind, gehen sie nicht signifikant auf Kosten der Darstellungsgeschwindigkeit.

a) RGB-Modus von VOXREN

b) Mit der Methode aus 9.1 aufgenommenes Reliefbild

Die Textur und Beleuchtung wurde jeweils aus einem Videobild der Vorlage übernommen und ist statisch.

Abbildung 4.12: RGB-Modus von VOXREN

4.2.2 Oberflächendarstellung (Shading)

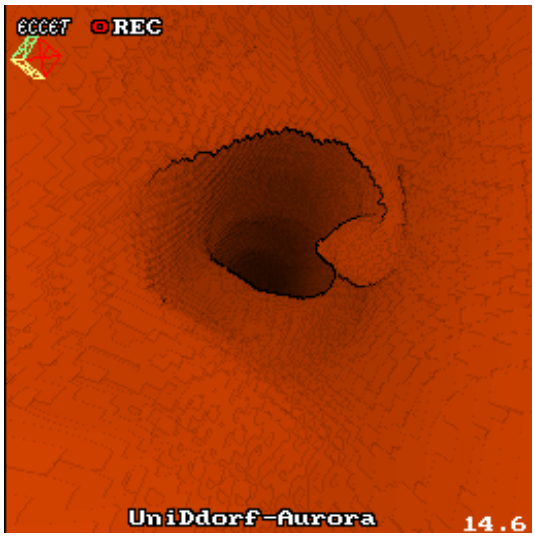
Hat man den gesuchten Punkt gefunden, stellt sich die Frage, mit welcher Farbe das zugehörige Pixel im Bild gezeichnet werden soll.

1. **Vorberechnete Beleuchtung**
   Im einfachsten Fall könnte man das Beleuchtungsmodell komplett vorberechnen und einfach den entsprechenden Farb- und Helligkeitswert (z.B. als RGB-Tripel) abspeichern.

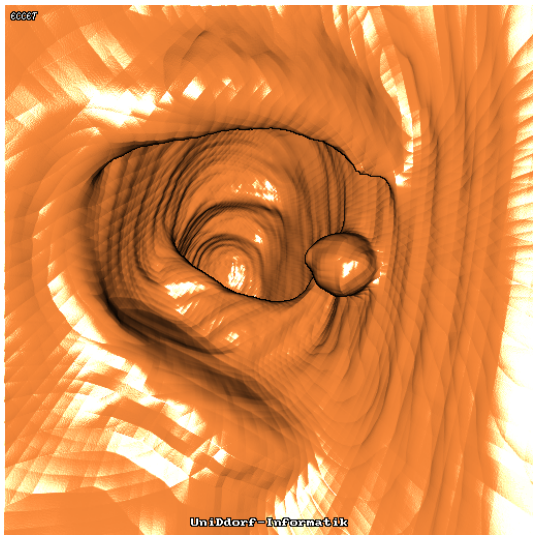
   VOXREN besitzt einen solchen Modus (RGB bzw. QRG), der allerdings für die klinische Praxis wenig relevant ist. Nur mit einer fest vorberechneten Beleuchtung zu arbeiten, erweist sich dort als unpraktisch, da man dann eben nicht durch Bewegen der Lichtquelle weiteren Aufschluss über unklare Strukturen erhalten kann. Des Weiteren soll ja die gewohnte Umgebung so weit wie möglich simuliert werden, was für eine der Hauptanwendungen — die virtuelle Endoskopie — eine Lichtquelle in direkter Nähe der virtuellen Kamera (Endoskopkopf) fordert.

   Dennoch sind im Bereich Virtual Reality/Augmented Reality Anwendungen für diese Shadingvariante denkbar.

   Zu Demonstrationszwecken existiert ein 3D-Scanning-Tool, das mit Hilfe eines



b) ZSH-Modus von VOXREN. Die Schattierung wird on-the-fly aus der Tiefenkarte des gerenderten Bildes berechnet.



c) ZSH-Modus von PLANEVIEW. Durch subpixelgenaue Berechnung einer 3D-interpolierten Tiefenkarte wird ein erheblich besseres Shading erreicht.

Abbildung 4.13: ZSH-Modus von VOXREN und PLANEVIEW

herkömmlichen steuerbaren Projektors (Beamer) und einer an den Rechner angeschlossenen Kamera (Videokamera und Framegrabber oder direkt angeschlossene Kamera) in der Lage ist, dreidimensionale Objekte als Relief zu rekonstruieren (s. Abschnitt 9.1). Dabei wird die mitaufgenommene Originaltextur der Vorlage über den RGB-Modus berücksichtigt (s. Abbildung 4.12).

2. **Z-Shading**

   Ein einfaches Verfahren, das keine weitere Information benötigt, ist das sog. Z-Shading (VOXREN-Modus ZSH). Es verwendet die gefundenen Schnittkoordinaten, um eine Tiefenkarte anzulegen, aus der anschließend mit Hilfe eines modifizierten Differenzenfilters, der die aktuelle Tiefe in Betracht zieht, die jeweilige „Neigung“ an einer Stelle des Bildes geschätzt wird.

   Dazu berechnet man den Abstand Δx, den zwei benachbarte Pixel in der Distanz z von der Kamera hätten (vgl. Abb 4.14).

   Zur Vereinfachung nehmen wir an, dass $\overline{C\,P1} \parallel \overline{C\,P3}$, also dass die Kamera im Verhältnis zu ihrer Auflösung relativ weit vom betrachteten Objekt entfernt ist. Damit gilt dann auch $\overline{C\,P1} \approx \overline{C\,P3} \approx$ z und $\alpha \approx \beta \approx 90° - \gamma$ und damit auch $\sin\alpha \approx \cos\gamma$.

   Ist nun Δz die Differenz der Tiefen der Voxel $P1$ und $P2$ in der Tiefenkarte, so gilt für den Quotienten $\frac{\Delta x}{\Delta z} \approx \tan(\alpha) \approx \sin(\alpha) \approx \cos(\gamma)$. Das heißt, er liefert

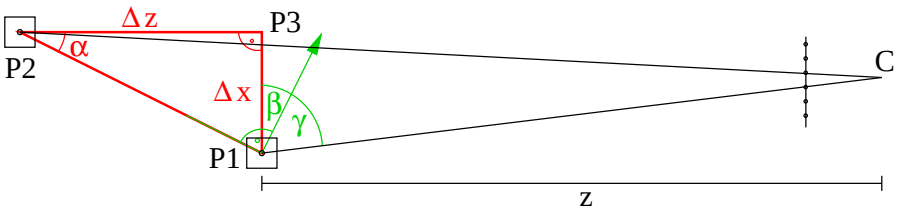


Abbildung 4.14: Z-Shading

eine Näherung für den Cosinus des Winkels γ zwischen dem Sehstrahl und der entsprechenden Normalen auf der Strecke $\overline{P1\,P2}$.

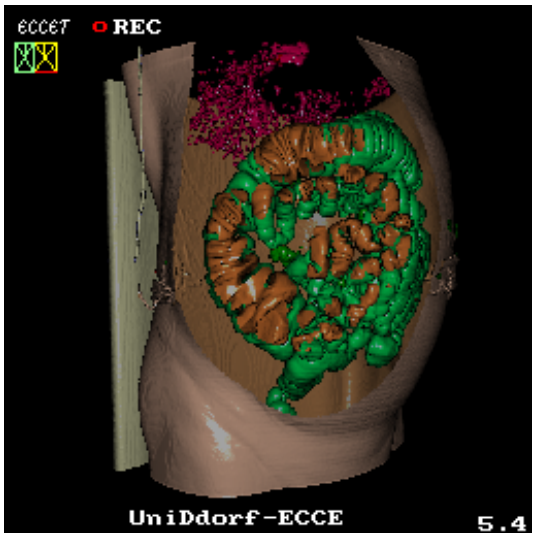
Diese zweidimensionale Betrachtung muss natürlich im Dreidimensionalen erweitert werden. Vergleicht man z.B. nur in x-Richtung, so werden Kanten, die parallel dazu verlaufen, nicht als solche erkannt. Man betrachtet daher sowohl den Nachbarn in x-, als auch in y-Richtung und verrechnet die beiden sich ergebenden Winkel γ_1, γ_2 geeignet. Im Sinne einer schnellen Verarbeitung wird in PLANEVIEW schlicht $\cos\gamma_{total} \approx \cos\gamma_1 \cdot \cos\gamma_2$ angenommen, was zwar geometrisch nicht ganz korrekt ist, aber den gewünschten Effekt hat, dass Kanten beider Richtungen dunkel erscheinen und flache Bereiche hell.

Problematisch erweist sich bei diesem Verfahren vor allem bei VOXREN, dass bei „Nahaufnahmen“ die Kanten der von den Voxeln induzierten Quader störend hervortreten.

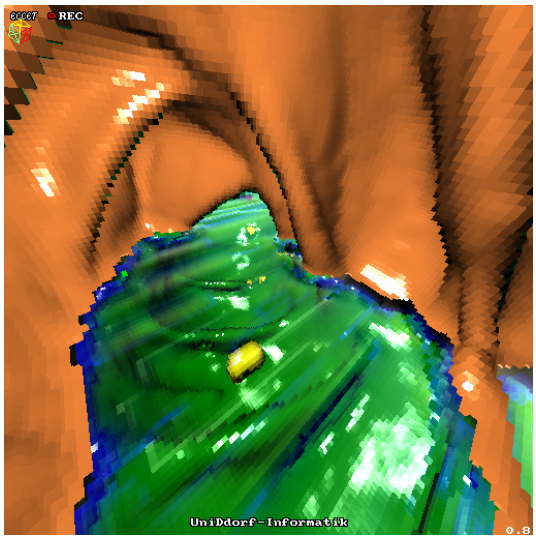
Das liegt daran, dass VOXREN mit einer harten Segmentierung arbeitet, so dass Δz immer um ganze Voxellängen springt.

Bei PLANEVIEW ergibt sich das Problem nicht, da dort das Raycasting-Verfahren nicht durch eine harte voxelbasierte Segmentierung abgebrochen wird, sondern durch eine Schwellwertentscheidung, die mittels trilinearer Interpolation eine Ausnutzung der in den Originaldaten vorhandenen Aliasing-Information (siehe Abschnitt 4.2.3) ermöglicht.

Um einen ähnlichen Effekt für VOXREN zu erreichen, müsste nachträglich interpoliert werden, wobei die Reichweite der Interpolation von der aktuellen Tiefe eines betrachteten Punktes abhängig gemacht werden müsste (nahe Voxel erscheinen größer und benötigen daher einen größeren Einzugsbereich). Aufgrund der hohen Rechenzeiten wurde von einer Implementation dieser Verbesserung in VOXREN Abstand genommen, zumal der ZSH-Modus nur recht selten benutzt wird, da bessere Darstellungsmöglichkeiten vorhanden sind.



a) XYZ-Modus von VOXREN . Die einzelnen Voxelklassen (als verschiedene Farben dargestellt) können einzeln ein- und ausgeschaltet werden.



b) XYZ-Modus mit Transparenz. Hier wurde ein teils mit Kontrastmittel gefüllter Darm („Fecal Tagging“) visualisiert. Die Flüssigkeitsoberfläche ist halbtransparent dargestellt.

Abbildung 4.15: Rendermodi unter Nutzung vorberechneter Normalen

3. **Vorberechnete Normalen**

   Besitzt man für alle opaken Voxel die Oberflächennormale (VOXREN Modus XYZ), so fällt eine aufwändige Berechnung derselben bei der Darstellung weg. Der Nachteil dieses Verfahrens ist der Zeitbedarf zum Berechnen der Oberflächennormale (vgl. 5.2), der aber als einmaliger Vorverarbeitungsschritt von untergeordneter Bedeutung ist, und der hohe Speicherbedarf. Um diesen in Grenzen zu halten, verwendet VOXREN eine quantisierte Darstellung der Normalen, die die Koordinaten θ und ϕ der Normalen bei Darstellung in einem sphärischen Koordinatensystem mit einer Auflösung von 1° enthält. Dazu benötigt man $181 \times 360 = 65160$ Werte, die gerade gut in 2 Byte kodiert werden können, die in Nicht-Oberflächen-Voxeln auch noch dazu verwendet werden können, die Originalgrauwerte der CT-Daten aufzunehmen und somit eine genauere Beurteilung zu ermöglichen, ohne die PLANEVIEW -Darstellung bemühen zu müssen. Diese Auflösung wurde gewählt, um bei wenig gekrümmten Oberflächen keine Quantisierungsartefakte (durch „springende“ Normalen) zu erhalten.

   Zwar ließe sich überschlägig gerechnet eine etwas bessere Codierung finden, denn anhand der Kugeloberfläche ($A_{\text{Kugel}} = 4\pi r^2$) und der approximierten Fläche eines $1° \times 1°$ großen Flächenstückes ($A_{1°\times 1°} \approx \left(\frac{1}{360} 2\pi r\right)^2$) ergibt sich damit ein minimaler Wertebedarf von ca. $\frac{A_{\text{Kugel}}}{A_{1°\times 1°}} \approx \frac{360^2}{\pi}$ also ca. 41250 Werten.

Dies erfordert sowieso die Verwendung von 16 Bit.

Im Sinne einer einfachen und leicht nachvollziehbaren Umrechnung wurde daher ein den Längen- und Breitengraden nachempfundenes Koordinatensystem für die Vektorlage eingesetzt.

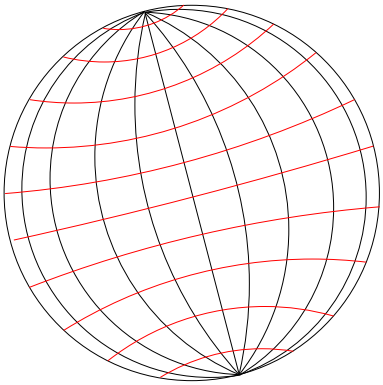


Abbildung 4.16: Kugelkoordinaten

Der Äquator wurde in 360 Grad-Schritte (Länge) eingeteilt, dazu kommt die Breite als Wert von $-90°$ - $+90°$ in $1°$ Schritten (siehe Abbildung 4.16). Diese Wahl stellt die Einhaltung von max. 1 Grad Abweichung sicher, auch wenn in der Nähe der Pole die Punkte unnötig dicht liegen, und ist leicht mit Hilfe einfacher Vektorarithmetik und Winkelfunktionen von und in die Vektordarstellung umzusetzen. Durch den geringen Werteumfang (65160 verschiedene Normalen) ist es sogar möglich, die Dekodierung der Quantisierung sehr schnell über eine Tabelle zu erledigen.

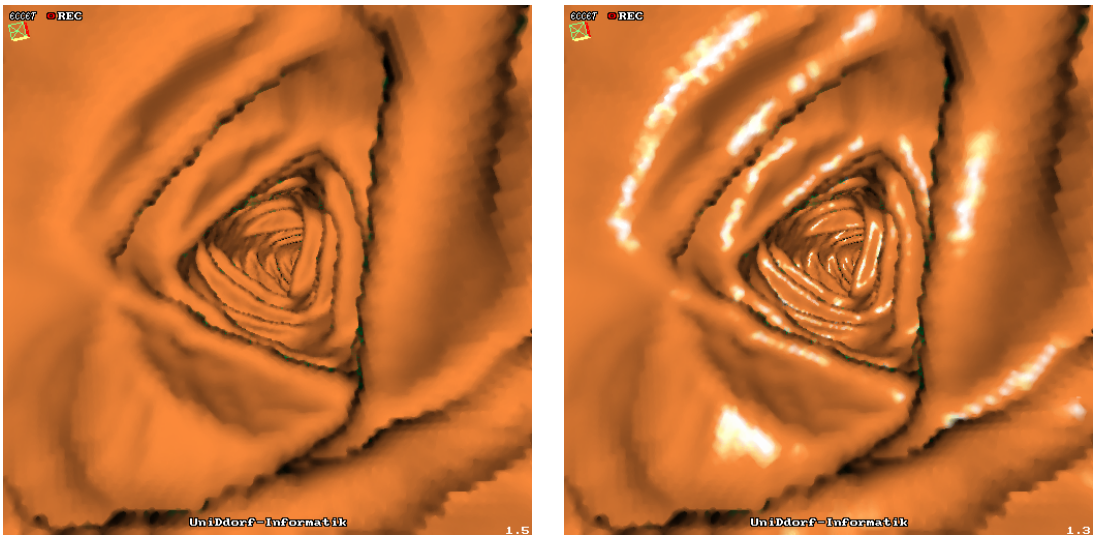
Die Berechnung der darzustellenden Voxelfarbe erfolgt, indem man den Cosinus des Winkels zwischen der Lichtquelle und der Oberflächennormalen bestimmt. Es gilt

$$\cos\alpha = \frac{\vec{n}\cdot\vec{v}}{|v||n|} \ . \tag{4.4}$$

Da man die Tabelle so gestaltet, dass $|n| = 1$ und den Lichtvektor ebenfalls gleich mit $|v| = 1$ bestimmt, vereinfach sich die Gleichung zu

$$\cos\alpha = \vec{n}\cdot\vec{v} \ , \tag{4.5}$$

was die ggf. sehr rechenzeitaufwändige Operation der Division erspart.



Ansicht eines Darmstückes ohne und mit Glanzlichtern.

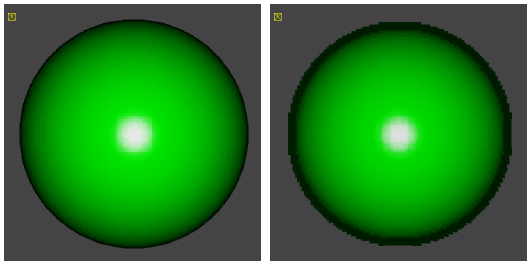
Abbildung 4.17: Glanzlichter

Damit erhält man die Helligkeit, mit der das Objekt durch diffuse Reflexion Licht abstrahlt (die diffuse Reflexion erfolgt gerade mit einem cosinusförmigen Profil bezüglich des Winkels zwischen Oberflächennormale und Lichtquelle). Um die Farbe des Objektes zu bestimmen, wird dessen Voxelklasse ausgelesen und diese über eine Tabelle zur Farbzuordnung genutzt.

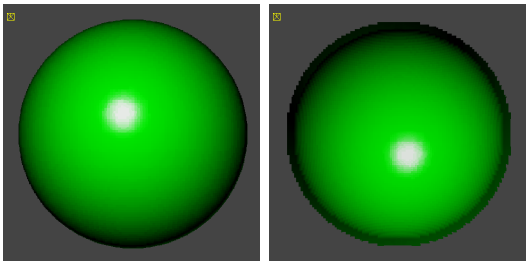
Man erreicht eine verbesserte plastische Wirkung im Zusammenhang mit in der Nähe der Kamera befindlichen Lichtquellen (wie sie von VOXREN ausschließlich unterstützt werden), indem man eine Farbtabelle (Color-Look-Up-Table, CLUT, LUT) verwendet, die in der Nähe von $\alpha = 0$ die Sättigung der Farbe verringert und sie so gleitend nach weiß übergehen lässt. Man simuliert damit Glanzlichter auf feuchten oder plastikartigen Oberflächen durch direkte Reflexion der Lichtquelle (specular highlighting, s. Abbildung 4.17).

Zur besseren Unterscheidung feiner Strukturen in flachen Bereichen kann die Position der Lichtquelle in Grenzen verändert werden. Dabei verändert sich die Position von Glanzlichtern und den (als Schatten empfundenen) Stellen, deren Normale nahezu senkrecht zur Lichtquelle steht, und liefert damit zusätzliche Informationen zur Oberfläche (s. Abbildung 4.18). Es werden allerdings keine echten Schatten berechnet, das wäre zu aufwändig. Aus diesem Grund werden auch nur dicht bei der Kamera befindliche Lichtquellen unterstützt, da sich dort die Problematik einer echten Schattenberechnung nicht stellt.

Entsprechend kann die Farbtabelle frei definiert und im laufenden Betrieb neu



a) Ansicht einer Voll- und einer halben Hohlkugel bei zentraler Beleuchtung. Die Objekte sind kaum zu unterscheiden.



b) Dieselben Objekte bei Beleuchtung von schräg links oben. Durch das Wissen über die Lage der Lichtquelle sind beide Objekte leicht zu unterscheiden.

Abbildung 4.18: Veränderung der Position der Lichtquelle

geladen werden, so dass eine weitere Verdeutlichung durch Streckung bestimmter Kontrastbereiche oder gar Falschfarbendarstellung erreicht werden kann. Ebenso ist es damit möglich, die simulierten Reflexionen (s.o., Abbildung 4.17) zu unterdrücken.

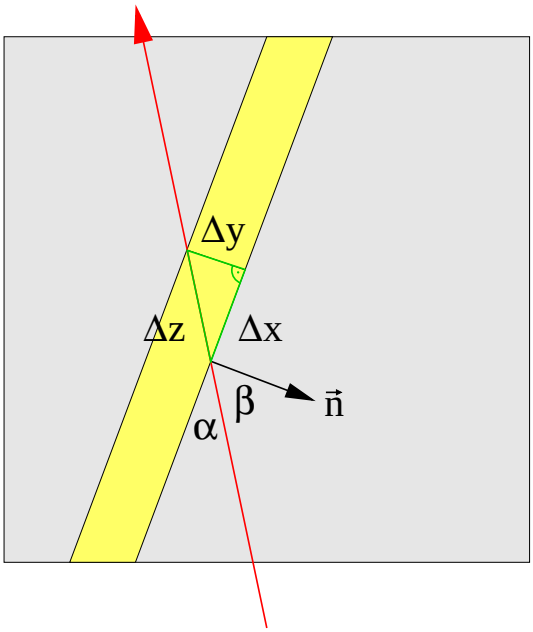
In speziellen Fällen ist jedoch eine sinnvolle Bestimmung der Oberflächennormalen nicht möglich, so zum Beispiel bei nur 1 Voxel dicken Strukturen, die theoretisch Normalen in „alle Richtungen“ haben müssten, wenn sie z.B. feine Gefäßsysteme oder kleine Kugeln repräsentieren.

Daher werden von den durch die Quantisierung noch nicht belegten 376 Werten 256 belegt, um Voxel mit einer festen, von allen Seiten gleichen Farbe belegen zu können.

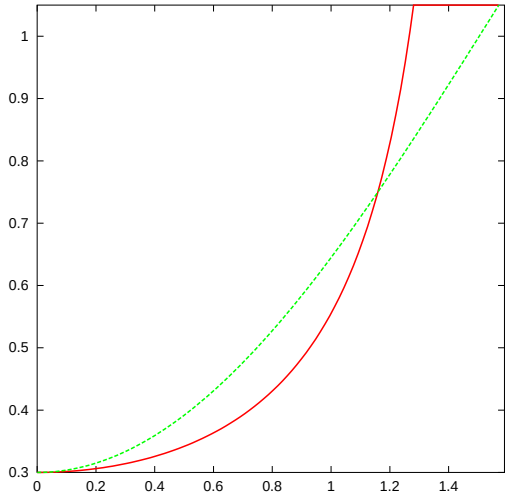
Von den danach verbleibenden 120 Byte werden 100 für die semitransparente Darstellung von Voxeln genutzt. Diese werden verwendet für Darstellungen von Hilfsoberflächen, bei der Segmentierung als unsicher eingestuften Voxeln, zur Erzeugung von echten Volumen-Nebeleffekten oder auch zur Approximation von mit Subvoxelgenauigkeit extrapolierten Daten.

Die verbleibenden 20 Werte sind noch für zukünftige Erweiterungen reserviert.

4. **Simulierter Kontrasteinlauf**
   In der klassischen Röntgendiagnostik wird im Bereich der Dickdarmdarstellung oft die Technik der Kontrastaufnahme verwendet. Dazu wird dem Patienten ein Kontrastmittel verabreicht, das die Darmwände in einer dünnen Schicht benetzt. Anschließend fertigt man mehrere Aufnahmen in verschiedenen Lagen an. Der Darm erscheint dabei in einer Negativdarstellung, da die bezüglich der Aufnahme seitlichen Wände hell erscheinen (die Röntgenstrah-



a) Berechnung des Beitrages eines einzelnen Voxels zum Rendering im KE-Modus.



b) Näherung von $\min(\sqrt{1+\Delta y^2}, \frac{\Delta y}{\cos\beta})$ (rot) durch Multiplikation mit $(1-cos\beta)$ (grün) und geeigneter Skalierung und Verschiebung

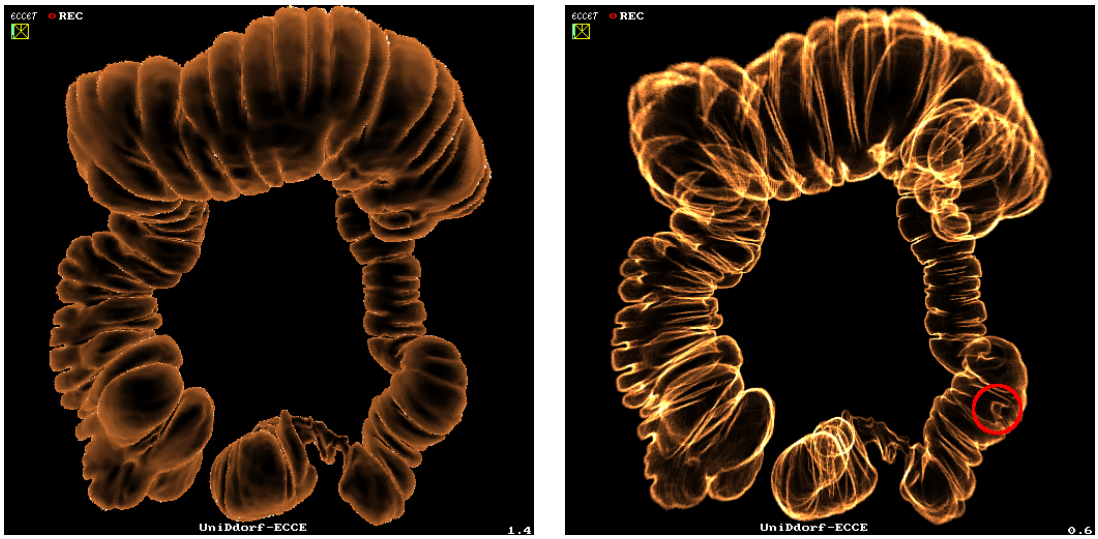
Abbildung 4.19: Simulierter Kontrasteinlauf - Berechnung

lung fällt hier streifend auf das Kontrastmittel, legt also einen langen Weg im Kontrastmittel zurück und wird entsprechend stark geschwächt), während zur Strahlrichtung senkrecht verlaufende Wände recht transparent bleiben (kurze Durchquerung der dünnen Schicht). Diese Aufnahmen erlauben insbesondere, größere geometrische Anomalien wie Polypen in ansonsten glatten Darmbereichen recht schnell zu erkennen. Sie zeichnen sich durch entsprechende kreis- oder u-förmige Strukturen gut ab.

Diese Methode kann virtuell natürlich nachgeahmt werden. Abbildung 4.19a illustriert die Berechnung der Schwächung der Röntgenstrahlung innerhalb eines einzelnen Voxels. Dazu wird eine Kontrastmittelschicht (in der Abbildung gelb dargestellt) der konstanten Dicke Δy angenommen, die das Voxel senkrecht zu dessen Normalenrichtung $\vec{n}$ durchzieht. Der Weg des Strahles durch das Kontrastmittel ist dann

$$\Delta z = \frac{\Delta y}{\sin\alpha} = \frac{\Delta y}{\cos\beta} \; . \tag{4.6}$$

Allerdings ist Δz bei $\sqrt{1+\Delta y^2}$ abzubrechen, da ja nur der Anteil innerhalb eines einzelnen Voxels berechnet werden soll. Zur Vermeidung der langsamen Division wird sie in der Implementation durch eine Multiplikation mit $1-\cos\beta$



a) KES-Modus. Das Rendering wird zugunsten der Geschwindigkeit nach der ersten Oberfläche abgebrochen.

b) KE-Modus. Man beachte den großen Polypen (roter Kreis).

Abbildung 4.20: simulierter Kontrasteinlauf

angenähert (s. Abb. 4.19b — man beachte, dass die rote Linie am Ende durch das Abschneiden nach oben waagerecht weiterläuft).

Nun kennen wir die Schwächung des simulierten Röntgenstrahles für ein einzelnes Voxel. Da dieses alleine in der Regel nicht bereits alle Strahlung schluckt, darf im Gegensatz zu den bisherigen Modi die Berechnung noch nicht abgebrochen werden, wenn der erste Schnitt mit der Voxelmenge gefunden wurde. Die Suche wird stattdessen so lange fortgesetzt, bis der mit Voxeln gefüllte Quader wieder verlassen wurde und somit keine weiteren Schnitte mehr möglich sind. Die gefundenen Helligkeiten werden analog zum physikalischen Modell des Vorbildes addiert (mit Limitierung am oberen Ende). Die Gewichtung kann durch den Benutzer eingestellt werden, um den gewünschten Kontrast zu erhalten. Abbildung 4.20 zeigt den Unterschied zwischen einer Darstellung mit Abbruch nach der ersten Oberfläche (a) und der Darstellung ohne Abbruch (b).

Im letzteren Fall müssen erheblich mehr Berechnungen angestellt werden: je eine Schwächungsberechnung pro zusätzlich getroffenem Oberflächenvoxel. Dies führt zu einer typischerweise um einen Faktor von 2-3 geringeren Geschwindigkeit, was aber in der Regel durch die bessere Übersicht ausgeglichen wird. So kann in Abbildung 4.20b der Polyp sofort erkannt werden, obwohl er in der Hinterwand lokalisiert wird, die in Abbildung 4.20a gar nicht dargestellt wird. Der KES-Modus wird daher nur zur schnellen und präzisen Einstellung einer

Ansicht genutzt, um dann wieder auf den KE-Modus zu schalten.

Gegenüber der klassischen Methode besitzt dieses Verfahren allerdings eine Reihe von Vorteilen:

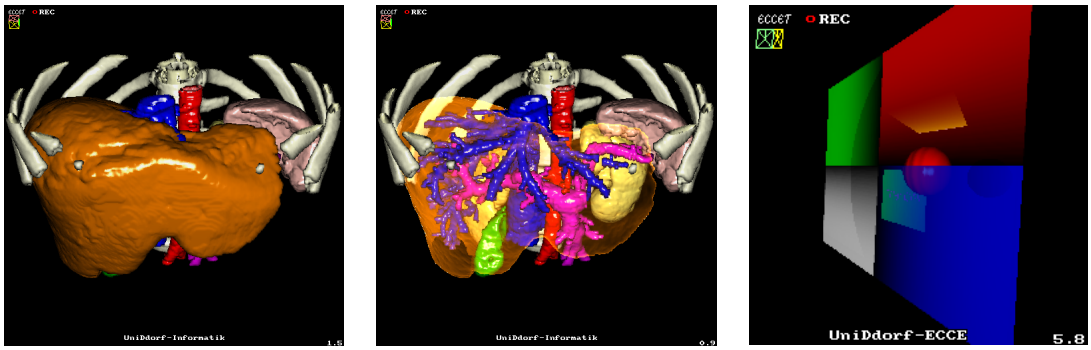
- Die Strahlendosis einer CT-Untersuchung ist bei modernen Geräten in Ultra-Lowdose-Technik ($0,7 - 1,15\,\mathrm{mSv}$ [VCS], [VCSA]) durchaus vergleichbar mit der, die beim klassischen Verfahren durch die Verwendung mehrerer Aufnahmen anfallen würde ($0.7\,\mathrm{mSv}$ [SER]). Somit ist das Verfahren für den Patienten schonender — es spart Zeit sowie das ggf. unangenehme Umlagern in verschiedene Positionen.
- Belichtung und Lage der einzelnen Projektionen kann im Nachhinein optimal gewählt werden, Detailansichten sind ohne zusätzliche Belastung des Patienten möglich.
- andere anatomische Strukturen mit starker Röntgenabsorption (Knochen) erscheinen nicht auf den Aufnahmen. Damit werden Blickwinkel möglich, die normalerweise aus diesem Grund oder weil Röntgenquelle und Aufnahmeplatte nicht sinnvoll dort positioniert werden könnten nicht möglich wären (z.B. direkt von unten nach oben).
- Es sind auch KE-Aufnahmen von Bereichen möglich, die normalerweise nicht mit Kontrastmittel gespült werden können. Ein Beispiel aus der Anwendung ist die Darstellung funktionaler Magnetenzephalographiedaten mit VOXREN. Im XYT-Modus wird dazu die **Gehirnoberfläche** als KE-Darstellung dem eigentlichen Bild (aktive Zentren) überlagert und dient als Orientierungshilfe.

5. **Kombinierte Darstellungen und Semitransparenz**
   Manchmal — wie im eben geschilderten Beispiel — ist es nützlich, die oben geschilderten Verfahren zu kombinieren.

   Eine weitere nützliche Anwendung ist die Visualisierung des Gefäßsystems der Leber. Dabei ist es wichtig, das Organ gleichzeitig mit dem Gefäßbaum darstellen zu können, um eine Zuordnung der relativen Lage zu ermöglichen.

   Die Leber kann dabei nicht wie üblich opak dargestellt werden, da sie ja den Gefäßbaum umschließt und diesen somit verdecken würde.

   Daher kann VOXREN im XYT-Modus Voxelklassen einzeln semitransparent (im Sinne einer KES-Darstellung) oder opak darstellen.



a) Die segmentierte Leberoberfläche verdeckt den Blick auf das Gefäßsystem.

b) Semitransparent eingeblendet ermöglicht sie eine bessere Orientierung bezüglich der Lage des Gefäßbaumes relativ zur Organoberfläche.

c) Synthetisches Beispiel für die pro Voxel einstellbare Transparenz

Abbildung 4.21: Kombinierte Darstellungen

Dabei werden auch mehrere transparente Voxelklassen hintereinander dargestellt, aber immer nur deren **erste** Oberfläche. Dies liefert die gewünschte relative Lageinformation, ohne durch Darstellung verdeckter (auf der Rückseite befindlicher) Strukturen zu verwirren (s. Abbildung 4.21a und b).

Alternativ können Voxeln explizit Transparenzwerte zugewiesen werden (s. Abbildung 4.21c). Damit können ähnliche Effekte erreicht werden oder auch Elemente mit Subvoxel-Ausmaßen approximativ dargestellt werden.

4.2.3 Raycasting mit Antialiasing in PLANEVIEW

Mit Hilfe der Originalgrauwerte ist es möglich, durch Interpolation „glattere“ Ränder zu erzeugen als mit einem hart segmentierenden Voxelverfahren. Dazu werden zum Rendering Grauwertisoflächen benutzt, die sich durch Interpolation zwischen den Voxelpositionen ergeben. Hierdurch wird Subvoxelgenauigkeit erreicht.

Das verwendete Verfahren ähnelt dem in 4.2.1 beschriebenen, allerdings mit folgenden Änderungen:

– Es steht kein Platz für ein Distanzfeld zur Verfügung, da PLANEVIEW im Sinne eines günstigen Gesamtsystems weniger Anforderungen an die Maschine stellen sollte. Im reinen 2D-Betrieb verbraucht PLANEVIEW mit 2 Byte/Voxel nur die Hälfte des Speichers von VOXREN. Entspre-

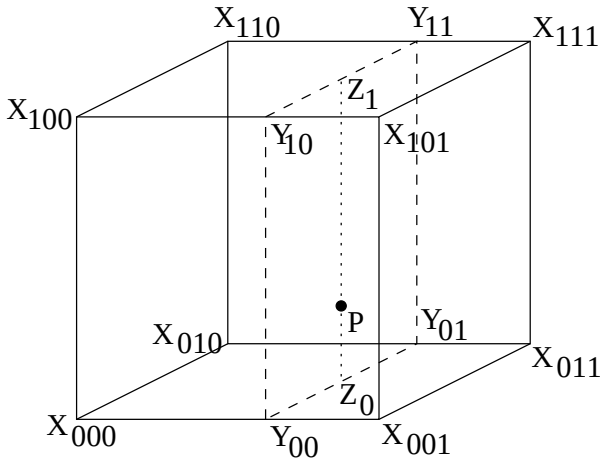


Abbildung 4.22: Trilineare Interpolation

chend ist die Optimierung der Schrittweite per GDA nicht möglich. Da das Rendern von hochaufgelösten Bildern aber selten notwendig ist (zur Dokumentation oder zur genaueren diagnostischen Klärung unklarer Stellen), erschien der „Preis“ des erhöhten RAM-Verbrauchs im Verhältnis zum Nutzen zu hoch. (Aktuell wird allerdings aufgrund fortschreitender Hardwareentwicklung erwogen, diese Möglichkeit optional zu machen, um ggf. auch hochwertiges Rendering in nahezu Echtzeit zu ermöglichen, s. Kapitel 10.)

- Die Entscheidung „Voxel ist Teil des darzustellenden Volumens“ wird nicht anhand einer Voxelmarkierung getroffen wie bei VOXREN , sondern anhand eines Schwellwertes, der über- bzw. unterschritten werden muss.
- Tritt eine Überschreitung des Grenzwertes bei der Strahlverfolgung auf, so wird iterativ mit binärer Suche nach dem exakten Durchtrittspunkt (mit Subvoxelgenauigkeit) gesucht.
- Sehr dünne Objekte können (ähnlich wie in Abbildung 4.5 bereits illustriert) natürlich auch bei dieser Vorgehensweise scheinbare „Löcher“ bekommen.
- Die Ränder von Objekten erscheinen durch die Interpolation deutlich „glatter“.

Die trilineare Interpolation

Als Interpolationsverfahren wurde die trilineare Interpolation gewählt, da sie relativ unaufwändig in der Berechnung ist, was aufgrund der häufigen Aufrufe

entscheidend die Laufzeit bestimmt.

Diese berechnet man am einfachsten schrittweise, wie in Abbildung 4.22 dargestellt. Sei P der Punkt für den die Interpolation berechnet werden soll und seien X_{ijk} mit $i, j, k \in \{0, 1\}$ die acht benachbarten Punkte auf dem regulären Gitter, so werden zunächst Interpolationswerte für die Punkte Y_{ij} gewonnen, indem

$$g(Y_{ij}) = \frac{g(X_{ij0}) \cdot \overline{Y_{ij}X_{ij1}} + g(X_{ij1}) \cdot \overline{X_{ij0}Y_{ij}}}{\overline{X_{ij0}X_{ij1}}} \tag{4.7}$$

berechnet wird, um dann in zwei weiteren Schritten

$$g(Z_i) = \frac{g(Y_{i0}) \cdot \overline{Z_iY_{i1}} + g(Y_{i1}) \cdot \overline{Y_{i0}Z_i}}{\overline{Y_{i0}Y_{i1}}} \tag{4.8}$$

und

$$g(P) = \frac{g(Z_0) \cdot \overline{PZ_1} + g(Z_1) \cdot \overline{Z_0P}}{\overline{Z_0Z_1}} \tag{4.9}$$

zu berechnen.

Wie man leicht nachrechnet, ergibt sich dadurch natürlich keine Asymmetrie bezüglich der Reihenfolge der Berechnung in den drei Achsen.

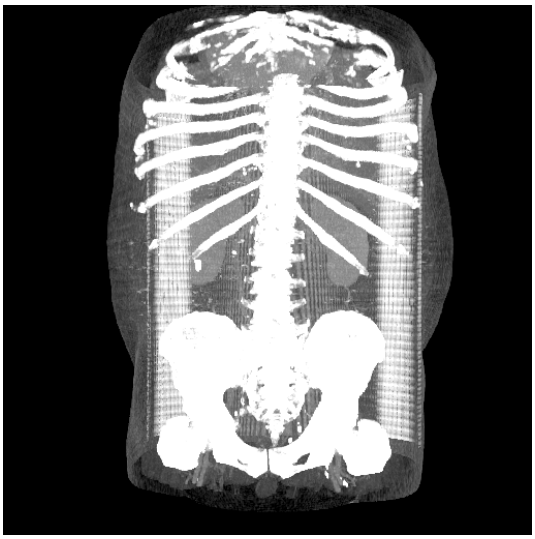
Da die Punkte X_{ijk} im gegebenen Problem jeweils auf ganzzahligen Koordinatenpositionen sitzen, sind die oben vorkommenden Teilstrecken jeweils leicht aus den Nachkommastellen der Koordinaten von P zu berechnen.

Offensichtlich ist die trilineare Interpolation an den Rändern zwischen zwei Voxeln stetig (die entsprechenden Strecken, die für ein Zumischen der Voxel des gegenüberliegenden Randes sorgen, werden dort gerade 0) - allerdings nicht mehr unbedingt eindeutig differenzierbar.
Anschaulich gesprochen kann die interpolierte Isofläche an Voxelrändern zwar knicken, wird aber ohne Sprung fortgesetzt.

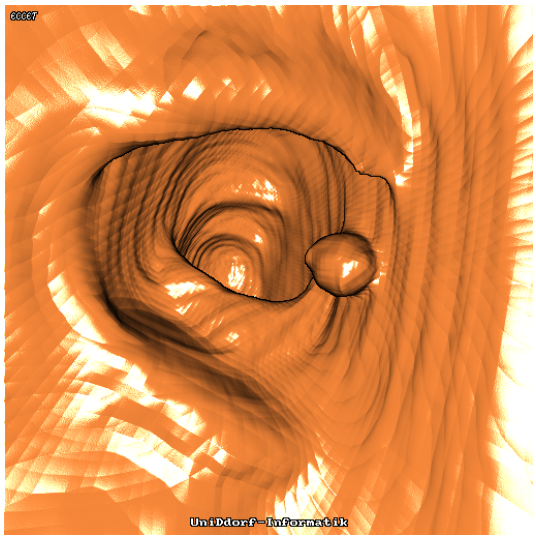
Oberflächendarstellung

Parallel zu 4.2.2 ist natürlich auch hier wieder das Problem zu lösen, in welcher Farbe das nun gefundene Pixel dargestellt werden soll.
Dazu wurden folgende Lösungen implementiert:

1. Maximumsprojektion
   Bei dieser Variante wird der Stopp des Tracings ähnlich dem KE-Modus in



a) Maximumsprojektion. Offensichtlich kann dieser Modus maximal zur Darstellung von dichten Strukturen (Knochen) bzw. per Kontrastmittel entsprechend angefärbten Regionen dienen.

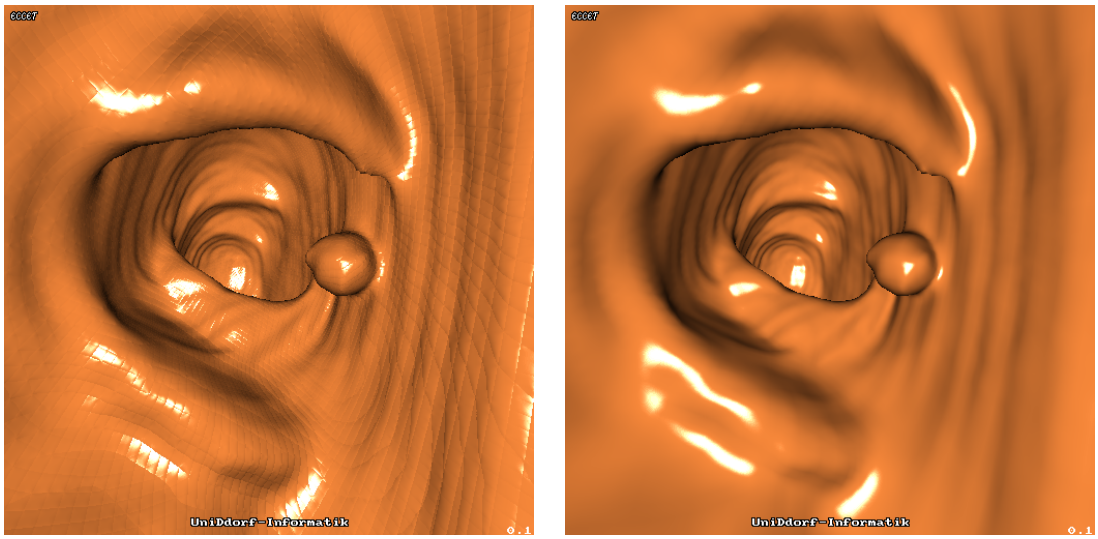


b) Das Z-Shading von PLANEVIEW wirkt aufgrund der möglichen höheren Auflösung durch die 3D-Interpolation bei der Oberflächenbestimmung deutlich „glatter“ als das von den Voxelkanten bestimmte von VOXREN .

Abbildung 4.23: Planeview: Maximumsprojektion und Z-Shading

4.2.2 aufgehoben und schlicht der maximale Grauwert entlang der einzelnen Sehstrahlen dargestellt. Diese Darstellung erinnert an ein klassisches Röntgenbild und ist daher gelegentlich nützlich, um eine Orientierungsansicht zu berechnen.
Ansonsten ist sie von wenig praktischem Nutzen, da in vielen Fällen stark absorbierende Strukturen wie Knochen störend vor den eigentlich interessierenden Objekten dominieren (s. Abbildung 4.23a). Zusätzlich benötigt die Maximumsprojektion noch eine lange Renderzeit, da alle im Sichtkegel gelegenen Voxel überprüft werden müssen.

2. Z-Shading
   Das Prinzip dieses Verfahrens wurde schon ausführlich in 4.2.2 dargestellt. Durch die subpixelgenaue Berechnung der Auftreffpunkte, die auch in die Δz-Abstände eingehen, erreicht man allerdings eine deutlich schönere Darstellung als bei VOXREN (vgl. Abbildung 4.13), vor allem bei sehr nahen Voxeln, da dort nicht mehr die großen „Sprünge“ an den Voxelkanten auftreten. Allerdings bleiben aufgrund der verwendeten trilinearen Interpolation Unstetigkeiten in der ersten Ableitung (die Grauwerte selbst gehen stetig über) an den Kanten. Dadurch erscheint das sich ergebende



a) Schätzung nach 5.2 mit Samplingpunkten auf dem Voxelgitter

b) wie a) mit Sampling auf Zwischengitterpunkten

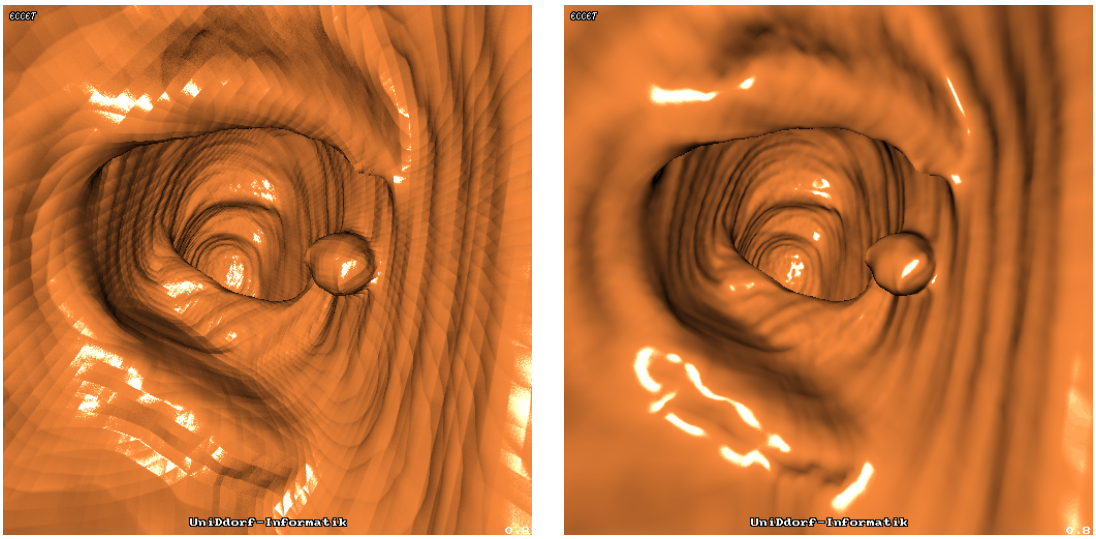
Abbildung 4.24: Normalenschätzung mit verschiedenen Abtastgittern

Bild aus kleinen Kacheln zusammengesetzt (s. Abbildung 4.23b) zu sein, deren Kanten auf den Schnittflächen des Voxelgitters liegen. Einerseits liefern diese Linien als „Höhenlinien“ eine zusätzliche dreidimensionale Information, andererseits wirken sie gerade für das ungeschulte Auge störend.

3. Oberflächennormalenschätzung nach Abschnitt 5.2
   Natürlich kann auch punktweise die bei VOXREN vorberechnete Normalenschätzung durchgeführt werden. Dazu wird im Wesentlichen derselbe Algorithmus verwendet. Er wird jetzt allerdings punktweise online berechnet und die Summe läuft nun über die gesamte Umgebung und enthält für jeden Punkt seinen Grauwert als Gewichtungsfaktor. Damit ergibt sich bereits eine Verbesserung, da nun anhand des Grauwertes berücksichtigt wird, zu „welchem Grad“ ein Voxel zu der helleren Klasse gehört.

   Die punktweise Berechnung ist insbesondere nötig, da jetzt die Zentren der Berechnung ($\vec{x}$) auf Zwischengitterpunkten liegen können. Eine vollständige Vorberechnung ist daher sowieso nicht mehr möglich.

   Diese Übertragung ins Kontinuum schafft ein weiteres Problem: Da die Summe in ein Integral übergeht, dessen Auswertung nicht trivial ist, muss dieses numerisch approximiert werden. Der einfachste Ansatz hierzu ist die Auswertung auf einem regulären Gitter um $\vec{x}$, wodurch die Aufgabe wieder mit den Methoden von 5.2 gelöst werden kann.



a) Gradientenschätzung mit Schrittweite 0,1 b) Gradientenschätzung mit Schrittweite 1

Abbildung 4.25: Gradientenschätzung mit kleiner und großer Reichweite

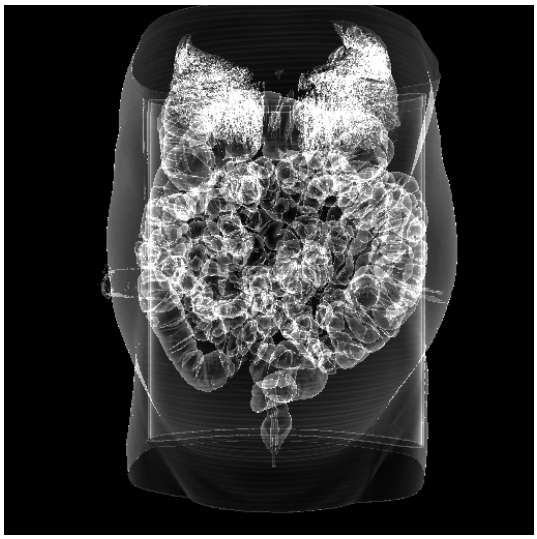
Dazu wurden zwei verschiedene Gitter verwendet:

a) Das von den Voxeln induzierte natürliche Gitter
Dieses Verfahren hat den Nachteil, dass bei Verschiebung des Aufschlagpunktes außerhalb eines durch das Voxelgitter induzierten Würfels, das verwendete Gitter ruckartig „springt“. Sofern die verwendete Umgebung nicht sehr groß ist, neigt diese Variante daher dazu, an den Voxelkanten Sprünge zu erzeugen, die ähnlich wie beim Z-Shading Linien an den Voxelkanten induzieren. Sehr große Kernweiten mildern diesen Effekt, wirken sich hingegen in der Nähe feiner Strukturen störend aus, da die Normalenschätzung dann zu global erfolgt.

b) Das um den Aufschlagpunkt zentrierte reguläre Gitter
Damit wird die Asymmetrie von a) behoben und es ergibt sich ein sehr glattes Bild. Allerdings müssen dann die Grauwerte der Gitterpunkte jeweils zuvor mit trilinearer Interpolation berechnet werden, wodurch sich ein geringfügig höherer Rechenaufwand ergibt.

4. Gradientenschätzung
Eine besonders schnelle Methode, die Normale zu schätzen, besteht darin, den Gradienten approximativ zu bestimmen als

$$\text{grad } g(\vec{x}) \approx \begin{pmatrix} [g(\vec{x}+\Delta x) - g(\vec{x})]/\Delta x \\ [g(\vec{x}+\Delta y) - g(\vec{x})]/\Delta y \\ [g(\vec{x}+\Delta z) - g(\vec{x})]/\Delta z \end{pmatrix} \tag{4.10}$$



Da PLANEVIEW normalerweise keine harte Segmentierung der Voxeldaten mit Einteilung in verschiedene Klassen vornimmt, leidet die Übersicht in der KE-Darstellung stark. Alle Luft-/Gewebeübergänge werden gleichzeitig dargestellt.

Abbildung 4.26: Kontrasteinlaufsimulation bei PLANEVIEW

und das Ergebis als Normalenschätzung zu interpretieren. Natürlich zeigt sich auch bei dieser Methode die in zweiter Ableitung nicht stetige Interpolation als Linienmuster auf dem gerenderten Bild.

Dieser Effekt kann minimiert werden, indem $\Delta x, \Delta y, \Delta y$ recht groß (optimal ist ein Betrag von 1) gewählt werden. In diesem Fall wertet man nicht mehr die lokale, durch die Interpolation induzierte, Neigung aus, sondern schätzt die Normale aus der Aliasing-Information der umgebenden Voxel (s. Abbildung 4.25).

5. Kontrasteinlaufsimulation
   Analog zu VOXREN kann natürlich auch diese Darstellung bereitgestellt werden. Dabei ergibt sich aber ein prinzipielles Problem:

   Während VOXREN durch seine voxelklassenorientierte Arbeitsweise die Darstellung auf interessante Bereiche (z.B. Darmoberfläche) beschränken kann, liegt eine solche Information in PLANEVIEW nicht vor.

   Entsprechend sind KE-Darstellungen normalerweise nicht sinnvoll, da zu viele Überdeckungen mit anderen Strukturen, die im gleichen Grauwertbereich liegen, auftreten (s. 4.26).

Beschränkung des darzustellenden Bereiches

Durch Nutzung der üblicherweise nicht verwendeten oberen 4 Bit der Grauwertinformation wird es möglich, bis zu 3 Voxelklassen (1 Bit verbleibt für

a) von VOXREN übernommene Markierung in der Maximumsprojektion eingeblendet

b) Damit wird eine isolierte **Außen**ansicht des Dickdarmes möglich.

c) Auch die KE-Darstellung ist nun wieder übersichtlich.

Abbildung 4.27: Verwendung einer Rendermaske bei Planeview

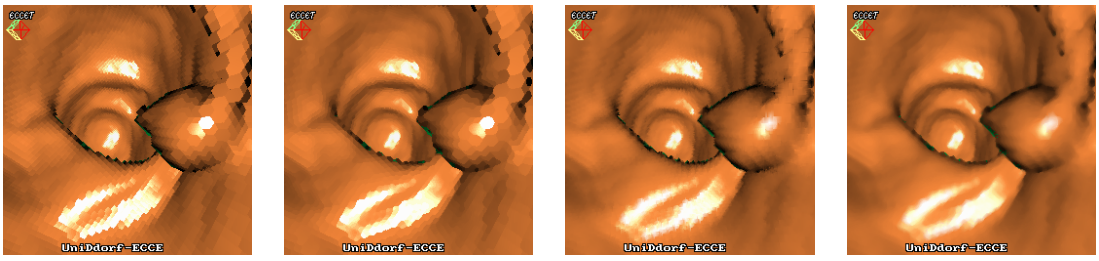
interne Zwecke reserviert) als Overlay von VOXREN auf PLANEVIEW zu übertragen. Die Darstellung kann auf markierte Bereiche beschränkt werden, so dass eine Selektion analog zu VOXREN möglich wird.

Da die Renderinginformation (Oberflächenposition und Normalenschätzung) aber immer noch aus dem Grauwert bezogen wird, heißt das allerdings nicht, dass auf diese Weise beliebige Selektionen aus VOXREN von PLANEVIEW in der hohen Renderingqualität wiedergegeben werden können.

Die erhöhte Qualität beruht im Wesentlichen auf der Annahme, dass die Grauwertdaten eine aufgrund von Partialvolumeneffekten „geglättete“ (korrekter: mit Aliasing versehene) Version der harten Segmentierung von VOXREN ist und die Segmentierung anhand eines Schwellwertes geschah.

Ist dies nicht der Fall (z.B. weil die Segmentierung mehrere Schwellwerte oder andere Verfahren genutzt hat), so wird auch keine korrekte Darstellung möglich sein.

Eine nützliche Anwendung der Möglichkeit, den darzustellenden Bereich auf markierte Voxel zu beschränken, ist in Abbildung 4.27 illustriert. Da das Rendering durch Nutzung der in VOXREN gewonnenen Maske nur auf den Bereich des Dickdarmes beschränkt wird, kann damit auch in PLANEVIEW eine übersichtliche KE-Darstellung erreicht werden.



Original Medianfilter Aliasfilter Beide Filter

Abbildung 4.28: Interpolation

4.2.4 Interpolation der Bilder

Der in VOXREN implementierte Algorithmus zur Berechnung der 3D-Ansichten ist zugunsten der Geschwindigkeit optimiert. Die erzeugten Bilder sind daher nicht so „glatt“ wie die aus interpolierten Daten von PLANEVIEW berechneten.

Um sie für den menschlichen Betrachter „gefälliger“ zu gestalten, wurden zwei Postprocessing Filter implementiert, die allerdings zusätzlich zum eigentlichen Bild noch Tiefen- und Zusammenhangsinformationen aus dem Rendervorgang benutzen.

Glättung von Ecken durch den Medianfilter

Dieses Filter ist im Wesentlichen ein klassisches Medianfilter. Allerdings wird die Filterweite bestimmt durch die aktuelle „Tiefe“ , die beim Rendervorgang mitgespeichert wurde. Nahe Elemente werden mit einer 5x5-Matrix gefiltert, weiter entfernte nur mit 3x3 und sehr weit entfernte gar nicht.

Hierdurch wird erreicht, dass die Kanten naher Voxel, die sonst „klotzig“ wirken, abgerundet werden, ohne dass detailreiche, weit entfernte Gebiete, in denen sich das Bild eines Voxels nicht über mehrere Pixel ausdehnt, verschmiert werden.

Glättung von Flächen mit das Aliasfilter

Dieser Filtertyp erfordert eine Zusatzinformation bei der Glättung, nämlich die Lage von Kanten. Diese Information wird in VOXREN beim Rendern gewonnen. Zusätzlich zur Bildinformation (Farbe) wird für jedes Pixel die Entfernung des getroffenen Voxels von der Kameraebene (Tiefe) und die Koordinaten des getroffenen Voxels gespeichert.

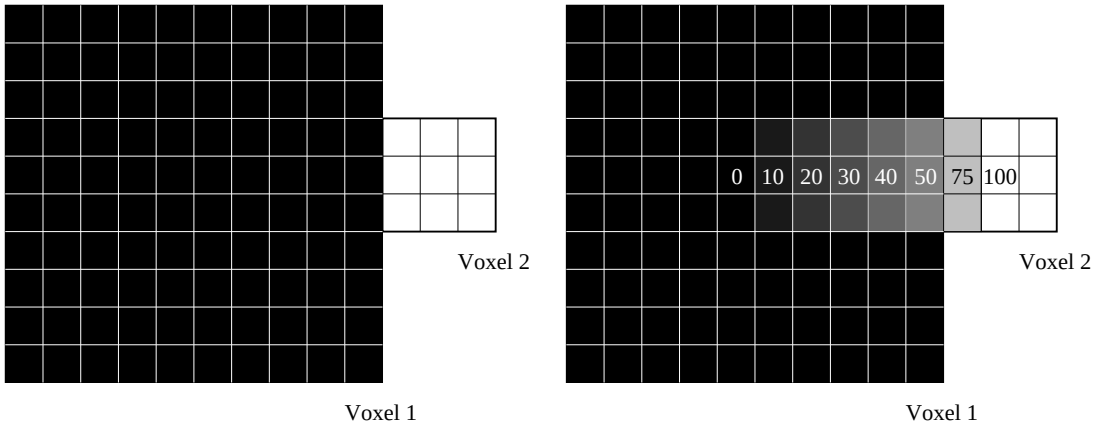


Abbildung 4.29: Funktionsprinzip Aliasfilter

Damit ist bekannt, welche Pixel von ein- und demselben Voxel verursacht werden. Wenn im Folgenden innerhalb dieses Abschnittes von „Voxel“ die Rede ist, so ist jeweils die Pixelmenge gemeint, die durch die Abbildung eines Voxels im Bild entsteht.

Die Idee ist nun, eine Interpolation von „Voxelmitte zu Voxelmitte“ mit 50%-Mischpunkten auf den Voxelkanten durchzuführen. Dadurch ergibt sich eine gute Glättung großer Flächen, ohne kleine Flächen zu verschmieren.

Dies ist am einfachsten anhand eines Beispiel zu erklären:

Nehmen wir an, in einer 3D-Ansicht wird ein nah an der Kamera befindliches Vordergrundvoxel (Grauwert 0) als 10 Pixel breites Quadrat dargestellt. Des Weiteren schließe sich in der Ansicht rechts davon direkt ein weiteres Voxel an, das weiter von der Kamera entfernt ist, und daher als nur 3 Pixel breites Quadrat dargestellt wird (s. Abbildung 4.29).

In diesem Fall wird die Farbe des großen Voxels von Pixel 5 bis Pixel 10 linear auf die 50%-Mischung zwischen den beiden Voxelfarben angepasst. Entsprechend wird auf dem restlichen Weg (Pixel 11) linear bis zur Farbe des 2. Pixels weiterinterpoliert (s. auch Abbildung 4.29):

Pixel	1	2	3	4	5	6	7	8	9	10	11	12	13
vorher:	0	0	0	0	0	0	0	0	0	0	100	100	100
nachher:	-	-	-	-	0	10	20	30	40	50	75	100	-

Die mit „-“ markierten Bereiche werden in diesem Schritt nicht berührt - sie werden bei der Betrachtung des vorherigen bzw. nachfolgenden Pixelpaares berücksichtigt.

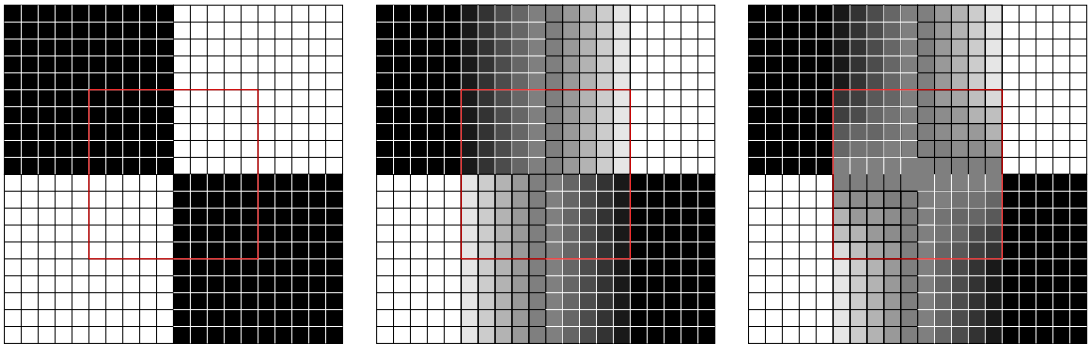


Abbildung 4.30: Kopplung der separierten Stufen

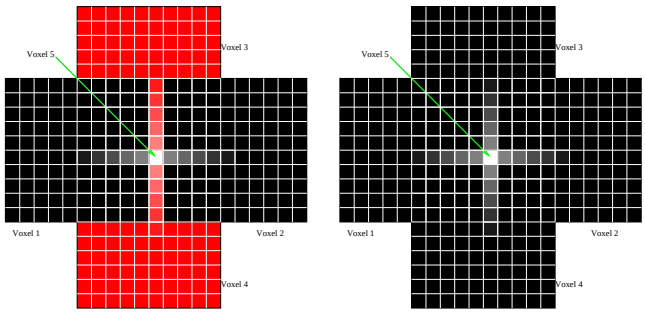
Dieses Filter wird separiert in x- und y-Richtung angewendet. Bei der zweiten Stufe stellt sich dabei natürlich die Frage, wie die Mischung dort anzuwenden ist, ohne den Effekt des ersten Schrittes zu negieren. Die folgende Heuristik hat sich dabei als brauchbar erwiesen:

Man verwendet bei der Mischung als Zielfarbe die erste Farbe, die zu einem neu auftretenden Voxel in einer Spalte gehört. Dies scheint zunächst der Idee „von Voxelmitte zu Voxelmitte zu interpolieren“ zu widersprechen. Sind die Voxel allerdings längs der aktuellen Kameraachsen ausgerichtet und in einer Ebene, so ist diese Farbe dieselbe wie die aller anderen zu diesem Voxel gehörigen Punkte dieser Spalte, da der vorherige Schritt ja senkrecht dazu verlief.

In diesem Fall liefert es also das erwartete symmetrische Verhalten. In den anderen Fällen wird zumindest ein stetiger Übergang an der Kante erzeugt, da ja das letzte Voxel vor der Kante zu 50% mit seinem Nachbarvoxel hinter der Kante gemischt wird, ebenso wie dann jener zu 50% mit seinem Vorgängervoxel gemischt wird. Beide Mischungen liefern das gleiche Ergebnis, es erfolgt ein stetiger Übergang zwischen den beiden Voxeln.

Diese Kopplung ist in Abbildung 4.30 illustriert. Man beachte, dass nur die im roten Kasten enthaltenen Pixel allein durch die vier dargestellten Voxel beeinflusst werden und daher nur diese Pixel als Ergebnis zu betrachten sind.

Dennoch können bei diesem Verfahren natürlich Artefakte auftreten, z.B. wenn Voxel aus dem Hintergrund durch ein kleines Loch in einer stark verkippten Vordergrundebene betrachtet werden. Es erscheint dann ggf. ein sternförmiges „Glanzlicht“ (s. Abbildung 4.31), das aber nur selten zu beobachten ist und nicht sehr störend wirkt.



Blickt man durch ein kleines Loch in einer Vordergrundebene (gebildet von den Voxeln 1-4) auf einen Hintergrundvoxel, so kann es zu sternförmigen Artefakten kommen. Die Vordergrundvoxel 1-4 sind links nur zur Verdeutlichung ihrer Ausdehnung verschieden gefärbt. Sie seien für den Algorithmus alle schwarz (rechts).

Abbildung 4.31: Mögliche Artefakte beim Aliasfilter

Vorteil des Aliasfilters gegenüber herkömmlichen Gaußfiltern:

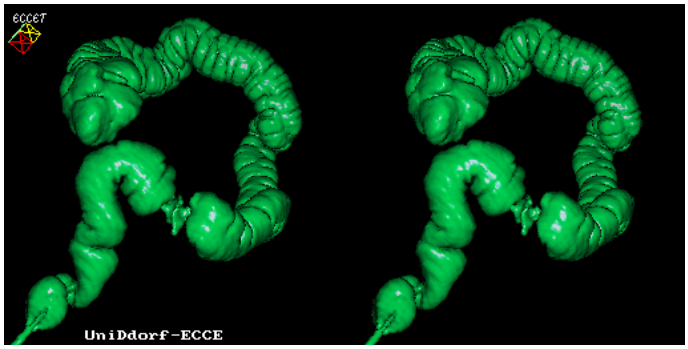
- Das Filter stellt sich durch die Berücksichtigung der Voxelgröße automatisch auf verschieden starke Verschmierung ein, abhängig davon, ob die dargestellten Objekte (deren Kanten bekannt sein müssen) großflächig oder klein sind.
- Die Stärke der Verschmierung wird dabei für die beiden aneinandergrenzenden Objekte getrennt bestimmt, so dass sich keine Verschmierung kleiner Details in der Nähe großflächiger Vordergrundobjekte ergibt.
- Das Filter benötigt nur einen zweimaligen Durchlauf durch das Bild, wobei jedes Pixel nur einen Read-Modify-Write-Zyklus durchläuft, plus einen Read-Zyklus für alle Kantenpixel.
- Das Filter kann in-situ laufen.

4.2.5 Erzeugung von stereographischen Ansichten

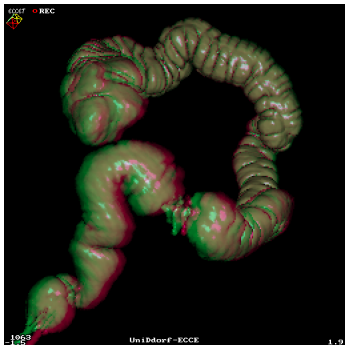
VOXREN ist in der Lage, stereographische Ansichten zu erzeugen. Das heißt, VOXREN rechnet **zwei** Bilder aus zwei verschiedenen Ansichten, wie sie die beiden menschlichen Augen hätten.

Dazu wird beim Raycasting für jeweils ein Bild die Kameraposition um einen einstellbaren Betrag verschoben und ihre Richtung optional noch um einen kleinen Winkel verdreht.

Um dem jeweiligen Auge das zugehörige Bild zu präsentieren, beherrscht VOXREN verschiedene Darstellungsmodi:



a) 3D-Darstellung zur direkten Fusion. Betrachten Sie die beiden Bilder ähnlich wie ein Random-Dot-Stereogram („magisches Auge“-Bilder), indem Sie versuchen „hindurchzusehen“. Die beiden Bilder fusionieren dann mit etwas Übung zu einem 3D-Bild.



b) Mit Hilfe einer Rot-Grün-Brille (erhältlich z.B. bei [PSP]) kann auch ohne Übung ein dreidimensionaler Eindruck erreicht werden, leider prinzipbedingt nur monochrom.

Abbildung 4.32: Erzeugung stereographischer Ansichten

Direkte Fusion

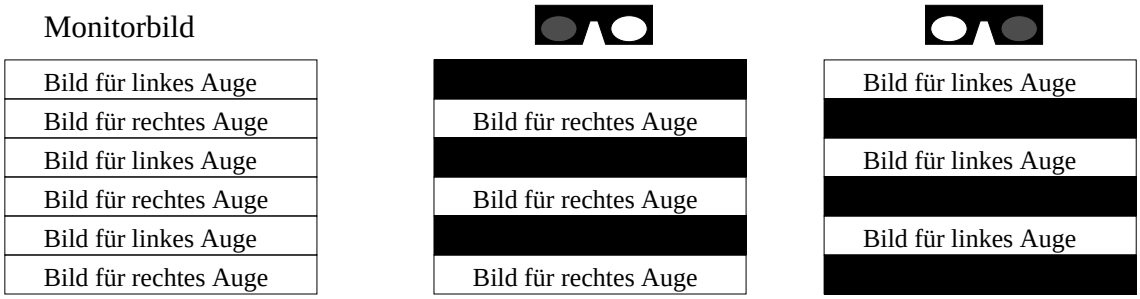
Stellt man beide Bilder einfach nebeneinander dar, kann der Betrachter mit etwas Übung die beiden Bilder fusionieren und somit ohne irgendwelche Hilfsmittel einen dreidimensionalen Eindruck der dargestellten Szene erhalten.

Allerdings ist das Verfahren für einen untrainierten Betrachter nicht ganz leicht zu erlernen, da Fusionspunkt und Bildschärfe des Auges entgegen unserer üblichen Gewohnheit nicht zusammenfallen.

Fusion mit optischen Hilfsmitteln

Mit Hilfe von Prismenbrillen kann die Fusion erleichert werden. Dazu werden die beiden Bilder übereinander dargestellt und durch zwei entgegengesetzt geneigte Prismen jeweils einem Auge zugeführt.

Allerdings ist dieses Verfahren stark vom Abstand des Betrachters von den Bildern und dem Abstand der beiden Bilder zueinander abhängig, so dass der Betrachtungskomfort in der Regel deutlich kleiner ausfällt als bei der direkten Fusion. Lediglich der Trainingsaufwand wird verringert, da er sich nun auf ein rein mechanisches Suchen des optimalen Betrachtungspunktes beschränkt, statt auf ein Trainieren der Augen.



Beim Lineblanking-Verfahren wird ein Bild erzeugt, das die Bilder für das linke und rechte Auge zeilenweise abwechselnd enthält. Ein zwischen Grafikkarte und Monitor geschaltetes Gerät blendet abwechselnd die geraden bzw. ungeraden Zeilen aus und steuert gleichzeitig die Shutterbrille an, so dass jeweils ein Auge abgedeckt wird.

Abbildung 4.33: Lineblanking-Verfahren

Rot-Grün-Brillen

Ohne Trainingsaufwand und mit kostengünstigen Materialien kommt dagegen die Rot-Grün-Darstellung aus. Dabei wird das Bild auf seine Helligkeitsinformation reduziert und eines der Bilder in Rot und das andere in Grün dargestellt. In Überlappungsbereichen werden die Farben additiv gemischt. Durch eine kostengünstige Pappbrille (ca. 50 Cent) mit je einem Rot- und einem Grünfilter wird erreicht, dass ein Auge nur noch das grüne, das andere nur noch das rote Bild sieht.

Dabei geht allerdings die Farbinformation bis auf Helligkeitsunterschiede und den Blauanteil (der von beiden Filtern durchgelassen wird) verloren.

Zeileninterleaving (Shutterbrille)

Unter Verwendung einer Shutterbrille (z.B. von [IArt]) kann jedem Auge explizit das korrekte Bild zugespielt werden. Am einfachsten geschieht dies im sog. Line-Blanker-Modus.

Dabei verdunkelt eine zwischen Monitor und Grafikkarte befindliche Schaltung in jedem geraden bzw. ungeraden Bild die jeweils geraden bzw. ungeraden Zeilen und steuert dabei gleichzeitig die rechte bzw. linke Hälfte der Shutterbrille dunkel.

Dadurch sieht das rechte Auge z.B. nur die geraden Zeilen und das linke die ungeraden. Die Software verschachtelt nun die berechneten Bilder entsprechend und verdoppelt dabei die Pixel in x-Richtung, um das Aspektverhältnis zu wahren.

Das Verfahren hat den Vorteil, dass auch ein untrainierter Beobachter den 3D-Effekt erfassen kann. Allerdings führt es zu einer Halbierung der effektiven Bildwiederholrate, was zumeist zu erheblichem Flimmern führt.

Dieser Nachteil ist, ebenso wie die verminderte mittlere Leuchtdichte, allerdings allen Shutterverfahren gemeinsam. Das Lineblanking hat den Vorteil, dass keine speziellen Treiber auf der Softwareseite erforderlich sind — allerdings ist ein leichter horizontaler Streifeneffekt festzustellen, der z.B. beim VSync-Doubling nicht auftritt.

Erst schnelle Shuttersysteme, die zeilenweise umschalten, würden dem Flimmern abhelfen.

Ein weiterer Nachteil ist, dass aus zwei Gründen keine LCD-Bildschirme zur Darstellung verwendet werden können:

1. Shutterbrillen verwenden Polfilter zusammen mit LCD-Elementen, um den Abdunkelungseffekt zu erreichen. Da ein LCD-Monitor dieselbe Technik benutzt, gibt er aber bereits polarisiertes Licht ab, so dass man ggf. gar nichts mehr sehen kann.
2. LCD-Monitore laufen meist nicht synchron zum Ansteuersignal, sondern werden mit einem konstanten, vom Steuersignal unabhängigen Takt betrieben. Das gilt ebenfalls für die meisten auf dem Markt befindlichen Projektoren. In der Folge stimmt das vom Lineblanker ausgestrahlte Synchronsignal nicht mit den auf dem Bildschirm angezeigten Daten überein.

Spalteninterleaving (3D-Displays)

Einige Displays (z.B. Dresden-3D [D3D]) arbeiten mit verschiebbaren Prismenmasken vor einem LCD-Schirm. Dadurch wird durch winkelgerichtete Abstrahlung der beiden Bilder jeweils einem Auge ein Bild zugeführt, diesmal allerdings zulasten der horizontalen Auflösung.

Da hier aber kein Shuttersystem notwendig ist, entsteht kein Flimmern bzw. es muss auch keine spezielle Brille getragen werden.

Für VOXREN ist das Verfahren im Wesentlichen dasselbe, nur dass eben horizontal verschachtelt und vertikal verdoppelt wird.

4.2.6 Alternative Tiefendarstellung mit Nebel-/Plasmaeffekten

Doch nicht immer können die stereographischen Verfahren genutzt werden, um die räumliche Tiefe eines Bildes zu transportieren — so z.B. wenn man nur ausgedruckte Bilder weitergeben kann, die ohne Hilfsmittel oder Training erfassbar sein sollen.

In diesem Fall ist es auch nicht möglich, das Problem mit Hilfe der interaktiven Beweglichkeit der Kamera in VOXREN zu lösen. Damit könnte man, anhand der unterschiedlichen Winkelgeschwindigkeiten von Vorder- und Hintergrundstrukturen bei Bewegung der Kamera, die Zuordnung zu treffen.

Die Tiefeninformation kann aber überaus wichtig sein: Wie Abbildung 4.34a recht deutlich zeigt, kann der Gefäßbaum der Leber recht komplex sein. Eine zweidimensionale Abbildung kann keine wirklich adäquate Darstellung mehr liefern, da es sehr schwer wird, die dreidimensionale Lageinformation nur noch aus Verdeckungen zu erschließen.

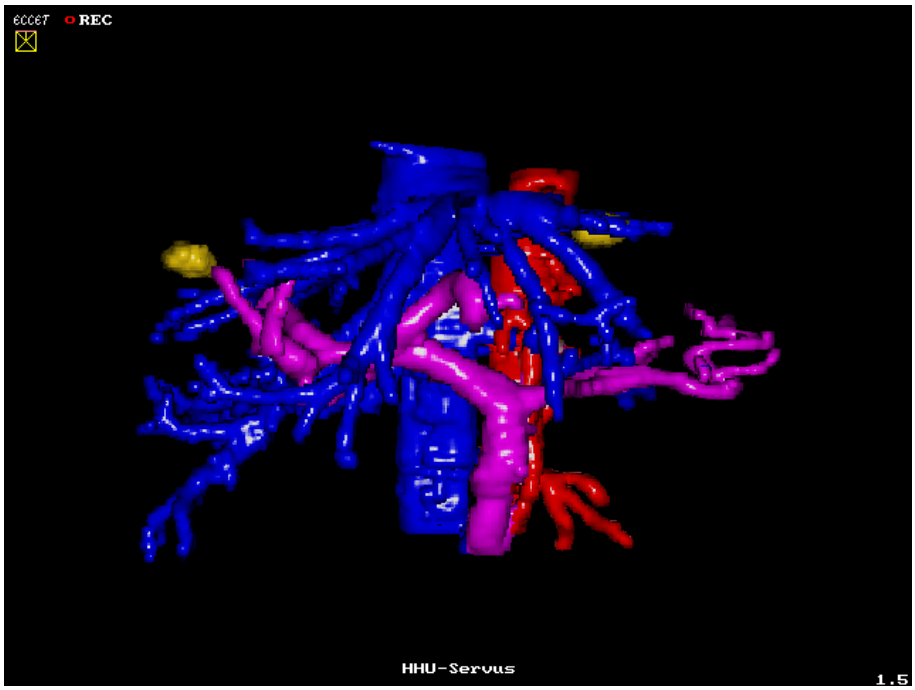
Eine alternative Methode, die es ermöglicht, die Tiefenwirkung mit nur einem Bild zu transportieren, ist die in der Kunst verbreitete Technik der Nutzung von Nebel.

VOXREN ist in der Lage einzelne Objekte so zu rendern, als wären sie mit einem semitransparenten, leuchtenden Plasmanebel gefüllt (s. Abbildung 4.34b).

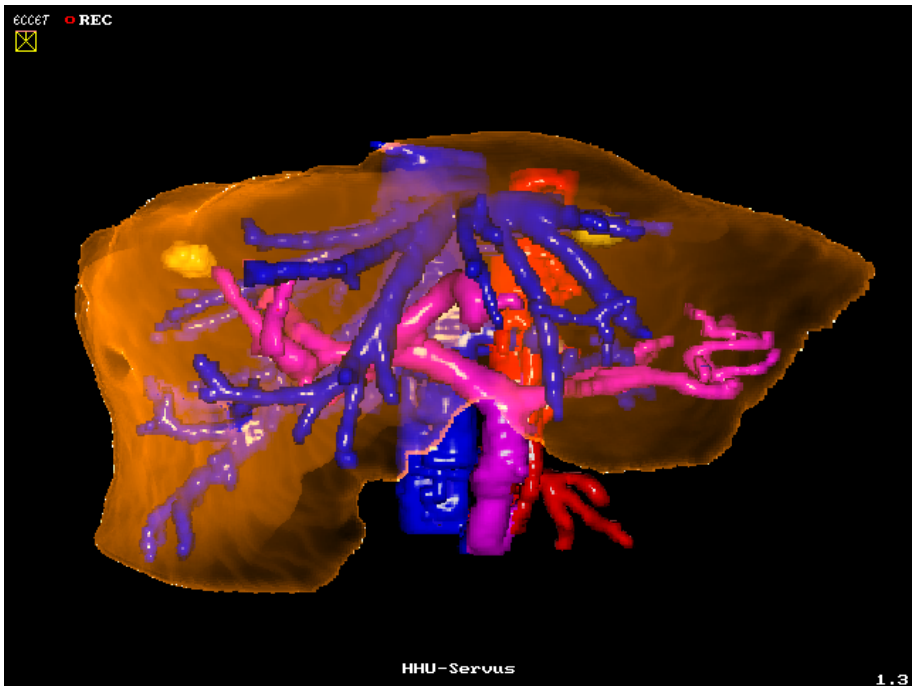
Das führt dazu, dass weiter hinten gelegene Objekte stärker abgedunkelt werden als weiter vorne befindliche und ebenso stärker von der Eigenfarbe des Nebels überdeckt werden.

Dieser Effekt ist aus der Alltagserfahrung vertraut und ermöglicht uns ein intuitives Erschließen der räumlichen Tiefe.

Die Geschwindigkeit der Darstellung leidet dabei nicht merklich gegenüber der einfachen semitransparenten Darstellung.



a) Gefäßsystem der Leber



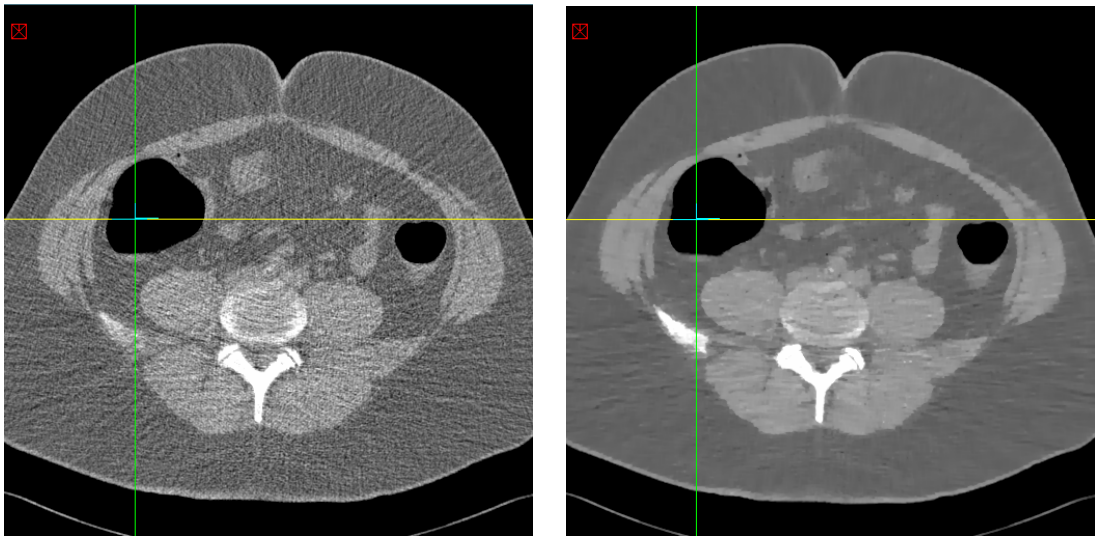
b) Durch den Plasmaeffekt kann die relative Lage der Gefäße zueinander wesentlich besser eingeschätzt werden.

Abbildung 4.34: Tiefendarstellung mit Nebel-/Plasmaeffekten

Kapitel 5

Vorverarbeitungsalgorithmen

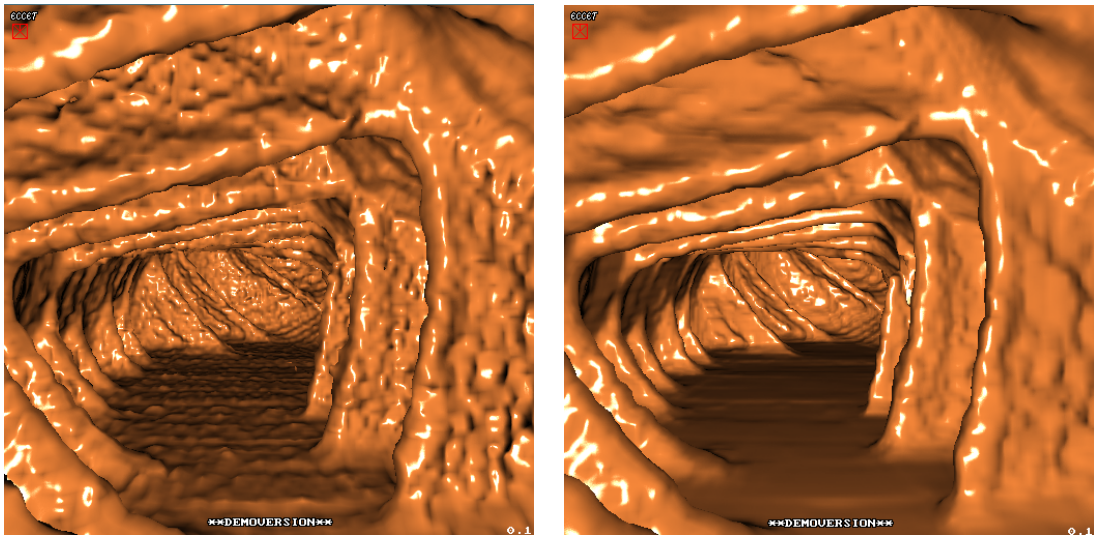
Um in der Visualisierung optimale Ergebnisse sowohl bezüglich der Qualität der angezeigten Bilder als auch der Detektion von Anomalien und nicht zuletzt auch bezüglich der Geschwindigkeit zu erhalten, bietet $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ die Möglichkeit, eine ganze Palette an Vorverarbeitungsalgorithmen auszuführen, von denen die wichtigsten hier kurz erklärt werden sollen.



a) Grauwertansicht eines mit Niedrigdosis aufgenommenen Darmvolumens

b) Ansicht nach der Filterung

Abbildung 5.1: Filterung mit dem nichtlinearen Gaußfilter



a) Mit PLANEVIEW gerenderte Ansicht vor der Filterung

b) Ansicht nach der Filterung

Abbildung 5.2: Filterung mit dem nichtlinearen Gaußfilter - 3D-Ansicht

5.1 Filterung

Zur Vorfilterung stark verrauschter Daten steht ein bereits früher am Institut entwickeltes nichtlineares Gaußfilter zur Verfügung, das zur einfacheren Bedienung in $\epsilon CC\epsilon\mathcal{T}$ integriert wurde.

Insbesondere bei der Anwendung als Diagnosetool am menschlichen Patienten unter Verwendung der Computertomographie erlaubt das Filter eine Reduktion der eingesetzten Strahlendosis, was für den Bereich der Prävention (Screening) das Verfahren erst sinnvoll anwendbar macht. Siehe dazu auch Abschnitt 6.4.

Für die genaue Funktionsweise des Verfahrens sei auf [NLG] und [NLG2] verwiesen. Es wird im Wesentlichen unverändert eingesetzt.

Dabei muss lediglich ein kleines Problem umgangen werden:

5.1.1 Speicherplatz

Da das NLG-Verfahren voraussetzt, dass ein Quell- und ein Zielvolumen vorhanden sind, wird theoretisch der doppelte Speicherplatz benötigt. Dieser steht jedoch nicht zur Verfügung.

Allerdings findet diese Vorverarbeitung zu einem Zeitpunkt statt, zu dem die Klassenzuordnung und die Tiefenabbildung noch keine Bedeutung haben. Die Implementation nutzt daher diese Bereiche als Zielvolumen und vermeidet damit die Verwen-

dung einer programmierfehleranfälligen Lösung mit gleitenden Fenstern zur Rettung später noch benötigter Originalwerte.

5.1.2 Einschränkung des zu filternden Volumens

Häufig ist eine Filterung des **kompletten** Volumens gar nicht erforderlich. Es genügt, nur die interessanten Bereiche zu filtern. Zum Beispiel machen bei einer typischen Koloskopieaufnahme die innerhalb des Darmes befindlichen Voxel nur ca. 2-4% der gesamten Aufnahme aus. Entsprechend kann die Ausführung der Filterung auf das ca. 20-fache beschleunigt werden, indem man zunächst grob den Darm lokalisiert und nur diesen Bereich filtert. Diese Optimierung wurde so implementiert, dass das Filter alle Grauwerte mit einem reservierten Wert (65535) ignoriert. Da das NLG-Filter Werte aus der Umgebung des zu filternden Bereiches verwendet, ist es notwendig, zu den interessierenden Strukturen einen geeignet breiten Rand hinzuzunehmen, den man durch Dilation (s. Abschnitt 6.3) gewinnt. Alle nicht in dieser Auswahl enthaltenen Voxel werden nun auf 65535 gesetzt und der NLG-Algorithmus ausgeführt.

Damit wurde es möglich auf einem 2x 1 GHz Pentium III PC die komplette Vorverarbeitung der Daten (Filterung, Segmentierung und Markierung von Polypen) für eine solche Untersuchung in einem vollautomatisierten Prozess zusammenzufassen, der typischerweise 6-8 Minuten benötigt. Eine detaillierte Beschreibung dieses Algorithmus findet man in 6.4.

5.2 Vorberechnung der Oberflächennormalen

Um eine schnelle und trotzdem optisch hochwertige Darstellung zu erreichen, verwendet VOXREN vorberechnete Oberflächennormalen, die im Voxelvolumen gespeichert werden.

5.2.1 Einfache integrierte Schätzung

In VOXREN integriert ist eine relativ einfache Methode, die die Oberflächennormalen auf dem Rand einer massiven Voxelmenge A approximiert:

Die Schätzung der Normalen erfolgt durch gewichtete Summation der Vektoren zu allen Nachbarpunkten eines Randpunktes, die in A liegen:

$$\vec{n}(\vec{x}) = \sum_{\vec{y} \in A} \frac{(\vec{y} - \vec{x})}{|\vec{y} - \vec{x}|} \cdot e^{-\frac{(\vec{y}-\vec{x})^2}{2\sigma^2}} \tag{5.1}$$

In der diskreten Implementation wird die Summation natürlich auf endlich viele $\vec{y}$ begrenzt. Da die Beiträge mit $|\vec{y} - \vec{x}|$ exponentiell abfallen, begrenzt man diesen Term auf typisch 4σ.

Dieses Verfahren funktioniert für einigermaßen dicke Objekte mit einer ausreichenden Genauigkeit, ist sehr schnell und stabil.

Es ist nicht geeignet zur Darstellung von extrem dünnen Objekten (Linien, Oberflächen).

Allerdings kann auch für diese Fälle eine akzeptable Darstellung erreicht werden. Es kann ein Parameter eingestellt werden, mit dem der Betrag des Resultatvektors verglichen wird. Bei zu kleinem Resultatvektor wird das betreffende Voxel als „von allen Seiten voll opak darzustellen“ markiert. Über einen weiteren Parameter kann darüber hinaus erreicht werden, dass bei noch kleineren Resultatvektoren das Voxel mit sinkender Resultatlänge zunehmend als transparent dargestellt wird. Bei der Berechnung von dünnen Objekten wie den Enden von Aderngeflechten ergeben sich hierdurch weniger Flimmerartefakte durch die mehr oder weniger zufälligen Richtungen der ansonsten aus dem kleinen Resultatvektor berechneten Normalen.

5.2.2 Erweiterte Verfahren

In der Arbeit [SGR] werden z.T. aufwändigere Alternativverfahren untersucht, die unter Nutzung erweiterter Kenntnisse (z.B. der Originalgrauwerte) bzw. erhöhtem Rechen- und/oder Speicheraufwand noch geringfügig bessere Ergebnisse liefern und Spezialfälle (dünne Wandungen z.B.) besser approximieren.

Eine Integration in VOXREN erfolgte wegen der erhöhten Speicheranforderungen nicht. Allerdings ist es natürlich möglich, diese Verfahren extern laufen zu lassen und die Ergebnisse in VOXREN zu importieren, wenn eine spezielle Anwendung dies erforderlich machen sollte.

Da PLANEVIEW keinen Platz für die Speicherung einer Normalen bietet, dafür aber die Originalgrauwerte zur Verfügung hat, wird dort für jeden dargestellten Punkt die Normale während der Berechnung des Bildes geschätzt. Die dazu eingesetzten Verfahren sind in 4.2.3 erläutert und werden zum Teil auch in [SGR] untersucht.

5.3 Geometrische Distanzabbildung

In VOXREN wird an verschiedenen Stellen die geometrische Distanzabbildung genutzt, z.B. zur Beschleunigung der Darstellung und zum interaktiven Segmentieren des Darmes. Worum handelt es sich dabei, und wie kann sie auf diskreten mehrdimensionalen Feldern effizient berechnet werden?

Definition:

Gegeben sei ein metrischer Raum X mit Metrik d sowie eine Teilmenge $A \subset X$.
Die geometrische Distanzabbildung $g_A(x)$ ist dann definiert als

$$g_A(x) := d(x, A) = \min\left(d(x, a) : a \in A\right) \ . \tag{5.2}$$

Anschaulich gesprochen liefert sie die Entfernung von einem Punkt zum nächstgelegenen Punkt der Menge A.

Auf diskreten Gittern lässt sich die geometrische Distanzabbildung für bestimmte Metriken effizient berechnen bzw. approximieren. Insbesondere gilt dies für die von der 1- und der ∞-Norm induzierten Metriken.
Die euklidische Metrik (2-Norm) lässt sich mit diesen Verfahren leider nur approximieren.

5.3.1 Effiziente Berechnung

Die direkte Berechnung erfordert pro zu berechnendem Punkt $\#A$ Auswertungen der Distanz, was insbesondere bei großen Mengen A sehr viel Rechenzeit kostet.

Auf diskreten Gittern, wie sie für diese Arbeit von Interesse sind, gibt es eine Optimierungsmöglichkeit, falls man die 1- oder ∞-Norm verwendet. Deren zugrundeliegende Idee besteht darin, von der Menge auszugehen und dort zunächst alle benachbarten (sprich in der Kugel mit Radius 1 gelegenen) Punkte zu suchen. Sofern diese Punkte nicht ohnehin bereits zur Menge A gehören und somit per Definition den Abstand 0 haben, haben sie per Konstruktion den Abstand 1 von A. Von dort findet man alle Punkte mit Abstand 2 usw.

Allerdings wäre eine direkte Implementation mit Hilfe von z.B. Breitensuche zwar denkbar, aber in der Praxis ungünstig, da das RAM-Zugriffsmuster dadurch stark fragmentiert würde und entweder mit jeder Iteration die entsprechenden Startpunkte neu gesucht werden müssten oder Speicher für eine Liste belegt werden muss.

Es ist jedoch eine rekursive Implementierung bekannt, die den gleichen Effekt durch zwei lineare Durchgänge durch das gesamte Gitter erreicht.

Man beginnt mit einem Gitter, auf dem alle Punkte, die zu A gehören, mit 0, die anderen mit ∞ vorbelegt sind. In einem ersten Durchgang wird das Gitter zunächst vorwärts durchlaufen. Dabei durchsucht man für jeden Punkt die 1-Kugel nach dem Punkt mit dem kleinsten eingetragenen Abstandswert. Es genügt dabei, nur diejenigen Punkte zu betrachten, die man dabei selbst zuvor schon bearbeitet hat. Zum gefundenen kleinsten Abstandswert addiert man 1 und trägt ihn am aktuellen Punkt ein. Dabei bildet sich aufgrund der Rekursion ein Feld, das für jeden Punkt jeweils bereits die GDA bezüglich der Vorgängerpunkte korrekt enthält.

Im zweiten Durchgang wählt man die umgekehrte Laufrichtung, so dass letztlich für jeden Punkt alle Nachbarn in der Berechnung berücksichtigt wurden.

Dieses Verfahren erscheint inhärent seriell zu sein, was die Abarbeitung der beiden Teilschritte betrifft.

Da die Berechnung der GDA für schnelle Darstellungen und einige andere Algorithmen in $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ notwendig ist und relativ lange dauert, wäre eine Parallelisierung wünschenswert.

Betrachtet man das Verfahren allerdings aus einem leicht veränderten Blickwinkel, so wird es zum einen leichter verständlich und zum anderen erkennt man leicht eine Möglichkeit zur Parallelisierung:

5.3.2 Parallele Betrachtung

Betrachten wir die Vorgehensweise einmal beispielhaft für einen einfachen eindimensionalen Fall:

Gegeben sei ein eindimensionales Gitter, in das an allen Punkten $a \in A$ bereits 0 eingetragen ist.

G				0							0		

In einem ersten Schritt bewegt man sich nun von links nach rechts über das Gitter und trägt auf den freien Positionen jeweils den minimalen Abstand zum nächsten, *weiter links gelegenen* Randpunkt ein.

Dazu genügt es, jeweils den links von der aktuellen Position eingetragenen Wert auszulesen, um 1 zu erhöhen und einzutragen.

G_1				0	1	2	3	4	5	6	0	1	2

Im mehrdimensionalen Fall müsste über alle bereits vorher befüllten Felder das Minimum gebildet und 1 addiert werden.

Zum Verständnis des Verfahrens denken wir uns nun eine Kopie des ursprünglichen Gitters, die nach dem gleichen Verfahren, aber von rechts nach links behandelt wurde:

G_2	3	2	1	0	6	5	4	3	2	1	0	

Das Gitter G_1 beinhaltet nun die Abstände zu allen weiter links gelegenen Punkten, G_2 die zu allen weiter rechts gelegenen.

Bildet man nun aus beiden feldweise das Minimum, so erhält man die gewünschte Abbildung:

G_1				0	1	2	3	4	5	6	0	1	2
G_2	3	2	1	0	6	5	4	3	2	1	0		
$\min(G_1,G_2)$	3	2	1	0	1	2	3	3	2	1	0	1	2

In der Implementation erfolgt natürlich keine Trennung der 2. Berechnung vom Minimierungsschritt, sondern man belegt das das Gitter darstellende Array mit einem nicht erreichbaren Maximalwert vor und trägt an jeder Stelle $g(y)$ den neuen Wert $\min(g(x) : x \in \text{Vorgänger von y}) + 1$ nur ein, wenn dieser kleiner als der bisherige Wert ist.

Dieses Verfahren kann problemlos auf höhere Dimensionen erweitert werden. Allerdings funktioniert das nicht mit jeder Metrik (im Eindimensionalen fallen die 1-, 2- und ∞-Metriken zusammen), sondern nur, wenn die neuen Distanzen iterativ aus denen der Vorgängerpunkte berechnet werden können.

Bei der 1- und ∞-Norm kann dies durch geeignete Wahl der Vorgänger erreicht werden. Im zweidimensionalen Fall betrachtet man zur Berechnung der GDA bezüglich 1-Norm jeweils nur den unmittelbar oberhalb und den links vom aktuellen Punkt liegenden Nachbarn (bzw. unterhalb/rechts im 2. Schritt), während man für die ∞-Norm die vier Nachbarpunkte (oben-links, oben, oben-rechts, links) betrachtet.

Eine Näherung für die 2-Norm kann erreicht werden, indem die schräggelegenen Nachbarn mit ca. $\sqrt{2}$ bewertet werden. Es bleibt aber eine Näherung, da man lediglich „die Länge des kürzesten Weges längs im 45°-Raster liegender Geraden“ erhält.

5.3.3 Parallelisierung

Nun wird deutlich, wie sich dieses Verfahren teilweise parallelisieren läßt. Durch Nutzung von 2 Prozessoren gelingt es, das Verfahren um einen Faktor 2 gegenüber der traditionellen Implementierung zu beschleunigen.

Dazu betrachtet man die Vor- und Rückrichtung als getrennten Schritt, wie im letzten Abschnitt dargestellt. Der 1. und 2. Schritt kann völlig parallel ablaufen — man benötigt lediglich anschließend einen „Merge“ -Schritt. Allerdings ergibt sich das Problem, dass hierfür das Feld doppelt vorhanden sein müsste, was aus Speicherplatzgründen nicht erstrebenswert ist.

Die Idee zur Lösung dieses Problems besteht darin, das Volumen in der Mitte zweizuteilen und in der oberen Hälfte mit Schritt 1 zu beginnen, während man in der unteren Hälfte mit Schritt 2 beginnt. Da jeweils bezüglich der Schrittrichtung nur bereits bearbeitete Werte benutzt werden, ergeben sich keinerlei Locking- und Kohärenzprobleme — alle von einem Thread bearbeiteten Daten werden *ausschließlich* von diesem gelesen und geschrieben. Nun vertauscht man die Rollen und macht mit dem rückwärtslaufenden Schritt in der oberen Hälfte und dem vorwärtslaufenden Schritt in der unteren Hälfte weiter.

Der jeweils andere Schritt darf aber erst ablaufen, wenn für alle von diesem Schritt benötigten Umgebungsfelder der *vorherige* Schritt vollständig durchgeführt wurde. Diese Bedingung wird bei den beiden in der Mitte adjazenten Ebenen verletzt. Daher wird zuvor Schritt 1 noch für die oberste Ebene des unteren Teilstapels durchgeführt und analog Schritt 2 für die unterste des oberen Teilstapels. Durch sequentielle Ausführung dieses Schrittes vermeidet man die Lockingprobleme, die sich durch den wechselweisen Zugriff zweier Threads auf gemeinsame Daten ergeben würden.

Danach kann der Rollentausch stattfinden, der zweite Schritt läuft dann analog zum ersten so, dass immer nur ein Thread die Daten in seinem Teilfeld liest und schreibt.

Die sequentiell berechneten 2 Ebenen sind in der Regel gegenüber der Gesamtzahl von typisch 600-800 Ebenen zu vernachlässigen und man erhält in etwa die doppelte Geschwindigkeit.

Kapitel 6

Segmentierungsalgorithmen

VOXREN basiert auf einer „harten“ Segmentierung, also der Einteilung der vorhandenen Voxel in zwei Klassen (gehört zum Objekt bzw. gehört nicht dazu). Erst dann können die zur Visualisierung notwendigen Daten berechnet werden.

Diese Einteilung kann durch verschiedene Verfahren getroffen werden, deren Auswahl von der gestellten Segmentierungsaufgabe abhängt.

6.1 Einfache Schwellwertentscheidung

Im einfachsten Fall ist das zu segmentierende Organ bzw. Objekt durch einen Grauwertbereich gekennzeichnet, der in seiner Umgebung nicht auftritt.

Dieser Fall tritt z.B. bei Werkstücken auf, deren Materialien sich klar unterscheiden, aber z.B. auch bei der virtuellen Koloskopie.

In letzterem Fall wird der Dickdarm (Colon) vor der Untersuchung mit Gas gefüllt, was sowohl eine Aufweitung (Dilation) als auch einen klaren Kontrast erzeugt, da Gas die Röntgenstrahlung im Gegensatz zu Gewebe fast nicht absorbiert.

Allerdings ist natürlich nicht jeder auf einer Aufnahme zu sehende dunkle (also gasgefüllte) Bereich dem Dickdarm zugehörig; zumeist gelangt auch etwas Luft in den Dünndarm und des Weiteren gibt es auch natürliche luftgefüllte Hohlräume (Lunge) und auch die außerhalb des Körpers befindliche Luft wird natürlich von einer einfachen Schwellwertentscheidung mitmarkiert.

Diese unerwünschten Markierungen können auf zwei Arten unterdrückt werden:

- Durch Verwendung einer interaktiven Segmentierung, die Zusammenhangskomponenten a priori berücksichtigt (z.B. siehe Abschnitt 6.2) oder

a) Markierung von Knochen b) Markierung von Luft

Abbildung 6.1: Einfache Schwellwertentscheidung

- durch eine spätere interaktive Neuklassifizierung von Zusammenhangskomponenten, mit der es z.B. schnell möglich ist, den Darm, die Hautoberfläche, die Patientenliege sowie die sonstigen Lufteinschlüsse (Lunge, Dünndarm) zu trennen (siehe Abschnitt 6.6.1).

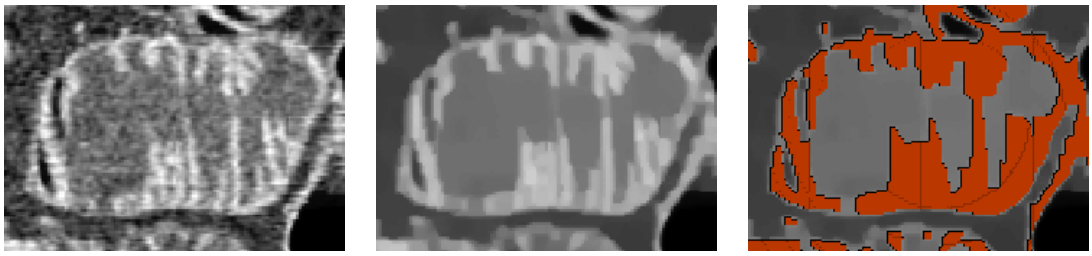
Dabei ist der ersten Methode der Vorzug zu geben, wenn nur der Dickdarm segmentiert werden soll, da sie eine sauberere Trennung ermöglicht.

Gerade Lufteinschlüsse im Dünndarm führen beim zweiten Verfahren oft zu Problemen: Bei der Erzeugung der Oberflächen wird im Interesse einer möglichst korrekten Darstellung des Darmlumens die Oberfläche auf der Darmwand (nicht im Darminneren) erzeugt. Da diese Wände teilweise sehr dünn sind (dünner als die Ausdehnung eines Voxels), kommt es häufig vor, dass benachbarte Dick- und Dünndarmwände innerhalb desselben Voxels zu liegen kommen. Folglich verschmelzen die Zusammenhangskomponenten der Wände, so dass ein nachträgliches Trennen über die zweite Methode nicht mehr möglich ist.

6.2 Mehrstufen–Füllalgorithmus

6.2.1 Problemstellung

Speziell für die Anforderung von Untersuchungen an Dick- und Dünndarm wurde ein mehrstufiger Füllalgorithmus entwickelt, um die Probleme, die herkömmliche



a) Rohdaten des Dünndarms b) nach Glättung gemäß 5.1 c) mit Stoppmarkierung

Abbildung 6.2: Mehrstufen–Füllalgorithmus - Vorbereitung

Algorithmen hier bereiteten, zu umgehen.

Besonders im Bereich des Dünndarms ist oft nur ein schlechter Kontrast zwischen Darmwand und dem Kontrastmittel (Paraffin) gegeben. Damit fallen einfache Schwellwertentscheidungen wie in 6.1 als Segmentierungsverfahren aus. Doch auch bei Einsatz aufwändigerer Verfahren wie z.B. vorheriger Kantenverstärkung bleiben häufig zumindest kleinere Löcher in der Darmwand, die einfachen Füllalgorithmen Probleme mit dem Auslaufen in den Außenraum bereiten.

In den Abbildungen 6.2 sehen Sie die Problematik illustriert:

a) Die ursprünglichen Aufnahmen sind relativ stark verrauscht (die Standardabweichung des Rauschens beträgt $\sigma \approx 18$ Grauwerte), der Kontrast zwischen Darmwand und Innerem ist nicht sehr hoch (ca. 70 Grauwerte) und stellenweise ist eine geschlossene Darmwand nicht mehr zu erkennen (z.B. links unten).

b) Nach Filterung ist zwar das Rauschen verschwunden, es bleibt aber die Problematik mit den „Löchern“ .

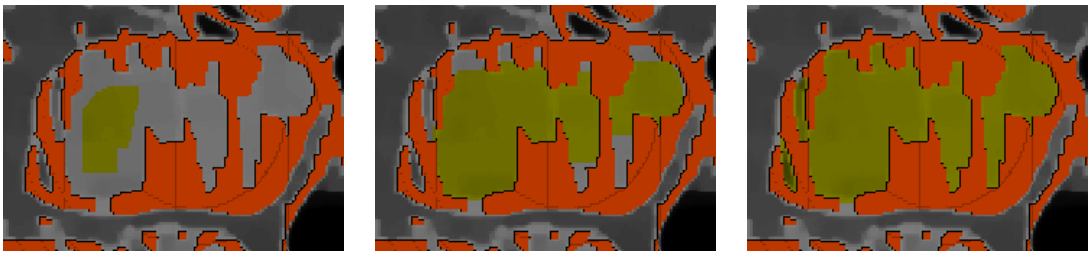
c) Eine schwellwertbasierte Segmentierung liefert hier keine geschlossene Figur mehr.

6.2.2 Algorithmus

Folgender Algorithmus ist in der Lage, einen nicht vollständig geschlossenen Bereich, dessen Löcher kleiner als ein vorgegebener Radius d sind, zu füllen:

Voraussetzung

Gegeben sei eine Menge A, die den Rand des zu füllenden Volumens bildet und keine Löcher mit einem Radius $\geq d$ enthalte. Das heißt, es sei unmöglich, eine Kugel mit



d) nach Skelettfindung

e) nach dem Expansionsschritt

f) Endzustand

Abbildung 6.3: Mehrstufen–Füllalgorithmus - Einzelschritte

Radius d von innen nach außen zu bewegen ohne die Menge A damit zu schneiden.[1]

Bezüglich der Menge A stehe auf dem Voxelgitter die geometrische Distanzabbildung zur Verfügung. Diese kann nach 5.3 effizient berechnet werden.

Weiterhin sei ein beliebiger Punkt x **innerhalb** des zu füllenden Volumens gegeben. Dieser Punkt wird vom Benutzer manuell vorgegeben.

Als weitere Parameter seien die maximale 2D-Taschengröße m und die Iterationsanzahl n vorgegeben[2].

Schritt 1 - Finden eines Startpunktes

Der Algorithmus benötigt zum Start einen Punkt s innerhalb des Volumens, für den $gda_A(s) > d$. In zweidimensionalen Schnittbildern sind diese Punkte für den Benutzer jedoch oft nicht leicht zu erkennen, da zwar auf dem Schnittbild alle sichtbaren Punkte von A weit entfernt sein können, aber sich ober- oder unterhalb des dargestellten Schnittes weitere nahegelegene Punkte von A befinden können.

Daher wird zunächst ausgehend vom durch den Nutzer vorgegebenen Punkt x per einfachem Gradientenabstieg nach einem lokalen Maximum gesucht.

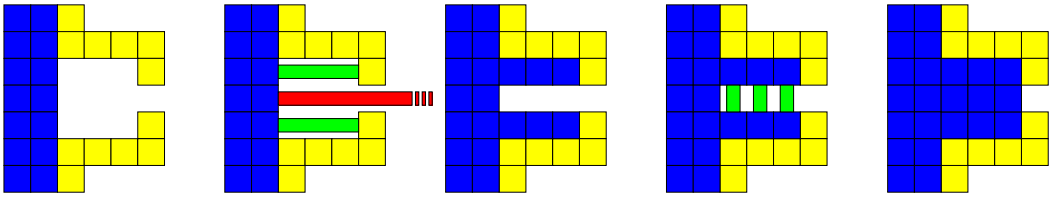
In der Regel erreicht man dabei einen Punkt s mit $gda_A(s) > d$. Ist dies nicht der Fall (z.B. weil man in eine kleine Ausbuchtung geklickt hat), so wird der Algorithmus hier abgebrochen und der Benutzer aufgefordert, einen anderen Punkt x zu wählen.

Schritt 2 - Skelett finden

Nun startet eine Breitensuche im gefundenen Punkt s. Dabei werden nur Punkte markiert, deren GDA größer als d ist (Ergebnis s. Abbildung 6.3d). Bei korrekter

[1] Im Beispiel wird diese Menge A mit Hilfe einer einfachen Schwellwertentscheidung gefunden.

[2] Hinweise zur sinnvollen Wahl dieser Werte folgen in den anschließenden Bemerkungen.



a) Tasche vor dem ersten Füllschritt.

b) Erster Schritt. Die mittlere Ebene läuft aus.

c) Daher ergibt sich eine Kerbe auf Höhe des Loches

d) Zweiter Füllschritt in orthogonaler Ebene. Kein Auslaufen.

e) Endergebnis. Das Loch wurde geschlossen.

Legende: Die Menge A ist gelb, die von den Schritten 2+3 gefundene Markierung blau dargestellt. Ebenen, die bei der Füllung weniger als m Voxel enthalten, erscheinen in grün, andere, die in den Außenraum auslaufen und daher mehr als m Voxel enthalten, in rot.

Abbildung 6.4: Mehrstufen–Füllalgorithmus - Taschenfüllung

Wahl von d erfolgt dann kein Auslaufen. Der markierte Bereich ist dann eine Art Skelett.

Schritt 3 - Skelett ausdehnen

Die gefundene Skelettmenge wird nun durch d weitere Schritte (per Breitensuche oder Dilation) ausgedehnt und nähert sich somit dem Rand des Volumens bis zur Berührung (s. Abbildung 6.3e[3]).

Schritt 4 - Taschen füllen

Übrig bleiben in diesem Stadium kleine Ausbuchtungen (Taschen), die noch gefüllt werden müssen. Dies erfolgt durch iterative Füllung innerhalb der xy-, xz- und yz-Ebenen.

Dazu wird an allen der bisherigen Markierung benachbarten Stellen eine erneute Breitensuche gestartet, die aber auf eine Ebene begrenzt ist.

Des Weiteren wird die Maximalzahl der neu zu markierenden Punkte auf m begrenzt. Wird diese überschritten, wird es als Auslaufen in den Außenraum interpretiert und die Markierung zurückgenommen (s. Abbildung 6.4b).

Enthält eine Wand also Löcher, die innerhalb der jeweiligen Ebene ein Auslaufen

[3]Man mag sich wundern, warum eine Ausdehnung des kleinen Bereiches in Abbildung 6.3d um nur 6 Voxel einen so großen Bereich ergeben kann. Das liegt daran, dass es sich ja um ein **dreidimensionales** Verfahren handelt. Unterhalb der dargestellten Schnittebene wurde das Darmsegment in seinem Inneren vom Skelettfüllschritt natürlich auch gefüllt. Von dort ausgehend wachsen natürlich auch Schichten auf, so dass sich der gezeigte große Bereich ergibt.

in den Außenraum ermöglichen, so erfolgt in dieser Ebene keine Erweiterung der bisherigen Markierung.

Damit würden sich in diesem Stadium unschöne Kerben ergeben (s. 6.4c), und es wäre nicht viel gewonnen.

Das Fehlen dieser Schichten wird aber, ebenso wie in der Geometrie vorhandene Abzweigungen, von den Folgeschritten kompensiert:

Bei der danach erfolgenden Füllung in xz-Ebene zeigen sich solche fehlenden xy-Ebenen als schmale rechteckige Bereiche. Diese erfüllen nun aber das Füllkriterium und somit erfolgt eine Ergänzung fehlender Schichten anhand der darüber- und darunterliegenden (s. Abbildung 6.4d).

Im Bereich von Wandlöchern ergibt sich somit eine sinnvolle Fortsetzung aus dem Verhalten der Füllung in nahegelegenen Ebenen ohne Löcher (s.Abbildungen 6.3f und 6.4e).

Die Gesamtsequenz der planaren Füllschritte in xy-, xz- und yz-Ebene wird in der Grundeinstellung nun n-mal iteriert. Dies dient nicht der Verbesserung der Füllung solcher Zwischenschichten (dort ändert sich nichts mehr), sondern dem Verfolgen von abzweigenden feinen Ästen, die nicht *zwischen* mehreren Schichten liegen, sondern nach oben bzw. unten von einer Schicht begrenzt werden.

6.2.3 Parameterwahl

Wahl von d

Die Lochgröße d der Menge A ist in der Regel nicht bekannt und muss durch Betrachtung dieser Menge oder aus Erfahrungswerten bezüglich des der Wahl von A zugrundeliegenden Auswahlalgorithmus geschätzt werden.

Wird d zu klein gewählt, so wird in Schritt 2 der Außenraum mit als zu füllendes Objekt angesehen.

Da der Außenraum oft sehr groß im Verhältnis zum zu füllenden Objekt ist, wurde in der Implementation eine Sperre vorgesehen, die die Füllung bei Überschreiten einer oberen Schranke von l_{max} gefüllten Voxeln mit einer Fehlermeldung abbricht.

Damit kann dann ohne größeren Zeitverlust ein neuer Versuch mit erhöhtem d gestartet werden.

Wahl der 2D-Taschengröße m

Aufgrund der Abbruchbedingung für Schritt 2 und 3 ist es möglich, dass vom Skelett Gänge abzweigen, die einen Radius von maximal d haben und noch nicht gefüllt worden sind. Daher hat sich zur Wahl von m ein Wert von ca. $(2d)^2$ bewährt. Ggf. ist dieser Wert zu erhöhen, falls **lange** Gänge dieser Art zu erwarten sind, oder zu verringern, wenn mit eher flachen Taschen gerechnet wird.

Wahl der Iterationszahl n

Die Anzahl der Iterationen ist nicht kritisch und bestimmt, wie stark gekrümmt abzweigende Äste sein dürfen, um noch entsprechend weit verfolgt zu werden. Drei erwies sich als für die Laufzeit günstige Wahl, die dennoch praktisch alle real vorkommenden Fälle im Bereich des Darmes abdeckte, ohne allzu tief in kleine stark gekrümmte Außenraumgebilde (die beim Darm ohnehin nicht vorkommen) zu laufen.

6.2.4 Optimierung

Übernahme der Randmengen

In Schritt 3 und 4 wird jeweils vom Rand der bisherigen Markierung ausgegangen, ebenso wie bei der Iteration von Schritt 4.

Da jeder Schritt als Breitensuche implementiert ist, fällt dieser Rand jeweils beim Vorgängerschritt an. Es handelt sich um diejenigen Voxel, die aufgrund der Abbruchbedingungen nicht mehr für die weitere Ausführung des Vorgängerschrittes in Betracht kommen.

Diese werden daher jeweils in eine gesonderte Schlange eingereiht, die vom nächsten Schritt direkt benutzt werden kann. Somit entfällt eine zeitaufwändige Neubestimmung des Randes.

6.2.5 Hintergrund zur Funktionsweise

Die Verwendung eines Skelettes und dessen anschließende Ausdehnung nach geeigneten Kriterien zur Füllung einer nicht vollständig geschlossenen Berandung liegt nahe. Nicht so offensichtlich ist die Designgrundlage für den Taschenfüllschritt.

Der Algorithmus wurde ursprünglich zur Segmentierung des Dünndarms entwickelt.

In diesem Fall besitzt man Wissen über die typische Gestalt der möglichen Ausbuchtungen:

Beim Darm handelt es sich dabei jeweils um schmale, typischerweise ringförmige Gebilde, divertikelähnliche Taschen oder flache Scheiben. Für den Außenraum hingegen sind meist eher große Zusammenhangskomponenten unterschiedlichster Form typisch.

Auf den ersten Blick scheint eine Unterscheidung anhand der Größe daher sinnvoll. Allerdings stellt man fest, dass insbesondere die mehr oder weniger ringförmigen Darmfalten erhebliches Volumen beinhalten können — in einer Größenordnung, die auch für Objekte im Außenraum auftreten kann.

Allerdings sind diese Gebilde in der Regel erheblich gekrümmt. Das führt dazu, dass die Zusammenhangskomponenten in zweidimensionalen Schnitten recht klein werden, im Gegensatz zu den meist eher flächigen Außenbereichen.

6.2.6 Anwendungen

Auf die Anwendung zur Füllung des Dick- und Dünndarmes wurde bereits eingegangen.

Untersucht man andere anatomische Strukturen, so stellt man oft fest, dass diese ebenfalls den Designkriterien des Algorithmus sehr gut entsprechen.

In vielen Fällen kommen bei den zu füllenden Taschen keine großvolumigen Elemente mit starker Krümmung (s.o.) vor. Dennnoch macht es Sinn, den zweidimensionalen Taschenfüllschritt zu verwenden, da er nicht nur der Erkennung dieser Elemente dient, sondern auch, wie in Abbildung 6.4 illustriert, ein sinnvolles Schließen der Löcher bewirkt.

Der Algorithmus erwies sich daher auch für andere Anwendungen als die Segmentierung von Darmlumina (z.B. beim Gehirn, vgl. Abbildung 6.18b,c und der Leber) als sehr nützlich. Wann immer es möglich ist, eine — wenn auch löcherige — Berandung[4] zu finden, kann damit das Objekt mit guter Genauigkeit segmentiert werden.

In der Praxis verwenden wir den Algorithmus für praktisch jedes Füllproblem, bei dem die Geschlossenheit der Berandung nicht gesichert ist. Um eine möglichst große

[4]Der Begriff Berandung wird hier in recht allgemeiner Form gebraucht. Im Gegensatz zum Darmlumen, das ja durch den Darm selbst berandet ist, besitzen Leber und Gehirn keine im CT sichtbare explizite Berandung. Sie unterscheiden sich aber im Grauwert an fast allen Stellen von Ihrer Umgebung. Damit wird es möglich einen solchen Rand an fast allen Stellen zu definieren, indem man alle Voxel mit Grauwerten, die im Zielobjekt nicht vorkommen, als Berandung ansieht.

Flexibilität zu erreichen, können alle wesentlichen Algorithmusparameter (maximaler Lochradius, maximale Taschengröße, Anzahl der Iterationen der Taschenfüllung, Anzahl der Iterationen bei der Erweiterung in Schritt 3) variiert werden.

Da es bei falsch eingestellten Parametern oder schlicht falschem Anklicken des Startpunktes zu einem Auslaufen des Algorithmus bzw. einer Fehlmarkierung kommen kann, wurde auch ein Undo/Fixate-Mechanismus implementiert, mit dessen Hilfe gefundene Markierungen wieder entfernt oder endgültig in die Auswahl übernommen werden können.

6.3 Dilation

Für einige Operationen wird eine Erweiterung der aktuellen Markierung eines Bereiches um einen gegebenen Rand benötigt.

Technisch gesehen handelt es sich um die Faltung eine Binärbildes, in dem die Markierung als „1“ abgelegt ist, mit einem gegebenen Strukturelement.

Aus Geschwindigkeitsgründen wählt man zumeist ein Rechteck als Strukturelement, da die Faltung dann trivial separiert.

6.3.1 Optimierte Implementation

Man zerlegt die Dilation mit einem quaderförmigen Strukturelement der Seitenlängen $(2a_x+1,\ 2a_y+1,\ 2a_z+1)$ mit $a_x, a_y, a_z \in I\!N_0$ zunächst in drei Dilationen längs der Achsen. Hintereinander ausgeführt ergibt sich die Gesamtoperation.

Durch die Separation erreicht man für den üblichen Fall des würfelförmigen Strukturelements der Kantenlänge $l = 2a_x + 1$ auf einem Volumen mit Kantenlängen s_x, s_y, s_z eine Geschwindigkeitssteigerung um den Faktor:

$$\frac{L_{direkt}}{L_{separiert}} = \frac{s_x \cdot s_y \cdot s_z \cdot l^3}{3 \cdot s_x \cdot s_y \cdot s_z \cdot l} = \frac{l^2}{3}$$

Eine weitere Verbesserung erreicht man, wenn man die verbleibenden eindimensionalen Faltungen rekursiv implementiert. Dazu zählt man schlicht, wie viele Pixel entfernt das letzte markierte Pixel war, und setzt neue Markierungen, wenn dieser Wert kleiner oder gleich a_x ist.

Allerdings muss man nun für jede Richtung das Volumen einmal vorwärts und einmal rückwärts durchlaufen (da das Verfahren ja nur vergangene Werte berücksichtigt).

Dies könnte durch Verwendung eines Zwischenspeichers vermieden werden, der entsprechend verschoben (um a_x) nach jeweils einer Zeile wieder in den Quellbereich kopiert wird. Letzteres bringt aber effektiv keinen Gewinn, da das Rückkopieren genauso aufwändig ist wie ein zweiter Durchgang.

Der Laufzeitgewinn liegt nun bei

$$\frac{L_{direkt}}{L_{separiert,rekursiv}} = \frac{s_x \cdot s_y \cdot s_z \cdot l^3}{6 \cdot s_x \cdot s_y \cdot s_z} = \frac{l^3}{6} .$$

Der kleinste sinnvolle Parameter für l ist 3 (Dilation um 1), so dass diese Optimierung auf jeden Fall lohnend ist.

Man beachte, dass auf realen Maschinen bei Operationen, die im Speicher in großen Abständen hüpfen, der Cache nicht mehr effizient funktioniert. Im Falle des vorliegenden Algorithmus ist dieser Effekt sehr deutlich zu sehen: Die Filterung in x-Richtung geht deutlich schneller als die in y-Richtung, die wiederum schneller geht als in z-Richtung.

So dauert z.B. eine Dilate-Operation in einem 512^3 Voxel großen Volumen, das einen Darm als zu dilatierendes Objekt enthielt:

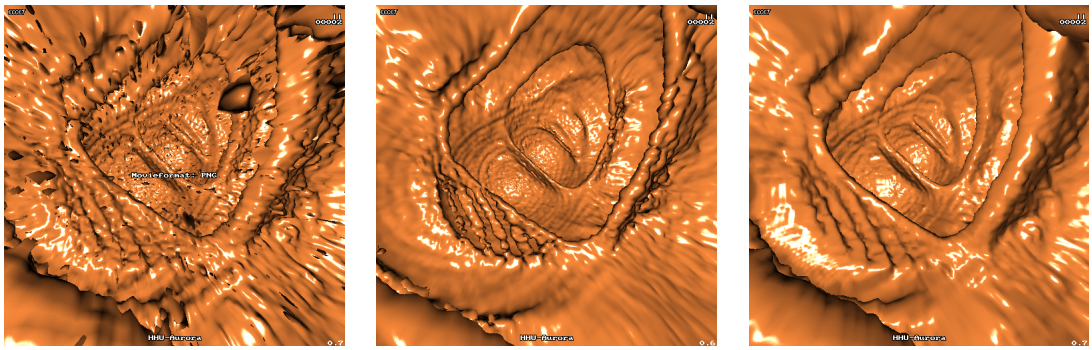
l	x	y	z	gesamt
3	3.4s	13.9s	27.9s	45.2s
11	3.4s	13.9s	28.0s	45.3s
21	3.4s	13.9s	28.2s	45.5s
41	3.4s	14.0s	28.4s	45.8s

Man sieht sehr gut, dass dieser Algorithmus fast nicht mehr von der Größe der Maske abhängt. Die leichte Abhängigkeit kommt von der größeren Zahl von Schreiboperationen.

Man beachte, dass die drei Richtungen bei der Messung verkettet abliefen, und zwar in der Reihenfolge $z \rightarrow y \rightarrow x$. D.h. die y- und x- Schritte hatten sozusagen noch **mehr** zu tun, da sie auch auf die bereits dilatierten Bereiche der Vorschritte reagieren mussten.

Dennoch sind die Richtungen, in denen in kleineren Schritten durch den Speicher gesprungen wird, deutlich bevorzugt. Das liegt daran, dass die Daten bei den Durchgängen in x-Richtung bezüglich ihrer Struktur im Speicher linear abgearbeitet werden, so dass Burst-Zyklen, Cache-Prefetching etc. optimal genutzt werden können.

Im gegebenen Fall ergibt sich allerdings immer noch eine signifikante Beschleuni-



a) ungefiltertes Volumen bei Schwelle 150

b) ungefiltertes Volumen bei Schwelle 500

c) gefiltertes Volumen bei Schwelle 150

Abbildung 6.5: Notwendigkeit der NLG-Filterung

gung. Die Messungen zeigen aber auch, dass eine Laufzeitbetrachtung, die rein operationenbasiert ist, nicht immer die Verhältnisse auf realen Maschinen widerspiegelt.

6.4 Vollautomatische Segmentierung des luftgefüllten Darmlumens

Für die virtuelle Koloskopie wurde ein automatisches Verfahren entwickelt, das den Dickdarm bei geringem Zeitaufwand vollautomatisch aus den Rohdaten extrahiert.

Da keine Interaktion des Benutzers mehr erforderlich ist, stellt ein solches Verfahren eine erhebliche Reduktion des Zeitaufwandes des diagnostizierenden Arztes dar und ist somit von großem Interesse.

Der Algorithmus hierfür läuft komplett mit Hilfe der Scriptfähigkeiten von VOXREN ab und wurde unter Nutzung von VOXREN als Experimentierplattform entwickelt.

6.4.1 Algorithmus

Problem

Es ist ein Algorithmus gesucht, der aus einer stark verrauschten CT-Aufnahme des Abdomens den Darm herauspräpariert.

Eine einfache Schwellwertentscheidung ist hierzu nicht ausreichend, da das starke Rauschen sich bei niedriger Schwelle als „Schnee“ im Bild äußert (s. Abbildung 6.5a), während bei zu hoher Schwelle aufgrund von Partialvolumeneffekten scheinbar Löcher in der Darmwand auftauchen (s. Abbildung 6.5b). Man kann zwar versuchen,

a) Volumen vor dem Start des Algorithmus

b) Markierung von Gewebe

c) Einfache Entfernung der Außenluft. Der verbleibende Luftspalt (dunkelrot) wird später gefüllt.

Abbildung 6.6: Vollautomatische Vorverarbeitung - Entfernen der Außenluft

über die Schwelle einen optimalen Arbeitspunkt zu finden, wird aber bei Niedrigstdosisaufnahmen nicht **beide** Störeffekte eliminieren können.

Unabhängig von der Schwelle sorgt das Rauschen weiterhin in der Nähe der Darmwand dafür, dass die Oberfläche sehr rauh erscheint (s. Abbildung 5.2), was die Suche nach Polypen erschwert.

Insbesondere stört das Rauschen auch den Polypendetektor, der dann vermehrt falsch positive Resultate liefert.

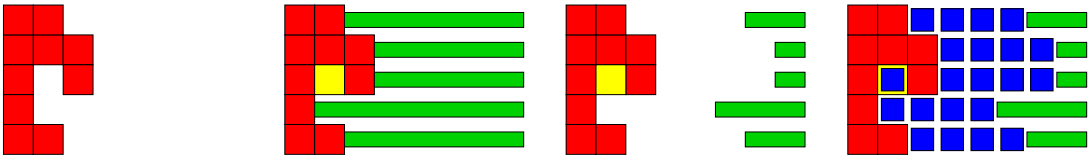
Die nichtlineare Gaußfilterkette (s. 5.1) eignet sich prinzipiell für die Aufgabe, diese Probleme zu beseitigen (s. Abbildung 6.5c). Allerdings benötigt sie für ein volles Volumen auf derzeitigen Rechnern mehrere Stunden Rechenzeit.

Ein möglicher Ausweg besteht darin, den Darm zunächst zu isolieren und nur in dessen unmittelbarer Umgebung das Filter anzuwenden.

Das Finden des Darmes hat den weiteren Vorteil, dass anschließend der Polypendetektor weniger falsch positive Ergebnisse aus Nicht-Dickdarm-Regionen liefert.

Voraussetzungen

Der Algorithmus erwartet als Eingabe einen Stapel Grauwertdaten vom Abdomen, der so aufgenommen wurde, dass der gesamte Dickdarm enthalten ist und er die obere Begrenzungsfläche nicht berührt.



a) Randkontur mit verdecktem Bereich

b) Markierung des Außenraumes ohne Spalt. Das gelb markierte Voxel bleibt ungefüllt.

c) Markierung mit 4 Pixel breitem Spalt. Der Zugang zum gelben Voxel bleibt frei.

d) Auffüllen des Spaltes mit Breitensuche. Verdeckter Bereich wird mitgefüllt

Abbildung 6.7: Vollautomatische Vorverarbeitung - Entfernen des Außenraumes

Schritt 1 - Markierung des Gewebes

Zuerst wird mit Hilfe einer Schwellwertentscheidung (Grauwert größer als 300) das Gewebe markiert (s. Abbildung 6.6b).

Aufgrund des ggf. starken Rauschens in den Originaldaten ergibt dies in der Regel keine glatten Oberflächen. Allerdings ist das Rauschen nicht so groß, dass bei der relativ hohen Schwelle luftgefüllte Volumina noch signifikante Mengen Rauschpixel enthalten.

Schritt 2 - Entfernung des Außenraumes

Ausgehend von den Seitenflächen des Voxelvolumens (vordere, hintere, linke und rechte Begrenzungsfläche) wird nun der Luftbereich bis auf einen schmalen Spalt gefüllt.

Dazu wird jeweils ausgehend von einer Begrenzungsfläche längs eines orthogonal zur jeweiligen Begrenzungsfläche gerichteten Strahles so lange das Volumen nach als Gewebe markierten Voxeln durchsucht, bis ein solches gefunden wird oder die gegenüberliegende Begrenzung erreicht wird.

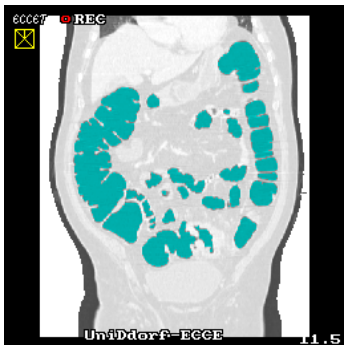
In ersteren Fall wird nun um die einstellbare Spaltbreite zurückgewichen und die Strecke zum Startpunkt wird grün markiert (s. Abbildung 6.6c).

In letzterem Fall wird die gesamte Zeile grün markiert.

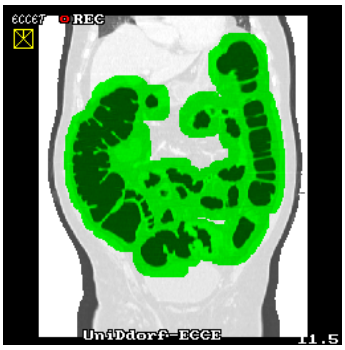
Der Spalt (dunkelroter Rand in der Abbildung 6.6c) dient dazu, kleine verdeckte Bereiche für den späteren Füllschritt zugänglich zu halten (s. Abbildung 6.7a-c).

Der so gefundene Bereich, der im Wesentlichen (bis auf den erzeugten Spalt) den Luftraum außerhalb des Körpers enthält, wird nun von der weiteren Bearbeitung ausgenommen.

d) Markierung der verbliebenen Luftvolumina



e) Entfernung der Zusammenhangskomponenten, die die oberste Ebene berühren



f) Dilation, um das Filterfenster nicht zu beschneiden

Abbildung 6.8: Vollautomatische Vorverarbeitung - Entfernen von Lunge und Hautoberfläche

Schritt 3 - Entfernung der Zusammenhangskomponenten mit Kontakt zur oberen Begrenzungsfläche

Durch eine erneute Schwellwertentscheidung werden die verbliebenen Luftbereiche (Darm, Lunge, verbliebener Luftspalt) erneut erfasst.

Der Spalt und die wesentlichen Anteile der Lunge werden nun entfernt, indem man alle verbliebenen Zusammenhangskomponenten, die mit der **oberen** Begrenzungsebene des Volumens Kontakt haben, entfernt.

Dazu wird an jedem Punkt der oberen Begrenzungsebene, der als verbliebenes Luftvolumen markiert ist, eine Breitensuche gestartet, die die jeweilige Zusammenhangskomponente löscht.

Schritt 4 - Entfernen kleiner Zusammenhangskomponenten

Nun werden sehr kleine Zusammenhangskomponenten (< 1000 Voxel) entfernt, um kleine Lufteinschlüsse, wie sie typisch im Dünndarm und im Bereich der Lunge vorkommen, zu entfernen.

Zum Dickdarm gehörende Komponenten sind in aller Regel erheblich größer.

Schritt 5 - Dilation

Da als nächster Schritt die Filterung erfolgen soll, werden die bisher gefundenen Bereiche um 15 Pixel dilatiert. Dies geschieht, da das Filter Werte aus der Umgebung eines Voxels benötigt, um die Glättung durchzuführen.

Alle nun nicht markierten Bereiche werden von der Filterung ausgeschlossen (s. Abbildung 6.8f).

Schritt 6 - Nichtlineare Gaußfilterkette

Nun kommt eine separierte Version der nichtlinearen Gaußfilterkette (s. 5.1) zum Einsatz, die nur auf dem eben gefundenen Teilvolumen arbeitet. Dadurch wird eine erhebliche Geschwindigkeitssteigerung erreicht: Typisch müssen nur noch 15% des Volumens bearbeitet werden (24 statt 160 Millionen Voxel).

Schritt 8 - Erneute Schwellwertentscheidung

Durch die kantenerhaltende Glättung kann nun mit einer einfachen Schwellwertentscheidung das Darmlumen segmentiert werden. Es ergeben sich jetzt keine Schneeeffekte im Darmlumen mehr, und auch die durch das Rauschen verursachten gezackten Ränder treten nicht mehr auf.

Bereiche, die oben schon von der Filterung ausgeschlossen wurden, werden natürlich von der erneuten Schwellwertentscheidung ebenfalls nicht mehr markiert.

Schritt 9 - Entfernen kleiner Zusammenhangskomponenten

Nun werden erneut sehr kleine Zusammenhangskomponenten (< 1000 Voxel) entfernt, um kleine Lufteinschlüsse, wie sie typisch im Dünndarm und im Bereich der Lunge vorkommen, wieder zu entfernen.

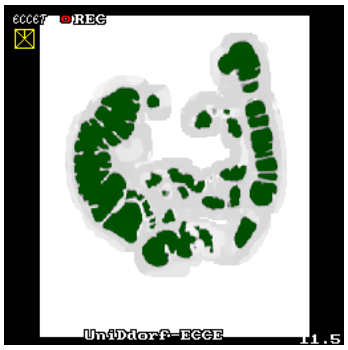
Diese können durch die Dilation dem Volumen vorübergehend wieder hinzugefügt worden sein.

6.4.2 Bemerkungen

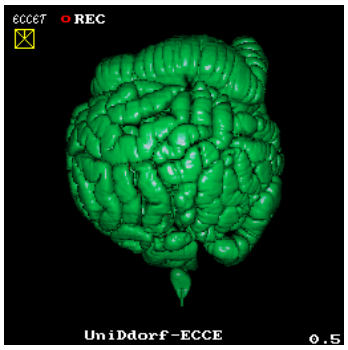
Geschwindigkeitsoptimierung

Natürlich könnte in Schritt 3 auch der gesamte Außenraum entfernt werden. Schritt 2 könnte also eingespart werden.

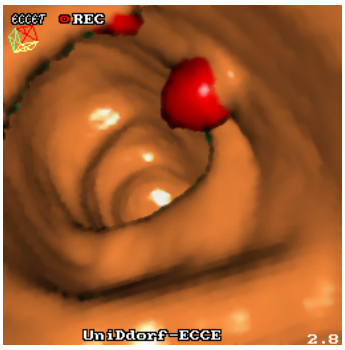
Allerdings läuft Schritt 2 aufgrund des einfacheren Algorithmus und des besseren Zugriffsmusters auf den Hauptspeicher erheblich schneller ab als eine Entfernung des gesamten Außenraumes per Breitensuche.



g) Segmentierung nach der Glättung



h) 3D-Rekonstruktion



i) Vom Polypenfinder markierter Polyp

Abbildung 6.9: Vollautomatische Vorverarbeitung - Rekonstruktion und Polypensuche

Suche von Nicht-Darm-Komponenten von oben

In Schritt 3 wird ausgehend von der obersten Schicht versucht, Nicht-Darm-Komponenten zu finden.

Man könnte versucht sein, den umgekehrten Weg zu gehen und in der untersten Schicht den Darmeingang zu finden. Das birgt aber das Risiko, dass durch unvollständige Entleerung oder durch Erkrankungen das Luftvolumen unterbrochen sein kann. Der Vorteil wäre, dass man keine teilweise mit Luft gefüllten Dünndarmanteile mehr mitsegmentieren würde.

6.4.3 Nachbearbeitung

Das verwendete Script bereitet das Ergebnis des Algorithmus direkt für den Anwender auf. Dazu werden noch 4 weitere Schritte ausgeführt, die nicht mehr unmittelbar mit der Segmentierung zusammenhängen:

- Die Oberfläche des gefundenen Objekts (Dickdarm plus größere Dünndarmstücke) und deren Normalen werden bestimmt.
- Das Volumen wird auf den kleinstmöglichen Teilquader eingeschränkt, der noch das komplette segmentierte Volumen erfaßt. Damit erreicht man eine verbesserte Darstellungsgeschwindigkeit.
- Die notwendige Beschleunigungsinformation für das Rendering (GDA) wird erzeugt.
- Der Polypensuchalgorithmus (s. 7.1) wird angewendet.

6.4.4 Bewertung und Verbesserungsmöglichkeiten

Dieser Algorithmus hat gegenüber manueller Selektion des Dickdarmes zwar den Nachteil, dass auch größere luftgefüllte Segmente des Dünndarmes erhalten bleiben, aber den Vorteil, dass er absolut ohne manuellen Eingriff innerhalb von 6-8 Minuten ablaufen kann.

Zur Zeit wird eine Erweiterung untersucht, bei der die eben in 6.4.2 angesprochene Technik der Suche des Dickdarms von unten dazu verwendet werden soll, in den Fällen, in denen der Dickdarm durchgängig ist, zusätzlich den Dick- vom Dünndarm zu trennen, so dass der Benutzer letzteren per Knopfdruck entfernen kann.

6.4.5 Entwicklung des Algorithmus

Die Entwicklung dieses Verfahrens illustriert sehr gut den Vorteil eines sowohl interaktiv bedienbaren als auch scriptgesteuert einsetzbaren Systems wie $\epsilon\mathcal{CC}\epsilon\mathcal{T}$. Der Algorithmus wurde aus seiner Grundidee interaktiv durch Nutzung vorhandener Operationen entwickelt, wobei die Vorgehensweise jeweils Schritt für Schritt getestet und durch die Integration mit der Visualisierungssoftware auf ihre Funktionalität überprüft werden konnte.

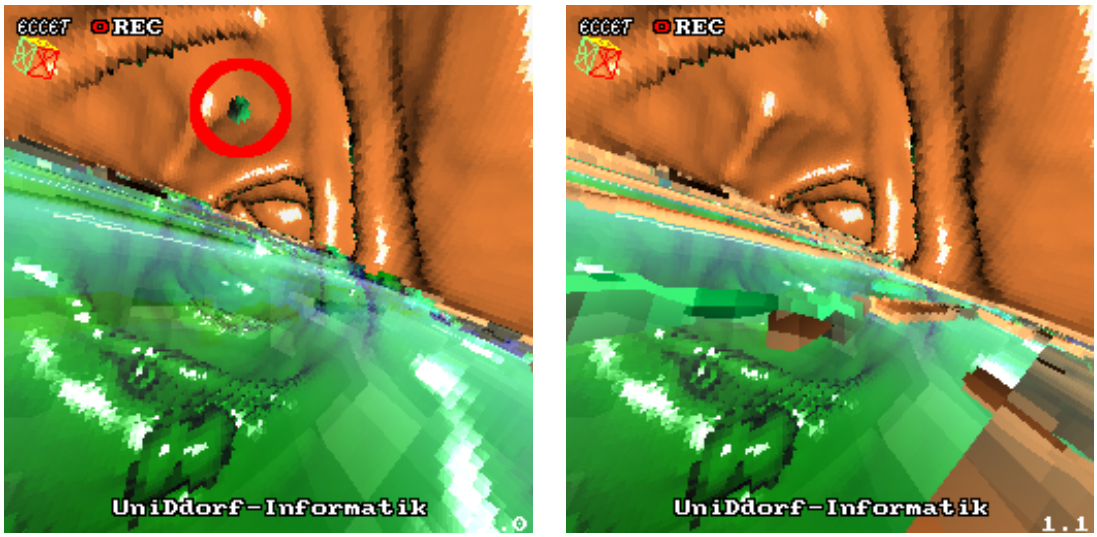
Dabei wurden die Einzelschritte dokumentiert und anschließend hieraus ein Script entwickelt, das den Vorgang automatisiert. Da mit dem Script eine programmiersprachenähnliche Beschreibung des Algorithmus vorlag, konnten dort noch einige Unzulänglichkeiten korrigiert werden, die bei der interaktiven Arbeit übersehen worden waren.

6.5 Normalenbasierte Segmentierung

Ein Problem bei der virtuellen Koloskopie sind Flüssigkeitsreste infolge unvollständiger Darmentleerung. Diese haben zumeist eine sehr ähnliche Dichte wie das Darmgewebe, so dass sie schwer von diesem zu trennen sind.

Ein möglicher Ausweg ist das „Fecal Tagging“ . Dazu wird dem Patienten während der Darmentleerung Kontrastmittel zugeführt. Eventuell unvollständig geleerte Segmente des Darmes sollten dann Kontrastmittel enthalten, so dass eine Unterscheidung vom Gewebe möglich wird.

Der luft- und der kontrastmittelgefüllte Bereich kann jeweils einzeln oder auch gemeinsam mit den bisher aufgeführten und noch folgenden Techniken segmentiert



a) „Loch“ in der Darmwand aufgrund einer benachbarten kontrastmittelgefüllten Dünndarmschlinge

b) Loch per REMARKNORM geschlossen - dafür Linienartefakte durch Bläschenbildung

Abbildung 6.10: virtuelle Koloskopie mit Kontrastmittel

werden.

Zur leichteren Diagnose ist es aber vorteilhaft, wenn beide zu einem zusammenhängenden Objekt vereinigt würden. Insbesondere stören die Oberflächen des Kontrastmittels.

An diesen Oberflächen entsteht durch Partialvolumeneffekte und Bläschenbildung, der Darm wird ja nach wie vor mit Luft dilatiert, eine Schicht von Voxeln, deren Grauwerte zwischen denen der Luft und des Kontrastmittels liegen, und damit leider auch im gleichen Bereich wie die Darmwände. Bei der Segmentierung bildet sich daher eine „Doppelwand“ um diese Schicht.

Diese Bereiche können heuristisch wie folgt gefunden werden:

- Finde alle Randpunkte des **Luftvolumens**, in deren Nähe (2-3 Pixel) sich Kontrastmittel befindet.

- Finde alle Randpunkte des **Kontrastmittelvolumens**, in deren Nähe sich Luft befindet.

Allerdings findet dieser Algorithmus oft auch Bereiche, in denen Darmschlingen aneinanderliegen bzw. sich Luft oder Kontrastmittel in Dünndarmschlingen in der Nähe befindet.

Da die oben gefundenen Bereiche anschließend (semi-)transparent gemacht werden sollen, um den Darm als Ganzes darzustellen, sind solche Fehler sehr störend, da sie als „Löcher“ in der Darmwand dargestellt werden (s. Abbildung 6.10).

Ein einfaches heuristisches Unterscheidungskriterium ist, dass die Kontrastmitteloberflächen aufgrund der Schwerkraft horizontale Flächen bilden.

Da die Normalen der Oberflächen bekannt sind, kann dieses Kriterium leicht zur weiteren Verbesserung der Unterteilung eingesetzt werden. Dazu werden nur die Bereiche, deren Normalen in etwa senkrecht liegen, transparent gemacht.

In Abbildung 6.10b wurden alle Voxel, die bisher als Kontrastmitteloberfläche markiert waren, aber nicht innerhalb von 20° um die Richtung der Schwerkraft lagen, wieder als Darmwand eingestuft. Dies schließt offensichtlich das Loch, trifft aber auch an der Oberfläche auf einige Voxel zu, die z.B. wegen Bläschenbildung stärker geneigt sind. Das führt zu schmalen „Graten“ auf der Oberfläche, die aber nicht weiter stören. Löcher in der Wand stören hingegen stark, da sie die Funktionsweise des Autopiloten (s. 8.1) beeinträchtigen.

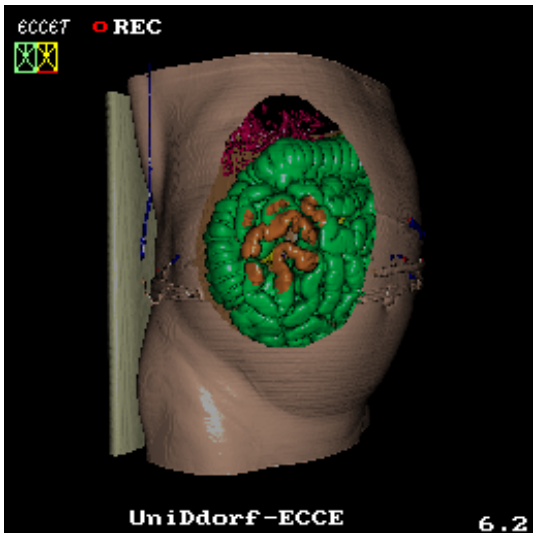
6.6 Manuell unterstützte Segmentierung

Nicht alle Segmentierungsprobleme lassen sich vollautomatisch lösen. Oft ist es effizienter, einen menschlichen Experten die Entscheidungen treffen zu lassen, wobei er allerdings von geeigneten Werkzeugen unterstützt werden muss. Insbesondere ist zum effizienten Arbeiten eine interaktive Kontrolle der Verarbeitungsschritte notwendig.

VOXREN ermöglicht dies durch die Vereinigung eines 2D/3D-Visualisierungssystems mit einer Vielzahl von Bearbeitungswerkzeugen, die in der Mehrzahl sehr schnell ausgeführt werden.

6.6.1 Umfärben von Zusammenhangskomponenten

Ein einfaches Verfahren zur interaktiven Segmentierung z.B. des Darmes besteht darin, zunächst eine einfache Schwellwertentscheidung zu treffen, die Luft und Gewebe trennt. Dabei werden aber zusätzlich zum Darmlumen auch noch die Luft im Außenraum und in der Lunge sowie andere Lufteinschlüsse in Hohlorganen (Magen, Dünndarm) markiert. Nun erzeugt man an allen Übergängen zwischen markierten und nicht markierten Bereichen Oberflächen.



Das nebenstehende Volumen wurde zunächst trivial mit Hilfe einer Schwellwertentscheidung segmentiert. Man erhält alle Luft-/Gewebeübergänge.
Mit wenigen Mausklicks erreicht man anschließend eine Aufteilung in die Bestandteile Patientenliege, Haut, Darm und Lunge, da sie in unterschiedliche Zusammenhangskomponenten zerfallen.

Abbildung 6.11: Manuell unterstützte Segmentierung (Zusammenhangskomponenten)

Ein menschlicher Experte kann die gefundenen Oberflächen nun klassifizieren, indem er interaktiv Zusammenhangskomponenten umfärbt. Hiermit können die Haut des Patienten sowie die Oberflächen der (zumeist hohl ausgeführten) Patientenliege gut erfasst werden. Problematischer ist der Bereich Lunge, da dort das Luftvolumen meist in viele kleine Einzelbereiche zerfällt. Hier erweist es sich als einfacher, den Darm, der meist aus nur wenigen Zusammenhangskomponenten besteht, neu zu klassifizieren.

Allerdings ist es damit in der Regel nicht möglich, Dünn- und Dickdarm zu trennen, da Kontaktpunkte zwischen Dick- und Dünndarmwand bestehen.

Eine Segmentierung mit Hilfe der in 6.2 beschriebenen Lösung erlaubt dagegen eine alleinige Selektion des Dickdarmes.

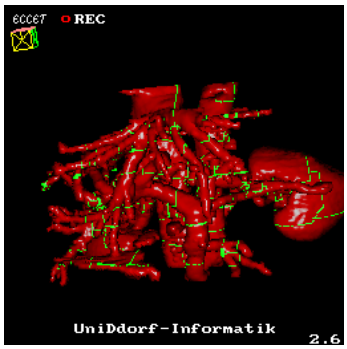
6.6.2 Finden von Verzweigungen

Bei der Analyse von Adersystemen kann es hilfreich sein, eine hierarchische Gliederung einzuführen, um die Übersicht zu erhöhen. Der wichtigste Schritt um eine solche Trennung durchführen zu können, besteht im Finden von Verzweigungspunkten.

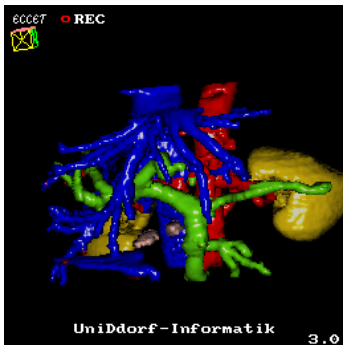
Problemstellung

Finde die Verzweigungspunkte in einem Röhrensystem und trenne das System so in Komponenten auf, dass sich nicht weiter verzweigende Teilstücke entstehen.

a) Segmentiertes Gefäßsystem des oberen Bauchraumes. Die gesamte Struktur ist zusammenhängend, insbesondere wurden die Nieren aufgrund ihres Grauwertes mitsegmentiert. Unübersichtlich.



b) Mit Hilfe des Verzweigungsfinders wird das Volumen schnell in relativ wenige Teile zerlegt.



c) ca. 10 Minuten später ist das Adernsystem vom Experten durch einfaches Anklicken der Teile in seine Bestandteile zerlegt worden. Das Ergebnis ist eine übersichtliche Darstellung.

Abbildung 6.12: Finden von Verzweigungen und ihre Nutzung zur Segmentierung

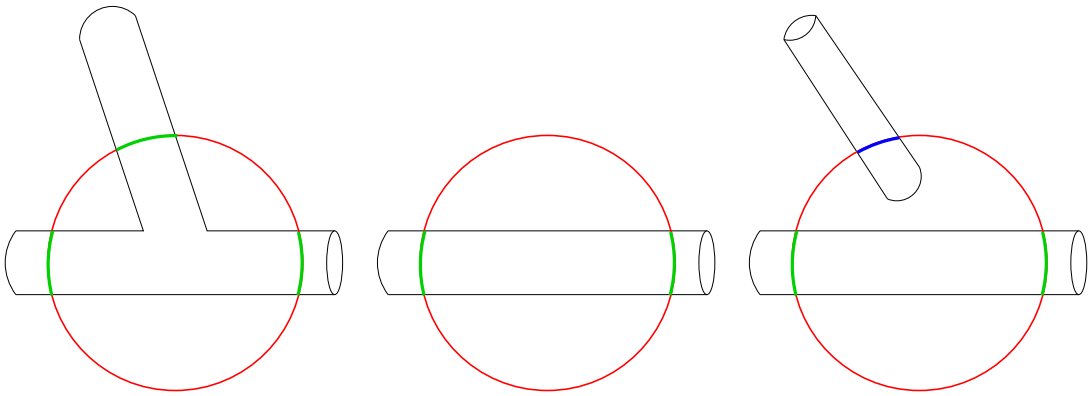
Voraussetzung

Das Röhrensystem liegt dem Algorithmus als hohle Voxelmenge vor. Die Segmentierung und Oberflächengenerierung hat bereits zuvor stattgefunden.

Verfahren 1

Der erste Algorithmus basiert auf der Anzahl der Zusammenhangskomponenten auf dem Schnitt mit einer Kugeloberfläche. An Verzweigungsstellen mit hinreichend weit ausgedehnten Teilästen ergeben sich mindestens drei Durchstoßungsfiguren des Objektes durch eine um den Verzweigungspunkt zentrierte, geeignet große Sphäre (s. Abbildung 6.13a). An Punkten ohne Verzweigung schneidet lediglich der „Zu- und Abfluss“ die umgebende Kugel (s. Abbildung 6.13b).

Das Problem bei diesem Verfahren ist das Finden einer geeigneten Kugel. Abbildung 6.13c zeigt eine mögliche daraus resultierende Fehlklassifikation: Durch Verwendung einer zu großen Kugel entsteht eine zusätzliche Schnittkomponente (blau), die nicht vom eigentlich verfolgten Objekt herrührt. Verkompliziert wird das Finden der jeweils passenden Kugel dadurch, dass es sich bei den Ausgangsdaten nicht um linienhafte Objekte (Skelette) handelt, sondern, z.B. im Fall von Adersystemen, um Röhren mit stark variierendem Durchmesser. Hierdurch gibt es weitere Fehlplatzierungs- und -dimensionierungsmöglichkeiten für die Kugel (kleine Kugel kann keine dicken Abgänge sehen, große Kugel kann kleine Stummel übersehen etc.).



a) Verzweigte Stelle. Die Kugeloberfläche schneidet das Gefäß an drei Stellen.

b) Nicht verzweigte Stelle. Nur zwei Schnittstellen.

c) Falschklassifikation durch ein nahes weiteres Objekt.

Abbildung 6.13: Finden von Verzweigungen - Verfahren 1

Es wurden verschiedene Versuche unternommen, dynamisch an jeder betrachteten Position ein geeignetes Zentrum und einen geeigneten Durchmesser für die Kugel zu bestimmen. Unter anderem wurde versucht mit Hilfe der bekannten Normalen auf den Gefäßwänden den Mittelpunkt des Gefäßes aufzusuchen und die Distanz zu diesem zur Schätzung einer geeigneten Kugelgröße zu verwenden. Eine deutliche Verbesserung der Problematik konnte jedoch nicht erreicht werden.

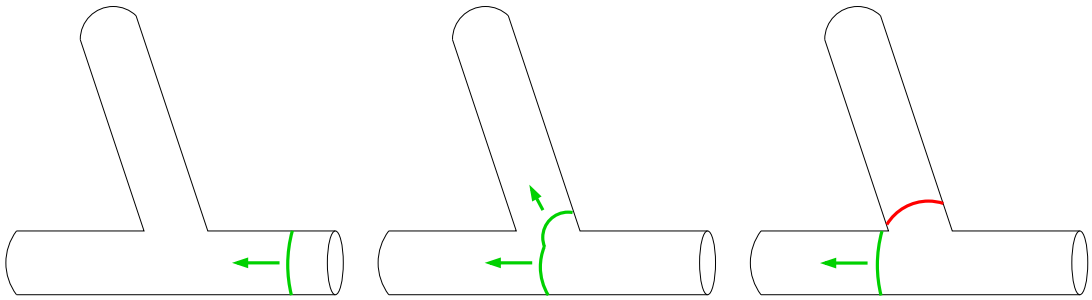
Verfahren 2

Verzweigung der Frontfläche einer Breitensuche

Aus dem in Abbildung 6.13c zu sehenden Problem lässt sich auch ein Lösungsansatz ableiten. Offensichtlich würde dieser Fall nicht auftreten, wenn die Zusammenhangskomponenten eine Rolle spielen würden.

So entstand der zweite Ansatz, bei dem eine Breitensuche an einem Ende der Ader gestartet wird. Startend von einem Einzelpunkt wird nun die zur Breitensuche verwendete Schlange schrittweise je einmal abgearbeitet. Den dabei entstehenden neuen Schlangeninhalt nennen wir die aktuelle Front der Breitensuche.

Nach jedem Schritt wird geprüft, ob diese neue Front der Breitensuche **nur eine** Zusammenhangskomponente enthält. Sie bildet solange jeweils einen Ring um die Ader, der sich schrittweise an ihr entlangschiebt. Zweigt nun ein Seitenast ab, so teilt sich die Front in zwei Zusammenhangskomponenten. In diesem Fall wird an einem der beiden Äste eine Trennungsmarkierung angebracht (die eine Zusammenhangskomponente wird umgefärbt), die andere Komponente wird weiterverfolgt.



a) Die Frontfläche läuft ein Gefäß entlang.

b) An der Verzweigung beginnt sich eine zweite Front abzuschälen.

c) Die Fronten trennen sich - der Algorithmus schneidet.

Abbildung 6.14: Finden von Verzweigungen - Verfahren 2

Später werden die abgetrennten Bereiche in weiteren Durchgängen mit demselben Algorithmus weiterbearbeitet.

Dabei sind folgende Punkte zu beachten:

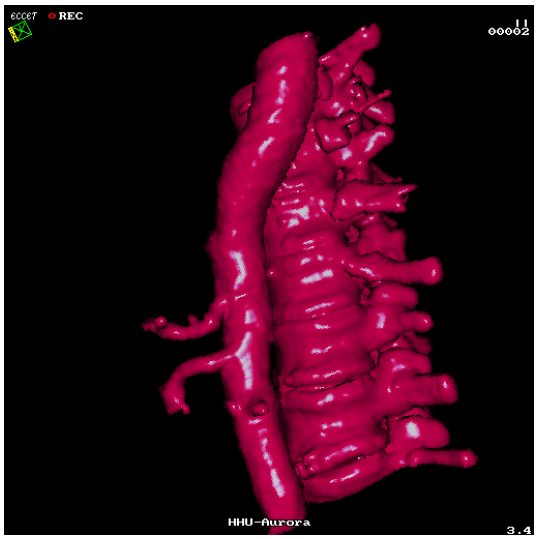
1. Die Form der Frontfläche entspricht nicht ganz den Erwartungen aus der Darstellung in Abbildung 6.14. Wir nehmen intuitiv eine zur Ader senkrechte Ausrichtung mit kürzestmöglichem Umfang an, wie sie ein um die Ader gespanntes Gummiband einnehmen würde. In der Implementation bestimmt die der Breitensuche zugrundeliegende Metrik die Form und Lage der Front. Im konkreten Fall neigt die Frontlinie dazu, sich längs der x-, y- und z-Achse des Volumens auszurichten und bildet entsprechend gern 90°-Knicke aus.

2. Bereits kleine Ausbuchtungen auf der Oberfläche werden oft erkannt und angezeigt - das Verfahren erscheint überempfindlich.

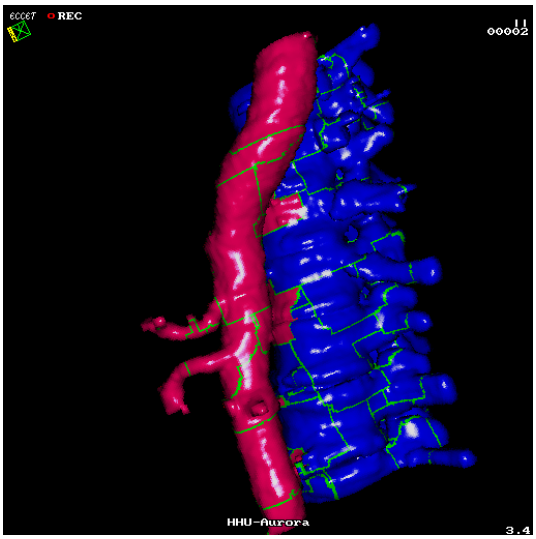
Punkt 2 kann verbessert werden, indem man fordert, dass die Mehrteiligkeit der aktuellen Frontfläche über mehrere Schritte erhalten bleibt. Bei einer Wahl von z.B. 3 Schritten ergibt sich in der Regel bereits eine sehr gute Unterdrückung von kleinen Beulen, ohne die Trennleistung signifikant zu beeinträchtigen.

Ergebnis

Verfahren 2 liefert eine gute Näherungslösung für das oben gestellte Problem. Kleine Abzweigungen können je nach Parametereinstellung abgetrennt oder beibehalten werden. Die Verzweigungen selbst werden dabei nicht gesondert getrennt, sondern willkürlich einem der Zuflüsse zugeordnet.



a) Gesamtobjekt mit mehreren schlecht erkennbaren Verbindungsstellen



b) Verzweigungsfindung, dann Auffüllen der erzeugten Flächen. Die Problemstellen sind nun klar erkennbar.

Abbildung 6.15: Trennung von Aorta und Wirbelsäule mit dem Verzweigungsfinder

Die eigentliche Hierarchiebildung kann anschließend anderen Verfahren oder dem Benutzer überlassen werden.

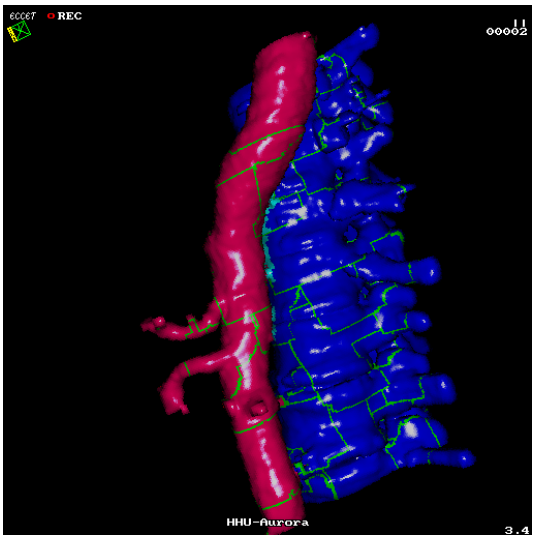
Anwendung

Abgesehen von der intendierten Anwendung, der Hierarchisierung verzweigter Adernsysteme, wurden noch verschiedene andere, teils unerwartete Anwendungen gefunden.

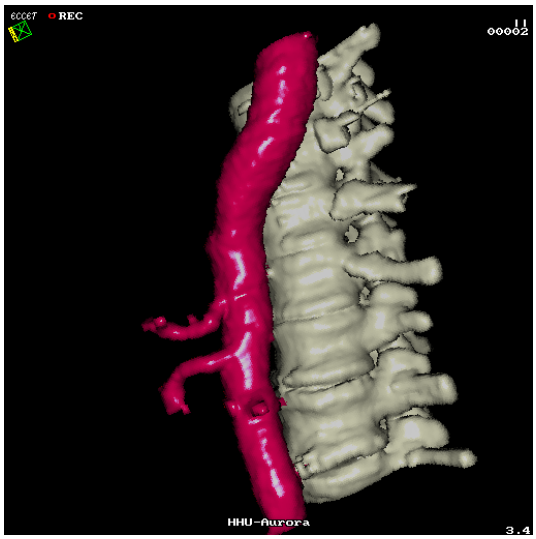
So kann das Verfahren insbesondere verwendet werden, um unklare, unerwünschte Übergänge, wie sie z.B. gerne bei der Trennung von Aorta und Wirbelsäule auftreten, zu lokalisieren.

Aorta und Wirbelsäule liegen anatomisch sehr dicht zusammen. Segmentiert man die Aorta über Ihren hohen Grauwert, so erhält man oft unvermeidlich gleichzeitig die Wirbelsäule. Eine Trennung über Zusammenhangsverfahren ist entsprechend nicht möglich. Aufgrund der komplizierten Struktur der Wirbelsäule ist es manuell nicht gerade leicht, die Stellen zu finden, an denen es Berührpunkte gibt (s. Abbildung 6.15a).

Wendet man nun das Verfahren 2 auf das noch ungetrennte Gesamtvolumen aus Wirbelsäule und Aorta an, wobei man die Toleranz für Stummel relativ hoch ansetzt, so wird die Aorta nur in sehr wenige Stücke zerteilt, da sie nur wenige große



c) Nach manueller Trennung der Problemstellen mit dem mausgesteuerten Skalpell



d) Endergebnis

Abbildung 6.16: Trennung von Aorta und Wirbelsäule mit dem Verzweigungsfinder

Abzweigungen besitzt.

Die Wirbelsäule hingegen wird in der Regel aufgrund ihrer Struktur (Dornfortsätze, mehrfache Verbindungen zwischen den einzelnen Wirbeln durch unvollständig erscheinende Bandscheiben etc.) in sehr viele Komponenten zertrennt.

Nun läßt sich die Aorta mit wenigen Mausklicks füllen (s. Abbildung 6.15b). Oftmals wurde die notwendige Trennung bei schmalen Übergängen bereits geschaffen (sie stellt dann ja eine Abzweigung dar). Bei breiten Übergängen kann es sein, dass dies ausbleibt. In diesem Fall bleibt die Ausbreitung des Umfärbens aber durch die starke Zerteilung der Wirbelsäule auf ein kleines Areal (typisch maximal ein Wirbelkörper) begrenzt. Durch dieses **begrenzte** Auslaufen der Füllung wird sofort klar, wo die unerwünschten Verbindungen liegen, so dass leicht mit Hilfe manuell gesetzter Schnitte die Trennung vervollständigt werden kann (s. Abbildung 6.16).

Eine andere nützliche Anwendung ist die Trennung großflächiger schwach gekrümmter Objekte (z.B. Organe) von stark verzeigten Geflechten (z.B. deren versorgende Blutgefäße).

Ein Beispiel für diese Anwendung sehen Sie in Abbildung 6.12 illustriert. Man sieht in Teilabbildung b sehr gut, daß die Niere in nur 4 Teile zerlegt wird, die schnell umgefärbt werden können. Zusätzlich wurden in diesem Beispiel die drei Gefäßsysteme der Leber nachträglich getrennt. Dazu beginnt man an einer Stelle an der die Zuordnung klar erkennbar ist mit der Umfärbung und verfolgt dann das Gefäß weiter.

6.6.3 Der Hüllenoperator

Für verschiedene Aufgaben ergab sich die Forderung nach einem Operator, der auch relativ dünn liegende Punktwolken zu massiven Körpern verdicken sollte. Dies erschien zunächst nützlich, um verrauschte Ränder zusammenzufügen, es fanden sich aber nach Implementation noch eine Reihe weiterer Einsatzmöglichkeiten.

Der gesuchte Operator sollte die Lücken füllen, ohne jedoch das Objekt nach außen zu verdicken, wie es eine Dilation tun würde, und ohne die Geometrie stark zu verändern, wie es bei einer Closing-Operation mit großem Strukturelement geschehen kann.

Was der Vorstellung am ehesten nahe kam, war eine konvexe Hülle.

Es sollten aber auch konkave Formen möglich sein, wenn deren Konkavität ein bestimmtes Maß nicht übersteigt, um die bearbeitbaren Objekte nicht unnötig einzuschränken.

Problemstellung

Eine gegebene dünne Voxelmenge soll zu ihrer konvexen Einhüllenden verdickt werden.

Optional sollen Stellen von parametrierbarer (geringer) Konkavität nicht wesentlich verändert werden.

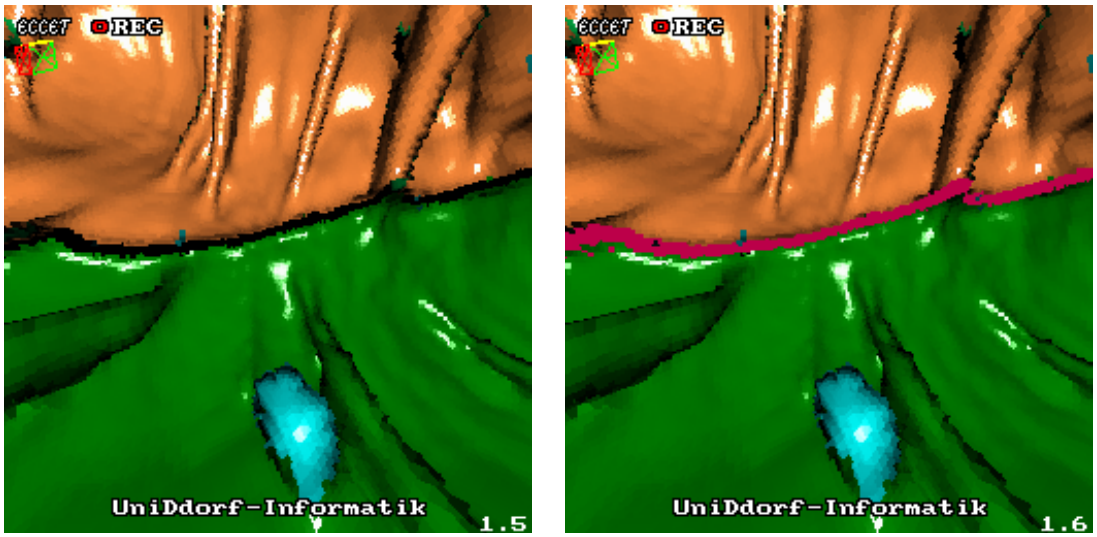
Voraussetzungen

Eingabe sind die Menge aller Voxel V, zwei Voxelmengen S und E (Start und Ende) sowie eine Maximaldistanz l. Die Mengenelemente stellen die einzelnen Voxel in Form ihrer Ortsvektoren dar. Der Algorithmus soll alle Verbindungslinien $\vec{s} \rightarrow \vec{e}$ mit $\vec{s} \in S, \vec{e} \in E$ mit Maximaldistanz l finden und die diesen Linien am nächsten liegenden Voxel einer neuen Voxelmenge O zuordnen.

Ausgabe ist diese neue Menge O.

Algorithmus

- Für alle $\vec{s} \in S$ führe aus:
  - Für alle $\vec{e} \in E$ mit $d(\vec{s}, \vec{e}) < l$



Anwendung des Hüllenoperators zum Schließen der Lücke zwischen den beiden Teilvolumina bei der kontrastmittelgestützten virtuellen Koloskopie. Die Flüssigkeitsoberfläche wurde zur Verdeutlichung ganz ausgeblendet.

Abbildung 6.17: Schließen von Lücken durch den Hüllenoperator

* Für jedes $\vec{x} := d \cdot \vec{s} + (1-d) \cdot \vec{e}$ mit $d \in [0,1]$ füge das Element $\vec{o} \in V$ zu O hinzu, für das $d(\vec{o}, \vec{x})$ minimal wird.

Implementation

Die Implementation durchsucht zunächst das Volumen nach Voxeln aus S.[5]

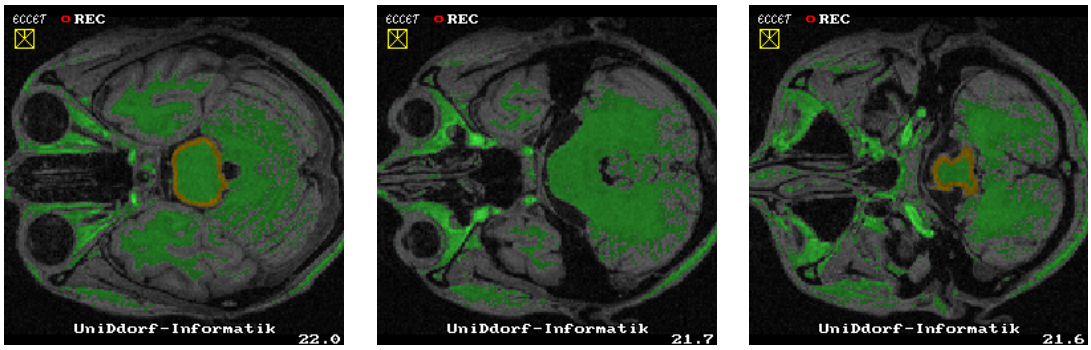
Für jedes gefundene Voxel $\vec{s}$, wird die Kugel mit Radius l um dieses Voxel nach Voxeln aus E durchsucht. Diese sind per Definition weniger als l von $\vec{s}$ entfernt.

Mit Hilfe des Bresenham-Algorithmus [BHM] wird nun eine diskrete Verbindungslinie[6] zwischen den beiden gefundenen Voxeln gezeichnet.

Dabei wird vermieden, Voxel, die bereits einer der Klassen S oder E angehören, neu zu markieren, da in der Implementation ein Voxel immer nur einer Klasse angehören kann. Diese Maßnahme ist ebenfalls erforderlich, um die Notwendigkeit eines Zielvolumens zu vermeiden.

[5] S und E sind als Voxelklassen im Volumen markiert.

[6] Der Artikel [BHM] diskutiert den Begriff einer diskreten Verbindungslinie und zeigt, dass der Bresenham-Algorithmus gerade die Punkte einzeichnet, die der kontinuierlichen Verbindungslinie am nächsten liegen.



In verschiedenen Schichten ist die Trennung unterschiedlich gut erkennbar. Insbesondere kann im Bereich der Kleinhirnschenkel (mittleres Bild) keine sinnvolle Trennlinie mehr angegeben werden. Dagegen kann die Trennung ober- und unterhalb dieses Bereiches problemlos erfolgen.

Abbildung 6.18: Trennung von Kleinhirn und Hirnstamm - Markierung

Laufzeit

Die Rechenzeit wird bestimmt von

a) der Anzahl der Voxel in S,

b) der Größe der zu durchsuchenden Umgebung l.
Aus diesen beiden Punkten ergibt sich eine Laufzeit von $\#S \cdot (2l+1)^3$.

c) der Anzahl und Entfernung von Voxelpaaren $\vec{s}, \vec{e}$ mit $d(\vec{s}, \vec{e}) < l$. Dieser Anteil induziert eine Laufzeit von $\sum_{d(\vec{s},\vec{e})<l} d(\vec{s}, \vec{e})$ (Laufzeit des Bresenham-Algorithmus ist linear mit der Streckenlänge).

Welcher Term dominiert, hängt stark von den Mengen S und E ab. Beide Terme zeigen eine starke Abhängigkeit von l. Der zweite Term scheint auf den ersten Blick nur linear abzuhängen, besitzt aber eine versteckte zusätzliche Abhängigkeit, da die Anzahl möglicher Paare $\vec{s}, \vec{e}$ in der Regel ebenfalls bei steigendem l steigt. Typischerweise steigt diese sogar mit l^3 aufgrund der entsprechenden Volumenvergrößerung der zu durchsuchenden Kugel.

Optimierungen

- Falls $S = E$, genügt es, aufgrund der Symmetrie eine Halbkugel zu durchsuchen. Damit vermeidet man, die Fälle, die durch Vertauschen von Start- und Endpunkt entstehen, doppelt zu betrachten.

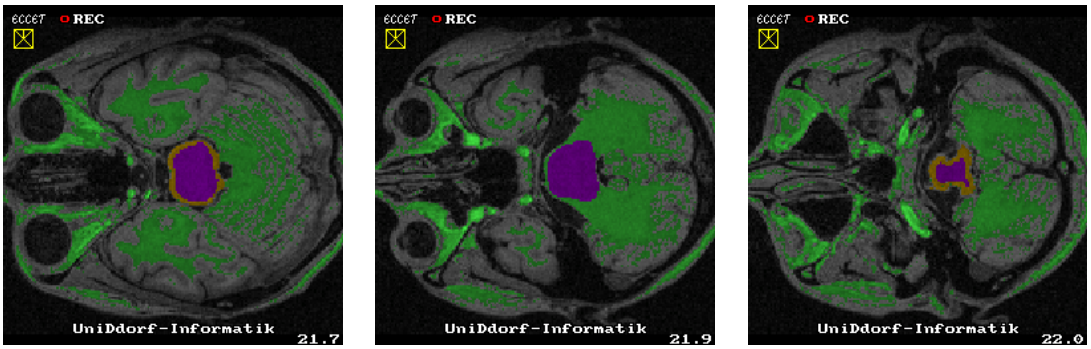


Abbildung 6.19: Trennung von Kleinhirn und Hirnstamm - Hüllenoperator

- Auch im Fall $S \neq E$ kann dieses Verfahren angewendet werden. Man sucht dann zunächst nach Punkten in $S \cup E$ und durchsucht dann die Halbkugel nach Punkten aus S, falls der gefundene Punkt in E lag und umgekehrt.

- Der Algorithmus zum Linienzeichnen muss so weit wie möglich optimiert werden, da er sehr oft aufgerufen wird. Bresenham bietet sich hier an, da er für die typisch relativ kurzen Strecken mit Länge $< l$ den Vorteil hat, wenig konstante Zeit für initiale Berechnungen zu benötigen.

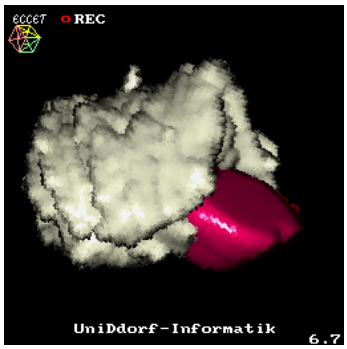
- Optional wurde ein fixedpoint-Arithmetik-basierter Algorithmus implementiert, der für kurze Linien durch eine sehr einfache Initialisierung noch etwas bessere Laufzeiten erreicht. Er verzichtet dazu auf eine korrekte Normierung des Richtungsvektors auf die größte Komponente (das würde eine Division erfordern), wodurch er zwar gelegentlich Voxel doppelt berührt, was aber bei den kleinen Längen durch den schnellen Setup-Anteil überkompensiert wird.

- Voxel in S, die vollständig von weiteren Voxeln in S eingeschlossen sind, tragen nicht signifikant zum Ergebnis bei. Detektiert man diese Voxel und ignoriert sie, kann eine massive Beschleunigung bei der Bearbeitung dichter Voxelmengen erreicht werden. Bearbeitet man dünne Mengen, in denen dieser Fall effektiv nicht vorkommt, kostet diese Überprüfung natürlich etwas Zeit.

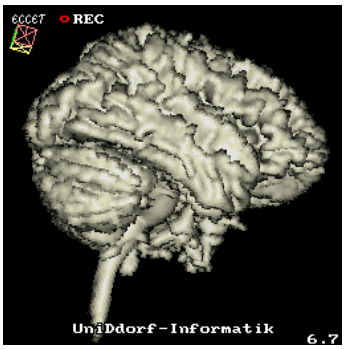
Anwendungen

a) Leberoberfläche
Bei der Segmentierung der Leberoberfläche erlaubt es der Algorithmus, Lücken zu schließen, die sich an einigen schwierigen Punkten, bei denen Kontakt zu einer Darmschlinge, einem Gefäß oder einer Rippe besteht, ergeben.

a) durch den Hüllenoperator ergänztes Stück (violett) und die beiden manuell erfassten Konturen (gelb)



b) Gesamtergebnis: Per Füllalgorithmus segmentiertes Kleinhirn und vom Hüllenoperator erzeugter Bereich



c) Ohne Verwendung des Hüllenoperators kann das Kleinhirn mit dem Füllalgorithmus nicht vom Hirnstamm und dem Großhirn getrennt werden.

Abbildung 6.20: Trennung von Kleinhirn und Hirnstamm - Ergebnis

b) Darmwand
Eine ähnliche Anwendung ergibt sich bei der Darstellung von virtuellen Koloskopieaufnahmen unter Kontrastmitteleinsatz. Bei dieser Technik entstehen zunächst zwei getrennte Teile des Darmlumens: zum einen der luftgefüllte, zum anderen der kontrastmittelgefüllte Teil. Zwischen beiden ergibt sich aufgrund von Partialvolumeneffekten (vgl. 6.5) in der Regel ein Spalt von wenigen Voxeln Höhe. Mit dem Hüllenoperator kann dieser Spalt leicht gefüllt werden. In diesem Anwendungsfall werden S und E verschieden gewählt; man verwendet die Klassen der beiden Teilvolumina. Damit wird erreicht, dass nur der Spalt geschlossen wird, ohne dass sich die sonstige Geometrie der beiden Halbschalen verändert (s. Abbildung 6.17).

c) Trennung von Kleinhirn und Hirnstamm
Bei der Segmentierung des Kleinhirns ergab sich die Fragestellung, wie dieses sinnvoll vom Hirnstamm, mit dem es durch zwei dicke Schenkel verbunden ist, abgetrennt werden kann. Es gibt hier bezüglich der Grauwerte in der MRT-Aufnahme keine Unterscheidungsmöglichkeit, und auch wenn man Experten befragte, konnten sie nicht in allen Ansichten klare Aussagen machen.

Weil das Volumen des Kleinhirns bestimmt werden sollte, war es wichtig, eine zwar ggf. arbiträre, aber reproduzierbare Trennlinie zu finden.

Hierzu kann der Hüllenoperator genutzt werden. Ober- und unterhalb des Bereiches der Kleinhirnschenkel ist die Entscheidung klar zu treffen (s. Abbildung 6.18, linkes und rechtes Bild). Man umfährt dort manuell einfach die Kontur

des Hirnstamms. Dazwischen (mittleres Bild in 6.18) kann keine klare Grenzlinie angegeben werden.

Durch Anwendung des Hüllenoperators können die vorgegebenen Konturen sinnvoll interpoliert werden. Das fehlende Zwischenstück wird dabei auf sehr natürlich wirkende Weise ergänzt, und es entsteht eine reproduzierbare Grenze im Bereich der Kleinhirnschenkel (Abbildung 6.19). Damit ist die kritische Stelle beseitigt, und man kann das Kleinhirn problemlos mit Hilfe des Füllalgorithmus (6.2) segmentieren (Abbildung 6.20).

Ohne Einführung einer künstlichen Trennung würde der verwendete Füllalgorithmus zum Großhirn hin auslaufen (rechtes Bild in Abbildung 6.20).

6.7 3D-Werkzeuge und virtuelle Chirurgie

Oftmals führt schon die Verwendung sehr einfacher automatischer Segmentierungsalgorithmen zu guten Ergebnissen. Allerdings werden dabei je nach verwendetem Algorithmus auch weitere Strukturen mit ähnlichen Eigenschaften mitsegmentiert.

Haben diese keinen Zusammenhang mit dem erwünschten Objekt, sind sie leicht zu entfernen. Schwieriger wird es, wenn diese mit dem Objekt zusammenhängen.

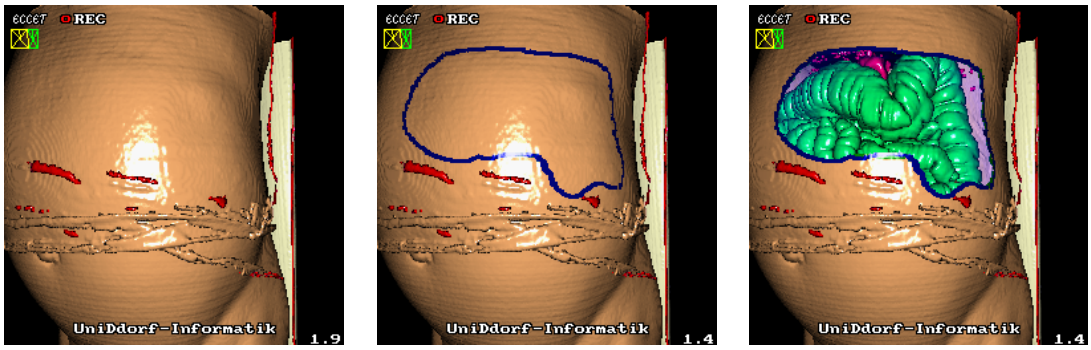
So findet ein Füllagorithmus, den man verwendet, um das Gefäßsystem der Leber mitsamt den zugehörigen Zu- und Abflüssen zu selektieren, in der Regel auch die zu den Nieren führenden Gefäße und markiert ggf. sogar die Nieren selbst mit.

Die Abgrenzung dieser Bereiche ist für den menschlichen Experten leicht möglich. Was benötigt wird, sind somit leicht und intuitiv bedienbare Werkzeuge, mit denen dieses Wissen auf die virtuelle Darstellung übertragen werden kann.

Für den Bediener am angenehmsten ist dabei die Nachahmung ihm bekannter Werkzeuge wie Skalpellen, wobei man oft noch zusätzliche Hilfen wie Schnitttiefenbegrenzung etc. anbieten kann.

6.7.1 Mausgesteuertes Skalpell

Durch die interaktive Performance ist es möglich, in Echtzeit Bereiche unter der Maus umzufärben, wobei Eindringtiefe und Schnittbreite einstellbar sind.



a) Außenansicht eines per Autosegmentierung und Umfärbung gewonnenen Volumens

b) Einzeichnen einer Umrisslinie mit dem mausgesteuerten Skalpell

c) Der ausgefüllte Umriss wird transparent geschaltet, so dass der Blick auf das Innere frei wird.

Abbildung 6.21: Manuelles Schneiden von Oberflächen

Voraussetzung

Gegeben sei eine Voxelmenge V, die mit Hilfe der Methoden von 4.2 als 2D-Bild visualisiert werden kann, ein Schittbreitenparameter d sowie eine Zielfarbe z und eine interaktive Eingabemöglichkeit zur Steuerung der Ansicht und Wahl eines angezeigten Pixels.

Algorithmus

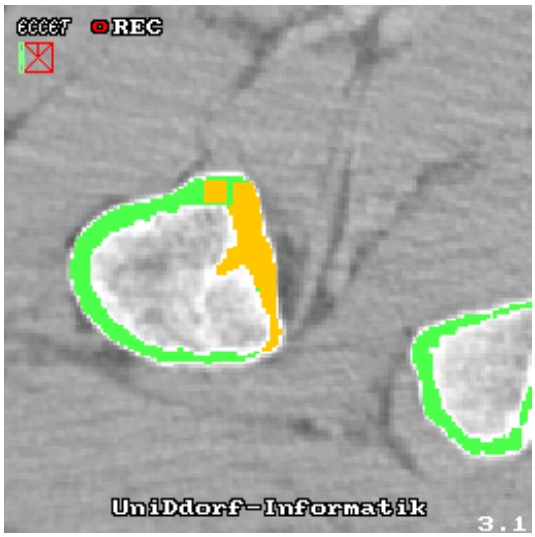
1. Berechne eine Ansicht des Volumens mit den Methoden von 4.2. Man beachte, dass dort jeweils zusätzlich zu den anzuzeigenden Pixeln auch die für das jeweilige Pixel verantwortlichen Voxel im Ergebnis gespeichert werden.

2. Warte auf Benutzereingaben, drehe und verschiebe ggf. die Ansicht gemäß den Benutzeranforderugen und fahre mit Schritt 1 fort. Fordert der Benutzer eine Schneideoperation an, stelle fest, auf welches Pixel der Mauszeiger gerade zeigt.

3. Stelle mit Hilfe des Ergebnisfeldes des Renderers aus Schritt 1 fest, welches Voxel die Anzeige des gewählten Pixels veranlasst hat. Ist dort nichts eingetragen, fahre bei Schritt 1 fort.

4. Stelle die aktuelle Klasse s des gewählten Voxels fest.

5. Färbe alle Voxel mit Klasse s innerhalb einer Kugel mit Radius d um das gewählte Voxel zur Klasse z um. Kehre zurück zu 1.



Segmentierung von Grauwertdaten mit der Maus - man beachte, dass das grüne Overlay den Wirkungsbereich des Werkzeuges begrenzt

Abbildung 6.22: Manuelle Segmentierung in der Grauwertansicht

Beispiel: Aufschneiden einer Oberfläche

Auf Oberflächen können mit dem Werkzeug beliebige Schnitte durchgeführt werden, ohne darunterliegende Schichten, die zu anderen Voxelklassen gehören, zu beschädigen. Nur wenn sich in der Nähe weitere Schichten befinden, die zur gleichen Klasse gehören, muss die Eindringtiefe gering genug gewählt werden.

Die Verwendung dieses Werkzeuges wird in Abbildung 6.21 illustriert. Die Linie in Abbildung 6.21b wird interaktiv mit der Maus eingezeichnet. Durch die interaktive Bedienbarkeit hat man jederzeit Kontrolle über den Schneidevorgang und kann ggf. auch das Volumen beliebig bewegen, um eine bessere Ansicht zu erhalten.

Anschließend wurde der von der Linie abgegrenzte Bereich umgefärbt und transparent gemacht (Abbildung 6.21c), um die dahinterliegenden Strukturen sichtbar zu machen.

Beispiel: Lokale Markierungen in der Grauwertansicht

Diese Funktion arbeitet nicht nur auf Oberflächen, sondern auch auf Grauwertdaten. Daher kann sie zur initialen Segmentierung ebenso genutzt werden wie zur Nachbearbeitung von Oberflächendaten.

Abbildung 6.22 zeigt den Einsatz auf Grauwertdaten.

Bei der Segmentierung ergibt sich manchmal die Situation, dass Objekte **lokal** gut mit bestimmten Grauwertfenstern korrespondieren, die jedoch nicht global gelten.

Das mausgestuerte Skalpell erlaubt, solche lokalen Segmentierungsentscheidungen schnell und präzise dem System mitzuteilen.

Hierzu blendet man in das Bild das gewählte Grauwertfenster (grüne Bereiche in 6.22a) ein und zeichnet die hierdurch vorgegebene Kontur mit der Maus nach. Die Breite des Pinsels wird durch den d-Parameter bestimmt. Man beachte, dass das Schnittwerkzeug zusätzlich vom aktiven Overlay begrenzt wird, so dass man eine Grauwertkontur lokal präzise herausarbeiten kann, ohne dabei mit kleinen Pinselbreiten arbeiten zu müssen. Die aktive Pinselbreite im Beispiel sehen Sie an der isolierten gelben Stelle links oben.

Beschränkungen/Erweiterungen

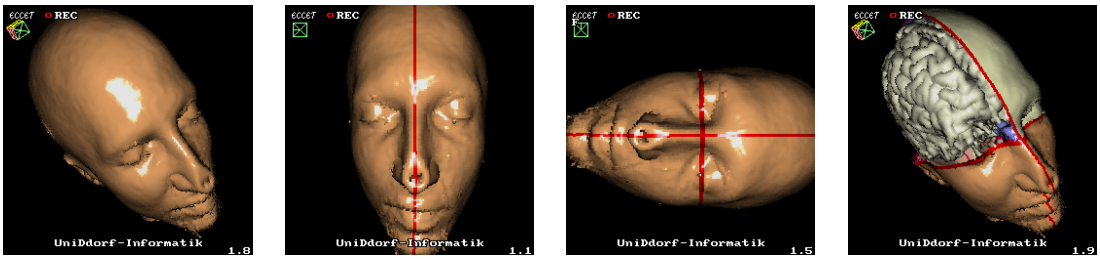
Zum Durchtrennen größerer Strukturen (wie großen Blutgefäßen) wäre es nützlich, Schnittbreite und -tiefe einzeln einstellen zu können. In diesem Fall möchte man ja einen schmalen Schnitt, der gerade so tief ist wie die zu durchtrennende Struktur.

Diese mögliche Erweiterung wurde, ebenso wie die Implementation anderer Schnittformen, nicht implementiert, da sie ohne Anzeige des Werkzeuges und die Nutzung echter 3D-Eingabegeräte nur sehr schwer zu kontrollieren ist.

Einen möglichen Ausweg könnten die Multivolumen-Fähigkeiten, die VOXREN neuerdings bietet, darstellen. Mit ihnen wird es möglich, die Lage des Werkzeuges darzustellen. Für interaktive Schnitte bleibt allerdings immer noch das Problem der Steuerung.

6.7.2 Ebene, zusammenhängende Schnitte

Der eben angesprochene Vorgang des virtuellen Durchtrennens von Gefäßen kann im Rechner automatisiert werden. Dazu wurde ein Werkzeug geschaffen, bei dem man das zu durchtrennende Objekt zunächst so ausrichtet, dass die gewünschte Schnittebene in der Bildschirmansicht senkrecht von oben nach unten verläuft, und dann lediglich auf einen Punkt auf der Oberfläche des Objektes klickt, durch den der Schnitt verlaufen soll.



a) Außenansicht eines segmentierten Kopfes b) Einzeichnen des ersten Schnittes c) Einzeichnen des zweiten Schnittes d) Endergebnis

Abbildung 6.23: Nutzung zusammenhangsgesteuerter Schnitte

Voraussetzung

Gegeben sei eine Voxelmenge O, ein Punkt $\vec{s} \in O$ (Startpunkt) und eine Ebene E mit $\vec{s} \in E$.

Algorithmus

- Bilde eine neue Menge $O' \subseteq O$ mit $O' := \{\vec{o'} : d(\vec{o'}, E) <= 1\}$.
- Starte ausgehend von $\vec{s}$ eine Breitensuche in O'. Markiere alle gefundenen Voxel um.

Bemerkungen

Die Konstante 1 in der Bedingung $d(\vec{o'}, E) <= 1$ stellt sicher, dass auch auf diskreten Gittern der Schnitt noch eine im Sinne der Breitensuche zusammenhängende Menge ergibt.

Die Ebene E wird über die aktuelle Lage der Ansicht bestimmt, indem die Lage der durch Blickrichtung und Orientierung der Kamera definierten Schnittebene in das Volumen projiziert und parallel verschoben wird, um $\vec{s} \in E$ zu erreichen.

Anwendungen

Es ergibt sich ein sauberer Schnitt in der vom Benutzer über die Ansicht vorgegebenen Richtung, der das angeklickte Objekt vollständig durchtrennt, aber keine in der Nähe befindlichen Strukturen beschädigt.

In Abbildung 6.23 sehen Sie eine mögliche Anwendung; dort soll zu Illustrationszwecken ein Segment der umhüllenden Schädeldecke entfernt werden, um den Blick auf das darunterliegende Gehirn freizugeben. Dazu dreht man das Objekt zunächst

a) Gefäßsystem mit Leber und den beiden Nieren

b) Mit zwei Schnitten werden die Nieren abgetrennt.

Abbildung 6.24: Nutzung zusammenhangsgesteuerter Schnitte II

so, dass man von vorne auf den Kopf blickt und dieser gerade ausgerichtet ist. Mit einem Klick auf die Nasenwurzel setzt man die erste Trennlinie (b). Diese folgt automatisch der Kopfkontur und schneidet die äußere Schicht durch, ohne das darunterliegende Gehirn zu beschädigen.

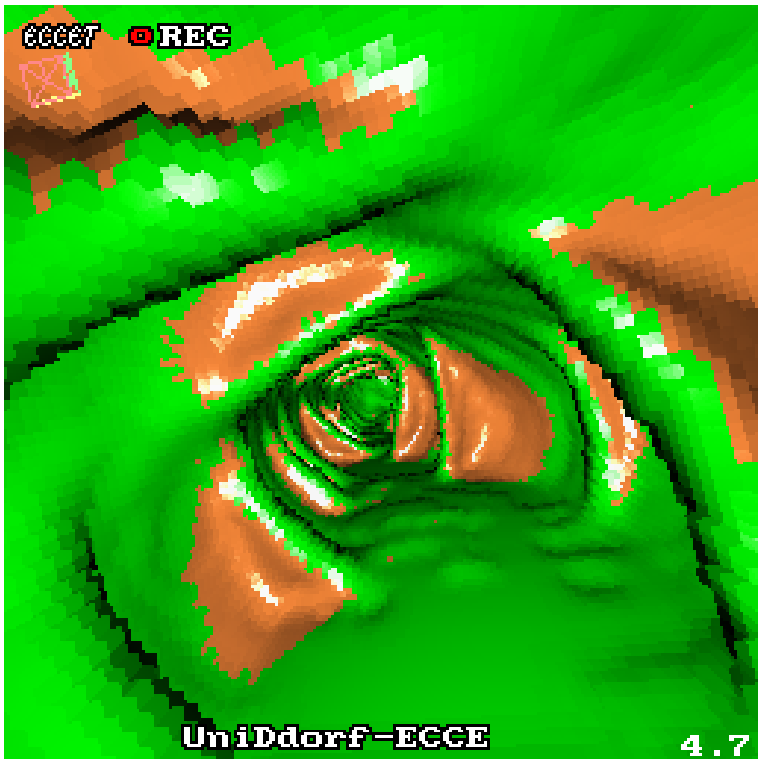
Im Beispiel ist dies schon dadurch ausgeschlossen, dass es einer anderen Voxelklasse angehört. Es würde aber auch bei gleichen Klassen funktionieren, sofern innerhalb von O' kein Zusammenhang zwischen der zu schneidenden Oberfläche und dem Gehirn besteht.

Danach wird der Kopf um 90° gedreht, um den zweiten Schnitt (c) zu setzen. Die entstehenden Segmente werden umgefärbt und eines davon transparent gemacht (d).

Selbstverständlich könnte ein ähnlicher Effekt auch mit den frei definierbaren Schnittebenen in VOXREN erreicht werden.

Eine typischere Anwendung stellt eher das gezielte Durchtrennen von Gefäßen dar, z.B. um Organe vom sie versorgenden Gefäßsystem zu trennen.

Abbildung 6.24 zeigt einen solchen Fall. Dort wurden die Nieren mitsamt Gefäßsystem segmentiert und sollen nun von diesem getrennt eingefärbt werden. Dazu legt man je einen Schnitt durch die versorgenden Gefäße, überprüft, dass es keine weiteren Verbindungen gibt, und färbt die Nieren dann um.



Darstellung von nicht gesehenen Bereichen bei der Koloskopie. Auf dem Hinflug wurden alle gesehenen Voxel grün markiert. Nicht gesehene Bereiche erscheinen auf dem Rückflug in der ursprünglichen Farbe (braun).

Abbildung 6.25: Visualisierung gesehener Bereiche

6.7.3 Ummarkierung aller sichtbaren Voxel

Problem/Voraussetzung

Gegeben sei eine vom Renderer generierte Ansicht einer Voxelmenge. Markiere alle Voxel, die in der Ansicht zu sehen sind.

Algorithmus

Markiere während der Laufzeit des Renderalgorithmus jedes Voxel, das zur Darstellung beiträgt.

Alternativ kann dies auch nachträglich anhand der entsprechenden Information im Renderergebnis geschehen.

Anwendung

Diese Methode eignet sich scheinbar primär zur Markierung flächiger Bereiche. Allerdings erfordert diese Anwendung große Sorgfalt bei der Auswahl des Bildausschnittes (es dürfen keine anderen Bereiche sichtbar sein) und funktioniert auch nur bei glatten Flächen und hinreichender Vergrößerung störungsfrei.

Wesentlich nutzbringender kann der Algorithmus zur Visualisierung **nicht** gesehener Bereiche, wie sie z.B. bei der realen Endoskopie systembedingt vorkommen, eingesetzt werden.

Dazu geht man wie folgt vor:

Man aktiviert die Ummarkierung aller sichtbaren Voxel und durchfliegt den Darm in Vorwärtsrichtung mit realitätsnahem Kameraöffnungswinkel ($110° - 140°$). Dabei werden alle Voxel, die man gesehen hat, ummarkiert. Nicht gesehene Voxel verbleiben in ihrer alten Farbe.

Dann deaktiviert man den Modus wieder und durchfliegt den Darm in Gegenrichtung, was mit dem virtuellen Endoskop problemlos möglich ist. [7] Da nur die schon gesehenen Voxel ummarkiert wurden, erkennt man nicht gesehene Bereiche an der unterschiedlichen Färbung.

Abbildung 6.25 zeigt ein Dickdarmstück, das auf diese Weise behandelt wurde, auf dem Rückflug. Man erkennt, dass trotz 140° Öffnungswinkel der virtuellen Endoskopkamera größere Bereiche hinter Falten nicht gesehen wurden (braun dargestellt).

[7] Ein reales Endoskop hat zwar zumeist die Möglichkeit, den Kopf zu verkippen um auch hinter Falten zu sehen, doch kann es dann in der Regel nicht mehr ungehindert bewegt werden und der Schlauch behindert natürlich weiterhin die Sicht.

Kapitel 7

CAD-Algorithmen

Zur Auswertung der immer detaillierteren Daten, die moderne Untersuchungsmethoden liefern, wird es zunehmend wichtiger, dem Arzt unterstützende Werkzeuge anzubieten, die die Diagnosefindung erleichtern.

$\epsilon\mathcal{CC}\epsilon\mathcal{T}$ enthält einige solche CAD(Computer Aided Diagnosis)-Algorithmen, die helfen sollen, gesuchte Pathologien in der großen Datenmenge zu finden.

7.1 Polypenfinder

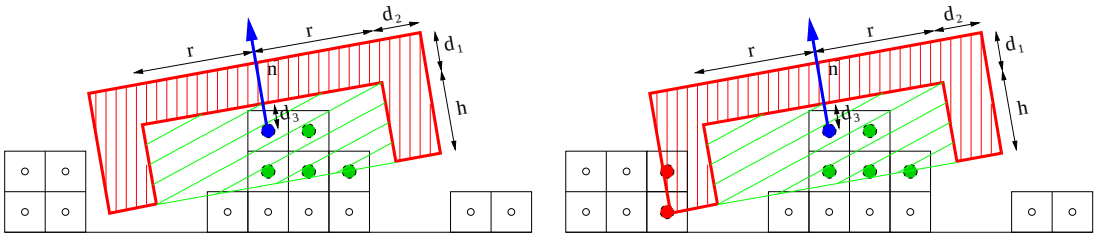
Eine wesentliche Aufgabe der virtuellen Koloskopie ist das Finden von Polypen und Tumoren. Für eine automatische Detektion bieten sich insbesondere Polypen an, da sie charakteristische geometrische Eigenschaften haben, die von denen der Darmwand unterscheidbar sind.

Polypen sind nach innen in den Darm ragende Ausbeulungen von unterschiedlicher Form, zumeist aber mit einem (halb-)kugeligen Kopf und ggf. einem tragenden Stiel.

Zumindest im Kopfbereich zeichnet sich ein solcher Polyp durch eine hohe, allseitig positive Krümmung aus.

Darmfalten hingegen sind zwar auch in einer Hauptkrümmungsrichtung stark positiv gekrümmt, in der anderen aber allenfalls sehr schwach positiv, in der Regel sogar eher negativ.

Es gilt nun ein diskretes Verfahren zu finden, das *stabil* solche Punkte findet und markiert. Das Problem besteht darin, dass die differenzielle Größe Krümmung sich im Diskreten schlecht numerisch stabil fassen lässt.



a) Polypenfinder (Dosenwandverfahren) b) Probleme bei eng benachbarten Strukturen

Abbildung 7.1: Polypenfinder - Dosenwandverfahren

7.1.1 Dosenwandverfahren

Anschaulich-geometrisch gesprochen versucht dieses Verfahren, geeignet große Dosen jeweils über einen Polypen zu stülpen.

Voraussetzung

Gegeben sei eine Voxelmenge O (Darmoberfläche), bei der zu jedem Element $\vec{o} \in O$ die zugehörige Normale $\vec{n}(\vec{o})$ bekannt ist. Die Richtung der Normalen weise in Richtung derjenigen Oberfläche, auf der die Polypen gefunden werden sollen.

Gegeben seien ferner eine vom Benutzer vorgegebene Mindesthöhe h und eine über den Radius r vorgegebene Krümmung $\frac{h}{r}$.

Algorithmus

- Für alle $\vec{o} \in O$ erzeuge eine analytische Darstellung einer Dose (Zylinermantel plus eine Bodenscheibe) mit einer Wandstärke von d_1 bzw. d_2. Die Dose sei so orientiert, dass der Boden senkrecht zu $\vec{n}(\vec{o})$ ausgerichtet ist und der Mantel sich entgegen der Richtung von $\vec{n}(\vec{o})$ ausdehnt. Der Mittelpunkt des Dosenbodens liege bei $\vec{o} + d_3\vec{n}(\vec{o})$.

  - Für alle $\vec{p} \in O$ mit $d(\vec{p}, \vec{o}) \leq max(r + d_2, h + d_1)$ (mit d=Maximumsnorm) prüfe, ob $\vec{p}$ innerhalb der **Wandung** (rote Schraffur in Abbildung 7.1) der Dose liegt. Findet sich zumindest ein solches $\vec{p}$, so ist der Punkt kein Polyp. Fahre fort mit nächstem $\vec{o} \in O$.

  - Ansonsten markiere jeden Punkt der innerhalb des **Innenraumes** (grüne Schraffur in Abbildung 7.1) der Dose liegt.

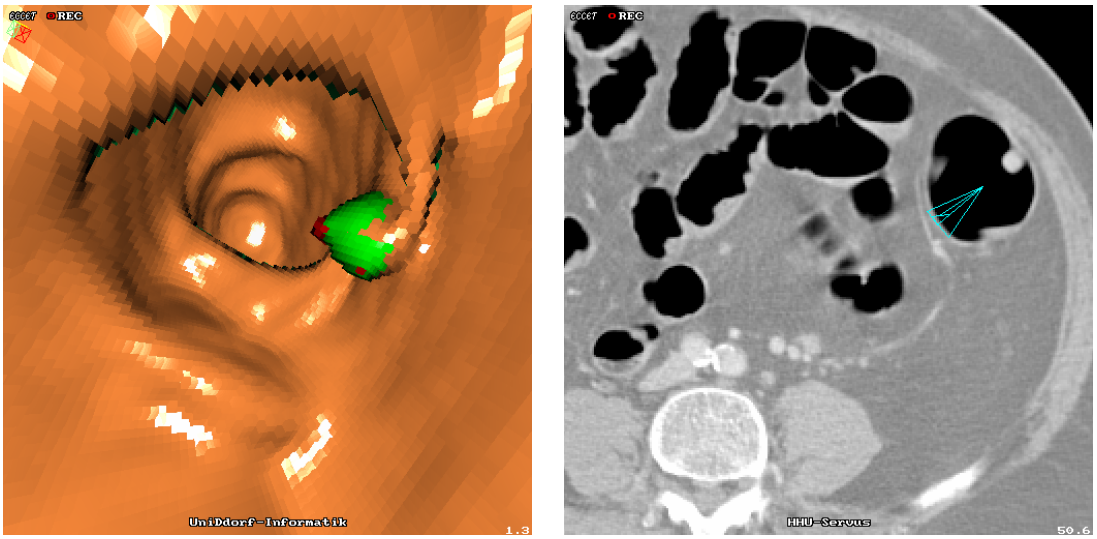


Abbildung 7.2: Polypenfinder - großer Polyp (10 mm)

Implementation

In Schritt 2 des obigen Algorithmus wird ein hinreichend großer Kubus (größer als die Dose in beliebiger Lage einnehmen kann) durchlaufen und nun für alle darin gefundenen Voxel aus O deren Relativkoordinaten bezüglich $\vec{o}$ auf Zylinderkoordinaten mit Achse längs $\vec{n}(\vec{o})$ projiziert. Nun kann anhand dieser leicht entschieden werden, ob ein Voxel im Innenraum, in der Wand oder außerhalb der Dose liegt.

Die Richtung des Normalenvektors steht in VOXREN sowieso zu Renderingzwecken zur Verfügung, so dass keine gesonderte Berechnung erfolgen muss.

Schritt 3 dient nur der **deutlichen** optischen Markierung des Polypen. Prinzipiell würde es natürlich reichen, nur das von $\vec{o}$ bezeichnete Voxel zu markieren.

Bemerkungen

Anschaulich bedeutet der Algorithmus, dass ein Voxel als zu einem Polypen gehörig kategorisiert wird, wenn es möglich ist, eine Dose mit Radius r und Höhe h mittig und mit ihrer Symmetrieachse parallel zu $\vec{n}(\vec{o})$ auf das Voxel aufzulegen, ohne dass die Dose mit ihrem Boden oder ihrer Wandung (jeweils rot in Abbildung 7.1a) weitere Voxel durchdringt.

Die dicke Wandung in der Implementation ist nur notwendig, um auch auf dem diskreten Gitter sicher einen Schnitt zu verursachen.

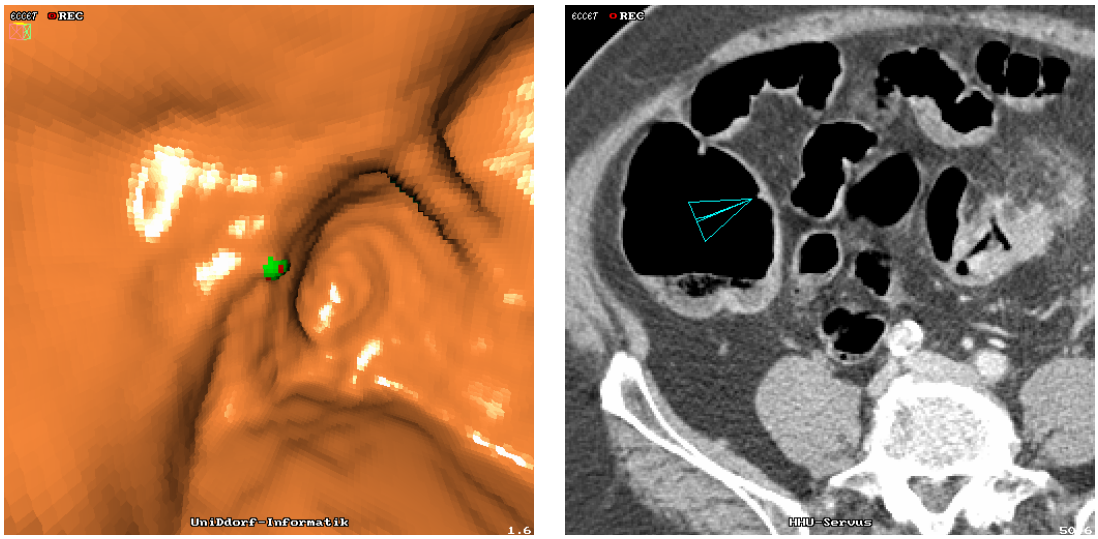


Abbildung 7.3: Polypenfinder - kleiner, flacher Polyp (3 mm)

Grenzen des Verfahrens

In der Nähe eines Polypen befindliche weitere Erhebungen (Falten, andere Polypen) können einen Schnitt mit der Dose auslösen und somit eine Markierung verhindern (s. Abbildung 7.1b).

Dies lässt sich zwar durch Verwendung verschieden großer Dosen abmildern, aber nicht völlig vermeiden. Außerdem steigt die Rechenzeit natürlich entsprechend an.

7.1.2 Inselverfahren

Die Grundidee ähnelt dem Dosenverfahren, allerdings betrachtet man nun die **Zusammenhangskomponente** des aktuellen Voxels innerhalb eines Zylinders mit Radius r und Höhe h, der analog zum ersten Verfahren auf das betrachtete Voxel aufgelegt wird.

Bei einem Punkt mit positiver Krümmung liegt die gesamte Zusammenhangskomponente innerhalb dieses Zylinders.

Voraussetzungen

Wie beim vorigen Verfahren.

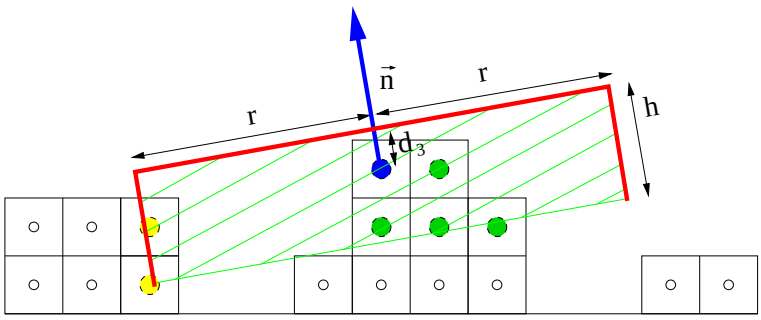


Abbildung 7.4: Polypenfinder - Inselverfahren

Algorithmus

- Für alle $\vec{o} \in O$ erzeuge eine analytische Darstellung eines Vollzylinders mit Radius r und Höhe h. Der Zylinder sei so orientiert, dass die Grundflächen senkrecht zu $\vec{n}(\vec{o})$ ausgerichtet sind. Die Mittelpunkte der Grundflächen liegen bei $\vec{o} + d_3\vec{n}(\vec{o})$ und $\vec{o} - (h - d_3)\vec{n}(\vec{o})$.
  - Finde nun per Breitensuche ausgehend von $\vec{o}$ schrittweise die Zusammenhangskomponente von $\vec{o}$. Für jedes dabei neu gefundene Voxel $\vec{p} \in O$ prüfe:
    - Liegt $\vec{p}$ innerhalb des Zylinders? Wenn ja, füge $\vec{p}$ der Schlange hinzu und fahre fort.
    - Liegt $\vec{p}$ unterhalb der **unteren** Zylinderfläche? Wenn ja, füge $\vec{p}$ **nicht** der Schlange hinzu und fahre fort. Aufgrund der finiten Anzahl im Zylinder enthaltener Voxel terminiert der Algorithmus durch diese Einschränkung.
    - Liegt $\vec{p}$ oberhalb der **oberen** Zylinderfläche oder außerhalb der Seitenflächen (in Abbildung 7.4 jeweils rot dargestellt)? Dann breche die Breitensuche ab. Der betrachtete Punkt $\vec{o}$ ist kein Polyp.
  - Wurde die Breitensuche nicht wegen der dritten Bedingung abgebrochen, sondern hat aufgrund des Aussterbens der Ausbreitung durch die zweite Bedingung terminiert, so markiere alle während der Breitensuche gefundenen und nicht verworfenen Punkte.

Implementation

Die Implementation erfolgt analog zum vorigen Verfahren — es wird lediglich die unspezifizierte Suche durch Breitensuche ersetzt und man benötigt keine dicken

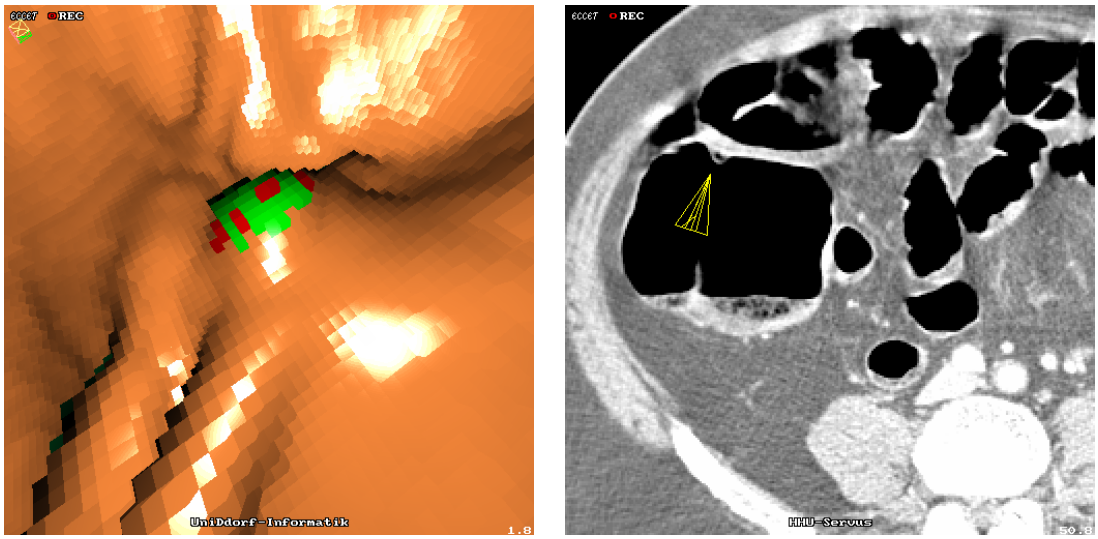


Abbildung 7.5: Polypenfinder - Stuhlrest. In der 2D-Ansicht erkennbar am Lufteinschluss

Wände mehr.

Bemerkungen

Anschaulich gesprochen bedeutet der Algorithmus Folgendes:

- Kippe das Volumen so, dass die Normale des aktuell betrachteten Voxels $\vec{o}$ nach oben weist.
- Fülle so lange Wasser ein, bis der Pegel so hoch steht, dass $\vec{o}$ gerade h über Wasser steht.
- Prüfe, ob die Küste der zu h gehörenden Insel an keinem Punkt weiter als r (zweidimensional von oben betrachtet, dreidimensionale Entfernung wäre natürlich $\sqrt{r^2+h^2}$) von $\vec{o}$ entfernt ist.

Die geforderte Stärke der Krümmung kann bei diesem Algorithmus ebenso wie beim vorherigen mit Hilfe des Verhältnisses von r zu h eingestellt werden.

Vorteile des Verfahrens

Gegenüber dem Dosenverfahren ergeben sich folgende Vorteile:

- Keine Effekte benachbarter Objekte, solange ein mindestens h tiefes Tal diese voneinander trennt. Ein Beispiel hierfür finden Sie in Abbildung 7.4, die übri-

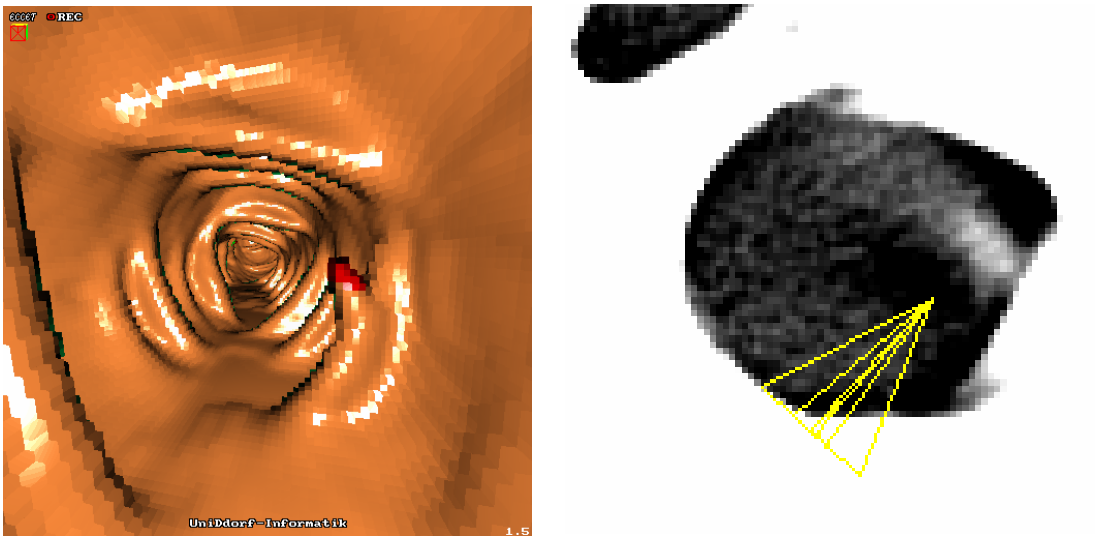


Abbildung 7.6: Polypenfinder - Strahlenartefakt. Vom Artefakterkenner rot markiert (vgl. Abbildungen 7.2,7.3 und 7.5).

gens die gleiche Situation zeigt wie Abbildung 7.1b. Die gelb markierten Voxel werden ignoriert, da es keinen innerhalb der Zylinderscheibe verlaufenden Weg zwischen ihnen und dem Startpunkt gibt.

- Infolgedessen ist es auch nicht notwendig, den Algorithmus mit verschieden großen Werten für r mehrfach auszuführen. Diese Notwendigkeit ergibt sich beim Dosenverfahren daraus, dass bei zu großen Dosen die Gefahr besteht, dass in der Nähe befindliche Störobjekte die Erkennung verhindern.

- Die Bearbeitung erfolgt schneller, da nur die Zusammenhangskomponente, nicht eine (potentiell große) Umgebung betrachtet wird. Aufgrund der Abbruchbedingungen werden maximal so viele Voxel betrachtet, wie in den Zylinder passen.

7.2 Erkennen von Artefakten

Besonders beim Einsatz von Multislice-Spiral-Computertomographen ergibt sich ein Problem mit strahlenförmigen Artefakten, die besonders an starken Kontrasten entstehen.

In der 3D-Rekonstruktion erscheinen diese Artefakte oft als kleine Zapfen, die den obigen Polypenbedingungen genügen.

Als Hilfe bei der Beurteilung der gefundenen Polypenkandidaten wurde daher ein

weiterer Algorithmus implementiert, der in vielen Fällen diese Klasse von „false positives“ erkennt.

Dazu wird die Umgebung der bisher gefundenen Polypenkandidaten nach Voxeln durchsucht, die einen hinreichend hohen Grauwert aufweisen, um als Gewebe klassifiziert zu werden. Dies unterscheidet die meisten Artefakte, die mit sehr flachen Rampen vom Schwarz der Luft zum Grau des Gewebes übergehen, von echten Polypen, bei denen dieser Übergang erheblich schneller erfolgt.

Technisch ist dies durch einen Algorithmus implementiert, der die Umgebung einer vorgegebenen Voxelklasse nach Voxeln durchsucht, deren Grauwert größer (oder optional auch kleiner) als eine vorgegebene Schwelle ist. Dieses Verfahren kann auch zu anderen Zwecken (s. 6.5) genutzt werden.

Die Polypen in den Abbildungen 7.2, 7.3, 7.5 und 7.6 wurden alle vom Artefakterkenner behandelt. Als „vermutlich echt“ erkannte Polypen wurden dabei grün, vermutete Artefakte rot markiert.

7.3 Marker

7.3.1 Anwendung

Gelegentlich treten bei der Befundung unklare Situationen oder evtl. auch besonders lehrreiche Beispiele auf, die man aus verschiedenen Gründen (zweite Meinung, Lehre etc.) einem Kollegen vorlegen möchte.

Eine Möglichkeit dazu ist natürlich der Ausdruck von Bildern. Alternativ können auch im Volumen entsprechende Marker angebracht werden und dem Kollegen die Daten geschickt werden.

Damit hat er die Möglichkeit, sich nicht nur **ein** Bild der Situation zu betrachten, sondern das markierte Objekt selbst von allen Seiten zu begutachten.

7.3.2 Implementation

Marker werden getrennt vom Volumen als Wertepaar (3D-Koordinate,Markername) gespeichert. Da die Kameradaten bekannt sind, kann die 3D-Koordinate in das Kamerasystem übersetzt werden.

Man berechnet dazu, wo in der Kameraebene die Projektion eines Punktes mit den gegebenen Koordinaten liegen würde. Damit ist die Position innerhalb des geren-

derten Bildes bekannt.

Man könnte nun einfach dort den Marker immer einzeichnen. Allerdings wären die Marker dann immer sichtbar, egal ob sie von anderen Strukturen verdeckt sind oder nicht.

Dies ist zwar manchmal nützlich (z.B. bei Außenansichten, da dort in der Regel alle Markierungen verdeckt sind — in Abbildung 7.7b sieht man eine solche Situation), manchmal aber auch eher verwirrend.

Optional kann daher eingestellt werden, dass nur Marker angezeigt werden, die nicht von gerenderten Oberflächen verdeckt werden.

Dazu wird zusätzlich der Abstand des Markers von der Kamera geprüft und mit dem Eintrag im Tiefenfeld des gerenderten Bildes verglichen. Ist der Abstand des Markers zur Kamera kleiner als der des betreffenden Voxels, wird der Marker angezeigt. Ansonsten ist er verdeckt und wird nicht gezeichnet. Diese Variante eignet sich besonders bei Innenansichten, da es dort verwirrend wirkt, wenn hinter Wänden gelegene Marker angezeigt werden.

Als weitere Option lassen sich die Marker auch ganz abschalten.

7.4 Längenmessungen

Im klinischen Einsatz ergab sich die Forderung nach Streckenmessungen in der virtuellen Ansicht, um z.B. die Größe von Polypen direkt beurteilen zu können.

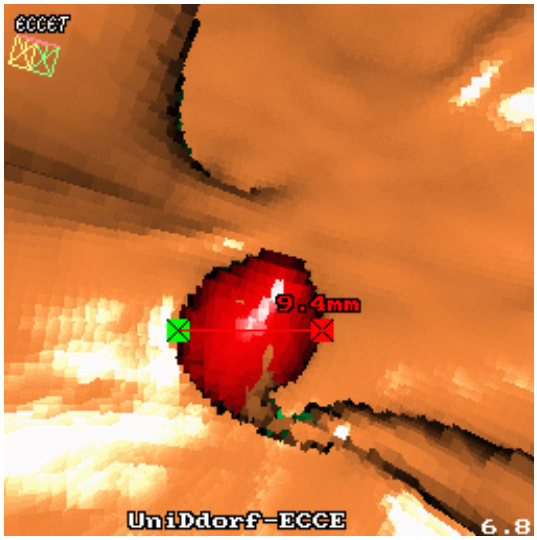
Dazu kann einfach mit der Maus auf zwei Punkte im Bild gezeigt werden, dann erfolgt eine Anzeige des euklidischen Abstandes.

Die dazu notwendigen Daten über die Kantenlängen eines Voxels l_x, l_y, l_z werden den Dateiköpfen der eingelesenen Daten entnommen, soweit vorhanden.

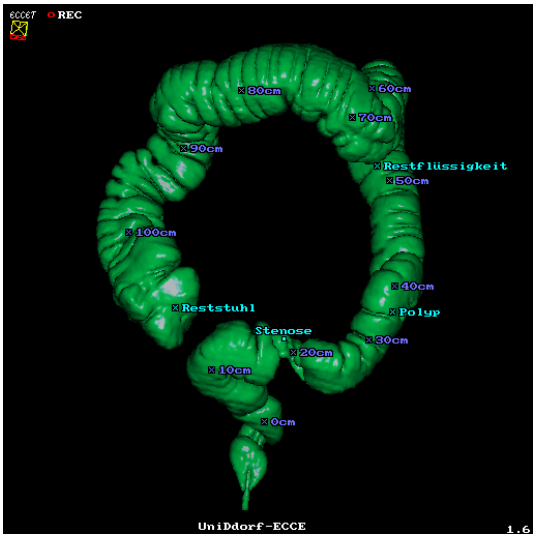
Beim Anklicken eines Pixels in der Darstellung, wird in den vom Renderer gelieferten Daten nachgesehen, welches Voxel die Darstellung dieses Pixels verursacht hat. Auf diese Weise erhält man die Koordinaten s_x, s_y, s_z und e_x, e_y, e_z der beiden Voxel deren Abstand d man finden möchte. Dieser ist nach Pythagoras gegeben durch

$$d = \sqrt{(l_x \cdot (s_x - e_x))^2 + (l_y \cdot (s_y - e_y))^2 + (l_z \cdot (s_z - e_z))^2} \; . \tag{7.1}$$

Eine Anwendung dieses Verfahrens zur Messung des Durchmessers eines Polypen zeigt Abbildung 7.7a.



a) Abstandsmessungen



b) Distanz „ab ano“ und einige frei definierbare Marken

Abbildung 7.7: Messungen und Marker

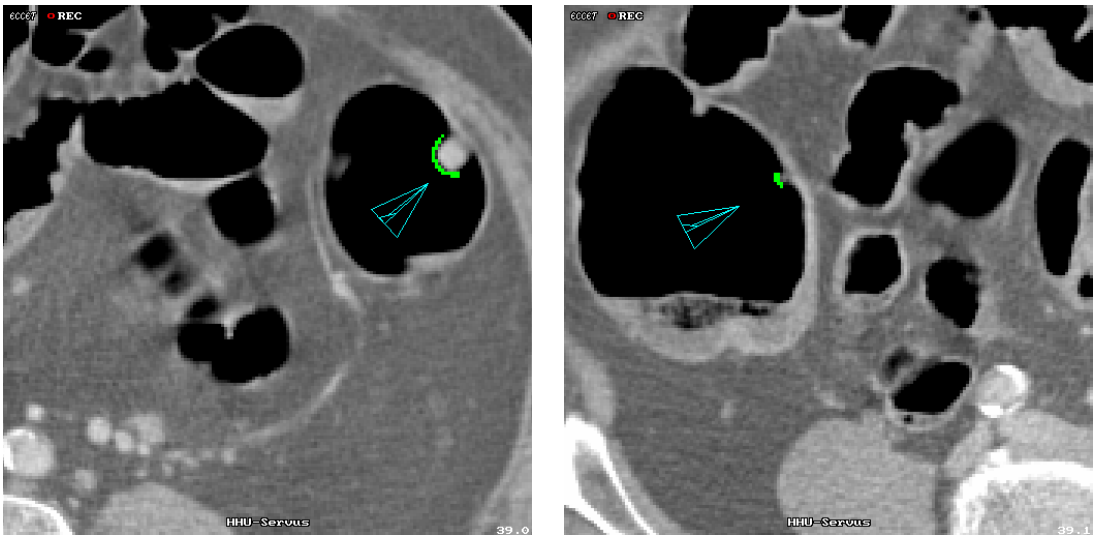
7.5 Markierung des Abstandes „ab ano“

Zum Vergleich von virtueller und realer Koloskopie ist es nützlich, die gleichen Ortsangaben wie bei der Endoskopie zu verwenden. Dies sind zum einen anatomische Angaben wie „in der rechten Flexur“ und zum anderen ist es die Angabe des Abstandes „ab ano“. Letzterer gibt an, wie weit das Endoskop in den Darm vorgedrungen ist, analog zur Positionsmessung mit dem realen Endoskop. Wird aufgrund einer Beobachtung in der virtuellen Koloskopie eine reale Koloskopie erforderlich, ist diese Positionsangabe eine sehr nützliche Information für den Gastrologen.

Es sei hier darauf hingewiesen, dass solche Angaben natürlich aufgrund des nicht genau bekannten Verlaufs des Endoskopweges im Darm (mehr oder weniger gewunden etc.) nur eine Groborientierung geben.

Die Erzeugung einer solchen Skala ist eine nützliche Anwendung des Autopiloten (s. auch 8.1). Da dieser in der Lage ist, einen etwa mittig verlaufenden Weg durch den Darm zu finden, kann er den Weg des Endoskopes so annähernd nachverfolgen.

Durch eine Option wird der Autopilot veranlasst, dabei seine Flugstrecke schrittweise genau wie in 7.4 zu berechnen und aufzuaddieren. Immer wenn dabei ein Vielfaches von 10 cm überschritten wird, wird das Volumen mit einer Markierung versehen, die mit dem aktuellen Abstand „ab ano“ beschriftet wird. Die reale Skalierung wird genau wie in 7.4 aus den entsprechenden Informationen in den DICOM-Dateien entnommen.



a) großer Polyp aus Abbildung 7.2 b) kleiner Polyp aus Abbildung 7.3

Abbildung 7.8: 2D-Suchlauf

Eine Möglichkeit, die aktuelle Kameraposition zu bestimmen, ist nun, aus der Position der nächstgelegenen Marker (die in der Regel zu sehen sind) die eigene Position zu schätzen.

Alternativ kann die Möglichkeit von VOXREN genutzt werden, mehrere Kameras gleichzeitig zu verwenden. Man öffnet einfach eine weitere Ansicht und begibt sich außerhalb des Darmes. Normalerweise werden die Marker nun durch den Darm verdeckt; sie können aber durch Knopfdruck wieder eingeblendet werden (s. Abbildung 7.7b). Da alle Kameras ebenfalls Marker tragen, lässt sich nun die Position der Kamera von außen leicht bestimmen.

7.6 2D-Suchlauf

Mit dem Polypenfinder steht ein leistungsfähiger Algorithmus zur Verfügung, der dem Arzt die Sichtung des umfangreichen Datenmaterials erleichtern kann.

Es ist nun natürlich nicht in dessem Sinne, dass er dazu den gesamten Darm in 3D durchfliegen muss, um nun diese bereits markierten Stellen zu suchen. Insbesondere würde diese Methode ja wieder die Gefahr bergen, einzelne Stellen zu übersehen (vgl. 6.7.3).

Daher wurde die Möglichkeit geschaffen, Voxelklassen aus VOXREN als Overlays in PLANEVIEW zu übernehmen (s. auch 4.1.4).

Nun könnte man schrittweise die 2D-Schichten durchsehen, was der herkömmlichen

Vorgehensweise entspricht, die aber bei 600 Schichten doch recht ermüdend ist.

Daher kann mit einem Tastendruck ein Suchlauf gestartet werden, der in der nächsten Ebene, in der eine Markierung gefunden wird, anhält.

Damit wird die Suche beschleunigt, und die Gefahr, eine kleine Markierung zu übersehen, schwindet.

Die Achse, in der die Suchrichtung verläuft (x, y, oder z), kann vom Benutzer gewählt werden.

Ist eine Ebene gefunden, so sieht man an den eingeblendeten Markierungen, warum auf dieser Ebene angehalten wurde bzw. welche Bereiche zu betrachten sind. Ist eine Markierung einmal schlecht zu sehen, kann per Tastendruck die Darstellung auf „nur Markierungen“ geändert werden. In diesem Modus sieht man die Markierung dann sofort und kann anschließend die Grauwertdarstellung wieder zuschalten.

Kapitel 8

Navigationsalgorithmen

8.1 Autopilot

Die Navigation in Hohlorganen kann für einen menschlichen Bediener sehr ermüdend und fehleranfällig sein, besonders wenn wiederholt starke Krümmungen und Engstellen auftreten. Außerdem lenkt diese Tätigkeit von der eigentlichen Aufgabe der Befundung ab.

Um den Bediener zu entlasten, implementiert VOXREN daher einen Autopiloten, der röhrenförmige Hohlorgane automatisch durchfliegt.

8.1.1 Voraussetzung

Gegeben sei eine Voxelmenge O, die ein röhrenförmiges Hohlorgan darstellt, und eine virtuelle Kamera, von der Position, Blickrichtung und Orientierung (bzgl. Rollbewegungen) bekannt sind.

8.1.2 Problemstellung

Finde einen Weg durch das von O dargestellte Hohlorgan, wobei eine möglichst zentrierte Flugbahn anzustreben ist und die Blickrichtung immer möglichst längs der aktuellen Röhrenachse liegen soll.

Optional können noch weitere Restriktionen gemacht werden, um auch die Orientierung festzulegen.

Der Autopilot soll ausgehend von einem manuell relativ beliebig vorgegebenen Startpunkt im Hohlorgan einen solchen Flug starten können.

8.1.3 Lösungsidee

Der oft verwendete Ansatz, zunächst nach einer Art „Zentrale“ im Hohlorgan zu suchen und dieser dann zu folgen, wurde aus zwei Gründen nicht verfolgt:

1. Eine solche Zentrale ist nicht immer leicht zu finden, insbesondere wenn es sich um ein verzweigtes Röhrensystem handelt.
2. Dieser Ansatz widerspricht der Forderung an einem beliebigen Startpunkt beginnen zu können.

Stattdessen wurde der Autopilot als diskretes dynamisches System implementiert.

Die Dynamik dieses Systems beschreibt, wie der Zustand (im konkreten Fall die Position, Blickrichtung und Orientierung der Kamera) zu einem Zeitpunkt t+1 aus den entsprechenden Daten zum Zeitpunkt t hervorgeht.

Die Modellierung der Dynamik wurde angelehnt an das Verhalten eines menschlichen Piloten, dem die Aufgabe des Durchfliegens des Hohlorgans gestellt wurde. Der Mensch benutzt hierzu seine räumliche Vorstellung von der Szene, insbesondere die Tiefeninformation.

Diese steht aufgrund des verwendeten Renderingsystems (s. 4.2.1) auch einem automatischen Piloten zur Verfügung.

8.1.4 Algorithmus

Im gegebenen Fall separiert die Dynamik des Systems recht gut in die Dynamik von Blickrichtung, Position und Orientierung.

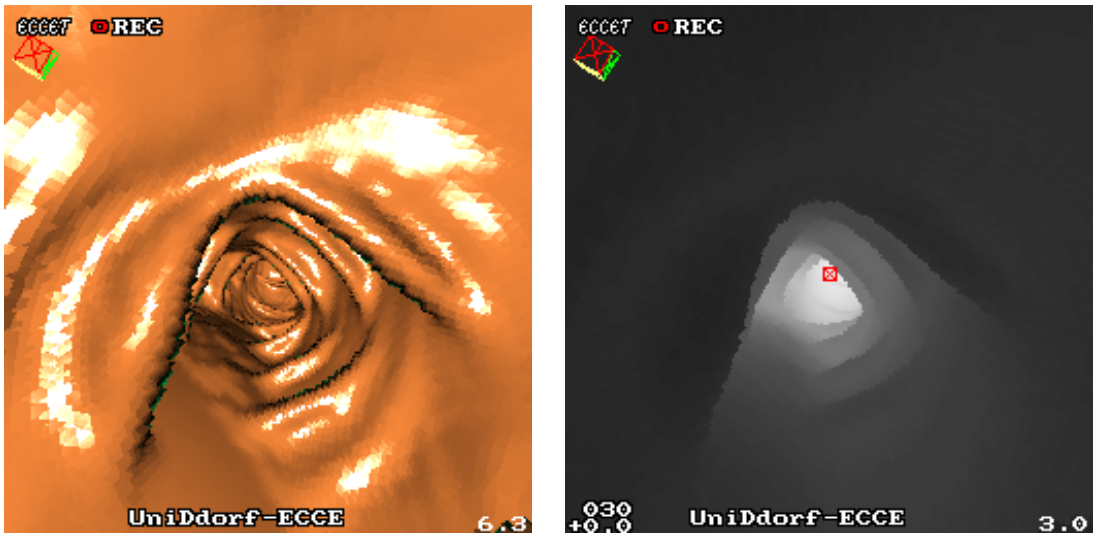
Wir werden diese daher zunächst getrennt betrachten.

Wahl der neuen Blickrichtung

Mit der Kamera im Zustand $Z(t)$ wird nun zunächst mit Hilfe des Renderers ein Bild berechnet.

In diesem Bild sucht der Algorithmus nun den am weitesten entfernten Punkt im Bild (s. Markierung in Abbildung 8.1b).

Da die Kameraparameter (Öffnungswinkel) bekannt sind, könnte dieser Punkt nun direkt in die Bildmitte gebracht werden. Das würde allerdings sehr ruckelnde Bewegungen erzeugen, die nicht natürlich wirken. Daher wird nicht direkt die Richtung



a) Gerenderte Ansicht b) Tiefenfeld mit markiertem tiefsten Punkt

Abbildung 8.1: Der „Tiefenblick“ des Autopiloten

neu gesetzt, sondern durch kleine Drehungen in Richtung des gefundenen Punktes korrigiert. Die Entfernung des gefundenen Punktes vom Bildmittelpunkt geht dabei linear ein, so dass starke Abweichungen schnell korrigiert werden, aber ein Überschwingen vermieden wird.

Des Weiteren wird das Ergebnis mit einem gleitenden exponentiellen Mittelwertfilter noch tiefpassgefiltert. Das vermeidet einerseits allzu starkes Rucken, wenn der am weitesten entfernte Punkt einmal springt (z.B. weil ein neues Segment sichtbar wird) und unterdrückt ebenfalls kleine Zitterbewegungen um die Mitte. Zusätzlich besitzt die Mittelpunktsuche einen kleinen Totbereich, was ebenfalls der Vermeidung von Zittern dient.

Seitliches Ausweichen

Besonders in den stark gefalteten Dickdärmen bestünde nun aber noch die Gefahr einer Kollision mit den Wänden bzw. der Autopilot könnte ggf. einfach dicht an den Wänden entlangfliegen, was aufgrund des ungünstigen unsymmetrischen Blickes nicht erwünscht ist.

Daher wird zusätzlich die Tiefeninformation am Bildrand ausgewertet. Jeweils am rechten und linken sowie am oberen und unteren Bildrand wird verglichen, wie nahe alle dort befindlichen Voxel zur Kamera sind. Befinden sich z.B. rechts viele nahe Voxel und links eher weit entfernte, so erfolgt ein Ausweichen nach links.

a) Vom Autopiloten gewählter Pfad in einem Dickdarm mit einer starken Stenose

b) Pfad durch ein großes Gefäß. Die Richtung wurde an der Abzweigung manuell beeinflusst, um das untere Gefäß zu wählen.

Abbildung 8.2: Vom Autopiloten durchflogener Pfad

Technisch gesehen wird für jeden Rand die Summe

$$\sum_{\text{Rand}} \frac{1}{1 + \text{Tiefe}} \tag{8.1}$$

gebildet. Diese Summe wird groß, wenn sich Objekte nahe an der Kamera befinden.

Die Summen zweier jeweils gegenüberliegender Ränder werden voneinander subtrahiert und auf die Anzahl der beteiligten Pixel pro Rand normiert.

Nun erfolgt ein Ausweichen in Richtung des Randes mit der kleineren Summe, wobei die normierte Differenz linear in die Schrittweite eingeht.

Um ein unnötiges Hin- und Herzittern zu vermeiden, wird ein Totbereich verwendet, der die Seitwärtsbewegung bei geringen Unterschieden unterdrückt. Sehr große Unterschiede werden begrenzt, um ein plötzliches Hüpfen zu vermeiden.

Vorschub

Normalerweise geht der Autopilot mit einer einstellbaren Schrittweite in jedem Zeitschritt vorwärts. Alle Bewegungen werden mit dieser Schrittweite skaliert, um unabhängig von dieser ein ähnliches dynamisches Verhalten zu erreichen.

An Engstellen nähern sich von allen Seiten Objekte. Die eben besprochene Logik sorgt immer noch dafür, dass der Autopilot zentrisch auf ein in der Mitte vorhandenes Loch zufliegt. Allerdings werden die Summen nach Gleichung 8.1 in dieser

Situation allseitig sehr groß. Kleine Änderungen bewirken nun schon große Änderungen in der Differenz von je zwei Summen, wodurch starke seitliche Ausweichmanöver ausgelöst werden. Will man ein kleines Loch durchfliegen, so ist dies kein wünschenswertes Verhalten.

Engstellen lassen sich jedoch gerade daran erkennen, dass die einander zugehörigen (rechten/linken bzw. oberen/unteren) Summen beide sehr groß werden. Daher wird die Schrittweite an solchen Stellen automatisch verringert.

Behandlung von Sackgassen

Biegt der Autopilot einmal falsch ab oder ist das Ende der Röhre erreicht, so soll er sich möglichst selbst wieder einen Ausweg suchen.

Dazu wird bei der Suche nach dem tiefsten Punkt auch dessen Tiefe bestimmt. Liegt diese nur noch bei wenigen Voxeln, so nimmt der Autopilot an, in eine Sackgasse geraten zu sein, und leitet eine Rechtsdrehung ein, um einen Ausweg zu finden.

Rollen/Gravitationssimulation

Die bisherigen Algorithmen wirken jeweils nur auf Position und Blickrichtung der Kamera. Die Orientierung des Bildes ist noch undefiniert. Für viele Fälle ist die Orientierung aber auch nicht relevant.

Optional kann der Autopilot angewiesen werden, die Lage der aktuellen Orientierung relativ zur physischen Richtung „unten“ bei der Aufnahme auszuwerten. Dazu wird das Skalarprodukt des von der Kamera — bezüglich des von ihr gemachten Bildes — nach rechts weisenden Vektors und des Gravitationsvektors gebildet. Das Ergebnis wird geeignet skaliert und in eine Rollbewegung umgesetzt. Dadurch wird erreicht, dass sich die Kamera immer mit ihrer Unterseite in Richtung der physischen Richtung „unten“ befindet (das Skalarprodukt wird dann gerade 0).

Wenngleich daraus gelegentlich etwas kompliziert anmutende Flugmanöver resultieren (z.B. ein Immelmann-Turn bei u-förmigen Biegungen übereinanderliegender Strukturen), hat es den Vorteil, dass man gravitationsbedingte Artefakte wie Restflüssigkeit (die in dieser Ansicht als glatte „Seen“ im unteren Bildbereich erkenntlich werden) und Reststuhl (der sich auch bevorzugt unten befindet) leichter als solche erkennen kann, da sie sich an der erwarteten Position (unten) befinden.

8.1.5 Bemerkungen

Dieser Algorithmus arbeitet sehr robust und findet in nahezu allen untersuchten Därmen, deren Volumina zusammenhängend sind, einen Weg von einem Ende zum anderen.

In den wenigen Fällen, in denen der Algorithmus versagt, hat in der Regel auch der menschliche Beobachter erhebliche Schwierigkeiten bei der Wegfindung. Dies tritt z.B. auf bei sehr verwundenen, steil abknickenden Röhren, bei denen auch für den menschlichen Piloten aus dem Bild nicht mehr abzulesen ist, wo es weitergeht. Ein typisches Beispiel sind enge 180°-Kehren, bei denen der tiefste Punkt des Bildes der Scheitel der Kehre ist. Der Autopilot fliegt darauf zu und wird dann schließlich die Sackgassenbehandlung einleiten und dann in eine der beiden Röhren zurückfliegen.

An solchen Stellen kann man sich aber leicht mit einer Außenansicht behelfen, dort den weiteren Verlauf lokalisieren und den Autopiloten per manuellem Eingriff wieder auf den richtigen Weg bringen.

8.1.6 Vergleich mit anderen Verfahren

In anderen Softwarepaketen wird oft eine Wegplanung verwendet, d.h. es wird explizit nach einem zentralen Pfad durch den Darm gesucht und die Kamera längs dieses Pfades bewegt. Vergleicht man diese Verfahren mit dem hier gewählten dynamischen System, so ergeben sich folgende

Vor- und Nachteile:

- Das Finden eines Pfades ist nicht gesichert. Man kann explizit Situationen konstruieren, in denen der Autopilot ein erreichbares Ziel nicht erreicht. Z.B. kann man ihn in einer kleinen Kugel plazieren, deren Ausgang oben liegt. Er wird dann aufgrund der Sackgassendetektionsstrategie unbegrenzt nach rechts drehen. Analog kann man steile Kehren konstruieren, die ein Zurückfliegen in eine bestimmte Richtung erzwingen. Diese Fälle sind jedoch in den üblichen Anwendungensgebieten (Darm, große Gefäße) extrem selten.

- Der eingeschlagene Weg ist nicht unbedingt der kürzeste.

+ Der Autopilot kann jederzeit ein- und ausgeschaltet werden, er kann von jedem Punkt innerhalb der Röhre starten.

+ Das Bewegungsverhalten wird vom menschlichen Betrachter als sehr natürlich empfunden.

+ Der Betrachter kann jederzeit (auch bei aktivem Autopiloten) in die Steuerung eingreifen und den Zustand des Autopiloten verändern. Damit ist es möglich, den Autopiloten auf intuitive Art zu unterstützen.

+ Der Autopilot kann jederzeit kurz ausgeschaltet werden, um z.B. eine interessante Struktur näher zu betrachten, und anschließend kann der Autopilot auch aus dem geänderten Zustand heraus seine Arbeit wieder aufnehmen.

+ Es können einzelne Teilsysteme des Autopiloten ein- und ausgeschaltet werden, so dass er einen menschlichen Bediener auch bei komplizierteren Aufgaben, die der Autopilot alleine nicht lösen kann, sinnvoll unterstützen kann. Mehr dazu erfahren Sie im nächsten Abschnitt.

+ Er ist relativ einfach zu programmieren.

Die beiden Negativpunkte erweisen sich übrigens in der Praxis als wenig relevant. Typische Därme werden problemlos durchflogen und der Pfad liegt nahe an einer idealen Mittellinie (s. Abbildung 8.2).

8.1.7 Typische Anwendungen

Der Autopilot wurde für die Koloskopie entwickelt, kann aber durch die individuelle Schaltbarkeit seiner Einzeldynamiken auch für andere Aufgaben verwendet werden.

Bei verzweigten Röhrensystemen (Gefäßen z.B.) trifft der Autopilot, wenn alle Optionen aktiv sind, immer die Entscheidung, in das Gefäß zu fliegen, in das man im Moment weiter hineinsehen kann. Dies ist oft nicht erwünscht. Schaltet man die automatische Richtungsvorgabe aus, kann die Richtung vom Benutzer per Maussteuerung vorgegeben werden. Die weiterhin aktive Kollisionsvermeidung durch seitliches Ausweichen und der automatische Vorschub sorgen dafür, dass der Benutzer sich nur um die Richtung kümmern muss.

Analog wird man bei Anwendungen, in denen die Orientierung nicht relevant ist, auch das automatische Rollen unterdrücken.

Auch die alleinige Nutzung des automatischen Vorschubes kann nützlich sein.

Messung der Flugpfadlänge

Der Autopilot kann genutzt werden, um im Darm die Distanz „ab ano“ zu markieren, siehe dazu den Abschnitt 7.5.

8.2 Pfadgenerierung durch Interpolation

Insbesondere zur Erzeugung von Filmen wird oft ein sowohl glatter (d.h. in allen Kameraparametern stetiger), als auch vom Benutzer vorgebbarer Kameraflugpfad gewünscht.

Der Autopilot erzeugt zwar einen glatten Pfad, jedoch ist er beschränkt auf die durch seine Betriebsarten und die vorgegebene Szene erreichbaren Pfade.

Eine manuelle Vorgabe des Pfades erlaubt dem Benutzer zwar völlige Freiheit bezüglich der Wahl des Pfades, wirkt aber oft sehr ruckelig und ungleichmäßig in der Geschwindigkeit.

Dieses Problem ist im Bereich der 3D-Animation bekannt und wird dort in der Regel mit sog. Keyframes und Interpolation gelöst.

Dabei gibt der Benutzer einzelne Kameraeinstellungen des Filmes vor und der Pfad zwischen diesen Einstellungen wird berechnet.

Es wurde ein Algorithmus implementiert, der diese Pfadgenerierung leistet:

8.2.1 Voraussetzungen

Gegeben sei eine Folge von n Kamerapositionen $\vec{k}_1, \vec{k}_2, ...\vec{k}_n$ und zugehörigen Kameraparametern $p_1, p_2, ...p_n$ sowie eine vorgegebene Schrittzahl $m > n$.

8.2.2 Problem

Finde einen Pfad S und eine Folge $\vec{s}_1, \vec{s}_2, ...\vec{s}_m$ mit $s_i \in S$ sowie neue Kameraparameter $p'_1, p'_2...p'_m$ so, dass

- der Pfad S stetig ist und die Positionen $\vec{k}_1, \vec{k}_2, ...\vec{k}_n$ in dieser Reihenfolge enthält,
- die Kamerageschwindigkeit $v_i := \vec{s}_i - \vec{s}_{i-1}$ betraglich möglichst konstant und ebenfalls stetig ist,

- die Blickrichtung und Orientierung der Kamera sowie sonstige Parameter (z.B. Öffnungswinkel) ebenfalls stetig geändert werden.

8.2.3 Erzeugung von kubischen Splines

Im untenstehenden Algorithmus werden kubische Splines benutzt. Daher werden diese hier kurz anhand einer effizienten Berechnungsmöglichkeit definiert:
Seien $\vec{k}_1, \vec{k}_2, \vec{c}_1, \vec{c}_2$ Elemente eines Vektorraumes und $x \in [0;1]$.
Anschaulich sind $\vec{k}_1$ und $\vec{k}_2$ Anfangs- und Endpunkt des Splines, $\vec{c}_1$ und $\vec{c}_2$ die beiden Kontrollpunkte für die Tangente an Anfangs- und Endpunkt und x der Interpolationsparameter.
Sei weiter die Funktion mix für $x \in [0;1]$ definiert als

$$\text{mix}(\vec{a}, \vec{b}, x) := (1-x)\vec{a} + x\vec{b}\ . \tag{8.2}$$

Dann definiert

$$\vec{S}(\vec{k}_1, \vec{k}_2, \vec{c}_1, \vec{c}_2, x) := \text{mix}\Big\{\begin{array}{rl}\text{mix} & \big[\text{mix}(\vec{k}_1, \vec{c}_1, x), \text{mix}(\vec{c}_1, \vec{c}_2, x), x\big] \quad , \\ \text{mix} & \big[\text{mix}(\vec{c}_1, \vec{c}_2, x), \text{mix}(\vec{c}_2, \vec{k}_2, x), x\big] \quad , x\end{array}\Big\} \tag{8.3}$$

einen kubischen Spline.

Stetigkeit des Splines

Offensichtlich ist $\vec{S}$ längs der Parametrierung x stetig (wegen der Stetigkeit von mix) und es gilt $\vec{S}(\vec{k}_1, \vec{k}_2, \vec{c}_1, \vec{c}_2, 0) = \vec{k}_1$ und $\vec{S}(\vec{k}_1, \vec{k}_2, \vec{c}_1, \vec{c}_2, 1) = \vec{k}_2$.
Dies ist noch leichter zu erkennen, wenn man $\vec{S}$ anders darstellt, indem man die mix-Funktion ausmultipliziert:

$$\begin{aligned}\vec{S}(\vec{k}_1, \vec{k}_2, \vec{c}_1, \vec{c}_2, x) = \\ (1-x)\Big[(1-x)\big((1-x)\vec{k}_1 + x\vec{c}_1\big) + x\big((1-x)\vec{c}_1 + x\vec{c}_2\big)\Big] + \\ x\Big[(1-x)\big((1-x)\vec{c}_1 + x\vec{c}_2\big) + x\big((1-x)\vec{c}_2 + x\vec{k}_2\big)\Big]\end{aligned} \tag{8.4}$$

Dies läßt sich umformen zu

$$\begin{aligned}\vec{S}(\vec{k}_1,\vec{k}_2,\vec{c}_1,\vec{c}_2,x) = (1-x)^2\big((1-x)\vec{k}_1 + x\vec{c}_1\big)+ \\ x(1-x)\big((1-x)\vec{c}_1 + x\vec{c}_2\big)+ \\ x(1-x)\big((1-x)\vec{c}_1 + x\vec{c}_2\big)+ \\ x^2\big((1-x)\vec{c}_2 + x\vec{k}_2\big)\end{aligned} \tag{8.5}$$

$$\begin{aligned}\vec{S}(\vec{k}_1,\vec{k}_2,\vec{c}_1,\vec{c}_2,x) =&(1-x)^3\vec{k}_1 + x(1-x)^2\vec{c}_1+ \\ &x(1-x)^2\vec{c}_1 + x^2(1-x)\vec{c}_2+ \\ &x(1-x)^2\vec{c}_1 + x^2(1-x)\vec{c}_2+ \\ &x^2(1-x)\vec{c}_2 + x^3\vec{k}_2\end{aligned} \tag{8.6}$$

$$\vec{S}(\vec{k}_1,\vec{k}_2,\vec{c}_1,\vec{c}_2,x) = (1-x)^3\vec{k}_1 + 3x(1-x)^2\vec{c}_1 + 3x^2(1-x)\vec{c}_2 + x^3\vec{k}_2 \tag{8.7}$$

Stetigkeit der Ableitung

Da wir beim Zusammensetzen von Splines auch die Stetigkeit der Ableitung fordern möchten, berechnen wir diese:

$$\begin{aligned}\vec{S}'(\vec{k}_1,\vec{k}_2,\vec{c}_1,\vec{c}_2,x) :&= \frac{d}{dx}\vec{S}(\vec{k}_1,\vec{k}_2,\vec{c}_1,\vec{c}_2,x) \\ &= 3(1-x)^2(-1)\vec{k}_1 + 3\big((1-x)^2 + 2x(1-x)(-1)\big)\vec{c}_1+ \\ &\quad 3\big(2x(1-x) + x^2(-1)\big)\vec{c}_2 + 3x^2\vec{k}_2 \\ &= -3(1-x)^2\vec{k}_1 + 3\big(1-2x+x^2-2x+2x^2\big)\vec{c}_1+ \\ &\quad 3\big(2x-2x^2-x^2\big)\vec{c}_2 + 3x^2\vec{k}_2 \\ &= 3\big[-(1-x)^2\vec{k}_1 + (3x^2-4x+1)\vec{c}_1 + (2x-3x^2)\vec{c}_2 + x^2\vec{k}_2\big]\end{aligned} \tag{8.8}$$

Auch diese ist offensichtlich stetig.

Anschlussbedingung

An den Enden des Splines gilt

$$\vec{S}(\vec{k}_1,\vec{k}_2,\vec{c}_1,\vec{c}_2,0) = \vec{k}_1 \tag{8.9}$$

$$\vec{S}(\vec{k}_1, \vec{k}_2, \vec{c}_1, \vec{c}_2, 1) = \vec{k}_2 \tag{8.10}$$

$$\vec{S}'(\vec{k}_1, \vec{k}_2, \vec{c}_1, \vec{c}_2, 0) = 3(\vec{c}_1 - \vec{k}_1) \tag{8.11}$$

$$\vec{S}'(\vec{k}_1, \vec{k}_2, \vec{c}_1, \vec{c}_2, 1) = -3(\vec{c}_2 - \vec{k}_2) \tag{8.12}$$

Möchte man also zwei Splines $\vec{S}_a(\vec{k}_{1a}, \vec{k}_{2a}, \vec{c}_{1a}, \vec{c}_{2a}, x)$ und $\vec{S}_b(\vec{k}_{1b}, \vec{k}_{2b}, \vec{c}_{1b}, \vec{c}_{2b}, x)$ so aneinandersetzen, dass der Übergang stetig differenzierbar erfolgt, so müssen folgende Anschlussbedingungen erfüllt sein:

$$\vec{S}_a(\vec{k}_{1a}, \vec{k}_{2a}, \vec{c}_{1a}, \vec{c}_{2a}, 1) = \vec{k}_{2a} = \vec{k}_{1b} = \vec{S}_b(\vec{k}_{1b}, \vec{k}_{2b}, \vec{c}_{1b}, \vec{c}_{2b}, 0) \tag{8.13}$$

$$\vec{S}'_a(\vec{k}_{1a}, \vec{k}_{2a}, \vec{c}_{1a}, \vec{c}_{2a}, 1) = -3(\vec{c}_{2a} - \vec{k}_{2a}) = 3(\vec{c}_{1b} - \vec{k}_{1b}) = \vec{S}'_b(\vec{k}_{1b}, \vec{k}_{2b}, \vec{c}_{1b}, \vec{c}_{2b}, 0) \tag{8.14}$$

Also verkürzt:

$$\vec{k}_{2a} = \vec{k}_{1b} \tag{8.15}$$

$$\frac{\vec{c}_{2a} + \vec{c}_{1b}}{2} = \vec{k}_{2a} \tag{8.16}$$

D.h., der Endpunkt von $\vec{S}_a$ muss mit dem Startpunkt von $\vec{S}_b$ zusammenfallen, und der Mittelpunkt der Verbindungsstrecke zwischen den beiden Kontrollpunkten $\vec{c}_{2a}$ und $\vec{c}_{1b}$ muss ebenfalls auf diesem Punkt zu liegen kommen.

8.2.4 Algorithmus

Erzeugung der Splines

Aus den gegebenen Stützpunkten $\vec{k}_1, ...\vec{k}_n$ werden zunächst $n - 1$ Splines $\vec{S}_i(\vec{k}_i, \vec{k}_{i+1}, \vec{c}_{1,i}, \vec{c}_{2,i}, x)$ erzeugt, die jeweils zwei aufeinanderfolgende Punkte mit einem glatten Pfad verbinden.

Diese Wahl der Start- und Endpunkte erfüllt automatisch 8.15.

Nun müssen noch $\vec{c}_{2,i}$ und $\vec{c}_{1,i+1}$ gemäß 8.16 gewählt werden, um einen stetig differenzierbaren Gesamtpfad zu erhalten.

Da außer den Stützpunkten keine weiteren Daten vorliegen, müssen diese Kontrollpunkte ohnehin vom Algorithmus sinnvoll gewählt werden. Dabei wird folgende Heuristik verwendet:

Berechne zunächst die Differenzvektoren von einem Stützpunkt zu dessen Nachbarstützpunkten

$$\vec{c}_{2,i-1}{}' := \vec{k}_i - \vec{k}_{i-1} \tag{8.17}$$

$$\vec{c}_{1,i}{}' := \vec{k}_i - \vec{k}_{i+1} \tag{8.18}$$

und stelle anschließend fest, welcher der beiden Abstände kleiner ist.

$$l_{min} := \min(|\vec{c}_{1,i}{}'|, |\vec{c}_{2,i}{}'|) \tag{8.19}$$

Berechne die mittlere Richtung der beiden Vektoren

$$\vec{c}_i{}' := \frac{\frac{\vec{c}_{2,i-1}{}'}{|\vec{c}_{2,i-1}{}'|} - \frac{\vec{c}_{1,i}{}'}{|\vec{c}_{1,i}{}'|}}{2} \tag{8.20}$$

und schließlich die Position der Kontrollpunkte:

$$\vec{c}_{2,i-1} := \vec{k}_i + \frac{l_{min}}{2}\vec{c}_i{}' \tag{8.21}$$

$$\vec{c}_{1,i} := \vec{k}_i - \frac{l_{min}}{2}\vec{c}_i{}' \tag{8.22}$$

Diese Wahl erfüllt per Konstruktion 8.16.

Approximative Bestimmung der Länge der Segmente

Jeder Spline wird nun in q Teilsegmente zerlegt, die für die Zwecke der Längenabschätzung näherungsweise als Geradenstücke betrachtet werden.
Dazu berechnet man $q+1$ Stützpunkte für jeden Spline S_i

$$\vec{S}_{i,j} := \vec{S}_i(\vec{k}_i, \vec{k_{i+1}}, \vec{c_{1,i}}, \vec{c_{2,i}}, \frac{j}{o}), \text{ mit } j \in \{0, 1, ...q\} \ , \tag{8.23}$$

die diesen Spline bezüglich des Spline-Parameters x gleichmäßig in q Teilsegmente unterteilen. Nun bestimmt man die Länge dieser Teilsegmente, indem man als Näherung den euklidischen Abstand der beiden das Teilsegment begrenzenden Stützpunkte berechnet.

$$l'_{i,j} := |\vec{S}_{i,j} - \vec{S}_{i,j-1}|, \text{ mit } j \in [1, 2, ...o] \tag{8.24}$$

Man summiert die Ergebnisse, um eine Näherung l'_i für den Weg l_i zwischen den beiden Endpunkten des Splines Nummer i zu erhalten:

$$l'_i := \sum_{j=1}^{q} l'_{i,j} \tag{8.25}$$

Daraus erhält man eine Näherung L' für den Gesamtweg L längs $S_1 ... S_{n-1}$,

$$L' = \sum_{i=1}^{n-1} l'_i \tag{8.26}$$

für den Weg zum Anfang jedes Segmentes,

$$L'_i = \sum_{j<i} l'_j \tag{8.27}$$

sowie den Weg zum Anfang eines Teilsegmentes

$$L'_{i,j} = L'_i + \sum_{h<j} l_{i,h} \tag{8.28}$$

Skalierung

Da m neue Kamerastützpunkte bestimmt werden sollen, wird die Kamera in Schritten von $\Delta x := \frac{L'}{m-1}$ bewegt.

Interpolation der Stützpunkte

Berechne für $h \in \{0, 1, ..., m-1\}$ die neuen Stützpunkte k_h wie folgt:
Bestimme zunächst das Segment, in dem der Punkt mit der Entfernung $h \cdot \frac{L'}{m-1}$ vom Startpunkt liegt:

$$i := \min(i : L'_i \leq h \cdot \frac{L'}{m-1}) \tag{8.29}$$

und darin das Teilsegment

$$j := \min(j : L'_{i,j} \leq h \cdot \frac{L'}{m-1}) \tag{8.30}$$

Damit ist das Teilsegment i, j bestimmt, in dem der aktuelle Punkt liegt.

Innerhalb dieses Teilsegmentes wird nun per linearer Interpolation der Interpolationsparameter $x(h)$ für den Spline bestimmt:

$$d_1(h) := h \cdot \frac{L'}{m-1} - L'_{i,j} \tag{8.31}$$

$$d_2(h) := L'_{i,j+1} - h \cdot \frac{L'}{m-1} \tag{8.32}$$

$$x(h) := \frac{d_2(h)\frac{j}{q} + d_1(h)\frac{j+1}{q}}{d_1(h) + d_2(h)} \tag{8.33}$$

Damit kann der neue Stützpunkt berechnet werden. Er ist

$$\vec{s}_h := \vec{S}_i(\vec{k}_i, \vec{k}_{i+1}, \vec{c}_{1,i}, \vec{c}_{2,i}, x(h)) \ . \tag{8.34}$$

Interpolation der skalaren Kameraparameter

Die skalaren Kameraparameter (Öffnungswinkel, Schnittebenen) werden auf die gleiche Weise linear interpoliert. Allerdings benötigt man hier die feinere Unterteilung j nicht, sondern bestimmt lediglich das Segment, in dem der neue Stützpunkt sich befindet:

$$i := \min(i : L'_i \leq h \cdot \frac{L'}{m-1}) \tag{8.35}$$

Man bestimmt die Mischparameter:

$$d_3(h) := h \cdot \frac{L'}{m-1} - L'_i \tag{8.36}$$

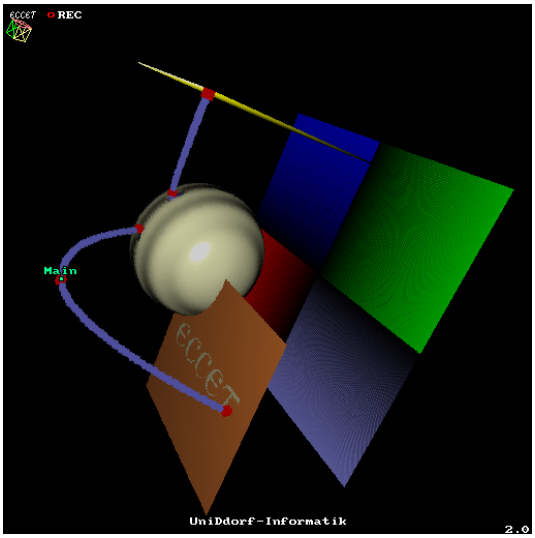
$$d_4(h) := L'_{i+1} - h \cdot \frac{L'}{m-1} \tag{8.37}$$

$$p'_h := \frac{d_4(h)p_i + d_3(h)p_{i+1}}{d_3(h) + d_4(h)} \tag{8.38}$$

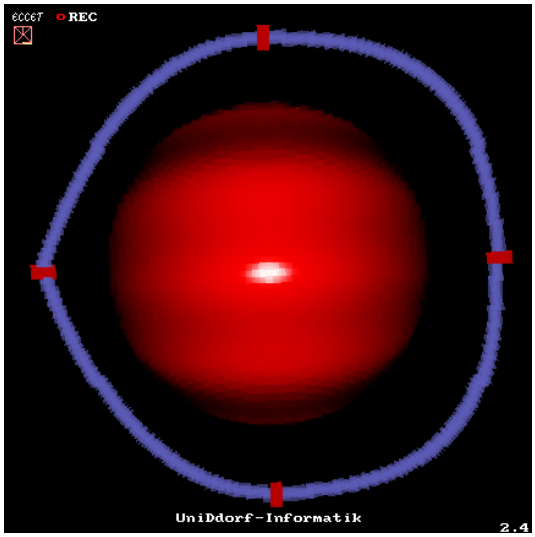
Interpolation der vektoriellen Kameraparameter

Blickrichtung und Orientierung der Kamera sind in Form von zwei Vektoren gegeben. Die Orientierung wird durch einen senkrecht auf der Blickrichtung stehenden Vektor gegeben, der im Kamerabild nach rechts weist.

Diese werden ebenfalls schlicht linear interpoliert und anschließend erneut normiert (s. nächster Abschnitt). Dies ist nicht ganz korrekt, da an sich eine Interpolation auf der Oberfläche einer Einheitskugel verwendet werden müsste, für die Praxis aber völlig ausreichend.



a) Pfad in einer künstlichen Szene. Die Stützpunkte des Pfades (rot) wurden an den kritischen Stellen (Ein-/Austritt aus der Kugel) gewählt.



b) Pfad mit 5 Stützpunkten (Anfang und Ende sind identisch) um eine Kugel. Der linke Punkt ist Start- und Endpunkt. Dort ergibt sich eine Asymmetrie, da dort kein Vorgänger/Nachfolger vorhanden ist.

Abbildung 8.3: Durch Interpolation gewonnener Pfad

Neunormierung des Kameradreibeins

Die Lage der Kamera wird definiert durch ihre Position und das Dreibein, das ihre Blickrichtung und Orientierung angibt.

Durch die lineare Interpolation wird dieses Dreibein verquetscht. Man richtet es mit Hilfe des Gram-Schmidt-Verfahrens wieder zu einem orthonormalen Dreibein auf.

Man beginnt mit der Blickrichtung und normiert sie auf Länge 1. Dann subtrahiert man die Projektion des nach rechts weisenden Vektors auf die Blickrichtung vom nach rechts weisenden Vektor und normiert das Ergebnis auf 1.

Der dritte Vektor des Dreibeins wird nun einfach per Vektorprodukt berechnet. Das stellt sicher, dass die „Händigkeit“ des Systems nicht wechseln kann.

8.2.5 Bemerkungen

Anschaulich lässt sich der Algorithmus wie folgt formulieren:

- Generiere $n-1$ Splines, die den geforderten glatten Pfad zwischen $\vec{k}_1, \vec{k}_2, ...\vec{k}_n$ erzeugen:
  - Wähle dazu jeweils die $\vec{k}_i$ als Start- und Endpunkte, so dass jeweils der

Endpunkt des Vorgängers gleich dem Startpunkt des darauf folgenden Splines ist.

- Wähle die Kontrollpunkte $c_{2,i}$ und $c_{1,i+1}$ so, dass deren Differenz zum gemeinsamen Start-/Endpunkt k_{i+1} gerade entgegengesetzt gleich ist und die Länge dieser beiden Differenzvektoren durch die kürzere der anliegenden Strecken bestimmt wird[1].
- Wähle die Lage der beiden Kontrollpunkte zusätzlich so, dass die Richtung des Differenzvektors zwischen Kontrollpunkt und zugehörigem Stützpunkt gerade mittig zwischen den Richtungen der beiden anliegenden Verbindungsstrecken zu den benachbarten Stützpunkten liegt.

- Bestimme die Länge der einzelnen Splines durch Aufteilen in q Teilabschnitte, die näherungsweise als Geradenstücke aufgefasst werden.
- Berechne die Gesamtlänge der Splinefolge und daraus die für m Stützpunkte erforderliche Schrittweite.
- Damit können die Stützpunkte $\vec{s}_h$ berechnet werden, wobei die Splineparameter $x(h)$ aus den oben näherungsweise bestimmten Längen berechnet werden.
- Interpoliere die sonstigen Kameraparamter entsprechend durch lineare Interpolation zwischen den beiden daneben liegenden Zuständen der Kamera.

[1]Das trägt dem Wunsch Rechnung, durch dichteres Setzen von Stützpunkten den Weg der Kamera genauer kontrollieren zu können. Durch obige Wahl sorgen dicht liegende Stützpunkte für nahe an den Stützpunkten liegende Kontrollpunkte, wodurch die Neigung zum Überschwingen unterdrückt wird.

Kapitel 9

Anwendungen

9.1 3D-Scanner zur Erzeugung von Reliefs

Zu Testzwecken wurde auch ein einfaches System zur Gewinnung von Reliefdaten entwickelt, das mit minimalem Hardwareaufwand in der Lage ist, dreidimensionale Scans von Objekten herzustellen.

Dazu wird lediglich eine an den Computer angeschlossene Kamera, ein computergesteuerter Projektor (Beamer) sowie eine Wand und/oder ein großes Blatt Papier benötigt.

Abbildung 9.1 zeigt den notwendigen Aufbau schematisch.

Die Erfassung des Reliefs erfolgt durch Triangulation seiner Oberflächenpunkte mit Hilfe von strukturiertem Licht.

Aufbau und Kalibrierung

Der Aufbau und die Kalibrierung des Systems geschieht wie folgt:

- Eine Kamera wird vor einer Wand so aufgebaut, dass sie senkrecht auf die Wand blickt.

- Seitlich gegen die Kamera versetzt wird der Projektor aufgebaut.
  Sein ausgeleuchteter Bereich wird so bestimmt, dass der von der Kamera erfasste Blickwinkel sowohl auf der Wand als auch auf der Kalibrierebene vollständig vom Projektor ausgeleuchtet wird.
  Dazu ist es nützlich, ein großes Blatt Papier in die Kalibrierebene zu halten.

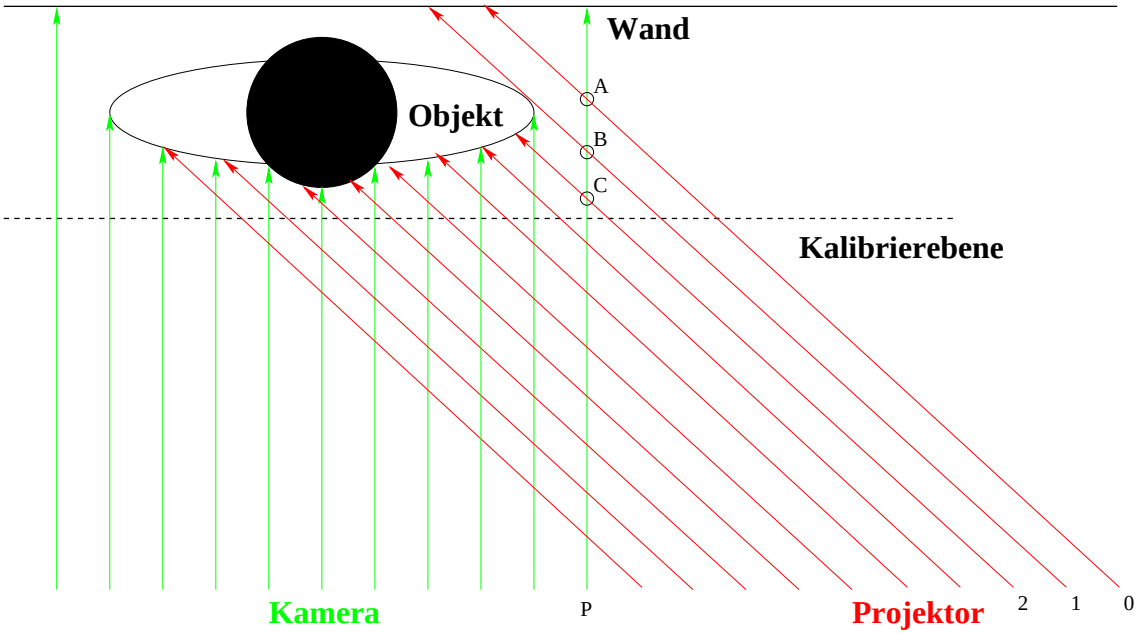


Abbildung 9.1: Einfache Erfassung von Reliefs mit Textur mit 3DSCAN

- Nun erfolgt die erste Kalibriermessung. Alle Objekte zwischen Kamera und Wand werden entfernt, und `scanit >bg.16hl` gestartet (mehr zu dessen Funktionsweise später). In diesem Durchgang wird die Wand als Nullreferenz trianguliert.

- Für die zweite Kalibriermessung wird ein Blatt Papier in der Kalibrierebene angebracht und `scanit >fg.16hl` gestartet. Dabei wird dieses Blatt als Vordergrundreferenz erfasst.

Der Kalibrieraufwand ist also minimal und kann in wenigen Minuten durchgeführt werden.

Erfassung von Reliefs

Dazu wird das Objekt zwischen Kalibrierebene und Wand eingebracht und `scanit object.ppm >object.16hl` gestartet.

Aus dieser Datei und den Kalibrierdatensätzen wird dann mit Hilfe von `reconst bg.16hl fg.16hl object.16hl object.ppm` *scale* `>out.3d32` ein Volumen für den RGB-Modus (s. Abb. 4.12) von VOXREN erzeugt.

Funktionsweise von 3DSCAN

3DSCAN erstellt eine Triangulation des aktuell von der Kamera gesehenen Objektes mit Hilfe strukturierten Lichtes, das zur Laufzeit von 3DSCAN selbst erzeugt und über einen Videoprojektor (Beamer) ausgegeben wird. (Hinweis: Dazu sollte die verwendete Auflösung des Ausgabesystems an die des Beamers angepasst sein, da sonst unerwünschte Interpolationseffekte auftreten können.)

Dazu wird zunächst ein Kalibrierrahmen ausgegeben, mit dessen Hilfe das zu erfassende Objekt positioniert werden kann. Der Projektor wird veranlasst, ein weißes Zielkreuz mit Rahmen auf die Wand zu projizieren, innerhalb dessen sich das Objekt befinden muss, und in der Mitte wird zusätzlich das aktuelle Kamerabild anzeigt, auf dem das aufzunehmende Objekt ebenfalls komplett erfasst sein muss.

Anschließend genügt ein Tastendruck, um die Aufnahme zu starten.

Dazu wird zunächst je ein Bild mit vollkommen schwarzem und vollkommen weißem Projektorbild erstellt und beide werden als Referenzbilder gespeichert.

Anschließend wird über den Projektor eine Graycode-Sequenz ausgegeben, bei der jedes Pixel seine eigene Koordinate als zeitliche Bitfolge ausgibt.

Jedes Bit wird jeweils einmal normal und einmal invertiert ausgegeben und dabei jeweils das aktuelle Bild erfasst.

Durch Vergleich mit den beiden Referenzbildern wird nun für jedes Bildpixel entschieden, ob es aktuell vom Projektor beleuchtet wird oder nicht.

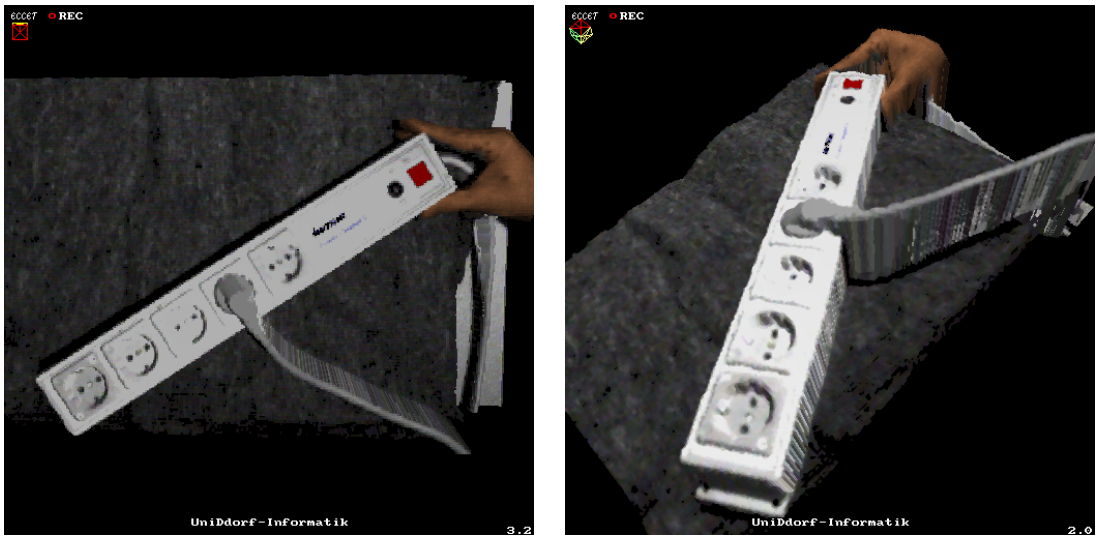
Durch die doppelte Ausgabe (normal und invertiert) je eines Codebits kann diese Entscheidung zweimal im Vergleich mit den beiden Referenzbildern und noch einmal zusätzlich im gegenseitigen Vergleich getroffen werden und somit auch eine Aussage über die Sicherheit der korrekten Messung eines Bits gemacht werden.

Sind alle Messungen übereinstimmend, so wird das Bit als korrekt angenommen. Differieren sie signifikant, so wird das Bit als unsicher markiert.

Durch den Graycode werden große Abweichungen durch 1-Bit Fehler vermieden, da einzelne falsch gemessene Bits immer nur eine Verschiebung des Messergebnisses um 1 bedeuten. Herkömmlicher Binärcode würde hingegen große Sprünge je nach Bitposition möglich machen (z.B. würde ein falsch gemessenes Bit 8 die Koordinate um 256 falsch bestimmen).

Am Ende der Messung wird bei voller Ausleuchtung durch den Projektor noch ein Kamerabild abgespeichert, das später zur Texturierung benutzt wird.

Anschließend wird der für jedes Kamerapixel abgespeicherte Graycode dekodiert.



a) Frontalansicht - Oberflächentextur wurde aus einer Kameraaufnahme gewonnen

b) Profil zur Illustration der gemessenen Tiefenkarte. Man beachte, dass es sich um ein Relief handelt.

Abbildung 9.2: Mit 3DSCAN erzeugtes Volumen

Dabei wird gleichzeitig die Anzahl der als unsicher markierten Bits ausgewertet. Ist diese höchstens 1, so ist damit zu rechnen, dass der Messfehler maximal ein (Projektor-)Pixel beträgt. Fehler in den unteren 4 Bit werden dabei nicht berücksichtigt — man toleriert also eine Abweichung um bis zu 16 Pixel. Dies ist sinnvoll, da der Projektor i.A. eine erheblich bessere Ortsauflösung (typ. 1024x768 Pixel) erreicht als die Kamera (typ. 320x240). Daher fallen ca. 2-3 Bit bereits in den Subpixelbereich der Kamera, der nicht mehr aufgelöst werden kann.

Pixel, die diesen Kriterien nicht genügen, werden speziell markiert, bevor sie abgespeichert werden. Sie werden später durch Interpolation ersetzt.

Funktionsweise von `reconst`

`reconst` lädt zunächst die beiden Kalibrierbilder fg.16hl und bg.16hl.

Diese enthalten jeweils eine Zuordnung von Kamerapixeln zu Projektorspalten für die Wand bzw. die Kalibrierebene als „Objekt“.

Unter der Annahme von Parallelprojektionen für Kamera und Projektor bestünde für die Entfernung d eines Punktes von der Kamera eine lineare Abhängigkeit zur Projektorspalte bei gegebenem Pixel, wie in Abbildung 9.1 für das Pixel P, exemplarisch skizziert:

Befindet sich ein Objekt an Position A, so wird es von Projektorspalte 0 beleuchtet,

und die Entfernung von der Kamera d ist gleich der Strecke AP. Wandert das Objekt nach vorne zu Punkt B, so wird es von Spalte 1 beleuchtet, befindet es sich bei C, von Spalte 2 etc.
Die Strecken AB, BC etc. sind jeweils gleich lang und ihre Länge hängt von der Verkippung von Projektor- und Kameraachse sowie der Breite eines Projektorpixels ab. Real gilt diese einfache Rechnung natürlich nicht, da sowohl Projektor als auch Kamera eher eine Zentralprojektion erzeugen, der Unterscheid ist bei geringen Öffnungswinkeln aber tolerabel.

Nun wird per linearer Interpolation zwischen den Messwerten der Kalibrierbilder jedem Punkt des Messbildes `object.16hl` eine Position zwischen 0.0 und 1.0 zugeordnet. Mit einem Medianfilter werden Ausreißer (und damit auch die als Messfehler markierten Punkte) entfernt, und das Ergebnis wird mit dem Parameter *scale* skaliert.

Für geometrisch korrekte Tiefenverhältnisse sollte dieser Parameter in etwa dem Abstand Wand zu Kalibrierebene, geteilt durch die Breite eines Kamerapixels in Projektion auf die Wand, betragen. Am einfachsten misst man zur Bestimmung dieses Wertes die Breite des Erfassungsbereiches der Kamera auf der Wand und teilt durch die Zahl der Kamerapixel. Wählt man einen größeren oder kleineren Wert, so kann eine Überhöhung oder Abflachung des Tiefeneffektes erreicht werden.

Das Ergebnisbild wird im 3D32-Format für VOXREN aufbereitet und ausgegeben.

9.2 Anwendung in der Medizin

9.2.1 Relevanz

Aufgrund zunehmend besserer Qualität und geringerer Strahlenbelastung wächst die Zahl der Einsatzmöglichkeiten für dreidimensionale Bildgebungsverfahren in der klinischen Routine immer mehr an.

Für einen praktischen Einsatz wird es allerdings auch zunehmend relevant, die immer größeren Datenmengen effizient sichten zu können. Eine klassische Sichtung aller entstehenden Einzelschichten wird mehr und mehr schon vom Zeitaufwand her unmöglich. Des Weiteren erlaubt die verbesserte Bildqualität auch zunehmend die Untersuchung von Fragestellungen, die sich in Einzelbildern nicht mehr in allen Fällen adäquat analysieren lassen. $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ strebt durch seine segmentierungsorientierte Zielsetzung gekoppelt mit Diagnosehilfen eine Lösung dieser Problematik an.

$\epsilon CCE\mathcal{T}$ sieht seine Aufgabe darin, den beurteilenden Mediziner mit Hinweisen auf mögliche Problemstellen und einer leicht verstehbaren Visualisierung zu unterstützen, ohne ihm zeitraubende Aufgaben aufzuerlegen.

So können z.B. bei der virtuellen Koloskopie als polypenverdächtig markierte Areale in die 2D-Darstellung der Schichten eingeblendet werden und automatisch der Reihe nach zur Anzeige gebracht werden. Das System lenkt die Aufmerksamkeit des Arztes gezielt auf die von ihm bereits gefundenen möglichen Polypen und vermeidet so ein eventuelles Übersehen.

Ein anderes Beispiel stellt die präoperative dreidimensionale Darstellung der Leber dar, die zur Zeit in Zusammenarbeit mit der Universitätsklinik Düsseldorf erforscht wird (s. auch Abschnitt 9.3.2). Dort soll dem Arzt eine plastische Vorstellung der relativen Lage aller wichtigen Strukturen gegeben werden. Der Arzt wird von der schwierigen Aufgabe entlastet, eine solche Vorstellung anhand der Schichtbilder selbst im Kopf zu entwickeln, und er kann sich auf andere Probleme (Blutversorgung verschiedener Areale, optimale Eintrittspositionen für operative Werkzeuge, Lage zu konservierender Strukturen, physiologische Besonderheiten des einzelnen Patienten) konzentrieren.

9.2.2 Einsatzorte

$\epsilon CCE\mathcal{T}$ wird praktisch eingesetzt

- an der Universitätsklinik Düsseldorf in der Radiologie und Gastroenterologie in enger Zusammenarbeit mit dem Institut für Informatik. Untersucht werden die Einsatzmöglichkeiten virtueller Koloskopie sowie neuerdings auch die Anwendungsmöglichkeiten zur präoperativen Visualisierung der Leber mit zugehörigen Gefäßsystemen und ggf. vorhandenen Pathologien.
- im MEG-Labor der Neurologie der Universitätsklinik Düsseldorf zur Visualisierung von funktionalen MEG-Daten.
- an der Neurologischen Universitätsklinik der Universität Essen in der Forschungsgruppe um Frau Prof. Dr. med. Dagmar Timmann-Braun zur Visualisierung und Segmentierung des Kleinhirns ausgehend von Magnetresonanztomographiedaten.
- am Klinikum Wuppertal in der Klinik für Radiologie (Prof. Dr. Bernhard Cramer).

9.2.3 Resonanz und Lernkurve

Die Resonanz erwies sich, nachdem einmal ein System installiert war, jeweils als außerordentlich erfreulich.

Als größte Hürde erwies sich zumeist die Anschaffung/Einrichtung einer Linuxworkstation. Deren anschließende Bedienung machte in der Regel wenig Probleme, da moderne graphische Oberflächen wie KDE und Gnome dem Benutzer weitgehend eine von anderen Betriebssystemen gewohnte Bedienung bieten.

Nach kurzer Einweisung und Durcharbeiten des $\epsilon\mathcal{CC}\epsilon\mathcal{T}$-Tutorials (s. [EcMan]) konnten die Anwender zumindest einfache Visualisierungsaufgaben selbständig erledigen.

Bei völlig neuen Segmentierungsproblemen war im Allgemeinen Hilfe durch die Entwicklerseite notwendig, da zur Entwicklung neuer nichttrivialer Segmentierungstechniken ein tiefes Verständnis der enthaltenen Bildverarbeitungsalgorithmen erforderlich ist. War allerdings erst einmal eine grundsätzliche Lösung gefunden, so konnte diese nach kurzer Trainingsphase durchweg von den Anwendern selbst auf verschiedene Fälle übertragen und umgesetzt werden.

Die Zusammenarbeit von Medizinern und Bildverarbeitungsexperten erwies sich dabei als äußerst fruchtbar, da der wechselseitige Zugriff auf das spezifische Fachwissen die Entwicklung komplexer Bildverarbeitungsmechanismen ermöglichte, die die medizinischen Anforderungen an das Ergebnis möglichst gut erfüllen.

9.2.4 Probleme

Ein der Verbreitung hinderliches Problem stellt die Wahl der Plattform Linux/Unix dar. Zumeist ist im medizinischen Umfeld kein entsprechendes System vorhanden und es besteht eine Hemmschwelle, technologisches Neuland zu betreten, für das oft auch kein Inhouse-Support verfügbar ist.

Ist die Entscheidung einmal getroffen, ist die Installation des Betriebssystems dank moderner Distributionen unproblematisch geworden. Es wurden lediglich des öfteren Probleme mit defekten RAM-Bausteinen beobachtet, die von Linux insbesondere beim Betrieb eines ressourcenintensiven Programmes wie $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ nicht toleriert werden. Diese Probleme entstehen natürlich auch auf anderen Plattformen, werden dort aber in der Regel zu anderen allgemeinen Instabilitäten attributiert.

Die Installation von $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ selbst ist durch Bündelung mit der verwendeten Lowlevel-Grafikbibliothek LibGGI [GGI] vollkommen unproblematisch. $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ kann als Demoversion von http://www.cs.uni-duesseldorf.de/~eccet/download.html

bezogen werden.

Die Demoversion hat volle Funktionalität, allerdings wird das Laden von medizinischen Formaten aus rechtlichen Gründen gestört. Da $\epsilon CC\epsilon\mathcal{T}$ kein nach dem Medizinproduktegesetz [MPG] zugelassenes Medizinprodukt ist, wird vom Anwender zunächst eine entsprechende Haftungsausschlusserklärung gefordert, nach deren Erhalt wir den Import von medizinischen Daten entsperren.

Diese Beschränkung behindert leider in vielen Fällen erheblich die Verbreitung von $\epsilon CC\epsilon\mathcal{T}$, so dass Überlegungen anstehen, $\epsilon CC\epsilon\mathcal{T}$ gemäß MPG zuzulassen. Leider erfordert dies einen nicht unerheblichen bürokratischen Aufwand.

9.3 Forschungsergebnisse

Mit Hilfe von $\epsilon CC\epsilon\mathcal{T}$ entstanden einige Publikationen, die unter anderem auch auf die Leistungsfähigkeit von $\epsilon CC\epsilon\mathcal{T}$ eingehen.

9.3.1 CT-Colonographie

In [RöFo] untersuchen die Autoren die Anwendbarkeit von Mehrschicht-Spiraltomographie bei Niedrigdosis zur Detektion von Läsionen am Colon.

Unter Verwendung von $\epsilon CC\epsilon\mathcal{T}$ in einer inzwischen nicht mehr aktuellen Entwicklungsstufe ergab sich eine Sensitivität und Spezifität für Tumoren und Polypen mit einem Durchmesser ab 5mm von $> 85\%$.
Polypen kleiner 5mm zeigten insbesondere eine starke Zunahme von falsch positiven Markierungen, aber auch eine verringerte Sensitivität (60%).
Dies ist zum einen begründet durch die begrenzte Auflösung (3 mm Schichtdicke, Untersuchungen zur Verwendung besser aufgelöster Aufnahmen laufen) als auch durch die in diesem Bereich leicht fehlzuinterpretierenden Multislice-CT-Artefakte.

Ein erweiterter Polypendetektor mit automatischer Klassifikation von echten Polypen und vermuteten Artefakten, wie er inzwischen implementiert ist, dürfte hier weitere Verbesserungen bringen.

Die Studie wird zur Zeit weiter ausgebaut - siehe [VCS] und [VCS2]. Aktueller Stand (gem. [VCSA], Artikel in Vorbereitung) ist:

Untersucht wurden 150 Patienten mit über 180 in der hochauflösenden Videokoloskopie gefundenen Läsionen.

Dabei wurde bei 35 Patienten die normale Röntgendosis (150mAs, s. Tabelle 9.1)

verwendet, bei 115 Patienten wurde die durch die Entrauschung mögliche Niedrigdosistechnik (10mAs, s. Tabelle 9.2) angewandt.

	Tumor	Polyp	Polyp	Polyp	flache	
		$\geq 10\,\text{mm}$	5-9.9 mm	$< 5\,\text{mm}$	Adenome	gesamt
Virt. Koloskopie	10	2	14	27	0	53
Endoskopie	10	2	9	14	0	35
falsch Positive	0	0	5	16	0	21
falsch Negative	0	0	1	4	0	5
Sensitivität	100%	100%	90%	73%		86%
Spezifität	100%	100%	80%	56%		49%

Tabelle 9.1: Vergleich virtueller Normaldosis-Koloskopie mit hochauflösender Videokoloskopie

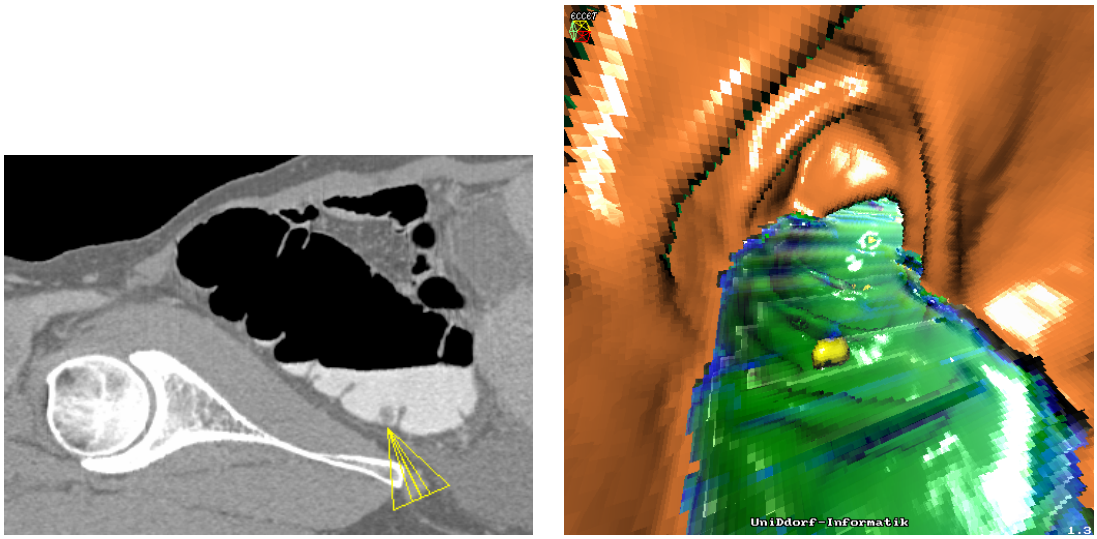
	Tumor	Polyp	Polyp	Polyp	flache	
		$\geq 10\,\text{mm}$	5-9.9 mm	$< 5\,\text{mm}$	Adenome	gesamt
Virt. Koloskopie	4	11	36	97	4	152
Endoskopie	4	9	33	24	8	150
falsch Positive	0	2	3	24	0	29
falsch Negative	0	0	3	23	4	30
Sensitivität	100%	100%	92%	76%	50%	80%
Spezifität	100%	97%	95%	70%	100%	66%

Tabelle 9.2: Vergleich virtueller Niedrigdosis-Koloskopie mit hochauflösender Videokoloskopie

Wie den Tabellen zu entnehmen ist, zeigt die virtuelle Koloskopie für Polypen $\geq$ 5mm Durchmesser eine brauchbare Sensitivität und Spezifität, insbesondere wenn man bedenkt, dass der Vergleichsmaßstab „**hochauflösende** Videokoloskopie“ noch nicht allgemein verbreitet ist, sondern häufig noch weit schlechter auflösende Verfahren im Einsatz sind.

Das schlechtere Abschneiden im Bereich $<$ 5mm und bei flachen Adenomen verwundert nicht, da für erstere die Auflösung der Grunddaten nicht wirklich ausreichend ist und flache Strukturen weder vom Polypendetektor erkannt werden können, noch im Röntgenbild besonders auffällig hervortreten.

Man beachte auch die Verbesserung von Sensitivität und Spezifität gegenüber der



a) 2D-Schnitt mit großem Polyp im Kontrastmittel

b) 3D-Ansicht dieses Polypen

Abbildung 9.3: Kontrastmittelgestützte virtuelle Koloskopie

vorherigen Untersuchung (85% → 92%). Dies dürfte sowohl auf verbesserte Algorithmen (Polypenfinder) und Darstellungsmethoden (Einblendung in 2D-Darstellung), als auch auf die breitere Erfahrungsbasis mit dem neuen System zurückzuführen sein.

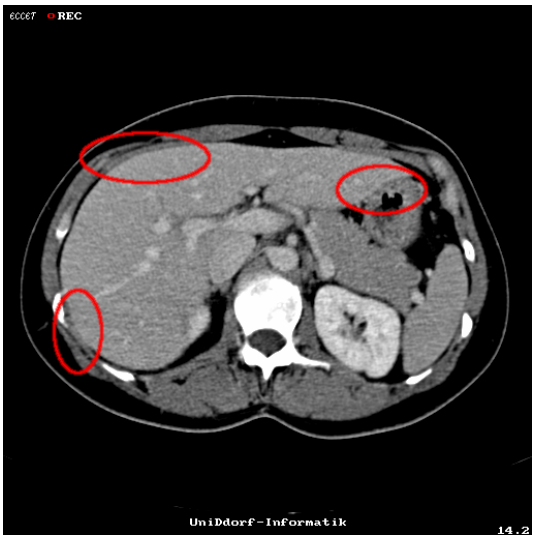
Weitere Ergebnisse bezüglich der Anwendung von $\epsilon CC\epsilon\mathcal{T}$ in der virtuellen Koloskopie und Einblicke in technische Einzelheiten geben [RöKoV], [RöKoD], [BVMS] und [BVMD].

9.3.2 Laufende Forschung

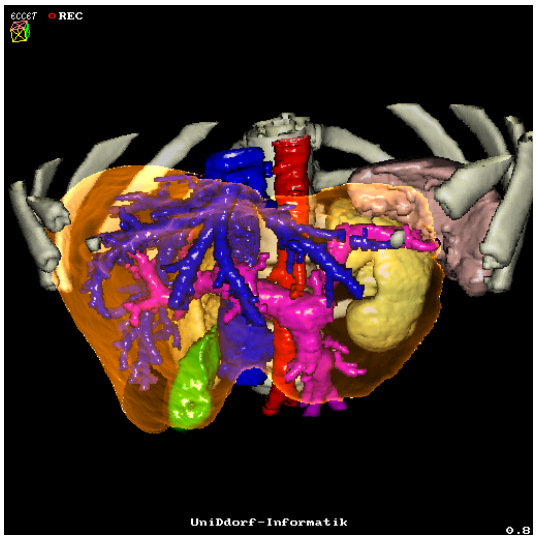
Kontrastmittelgestützte virtuelle Koloskopie

Zur weiteren Verbesserung der Ergebnisse der virtuellen Koloskopie und zur Verringerung der Patientenbelastung untersuchen wir zur Zeit in Zusammenarbeit mit der Universitätsklinik Düsseldorf kontrastmittelgestützte Verfahren.

Dabei wird dem Patienten während der Darmreinigung zusätzlich Kontrastmittel verabreicht. Flüssigkeitsgefüllte Darmabschnitte lassen sich damit rekonstruieren, wenngleich bedingt durch den geringeren Kontrast von Flüssigkeit zu Darmwand nicht mit der Präzision von luftgefüllten Bereichen (s. Abbildung 9.3).



a) 2D-Schnitt durch die Leber. Die Kreise markieren schlecht zu trennende Problemzonen.



b) 3D-Rekonstruktion der Leber mit Gefäßsystem, Nieren, Milz, Gallenblase und Rippen

Abbildung 9.4: Die Leber und ihr Gefäßsystem

Visualisierung der Leber und ihres Gefäßsystems

Im Rahmen eines weiteren Projektes, ebenfalls in Zusammenarbeit mit der Universitätsklinik Düsseldorf, wird die Anwendung von $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ zur präoperativen Visualisierung der Leber, ihres Gefäßsystems und eventueller Läsionen erforscht.

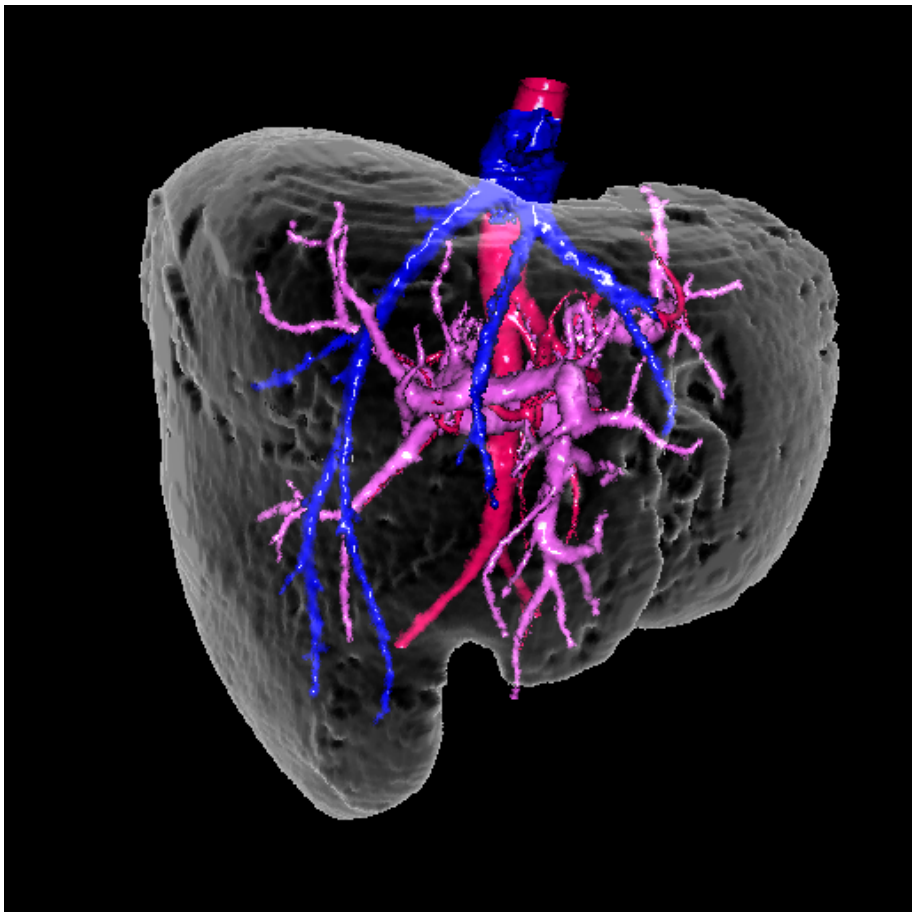
Wie diverse Publikationen in diesem Bereich zeigen ([HPV2], [RALS], [VOLS]), ist dies ein aktuelles und schwieriges Forschungsgebiet.

Speziell für diesen Zweck werden zur Zeit Algorithmen entwickelt, mit deren Hilfe die anspruchsvollen Aufgaben der Segmentierung der einzelnen Komponenten einfacher und schneller als mit bisherigen Verfahren (schichtweise Markierung einzelner Voxel, Live-Wires etc.) bewältigt werden sollen.

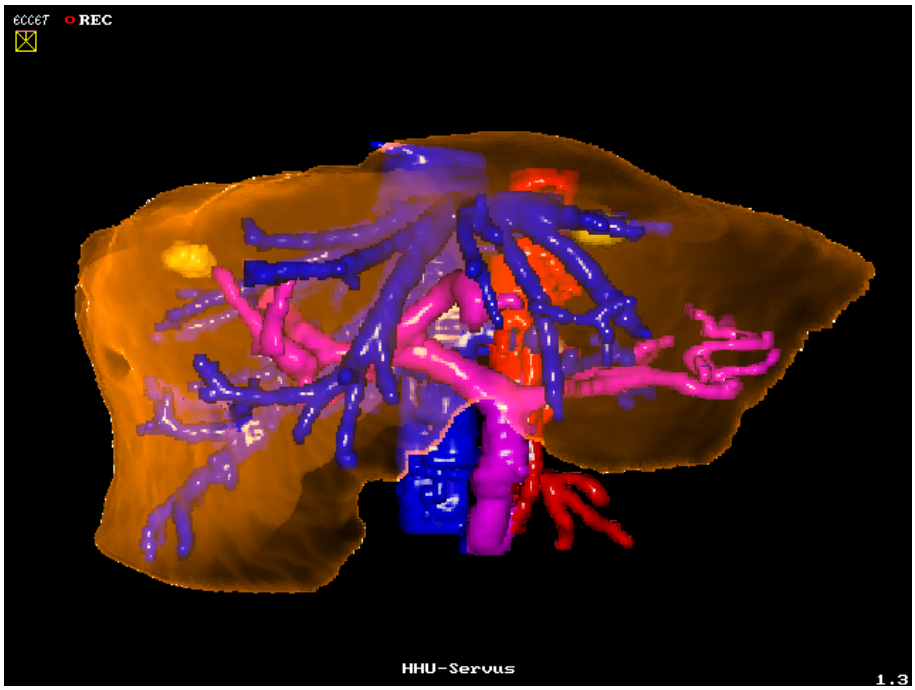
Des Weiteren wurden spezielle Darstellungsmodi entwickelt (s. Abbildung 3.7b und 9.4), um die komplexe Struktur des Gefäßsystems für den Betrachter besser zugänglich zu machen.

Wie das Bild ebenfalls illustriert, ist trotz der erst relativ kurzen Beschäftigung mit diesem Gebiet bereits die Erstellung einer hochwertigen Segmentierung und Darstellung mit Hilfe von $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ möglich, wenngleich noch zu zeitaufwändig (1-2h).

Wir hoffen jedoch, diese Zeit durch die Implementation neuer Segmentierungswerkzeuge noch deutlich verringern zu können.



a) Leber mit feiner aufgelöstem Gefäßsystem



b) Leber mit Gefäßsystem und zwei kleinen Zysten (gelb)

Abbildung 9.5: Segmentierungsergebnisse Leber

Kapitel 10

Ausblick

10.1 Technische Weiterentwicklung

$\epsilon\mathcal{CC}\epsilon\mathcal{T}$ wird laufend weiterentwickelt. Während die vorliegende Arbeit entstand, wurden eine ganze Reihe Erweiterungen implementiert, die hier nur zum Teil dargestellt werden.

Dieses Kapitel möchte einen kurzen Blick auf zur Zeit im Entstehen begriffene Funktionen werfen.

10.1.1 Verbesserte Rendergeschwindigkeit

Die Geschwindigkeit von PLANEVIEW konnte durch Optimierung des Programmcodes so weit gesteigert werden, dass inzwischen auf schnellen Maschinen eine interaktiv bewegliche, hochauflösende Darstellung möglich ist.

Dazu wurden Werte, die in den inneren Schleifen des Programms verwendet werden, soweit wie möglich vorberechnet, Fließkommaoperationen vermieden und redundanter Code wurde entfernt.

Die meisten dieser Optimierungen sind weitgehend plattformunabhängig, so dass keine feste Bindung an die Plattform entsteht.

Lediglich einige Fließkommaoptimierungen wurden speziell für die x86-Prozessorfamilie eingebaut, da diese aufgrund des historisch gewachsenen Konzepts (s. [FPO] und [FP2]) besonders lange für Konversionen von Fließkomma- zu Integerwerten braucht. Diese Optimierungen sind aber problemlos abschaltbar und sollten auch auf anderen Architekturen keine Probleme bereiten, sofern diese

Fließkommazahlen im üblichen IEEE-Format abspeichern.

Einige dieser Optimierungen konnten auch für VOXREN übernommen werden, so dass auch hier die Renderleistung um ca. 30% gesteigert werden konnte.

10.1.2 Integration von VOXREN und PLANEVIEW

Durch eine weitergehende Modularisierung wird zur Zeit angestrebt, die Teilprogramme VOXREN und PLANEVIEW in technischer Hinsicht zu verschmelzen. Aufgrund der unterschiedlichen Anwendungen werden beide sicherlich weiter dem Benutzer als Einzelprogramme zur Verfügung stehen. Technisch gesehen soll dies jedoch durch ein einziges Programm geschehen, das lediglich über seine Konfiguration entsprechend angepasst wird.

Besonders für PLANEVIEW ergibt sich der Vorteil, dass dort nun auch eine Bedienoberfläche über die TCP/IP-Schnittstelle angekoppelt werden kann.

Durch die Vereinheitlichung soll insbesondere erreicht werden, dass für spezielle Anwendungen leicht neue, maßgeschneiderte Bedienoberflächen erzeugt werden können, so dass der Lernaufwand sinkt und damit die Akzeptanz beim Nutzer erhöht wird.

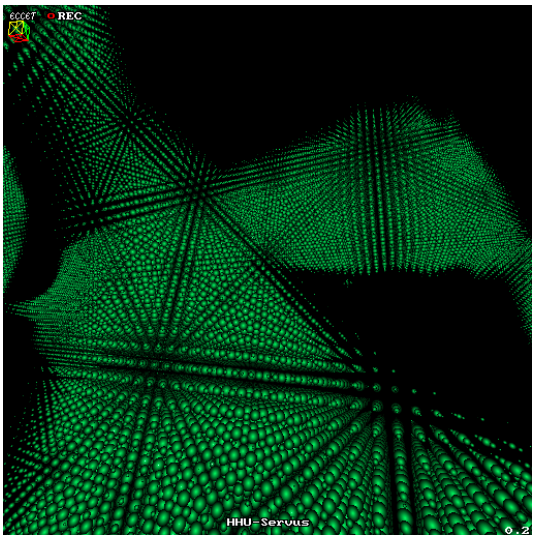
10.1.3 Modulare Renderer

Das Modulkonzept wurde dahingehend erweitert, dass auch Renderalgorithmen modular erstellt und nach Bedarf einzelnen Ansichten zugeordnet werden können. Ziel ist, den zur Zeit statischen Satz an möglichen Darstellungen auf die gleiche flexible Art änderbar zu machen wie die verwendbaren Kommandos. Es wird somit in Zukunft möglich sein, für spezielle Visualisierungsprobleme maßgeschneiderte Darstellungsroutinen zu erstellen und problemlos in das System zu integrieren.

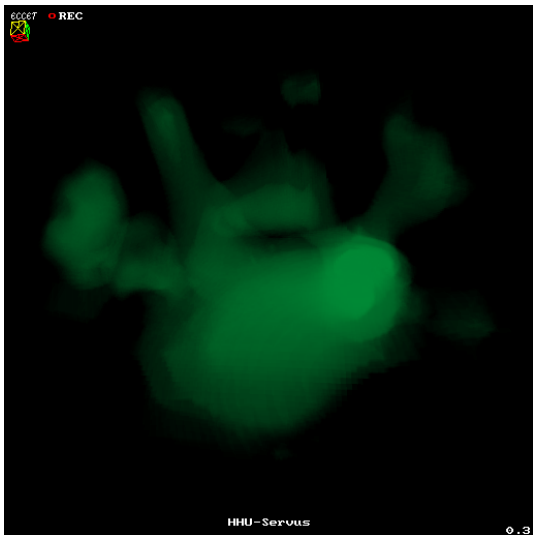
10.1.4 Neue Darstellungsmodi

Es wurden weitere Darstellungsmodi für die Renderer von VOXREN und PLANEVIEW entwickelt:

- Dünnschicht-MIP-Darstellung
  Wie im Abschnitt 4.2.3 erwähnt, eignet sich die Darstellung in der Maximum-Intensity-Projection lediglich zur Darstellung von stark positiv kontrastierten Volumina, wobei zusätzlich keine störenden weiteren stark absorbieren-



a) Dichtefeldvisualisierung über die Voxelgröße



b) Dichtefeldvisualisierung durch Volume Fogging

Abbildung 10.1: Dichtefeldvisualisierung eines MEG-Datensatzes

den Objekte im Volumen vorhanden sein dürfen. Durch Übernahme des 2-Schnittebenen-Konzeptes (s. Abbildung 4.9) aus VOXREN gibt es nun eine Möglichkeit, die Nützlichkeit dieses Modus deutlich zu steigern. Wird für die Berechnung der maximalen Intensität nämlich nur eine dünne Schicht (typ. 10-100 Voxel) verwendet, so können störende absorbierende Objekte im Außenbereich (z.B. die Rippen) unterdrückt werden.

Entsprechend kann es unter diesen Voraussetzungen Sinn machen, eine analoge Minimum-Intensity-Projection zu implementieren, die in einem Vollvolumen aufgrund des störenden äußeren Luftvolumens sinnlos ist.

- Beliebig orientierte Schnitte
  Analog zu VOXREN ist es nun auch in PLANEVIEW möglich, beliebig orientierte Schnitte durch das Volumen zu betrachten. Diese stellen in der Regel eine Schnittebene durch die Kamera dar, die in der Ausrichtung der der Kamera entspricht. Ist in der 3D-Ansicht ein Schnittebene aktiv, so wird diese dargestellt.
  Damit wird es möglich, Strukturen schnell in beliebigen Schnitten zu visualisieren. Man positioniert einfach die Kamera mitten in der Struktur und dreht die Kamera.

- Dichtefeldvisualisierung über die Voxelgröße
  VOXREN ist in der Lage den Grauwert eines Voxels durch dessen Größe darzu-

stellen. Damit wird es möglich, dreidimensionale Dichtedaten zu visualisieren (s. 10.1a).

- Dichtefeldvisualisierung durch Volume Fogging
  Analog ist es möglich, den Grauwert eines Voxels durch dessen Durchsichtigkeit anschaulich zu machen (s. 10.1b). Dazu wird derselbe Code benutzt wie bei der Visualisierung von explizit als transparent markierten Voxeln.

10.1.5 Mengenoperationen

Beim Segmentieren komplexer Objekte war bisher eine sehr sorgfältige Planung der Folgeschritte notwendig, um versehentliches Überschreiben bereits segmentierter Bereiche zu verhindern.

Es wird aktuell daran gearbeitet, ein Mengenkonzept in das System einzuarbeiten, das es für jede Operation erlaubt, die möglichen Quell- und Zielklassen festzulegen und damit auch Klassen vor dem versehentlichen Überschreiben zu sichern.

Des Weiteren ist dieses System bedeutend übersichtlicher und leichter in einem GUI darzustellen, so dass auch der Bereich Segmentierung für den Nicht-Experten besser zugänglich werden sollte.

10.1.6 Entwicklung einer eigenen GUI-Komponente

Zurzeit wird eine eigene einfache GUI-Library entwickelt, mit deren Hilfe die strenge Trennung von Renderengine und Bedienelementen visuell durchbrochen werden soll, da sie für den Benutzer nicht so intuitiv ist wie eine integrierte Oberfläche.

Dabei soll aber auf technischer Ebene weiterhin die Trennung weitgehend erhalten bleiben, um eine größtmögliche Flexibilität zu wahren. Es ist daher geplant, das Modulsystem zu nutzen, um zusätzlich zu Kommandos, Tastendrücken und Mausaktionen auch graphische Bedienelemente registrieren zu können. Diese wären dann zwar visuell integraler Bestandteil des Systems, könnten aber dennoch leicht ausgetauscht werden.

Ziel ist auch hier die Möglichkeit, mit geringem Aufwand Applikationen schaffen zu können, die für spezielle Anwendungen optimiert sind.

10.2 Verbesserung des Benutzerinterfaces

$\epsilon CC\epsilon\mathcal{T}$ wurde als experimentelles System konzipiert, mit dem neue Segmentierungs- und Visualisierungstechniken erprobt werden können. Aufgrund des großen Funktionsumfanges ist es für einen neuen Benutzer oft nicht ganz leicht zu bedienen, da er von der Funktionsvielfalt überwältigt wird.

10.2.1 Verbesserter Einstieg für neue Benutzer

Die oben erwähnten technischen Änderungen zielen zum großen Teil dahin, die Benutzeroberfläche flexibler zu gestalten, so dass es möglich wird, dem Benutzer zunächst mit einer einfachen und intuitiven Schnittstelle den Einstieg zu ermöglichen und ihm dann schrittweise über zusätzlich aktivierte Module den Zugang zu den komplexeren Funktionen zu öffnen.

Dazu wird es insbesondere notwendig sein, die unflexiblen und auch für den Programmierer ab einer gewissen Komplexität recht unübersichtlichen Tcl/Tk-Programme durch besser modularisierbare Komponenten zu ersetzen (z.B. 10.1.6).

10.2.2 Erstellung von Spezialanwendungen

Viele Anwendungen benötigen nur einen sehr kleinen Bruchteil der Fähigkeiten von $\epsilon CC\epsilon\mathcal{T}$. Durch die bessere Modularisierung wird es möglich, Spezialapplikationen zu konstruieren, indem $\epsilon CC\epsilon\mathcal{T}$ auf die für die jeweilige Anwendung relevanten Module eingeschränkt und mit einem hierfür spezifischen GUI ausgestattet wird.

Zurzeit in Planung ist insbesondere ein solches System für die virtuelle Koloskopie, das mit sehr wenigen Bedienelementen auskommt.

10.2.3 Konsistente Bedienung

Aus historischen Gründen verwenden insbesondere diverse Segmentieralgorithmen recht verschiedene Parameterformate und Randbedingungen. Insbesondere die in 10.1.5 angesprochene Verwendung von Ein- und Ausgabeklassenmengen sowie ggf. Parameterklassenmengen sollte hier die Bedienung in vielen Punkten vereinheitlichen.

Entsprechend wird auf längere Sicht angestrebt, sich gerade mit dem graphischen Benutzerinterface üblichen Standards, die dem Benutzer vertraut sind, zu nähern.

10.3 Ausweitung der Anwendungsbereiche

Durch verbesserte Benutzerführung und dem damit verbundenen leichteren Einstieg hoffen wir, auch neue Anwendungsbereiche zu erschließen.

$\epsilon CCE\mathcal{T}$ wird bereits jetzt zur Bildgebung auf sehr verschiedenen Feldern der Medizin (Neurologie, Gastroenterologie, Hepatologie) angewandt sowie zur Visualisierung von Messdaten (MEG) und realen Objekten (3DSCAN).

Als weitere Anwendungen sind vorstellbar:

- virtuelle Operationsplanung (insbesondere mit Hilfe der Multivolume-Fähigkeiten, die bisher noch kaum genutzt werden),
- Materialprüfung,
- Visualisierung mehrdimensionaler Messdaten,
- Visualisierung dynamischer Prozesse (durch die Multivolume-Fähigkeit können dreidimensionale Objekte animiert werden).

$\epsilon CCE\mathcal{T}$ wird laufend weiterentwickelt und ergänzt, um solche neuen Anwendungen zu erschließen.

Anhang A

Kommandoreferenz zu VOXREN

A.1 Interne Kommandos

Das Hauptprogramm von VOXREN implementiert einige Kommandos, die aufgrund ihrer Natur (z.B. Interaktion mit der Renderingengine) in recht direktem Zusammenhang mit dem Programm stehen und nicht austauschbar sind.

A.1.1 HELP

Synopsis

`HELP [MODULES|COMMANDS|KEYS|MOUSE] [`*MODULENAME*`]`

Beschreibung

Zeigt Hilfetexte für alle geladenen Module an.

Mit HELP MODULES kann eine Übersicht aller aktiven Module angezeigt werden.

Mit HELP COMMANDS erhält man eine Kurzhilfe für alle aktiven Kommandos. Ist der optionale Parameter *MODULENAME* gesetzt, werden nur die Kommandos des betreffenden Moduls gelistet.

Mit HELP KEYS kann analog eine Liste aller von einem Modul belegten Tasten und mit mit HELP MOUSE eine kurze Übersicht über die Maustastenbelegungen von Modulen angezeigt werden.

Beispiele

`HELP MODULES`

Zeigt alle aktiven Module an.

`HELP COMMANDS internal`

Zeigt alle internen Kommandos an.

`HELP KEYS move`

Zeigt alle Tasten, die Bewegungen auslösen (bzw. deren Aktionen im Modul „move“ implementiert sind) an.

Rückgabewert

Liste von Hilfetexten.

A.1.2 BCAST

Synopsis

`BCAST [0|1]`

Beschreibung

(De-)aktiviert die asynchrone Übertragung (Broadcast) von Positionsinformationen.

Wird kein Parameter übergeben, so wird der aktuelle Zustand als Zahl zurückgegeben. Ist 0 oder 1 angegeben, so wird der Broadcast aus- bzw. eingeschaltet.

Beispiele

`BCAST`

Zeigt an, ob die Broadcastfunktion für den aktuellen Kommunikationskanal aktiv ist.

`BCAST 1`

Schaltet die Übertragung von Positionsinformation ein.

Rückgabewert

0/1 bei der Abfrageform ohne Parameter, ansonsten Erfolgsmeldung (3xx) oder Fehlermeldung (5xx).

A.1.3 RECORD

Synopsis

```
RECORD [?|0|1]
```

Beschreibung

(De-)aktiviert die Aufzeichung für den aktiven View. Ohne Parameter wird der aktuelle Zustand ausgelesen, mit einem `?` als Parameter werden die möglichen Zustände ausgegeben, ansonsten wird der Parameter als neuer Zustand übernommen.
Das Aufzeichnungsformat wird vom Kommando `RECORDMODE` (s. A.1.4) bestimmt.

Beispiele

`RECORD`
Zeigt an, ob der aktuelle View aufgezeichnet wird (Antwort ist 0 oder 1).
`RECORD ?`
Antwortet mit „RECORD: 0 1“.
`RECORD 1`
Schaltet die Aufzeichnung ein.

Rückgabewert

Wird kein Parameter übergeben, so wird der aktuelle Zustand zurückgegeben, beim Parameter `?` wird eine Liste der möglichen Parameter zurückgegeben.
Wird ein gültiger Wert als Parameter angegeben, so wird eine INFO (3xx) Meldung über den neuen Wert ausgegeben. Ansonsten ein Fehlercode (5xx).

A.1.4 RECORDMODE

Synopsis

`RECORDMODE [?|`*Format*`]`

Beschreibung

Wählt das Aufzeichungsformat für den aktuell aktiven View. Ohne Parameter wird der aktuelle Zustand ausgelesen, mit einem `?` als Parameter werden die möglichen Zustände ausgegeben, ansonsten wird der Parameter als neuer Zustand übernommen.

Die Aufzeichnung wird durch das Kommando `RECORD` (s. A.1.3) gestartet und gestoppt.

Die verfügbaren Formate hängen vom installierten Save-Plugin ab und können mit `RECORDMODE ?` abgefragt werden.

Beispiele

`RECORDMODE PNG`

Setzt das Aufzeichungsformat auf PNG (Portable Network Graphics).

Rückgabewert

Wird kein Parameter übergeben, so wird der aktuelle Zustand zurückgegeben, beim Parameter `?` wird eine Liste der möglichen Parameter zurückgegeben.

Wird ein gültiger Wert als Parameter angegeben, so wird eine INFO (3xx) Meldung über den neuen Wert ausgegeben. Ansonsten ein Fehlercode (5xx).

A.1.5 REALTIMERECORD

Synopsis

`REALTIMERECORD [?|`*sec_per_frame*`]`

Beschreibung

Zeichnet einen Film so auf, dass die Wiedergabe einen Eindruck der Echtzeitperformance des Systems liefert. Dazu werden nach Bedarf Bilder mehrfach gespeichert oder ausgelassen, so dass sich ein Film mit der in *sec_per_frame* angegebenen Zeitauflösung ergibt.

Ist der Parameter negativ, so wird die Echtzeitaufzeichnung ausgeschaltet und es werden alle gerenderten Bilder je einmal gespeichert.

Hinweis: Arbeitspausen werden **nicht** aufgezeichnet, die zusätzlich zum Abspeichern benötigte Zeit wird nicht berücksichtigt.

Ohne Parameter wird der aktuelle Zustand ausgelesen, mit einem ? als Parameter werden die möglichen Zustände ausgegeben, ansonsten wird der Parameter als neuer Zustand übernommen.

Beispiele

`REALTIMERECORD 0.04`

Stellt die Echtzeitaufzeichnung auf eine Framerate von 25 Bildern pro Sekunde (entspricht 0.04 Sekunden pro Bild) ein.

`RECORD -1`

Schaltet die Echtzeitaufzeichnung aus.

Rückgabewert

Wird kein Parameter übergeben, so wird der aktuelle Zustand zurückgegeben, beim Parameter ? wird eine Liste der möglichen Parameter zurückgegeben.

Wird ein gültiger Wert als Parameter angegeben, so wird eine INFO (3xx) Meldung über den neuen Wert ausgegeben. Ansonsten ein Fehlercode (5xx).

A.1.6 COLORIZE

Synopsis

`COLORIZE [?|`*Voxelklasse*`]`

Beschreibung

Wählt die Voxelklasse für umfärbende Operationen ohne explizite Zielklassenangabe. Ohne Parameter wird der aktuelle Zustand ausgelesen, mit einem ? als Parameter werden die möglichen Zustände ausgegeben, ansonsten wird der Parameter als neuer Zustand übernommen.

Beispiele

```
COLORIZE 16
```

Wählt Voxelklasse 16 zum Umfärben mit z.B. FLAGS 32 (s. A.6.10).

Rückgabewert

Wird kein Parameter übergeben, so wird der aktuelle Zustand zurückgegeben, beim Parameter ? wird eine Liste der möglichen Parameter zurückgegeben.

Wird ein gültiger Wert als Parameter angegeben, so wird eine INFO (3xx) Meldung über den neuen Wert ausgegeben. Ansonsten ein Fehlercode (5xx).

A.1.7 AUTO

Synopsis

```
AUTO [?|0-63]
```

Beschreibung

Wählt den Betriebsmodus des Autopiloten. Es handelt sich dabei um ein Bitfeld bei dem folgende Optionen ein- und ausgeschaltet werden können.

1 Vorschub

2 Drehen. Der Autopilot richtet sich auf den tiefsten aktuell zu sehenden Punkt aus.

4 Ausweichen. Nahe Objekte an den Rändern des sichtbaren Bereiches veranlassen den Autopiloten, seitlich auszuweichen.

8 Rollen. Der Autopilot versucht sich durch Rollen so auszurichten, dass die physische Richtung „unten“ auch im Bild unten gehalten wird.

16 Sackgassenbehandlung. Fliegt der Autopilot in eine Sackgasse, so versucht er automatisch, diese durch eine Rechtsdrehung wieder zu verlassen.

32 Markerdrop. Der Autopilot setzt automatisch auf seinem Flugpfad alle 10 cm eine Markierung aus.

Ohne Parameter wird der aktuelle Zustand ausgelesen, mit einem ? als Parameter werden die möglichen Zustände ausgegeben, ansonsten wird der Parameter als neuer Zustand übernommen.

Beispiele

`AUTO 23` Aktiviert den Autopiloten für Därme ohne Gravitationssimulation.

`AUTO 31` Wie oben, aber mit Gravitationssimulation (Rollen).

`AUTO 21` Halbmanuelle Navigation. Der Autopilot steuert nur den Vorschub und weicht aus. Die Richtung wird per Hand (Maussteuerung) vorgegeben.

`AUTO 55` Markerdrop. Wie 23, zusätzlich wird alle 10 cm eine Markierung ausgesetzt.

Rückgabewert

Wird kein Parameter übergeben, so wird der aktuelle Zustand zurückgegeben, beim Parameter ? wird eine Liste der möglichen Parameter zurückgegeben.

Wird ein gültiger Wert als Parameter angegeben, so wird eine INFO (3xx) Meldung über den neuen Wert ausgegeben. Ansonsten ein Fehlercode (5xx).

A.1.8 PLAY

Synopsis

`PLAY [?|.|`*Filename*`]`

Beschreibung

Spielt ein Positionsfile ab, das mit RECORD (s. A.1.3) im RECORDMODE POS (s. A.1.4) erstellt wurde. Existiert die Datei nicht oder ist der Parameter '.', so wird eine evtl. gerade laufende Sequenz gestoppt und die zugehörige Datei geschlossen.

Ohne Parameter wird der aktuelle Zustand ausgelesen, mit einem ? als Parameter werden die möglichen Zustände ausgegeben, ansonsten wird der Parameter als neuer Zustand übernommen.

Beispiele

`PLAY /ct/Testpatient/pictures/Flug.pos`

Spielt die Datei /ct/Testpatient/pictures/Flug.pos ab.

`PLAY .`

Beendet das Abspielen.

Rückgabewert

Wird kein Parameter übergeben, so wird der aktuelle Zustand zurückgegeben, beim Parameter ? wird eine Liste der möglichen Parameter zurückgegeben.

Wird ein gültiger Wert als Parameter angegeben, so wird eine INFO (3xx) Meldung über den neuen Wert ausgegeben. Ansonsten ein Fehlercode (5xx).

A.1.9 PLAYDIR

Synopsis

`PLAYDIR [?|-5|-2|-1|0|1|2|5]`

Beschreibung

Beeinflusst die Abspielgeschwindigkeit eines Positionsfiles (s. A.1.8). Die Werte bedeuten:

-5 Schneller Rücklauf

-2 Einzelbild rückwärts

-1 Abspielen rückwärts

0 Pause

1 Abspielen vorwärts

2 Einzelbild vorwärts

5 Schneller Vorlauf

Andere Werte sind nicht definiert.

Ohne Parameter wird der aktuelle Zustand ausgelesen, mit einem ? als Parameter werden die möglichen Zustände ausgegeben, ansonsten wird der Parameter als neuer Zustand übernommen.

Beispiele

`PLAYDIR 0`

Stoppt das Abspielen der aktuellen POS-Datei.

`PLAYDIR 1`

Führt das Abspielen weiter.

Rückgabewert

Wird kein Parameter übergeben, so wird der aktuelle Zustand zurückgegeben, beim Parameter ? wird eine Liste der möglichen Parameter zurückgegeben.

Wird ein gültiger Wert als Parameter angegeben, so wird eine INFO (3xx) Meldung über den neuen Wert ausgegeben. Ansonsten ein Fehlercode (5xx).

A.1.10 LOADCLUT

Synopsis

`LOADCLUT [?|`*Filename*`]`

Beschreibung

Lädt eine CLUT (Color Look Up Table), mit deren Hilfe die Farbcharakteristik des dargestellten Bildes beeinflusst werden kann.

Eine CLUT ist ein PNM-Bild im P6-Format der Größe 512x512.

Symmetrisch um Zeile 256 angeordnet befinden sich jeweils die CLUTs einer Voxelklasse für die Vorder- und Rückseite. **Achtung:** Klasse 0 unterscheidet keine Vorder- und Rückseite.

Typisch wird eine solche Zeile einen Farbverlauf von schwarz zur vollen Sättigung der Farbe enthalten sowie ggf. am Ende eine Aufhellung ins Weiß, um Glanzlichter

zu simulieren.

Zeile 0 enthält ab Spalte 0 Färbungsinformation für die Voxelklassen 0-15. Diese sind in 8 Blöcke zu je 16 Farben eingeteilt. Der erste Block gibt dabei die normale Färbung an, die 7 weiteren Blöcke werden für je eines der möglichen `GREYMARK`-Overlays (s. A.6.21) verwendet.

Der bisher unbenutzte Teil der Zeile soll mit dem Wert für die Randhervorherbung (s. FLAGS=4 in A.6.10) belegt sein.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`LOADCLUT /ct/lib/eccet/clut/standard.ppm`

Lädt die Standard-CLUT.

Rückgabewert

Wird kein Parameter übergeben, so wird der aktuelle Zustand zurückgegeben, beim Parameter `?` wird eine Liste der möglichen Parameter zurückgegeben.

Wird ein gültiger Wert als Parameter angegeben, so wird eine INFO (3xx) Meldung über den neuen Wert ausgegeben. Ansonsten ein Fehlercode (5xx).

A.1.11 MARK

Synopsis

`MARK [?|`*min max class*`]`

Beschreibung

Markiert alle Voxel, die in den angegebenen Grauwertbereich (*min-max*) fallen, mit der Klasse *class*.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform

ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`MARK 0 200 2`

Ordnet allen Voxeln mit Grauwerten von 0 bis 200 die Klasse 2 (grün) zu.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.12 MARKD

Synopsis

`MARKD [?|`*min max class*`]`

Beschreibung

Markiert alle Voxel, deren Markierung im Tiefenfeld (GDA) im angegebenen Bereich (*min-max*) liegt mit der Klasse *class*.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`MARKD 1 5 2`

Markiert alle Voxel, die maximal 5 Pixel gemäß der GDA von den aktuellen Rändern entfernt sind, grün (Klasse 2).

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.13 MARKZ

Synopsis

`MARKZ [?|`*min max*`]`

Beschreibung

Markiert alle Voxel, die in den angegebenen Grauwertbereich (*min-max*) fallen, im GDA-Feld mit 0, alle anderen mit 1.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
MARKZ 200 65535
```

Markiert alle Voxel, deren Grauwert größer als 200 ist im GDA-Feld als „massiv“ und alle anderen als durchsichtig.

Anwendung: Instant-Rendering im ZSH-Mode oder Markierung von Grenzen beim interaktiven Segmentieren (s. 6.2).

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.14 MARKZD

Synopsis

`MARKZD [?|`*min max*`]`

Beschreibung

Markiert alle Voxel, die in den angegebenen GDA-Bereich (*min-max*) fallen, im GDA-Feld mit 0, alle anderen mit 1.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
MARKZD 1 1
```

Markiert alle Voxel, die bisher den „Rand“ des als 0 markierten GDA-Bereiches bildeten, als neue GDA=0.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.15 MARKE

Synopsis

`MARKE [?|`*min*`]`

Beschreibung

Markiert alle Voxel, in deren Umgebung sich eine Grauwertkante befindet, an der ein Grauwertunterschied von mehr als *min* vorliegt, im GDA-Feld mit 0, alle anderen mit 1.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
MARKE 100
```

Markiert alle Voxel in der Nähe sehr steiler Grauwertkanten.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.16 DESTROYGREY

Synopsis

`DESTROYGREY [?|`*class*`] [`*targetgrey*`]`

Beschreibung

Zerstört den Grauwert aller Voxel der angegebenen Klasse, indem es ihn mit *targetgrey* überschreibt. Wird *targetgrey* nicht angegeben, so wird 65535 angenommen.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der

Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`DESTROYGREY 1 0`

Setzt den Grauwert aller Voxel die zu Klasse 1 (blau) gehören auf 0.

Anwendungen

Schrumpfen von gezippten 3d32-Dateien, indem alle nicht benötigeten Planes auf 0 gesetzt werden, so dass `gzip` die Datei besser packen kann.

Präparation von Volumina für eine teilweise Filterung mit NLG oder NLGS. Diese Filteroperationen ignorieren Pixel mit Grauwert 65535.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.17 DENOISE

Synopsis

`DENOISE [?]` *size from small big*

Beschreibung

Bestimmt die Zusammenhangskomponenten aller Voxel mit Klasse *from* und vergleicht deren Größe mit *size*. Komponenten mit geringerer Größe werden auf *small* umgefärbt, andere auf *big*.

Achtung - *small* und *big* dürfen nicht mit *from* identisch sein.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der

Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`DENOISE 10 3 0 2`

Entmarkiert alle Komponenten der Klasse 3, sofern sie kleiner als 10 Voxel sind. Das Ergebnis befindet sich in Klasse 2.

Anwendungen

Entfernen kleiner „Schneeartefakte“ z.B. im Darminneren bei Niedrigdosisaufnahmen.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.18 DILATE

Synopsis

`DILATE [?]` *from border size*

Beschreibung

Dilatiert die Menge der Voxel mit Klasse *from* mit einem kubischen Strukturelement der Kantenlänge $2size+1$ und setzt die neu hinzukommenden Voxel in die Klasse *border*.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`DILATE 2 4 5`

Dilatiert die Voxelmenge der Klasse 2 um 5 Voxel in alle Richtungen (Strukturelement hat Kantenlänge 11). Die neu hinzugekommenen Voxel tragen die Klasse 4.

Anwendungen

Hinzufügen von Rändern, um z.B. Platz für Operationen mit größeren Operatorfenstern zu lassen.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.19 CUTVOLUME

Synopsis

`CUTVOLUME [?] cube|`*dx dy dz wx wy wz*

Beschreibung

Beschneidet das aktuelle Volumen auf den Quader, der durch die linke obere Ecke *dx dy dz* und die Ausdehnung *wx wy wz* beschrieben wird.

Wird anstatt von Koordinaten das Schlüsselwort `cube` verwendet, so wird der kleinstmögliche Quader bestimmt, der noch alle Oberflächen (GDA=0) enthält.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`CUTVOLUME cube`

Beschneidet das Volumen auf das zur Darstellung aller Oberflächen notwendige Minimum.

Anwendungen

Beschleunigung der Darstellung und weiterer Operationen, nachdem sicher ist, dass die Bereiche außerhalb dieses Quaders nicht mehr benötigt werden.

Die alte Numerierung der Voxel bleibt für die Kommunikation mit PLANEVIEW (und **nur** für diese) erhalten.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.20 BFS

Synopsis

`BFS [?]` *x y z* `[`*skeldepth*`] [`*pocketsize*`]`

Beschreibung

Startet den Multiskalen-Füllalgorithmus (modifizierte Breitensuche, daher Breadth First Search) nach Abschnitt 6.2 an der Position *x y z* im Volumen.

Die maximal erlaubte „Lochgröße“ für den Algorithmus wird auf 2·*skeldepth* eingestellt, die maximal erlaubte Taschengröße auf *pocketsize*.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`BFS -1 -1 -1 16 -1`

Setzt die „Lochgröße“ für spätere Aufrufe (insbesondere Aufrufe per Tastendruck) auf 16.

`BFS -1 -1 -1 -1 100`

Setzt die Taschengröße für spätere Aufrufe (s.o.) auf 100.

Anwendungen

Setzen der Parameter für den Multiskalen-Füllalgorithmus.

Die Anwendung des Algorithmus erfolgt in der Regel eher per Point-and-Click.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.21 NORM

Synopsis

`NORM [?] targetclass [sigma factor type minlen]`

Beschreibung

Generiert Normalen für die aktuelle Auswahl in Voxelklasse 2 (grün).

sigma bestimmt den Gewichtungsfaktor für den Einfluss von Nachbarpunkten (hohes σ ergibt glattere Oberflächen mit weniger Treppchenartefakten um den Preis einer ungenaueren Wiedergabe sehr lokaler Veränderungen). Standardwert 2.0 .

factor gibt an, wieviele Voxel weit das Operatorfenster reichen soll. Standardwert 4. Empfohlener Wert ca. 2·*sigma*.

type Falls type=2 ist, wird eine modifizierte Version der Normalenberechnung eingesetzt, die geringfügig bessere Ergebnisse liefert. In dieser Version ist der Parameter *minlen* unwirksam.

minlen bestimmt, welchen Wert das Summenregister nach der Berechnung mindestens enthalten muss, damit der Wert verwendet wird. Wird dieser Wert nicht erreicht, so wird das Pixel linear im Bereich 25-75% transparent geschaltet. Solche Voxel werden von allen Seiten mit der gleichen Intensität dargestellt und stellen damit am ehesten eine korrekte Darstellung für Voxel dar, deren Normale nicht sinnvoll bestimmt werden kann.

Des Weiteren simulieren sie durch die semitransparente Darstellung im XYT-Modus den häufigsten Grund für zu kurze Normalen: schmale Ausläufer z.B. an Gefäßbäumen von nur noch wenigen Voxeln Durchmesser.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
NORM 16
```

Generiere eine Oberfläche mit Klasse 16.

Anwendungen

Generierung von Oberflächen nach der Segmentierung im Grauwertbereich.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

Nach dem Ende des Kommandos wurde der Außenraum auf Klasse 1 (blau) umgefärbt. Die neu generierte Oberfläche wird in der GDA eingetragen. Eine Beschleunigung erfolgt aber noch nicht (s. A.1.22).

A.1.22 MAKEDIST

Synopsis

```
MAKEDIST [?|1]
```

Beschreibung

Erzeugt die für die Beschleunigung der Darstellung gebrauchte GDA-Information. Ist der optionale Parameter 1 gesetzt, so werden zuvor die GDA-Ränder neu berechnet, wobei alle aktuell sichtbaren Klassen als Randvoxel markiert werden. Hierdurch ist es möglich, Klassen so auszublenden, dass sie den Renderingvorgang nicht mehr bremsen.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
MAKEDIST
```

Erstellt die Beschleunigungsinformation.

Anwendungen

Beschleunigung der Darstellung nach der Neuerstellung von Oberflächen.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.23 QUITPROGRAM

Synopsis

`QUITPROGRAM [?|1]`

Beschreibung

Beendet VOXREN. Aufgrund der weitreichenden Konsequenzen dieses Befehls wird dieser nur ausgeführt, wenn er zusätzlich den Parameter 1 erhält.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`QUITPROGRAM 1`

Beendet VOXREN .

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.24 REPEAT

Synopsis

`REPEAT` *n COMMAND*

Beschreibung

Führt das nachfolgende Kommando *COMMAND* *n* mal aus. Achtung: Zwischen den Einzelschritten wird **nicht** neu gerendert. Der Befehl eignet sich also nicht zur

Erstellung von Filmen.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
REPEAT 5 FWD 1
```

Entspricht `FWD 5`.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.25 DISP

Synopsis

```
DISP 0|16
```

Beschreibung

Setzt die aktuellen Anzeigeeinstellungen. Sinnvoll sind nur die Werte 0 (normal) und 16 (forcedisp).

In der Einstellung forcedisp wird nach jedem Kommando, jedem Tastendruck und jeder Mausbewegung ein Rendering erzwungen.

Normalerweise werden erst alle noch anliegenden Befehle en bloc ausgeführt, bevor ein neues Bild gerendert wird, so dass eine flüssige Verarbeitung erreicht wird.

Dieses Verhalten ist in zwei Fällen störend:

1. Laufen längere Scripte ab, will der Benutzer im Allgemeinen deren Aktionen verfolgen. Da ein Script seine Kommandos sehr schnell an VOXREN übergibt,

würde das Rendern bis zum Ende des Scriptes angehalten. Setzt man `DISP 16`, so wird nach jedem Schritt ein Rendering erzwungen und der Benutzer erhält eine visuelle Rückmeldung über die Ergebnisse der einzelnen Verarbeitungsschritte.

2. Beim Rendern bestimmter Filmsequenzen (z.B. 360° Drehungen um ein Objekt) bietet sich die Nutzung von Scripten an. Hier will man natürlich auch alle Zwischenstufen angezeigt haben. Analog unterdrückt man auch das Zusammenfassen von Tastatur bzw. Mauskommandos, so dass ein flüssigerer Filmeindruck bei manueller Steuerung entsteht.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`DISP 16`

Schaltet das erzwungene Rendering nach jedem Kommando ein.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.26 LOAD

Synopsis

`LOAD` *filename* `[`*numpics*`]`

Beschreibung

Lädt ein neues Volumen aus der angegebenen Datei. Ist der optionale Parameter *numpics* gesetzt, werden bei Bildstapeln nicht alle Bilder geladen, sondern nur so

viele wie dort angegeben.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
LOAD /ct/Patient/3d32/colon.3d32.gz
```

Lädt die angegebene Datei.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.27 MERGE

Synopsis

`MERGE` *filename* [*numpics*]

Beschreibung

Lädt ein neues Volumen aus der angegebenen Datei zusätzlich zum aktuellen Volumen. Ist der optionale Parameter *numpics* gesetzt, werden bei Bildstapeln nicht alle Bilder geladen, sondern nur so viele wie dort angegeben.

Der Mergeschritt beeinflusst nur Voxel, die **im bereits geladenen Volumen** die Voxelklasse 0 besitzen. Insbesondere lässt er also existente Oberflächen im aktuellen Volumen unangetastet. Das nachgeladene Volumen muss die gleiche Größe wie das aktuelle Volumen besitzen.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der

Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`MERGE /ct/Patient/3d32/liver.3d32.gz`

Lädt die angegebene Datei zu der bereits geladenen hinzu.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

Falls neue Oberflächen geladen wurden, so ist die GDA im Allgemeinen nicht korrekt (sie wird ebenfalls für alle Voxel der Klasse 0 vom neu hinzugeladenen Volumen übernommen). Diese muss also ggf. mit `MAKEDIST` (s. A.1.22) neu erstellt werden.

A.1.28 SAVE

Synopsis

`SAVE` *filename*

Beschreibung

Speichert das aktuelle Volumen als 3d32-Datei ab. Endet der Dateiname mit „.gz“, so wird das Volumen automatisch gezippt. Falls die Grauwerte nicht mehr benötigt werden, empfiehlt es sich, diese mit `DESTROYGREY` (s. A.1.16) auf 0 zu setzen, um eine bessere Kompression zu erreichen.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`SAVE /ct/Patient/3d32/combined.3d32.gz`

Speichert das aktuelle Volumen komprimiert ab.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.29 SAVE16

Synopsis

`SAVE16` *filename class*

Beschreibung

Speichert das aktuelle Volumen als 16HL-Stapel ab. Ist *class* dabei -1, so werden die Grauwerte abgespeichert. Ist *class* eine gültige Voxelklasse (0-255), so wird diese in Bit 12 des Volumens als Markierung abgespeichert.

Die Folgedateinamen werden durch Hochzählen enthaltener Zahlen generiert.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`SAVE16 /ct/Patient/filtered/0001.gz`

Speichert das aktuelle Volumen als 16HL-Stapel ab.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.30 ECHO

Synopsis

`ECHO` *Text to display*

Beschreibung

Zeigt *Text to display* auf dem aktuellen View an.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
ECHO Please wait - script running.
```

Zeigt `Please wait - script running.` an.

Anwendungen

Anzeigen des Fortschritts bei komplexen Scripten.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.31 SLEEP

Synopsis

`SLEEP` *microseconds*

Beschreibung

Wartet *microseconds* Mikrosekunden, bevor das nächste Kommando interpretiert wird.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
SLEEP 3000000
```

Wartet 3 Sekunden.

Anwendungen

Bremsen von Demonstrationsscripts.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.1.32 PLANETRANS

Internes Kommando - Benutzung nur durch PLANEVIEW .

A.2 Modul: Multivolume

Dieses Modul ist noch experimentell und gestattet die Handhabung mehrerer Volumina innerhalb einer VOXREN -Instanz. Diese Funktionalität wird genutzt zur Darstellung bewegter Volumina und soll ausgebaut werden, um auch eine gleichzeitige Darstellung multimodaler Volumina zu ermöglichen.

Das Modul stellt die Kommandos SETVOLUME, ADDVOLUME, DELVOLUME, VOL_OFFSET, VOL_SCALE und VOL_TURN zur Verfügung.

Eine Dokumentation dieser Funktionen erscheint zum gegenwärtigen Zeitpunkt noch nicht sinnvoll.

A.3 Modul: Marker

Dieses Modul beinhaltet Funktionen zur Veränderung von Markern. Marker sind in das gerenderte Bild eingeblendete Orientierungspunkte, z.B. die Positionen der Kameras anderer Views, Messmarkierungen oder auch manuell gesetzte Hinweise auf interessante Punkte.

A.3.1 MARKERADD

Synopsis

`MARKERADD` *name x y z* [*color*]

Beschreibung

Fügt einen Marker an Position *x y z* (**in Kamerakoordinaten!**) mit der Bezeichnung *name* hinzu.

Ist das optionale Argument *color* vorhanden, so gibt dieses die zu verwendende Farbe für den Marker an. Standardfarbe ist Cyan (0000 ffff ffff). Die Farbe wird als 16-bittiges Hex-Triplet angegeben.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`MARKERADD Koordinatenursprung 0 0 0`

Setzt einen Marker am Koordinatenursprung.

`MARKERADD RoterMarker 12 34 56 ffff 0 0`

Setzt einen roten Marker mit Namen „RoterMarker“ an der Position 12 34 56.

Anwendungen

Manuelles Setzen von Markierungen. Die Kamerakoordinaten erhält man mit dem Befehl CPOS (s. A.9.22).

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.3.2 MARKERDEL

Synopsis

`MARKERDEL` *name*`|*`

Beschreibung

Entfernt den Marker mit der Bezeichnung *name*. Mit `*` als Argument werden alle benutzerdefinierten Marker entfernt.

Insbesondere kann hiermit der „0cm“ Marker gelöscht werden, wodurch der Autopilot im Drop-Modus neu zu zählen beginnt.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`MARKERDEL Koordinatenursprung`

Entfernt den Marker „Koordinatenursprung“ .

`MARKERDEL *`

Entfernt alle benutzerdefinierten Marker.

Anwendungen

Entfernen von Markierungen, Neustart der Zählung des Autopiloten im Drop-Modus.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.3.3 MARKERSHOW

Synopsis

`MARKERSHOW` *name*`|*`

Beschreibung

Zeigt alle aktuell gesetzten Marker und deren Positionen an.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`MARKERSHOW *`

Zeigt alle aktuell gesetzten Marker an.

```
303-Marker RoterMarker 12.00 34.00 56.00 ffff 0000 0000.
303-Marker Koordinatenursprung 0.00 0.00 0.00 0000 ffff ffff.
303-Marker !EN 0.00 0.00 0.00 ffff 0000 0000.
303-Marker !ST 0.00 0.00 0.00 0000 ffff 0000.
303-Marker !Main 551.59 -249.10 -144.26 0000 ffff 8fff.
303 List ends.
```

Zeigt alle aktuellen Marker an. Marker mit ! sind Systemmarker. Rechts vom Markernamen ist die Position und danach die Farbe des Markers angegeben.

Anwendungen

Anzeigen der gesetzten Marker.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.
Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.4 Modul: multiview

A.4.1 SETVIEW

Synopsis

`SETVIEW` *name*

Beschreibung

Setzt einen neuen View als aktuellen View. Alle auf einen View wirkenden Kommandos (Renderingeinstellungen, Kamerabewegungen etc.) wirken von nun an auf diesen View.
Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform

ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`SETVIEW Main`

Aktiviert den Hauptview.

Anwendungen

Scripts, die Multiview-Szenarios aufbauen.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.4.2 ADDVIEW

Synopsis

`ADDVIEW` *name*

Beschreibung

Erzeuge einen neuen View als Klon des aktuellen Views. Der neu erzeugte View ist automatisch aktiv.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`ADDVIEW Outside`

Erzeugt einen neuen View und nennt ihn „Outside“ .

Anwendungen

Scripts, die Multiview-Szenarios aufbauen.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.4.3 ADDSLAVEVIEW

Synopsis

`ADDSLAVEVIEW` $d_{forward}$ d_{right} d_{top} α_{right} α_{top} α_{roll} *name*

Beschreibung

Erzeugt einen neuen View, der dem aktuellen View folgt. Der neu erzeugte View ist automatisch aktiv.

Ein Slave-View bewegt sich automatisch mit, wenn sich der zugehörige Master bewegt. Seine Kameraposition, Zoom und Schnittebenenparamater sind mit der des Masters verknüpft.

Dabei wird der Slave allerdings um $d_{forward}$ in Blickrichtung der Kamera, um d_{right} nach rechts (in Kamerakoordinaten) und um d_{top} nach oben verschoben und anschließend um α_{right}, α_{top} und α_{roll} gedreht.

Damit kann eine beliebig orientierte, starr mit dem Master verbundene Kamera erzeugt werden.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der

Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
addslaveview 100 0 0 180 0 0 Rearview
```

Erzeugt einen neuen View namens „Rearview“ , der entgegen der Kamerablickrichtung schaut, aber 100 Einheiten im Kamerakoordinatensystem vor der aktuellen Kamera schwebt.

Anwendungen

Scripts, die Multiview-Szenarios aufbauen, z.B. zur Erzeugung von „Colon-Flattening“-Darstellungen.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.4.4 DELVIEW

Synopsis

`DELVIEW` *name*

Beschreibung

Zerstört den View *name*.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`delview Rearview`

Zerstört den View „Rearview“.

Anwendungen

Scripts, die Multiview-Szenarios aufbauen.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.5 Modul: surgery

A.5.1 FILLSURF

Synopsis

`FILLSURF` *x y z newclass*

Beschreibung

Markiert die Zusammenhangskomponente, zu der der Voxel mit den Koordinaten *x y z* gehört mit der Klasse *newclass* (Umfärben von Zusammenhangskomponenten).

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`FILLSURF 145 17 32 21`

Markiert die Zusammenhangskomponente des Punktes bei `145 17 32` als Klasse 21 (Knochen).

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.5.2 KILL_TOP

Synopsis

`KILL_TOP`

Beschreibung

Markiert alle Zusammenhangskomponenten zu Klasse 0 um, die die obere Begrenzungsfläche des Volumens (z=0) berühren.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Anwendungen

Entfernen unerwünschter Außenräume bei der Segmentierung z.B. des Colons.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.5.3 CUTSIZE

Synopsis

`CUTSIZE` *size*

Beschreibung

Setzt die Größe des Freihand-Schneidewerkzeuges.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Anwendungen

Anpassung der Schnittbreite und -tiefe an die zu schneidenden Objekte.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6 Modul: parms

Dieses Modul dient der Einstellung aller wesentlichen Renderingparameter.

A.6.1 JUMP

Synopsis

`JUMP [?|`*cutplanepos*`]`

Beschreibung

Setzt die Position der vorderen Schnittebene.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`JUMP 128`

Setzt die vordere Schnittebene 128 Pixel (in Kamerakoordinaten) vor die Kamera.

Anwendungen

Zeigen einer Schnittebene bei Vorverarbeitungsscripts, interaktive Manipulation der Schnittebene durch VOXCONTROL.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.2 MAXDEPTH

Synopsis

`MAXDEPTH [?|`*maxrenderdepth*`]`

Beschreibung

Setzt die maximale Rendertiefe und damit die Position der hinteren Schnittebene.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der

Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`MAXDEPTH 256`

Setzt die hintere Schnittebene 256 Pixel (in Kamerakoordinaten) vor die Kamera.

Anwendungen

Beschleunigung des Renderings in komplexen langgestreckten Strukturen, interaktive Manipulation der Schnittebene durch VOXCONTROL.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.3 ZOOM

Synopsis

`ZOOM [?|`*zoomfactor*`]`

Beschreibung

Setzt den Zoomfaktor und damit implizit den Öffnungswinkel der Kamera.

Es ist $zoomfactor = 2 * \tan(\frac{\alpha}{2})$, wobei α der volle Öffnungswinkel der Kamera in Richtung der Bild**höhe** ist.

Bei nichtquadratischen Ansichtsauflösungen ist der Öffnungswinkel in Richtung der Bildbreite entsprechend erhöht.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der

Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`ZOOM 2`

Setzt den Öffnungswinkel auf 90°.

Anwendungen

Interaktive Manipulation der Schnittebene durch VOXCONTROL.

Einstellen des Öffnungswinkels für nahtlose Mehrfachansichten (via SLAVEVIEWs - s. A.4.3).

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.4 LIGHT

Synopsis

`LIGHT [?|`*lightdepth*`]`

Beschreibung

Setzt die maximal durch die Lichtquelle erreichbare Entfernung fest.

Objekte, die weiter entfernt sind, werden nicht mehr dargestellt, sie werden linear nach schwarz ausgeblendet.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`LIGHT 2000`

Stellt die Reichweite der Lichtquelle auf 2000 (Kamera-)Pixel.

Anwendungen

Interaktive Manipulation der Ausleuchtung durch VOXCONTROL. Beschleunigung des Renderings in komplexen langgestreckten Strukturen.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.5 LIGHTMODE

Synopsis

`LIGHTMODE [?|`*mode*`]`

Beschreibung

Setzt das Beleuchtungsmodell (Position und Typ der Lichtquelle).

VOXREN unterstützt aus Geschwindigkeitsgründen nur eine einzige Lichtquelle.

Gültige Werte sind zur Zeit:

CENTER Punktförmige Lichtquelle im Zentrum der Kamera.

TOPC Punktförmige Lichtquelle oberhalb (Top) der Kamera.

TLC Punktförmige Lichtquelle links oberhalb (Top Left) der Kamera.

TRC Punktförmige Lichtquelle rechts oberhalb (Top Right) der Kamera.

BLC Punktförmige Lichtquelle links unterhalb (Bottom Left) der Kamera.

BRC Punktförmige Lichtquelle rechts unterhalb (Bottom Right) der Kamera.

PARALLEL Paralleles Licht in Richtung der Kameraachse.

PARC Paralleles Licht in Richtung der Kameraachse plus punktförmige Lichtquelle im Zentrum der Kamera.

TUNNEL Spezieller „Tunnelmodus“, bei dem das Licht vor der Kamera platziert zu sein scheint.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
LIGHTMODE TLC
```

Stellt die Reichweite der Lichtquelle auf Top-Left-Center Modus.

Eine oben plazierte Lichtquelle erreicht ein etwas plastischeres, natürlicher wirkendes Bild (der Mensch ist es gewohnt, dass Objekte von schräg oben [durch die Sonne] beleuchtet werden).

Anwendungen

Interaktive Manipulation der Beleuchtung durch VOXCONTROL.

Bessere Beurteilung unklarer Strukturen durch Betrachten in verschiedenen Beleuchtungseinstellungen.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.6 GLOBALOPACITY

Synopsis

`GLOBALOPACITY [?|`*opacity*`]`

Beschreibung

Setzt die globale „Undurchsichtigkeit“ von in KE-Darstellung eingeblendeten Strukturen.
Hinweis: Explizit transparent markierte Strukturen sind von diesem Regler **nicht** betroffen, sie werden einzig von ihrer per `MKTRANSP` (s. A.10.4) zugewiesenen Transparenz beeinflusst.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`GLOBALOPACITY 2000`

Setzt den Standardwert für die Helligkeit der semitransparenten Strukturen.

`GLOBALOPACITY 0`

Schaltet die Sichtbarkeit der semitransparenten Strukturen temporär ab.

Anwendungen

Interaktive Manipulation der Darstellung durch VOXCONTROL.

Kontrasteinstellung bei der Verwendung der Kontrasteinlauf-Simulation.

Anpassung der Transparenz an die aktuell geforderte Aufgabe (gute Erkennbarkeit der transparenten Orientierungsstrukturen bzw. minimale Verdeckung der dahinterliegenden Struktur).

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.
Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.7 RENDERMODE

Synopsis

`RENDERMODE [?|`*mode*`]`

Beschreibung

Setzt den aktiven Rendermodus. Zur Zeit sind folgende Modi verfügbar. Eine genauere Beschreibung der Funktionsweisen finden Sie in 4.2.2

RGB RGB-Modus zur Darstellung von vorgerenderten Daten mit Textur im RGB-Format.

QRG Spezialmodus zur Darstellung alter Daten, die noch im QRG-Format vorliegen.

XYZ Standard-Rendermodus — Transparenzeffekte sind abgeschaltet.

XYT Transparenzmodus. Wie XYZ, aber mit Transparenzeffekten.

ZSH Z-Shading. Instant-Rendermodus, der keine Normaleninformation benötigt.

KE Kontrasteinlauf-Simulation. (Achtung - Modus kann sehr langsam sein.)

KES Kontrasteinlauf-Simulation mit Abbruch nach der ersten Oberfläche. Wird genutzt zur Einstellung von KE-Positionen.

SMN Experimenteller Modus (Voxelshaping).

GRY Graustufen-Modus. Alle Voxel werden als Grauwerte angezeigt.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`RENDERMODE XYT`

Aktiviert die Transparenzeffekte.

`RENDERMODE KE`

Schaltet in die Kontrasteinlauf-Simulation.

Anwendungen

Interaktive Manipulation der Darstellung durch VOXCONTROL.

Wahl von geeigneten Darstellungsmodi durch Scripte. Insbesondere bei der Verwendung des Multiskalen-Füllalgorithmus (s. 6.2), bei der ZSH gewählt werden sollte, um die Sperrbereiche zu visualisieren.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.8 FINE

Synopsis

`FINE [?|0|1]`

Beschreibung

(De-)aktiviert die exakte Oberflächenberechnung. Ohne diese kann es zu geringfügigen Verzerrungen der Darstellung der Ränder einzelner Voxel kommen.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
FINE 1
```

Aktiviert die exakte Oberflächenberechnung.

Anwendungen

Während bei interaktiver Manipulation die geringen Fehler, die durch die (deutlich schnellere) Standardmethode verursacht werden, nicht augenfällig werden, treten diese beim Aufzeichnen von Filmen störend zutage.

Ohne die exakte Oberflächenberechnung kommt es zu einer Interferenz des von der Kamera induzierten Abtastgitters mit dem Voxelgitter, das Moiree-Effekte verursacht, die erst bei langsamer, gleichmäßiger Bewegung der Kamera als wellenförmige Artefakte erkennbar werden.

Beim Aufzeichnen von Filmen wird daher empfohlen, die exakte Oberflächenberechnung einzuschalten.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.9 ALIAS

Synopsis

```
ALIAS [?|0-7]
```

Beschreibung

Schaltet die Nachbearbeitungsfilter zur Bildglättung ein. Es handelt sich beim übergebenen Wert um ein Bitfeld mit folgenden Bedeutungen:

1 Aktiviert den Medianfilter nach 4.2.4

2 Aktiviert den Aliasfilter nach 4.2.4

4 Aktiviert echte 3D-Interpolation - veraltete Option, nicht mehr voll funktionsfähig.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
ALIAS 3
```

Aktiviert alle Nachbearbeitungsfilter.

Anwendungen

Beim Rendern mit großen Auflösungen und wenn sich die Kamera sehr nah an Objekten befindet, können die Voxelkanten als störend empfunden werden.

Bei interaktiver Bewegung des Objektes fällt dies in der Regel nicht auf, für Standbilder (z.B. für Publikationen) ergeben gefilterte Bilder meist einen besseren optischen Eindruck.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.10 FLAGS

Synopsis

```
FLAGS [?|0-127]
```

Beschreibung

Schaltet diverse Optionen des Renderers ein bzw. aus. Es handelt sich beim übergebenen Wert um ein Bitfeld mit folgenden Bedeutungen:

1 Verwendet symmetrische Palette — es erfolgt keine Unterscheidung mehr nach „Vorder- und Rückseite“ einer Oberfläche.

2 Invertiert die Bedeutung von „Vorder- und Rückseite“ für Oberflächen.

4 Zeigt die Voxel, die von der vorderen Schnittebene angeschnitten werden, mit einer Sonderfarbe (s. A.1.10) hevorgehoben an.

8 Entfernt alle unnötigen Dateneinblendungen.

16 Unbenutzt.

32 Aktiviert die Markierung aller sichtbaren Voxel (s. Abb. 6.11).

64 Aktiviert die Markierung des Voxels an der aktuellen Kameraposition.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`FLAGS 32`

Aktiviert die Markierung aller sichtbaren Voxel. Damit kann z.B. demonstriert werden, was bei einer konventionellen Koloskopie nicht gesehen werden kann. Dazu wählt man zunächst mittels `COLORIZE` (s. A.1.6) eine Sonderfarbe, mit der alle im Folgenden gesehenen Voxel markiert werden. Nun schaltet man mit `FLAGS 32` die Markierung aller sichtbaren Voxel ein und lässt den Autopiloten einmal vorwärts durch den Darm fliegen.

Dabei werden alle gesehenen Voxel in der Sonderfarbe markiert. Dann schaltet man mit `FLAGS 0` dieses Umfärben wieder aus, dreht um und läßt den Autopiloten zurückfliegen. Die zuvor nicht gesehenen (da durch Falten etc. verdeckten) Bereiche erscheinen in der Originalfärbung, alles andere in der Sonderfarbe.

Anwendungen

Entfernung störender Farbsäume in Innenräumen bei hohen Öffnungswinkeln oder dünnen Wänden durch Auswahl der symmetrischen Palette.

„Umklappen“ der Palette bei invers segmentiereten Volumina (vgl. auch das Kommando `FLIPSIDES`, s. A.10.1).

Visualisieren der vorderen Schnittebene (übersichtlicher).

Visualisieren der Anteile, die nicht gesehenen wurden (s. Beispiel).

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.11 BGCOLOR

Synopsis

`BGCOLOR [?|`*r g b*`]`

Beschreibung

Setzt die Hintergrundfarbe. *r g b* gibt dabei in Hexadezimaldarstellung ein RGB-Farbtriplet mit 16 Bit Auflösung an.

Achtung: Einige Transparenzfunktionen (insbes. KE-Aufnahmen) liefern etwas „merkwürdige“ Ergebnisse, wenn der Hintergrund nicht schwarz ist, da die Transparenz als Farbfilter angelegt ist, um leicht iterativ berechnet werden zu können.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`BGCOLOR 2000 4000 2000`

Setzt einen dunkelgrünen Hintergrund.

Anwendungen

Anpassung des Hintergrundes an das Ausgabemedium - so wirkt z.B. ein weißer Hintergrund bei Papierausdrucken angenehmer.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.12 SIZE

Synopsis

`SIZE [?|`x`]`

Beschreibung

Setzt die zu berechnende Größe des aktuellen Views auf $x \times x$ Pixel.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`SIZE 256`

Setzt die Standardgröße.

Anwendungen

Anpassung des Bildformates an die Geschwindigkeits- bzw. Qualitätsansprüche.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.13 SIZEX

Synopsis

`SIZEX [?|`x`]`

Beschreibung

Siehe A.6.12, wirkt aber nur auf die X-Ausdehnung. Zusammen mit A.6.14 können so beliebige Formate eingestellt werden.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`SIZEX 320`

Setzt die Breite des Views auf 320 Pixel.

Anwendungen

Anpassung des Bildformates an die Geschwindigkeits- bzw. Qualitätsansprüche.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.14 SIZEY

Synopsis

`SIZEY [?|`*y*`]`

Beschreibung

Siehe A.6.12, wirkt aber nur auf die Y-Ausdehnung. Zusammen mit A.6.13 können so beliebige Formate eingestellt werden.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
SIZEY 240
```

Setzt die Höhe des Views auf 240 Pixel.

Anwendungen

Anpassung des Bildformates an die Geschwindigkeits- bzw. Qualitätsansprüche.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.15 SCALE

Synopsis

`SCALE [?|`*f*`]`

Beschreibung

Ändert die Vergrößerung, mit der der aktuelle View angezeigt wird. Damit ist es möglich, schnelles Rendering zu erreichen (durch kleine SIZE-Einstellungen, s. A.6.12), ohne dass die Bildschirmdarstellung zu klein wird.

Das skalierte Bild ist natürlich gröber aufgelöst als eines, das mit einer Größe von SIZE×SCALE gerendert wurde.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`SCALE 2`

Zeigt den aktuellen View in 2-facher Vergrößerung an.

Anwendungen

Anpassung des Bildformates an die Geschwindigkeits- bzw. Qualitätsansprüche.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.16 ASPECT

Synopsis

`ASPECT [?|`*ratio*`]`

Beschreibung

Ändert das Seitenverhältnis der Pixel, mit denen der aktuelle View angezeigt wird. Damit ist es möglich, auf Ausgabemedien mit nichtquadratischen Pixeln (z.B. auf SVCD) eine korrekte Ausgabe zu erhalten.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`SCALE 1.33`

Setze ein Pixel-Aspektverhältnis von 4:3 (NTSC-VideoCD).

Anwendungen

Anpassung an die Pixelgeometrie des Zielformates.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.17 LAYER

Synopsis

`LAYER` *class* `[`*hide*`]`

Beschreibung

Bestimmt, ob die Klasse *class* dargestellt *hide* = 0 oder versteckt *hide* = 1 werden soll und welche Schnittebenen auf die Klasse wirken sollen.

Der Wert *hide* ist ein Bitfeld mit folgenden Bedeutungen:

- 1 Invertiere den Sichtbarkeitszustand. Ohne aktive Schnittebenen bedeutet 0 sichtbar und 1 unsichtbar. Sind Schnittebenen aktiv, so wird deren Gesamtergebnis invertiert.
- 2 Aktiviere die Wirkung von Schnittebene Nummer 0.
- 4 Aktiviere die Wirkung von Schnittebene Nummer 1.
- 8 Aktiviere die Wirkung von Schnittebene Nummer 2.
- 16 Aktiviere den „Nebeleffekt“.

Im XYT-Modus erscheinen versteckte Layer semitransparent.

Ist der *hide*-Parameter nicht angegeben, so wird der aktuelle Zustand ausgelesen.

Die Schnittebenen werden mit Hilfe der Kommandos `CUTPLANEGFC` und `CUTPLANESET` (s. A.6.22 und A.6.23) gesetzt.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
LAYER 17 1
```

Versteckt alle Voxel mit Klasse 17.

Anwendungen

Ausblenden gerade uninteressanter Strukturen zur besseren Erkennbarkeit dahinter verborgener Bereiche.

Aktivierung des Nebeleffektes zur besseren dreidimensionalen Orientierung bei komplexen Strukturen innerhalb von umhüllenden Objekten (z.B. das Gefäßsystem in der Leber).

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.18 GCENTER

Synopsis

`GCENTER [?|`*center*`]`

Beschreibung

Stellt das Zentrum des darstellbaren Grauwertfensters auf *center* ein.

Voxel mit diesem Grauwert werden auf dem Bildschirm als Mittelgrau (50%) dargestellt. Siehe auch A.6.19.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
GCENTER 1000
```

Setzt das Zentrum des Grauwertfensters auf 1000.

Anwendungen

Einstellung des Grauwertfensters während der Betrachtung von Grauwertstrukturen, z.B. bei der Segmentierung.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.19 GWIDTH

Synopsis

`GWIDTH [?|`*width*`]`

Beschreibung

Stellt die Breite des darstellbaren Grauwertfensters auf *width* ein.

Voxel mit dem Grauwert *center-width* oder kleiner werden auf dem Bildschirm schwarz, Voxel mit dem Grauwert *center+width* oder größer weiß dargestellt. Dazwischen wird interpoliert. Siehe auch A.6.18.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`GWIDTH 400`

Setzt die Breite des Grauwertfensters auf 400.

Anwendungen

Einstellung des Kontrastes des Grauwertfensters während der Betrachtung von Grauwertstrukturen, z.B. bei der Segmentierung.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.20 GAMMA

Synopsis

`GAMMA [?|`*gamma*`]`

Beschreibung

Stellt die Gammakorrektur für den verwendeten Bildschirm ein.

Dieser Befehl sollte nur einmal während der Installation und Kalibrierung von VOXREN benötigt werden.

Dazu laden Sie die mitgelieferte Datei gammatest.16hl und stellen den Drehpunkt auf VolumeCenter und drehen sie mit einer 90°-Drehung in die Sichtebene.

Sie sollten ein Schwarz-Weiß-Raster mit einem hellen Bereich in der Mitte sehen.

Stellen Sie nun GWIDTH **und** GCENTER auf 2048.

Experimentieren Sie nun mit dem Wert von Gamma (typ. im Bereich von 1.0-3.0, bei gammakorrigierten Monitoren evtl. auch um 1.0 herum), bis beide Flächen gleich hell erscheinen.

Sie sollten denselben Wert erhalten wie auch bei der Kalibrierung von PLANEVIEW.

Der ermittelte Wert sollte in $ECCET_DIR/lib/eccet/config/voxren eingetragen werden:

`Display/Gamma: 1.13`

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`GAMMA 1.0`

Schaltet die interne Gammaanpassung aus.

Anwendungen

Kalibrierung der Darstellung. **ACHTUNG**: Eine inkorrekte Kalibrierung kann fehlerhafte Filterergebnisse vortäuschen. So kann z.B. ein Glättungsfilter scheinbar die Gesamthelligkeit verändern.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.21 GREYMARK

Synopsis

`GREYMARK [?|`*set from to*`]`

Beschreibung

Markiert einen Grauwertbereich **optisch**. Dazu sind bis zu 7 Grauwertbereiche (durch *set*=0..6) einstellbar, die in jeweils verschiedenen Farben erscheinen.

Dies verursacht **keine** Änderung der Voxelklasse für die betroffenen Voxel.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
GREYMARK 1 800 1200
```

Markiert einen Grauwertbereich grün.

Anwendungen

Schnelle Voransicht von grauwertbasierten Segmentierungen.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.22 CUTPLANEGFC

Synopsis

`CUTPLANEGFC [?|`*plane*`]`

Beschreibung

Setze eine der drei möglichen frei definierbaren Schnittebenen auf die aktuelle vordere Kameraschnittebene.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`CUTPLANEGFC 0`

Setzt Schnittebene 0.

Anwendungen

Aufschneiden von einzelnen Voxelklassen.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.6.23 CUTPLANESET

Synopsis

`CUTPLANESET [?|`*plane normvector constant*`]`

Beschreibung

Setze eine der drei möglichen frei definierbaren Schnittebenen so, dass die Ebene die Normalengleichung $\vec{x} \cdot \vec{n} = c$ mit $n =$*normvector* und $c =$*constant* erfüllt.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
CUTPLANEGFC 0 1 0 0 128
```

Setze Schnittebene 0 so, dass sie parallel zu y- und z-Achse ausgerichtet ist und die x-Achse bei 128 schneidet.

Anwendungen

Aufschneiden von einzelnen Voxelklassen.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.7 Modul: stereo

A.7.1 EYE

Synopsis

`EYE [?|`*distance*`]`

Beschreibung

Setzt die Entfernung der beiden Kameras (für rechtes bzw. linkes Auge) bei der Erzeugung von Stereobildern (in Kamerakoordinaten).

Tipp: Durch Setzen eines negativen Abstandes können rechtes und linkes Auge vertauscht werden - nützlich für Brillen mit anderer Anordnung der Farbfilter.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`EYE 30`

Setzt den Standardwert für den Augenabstand.

Anwendungen

Einstellung der Tiefe des räumlichen Eindrucks.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.7.2 EYEDEG

Synopsis

`EYEDEG [?|`*degrees*`]`

Beschreibung

Setzt den „Schielwinkel“ zwischen den beiden Kameras bei der Erzeugung von Stereobildern.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`EYEDEG 2`

Setzt den „Schielwinkel“ auf 2°.

Anwendungen

Einstellung der z-Position des Bildes. Es sollten nur Winkel von wenigen Grad verwendet werden - zu große Differenzen können vom Auge i.A. nicht mehr sauber fusioniert werden.

Achten Sie aus dem gleichen Grund auch darauf, dass die Vorzeichen der Werte `EYE` und `EYEDEG` gleich sind bzw. `EYEDEG` höchstens **geringfügig** in die andere Richtung weist.

Am besten erfolgt die Einstellung des Schielwinkels interaktiv. Als Startwert stellt man diesen Parameter so ein, dass rechtes und linkes Bild in etwa übereinanderliegen. Danach kann durch geeignetes Verändern dieses Parameters das Gesamtobjekt in der Tiefe verschoben werden.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.8 Modul: nlg

A.8.1 FILTER

Synopsis

`FILTER [?|`*typ sx sy sz sw fx fy fz fw*`]`

Beschreibung

Wendet eine nichtlineare Gaußfilterung bzw. einen nichtlinearen Differenzenfilter auf den geladenen Bildstapel an.

Dabei kann *typ* folgende Werte annehmen:

0 NLG - Nichtlineare Gaußfilterung nach [NLG] und [NLG2]. Die Parameter *sx*, *sy* und *sz* geben die Sigmawerte für die drei Raumachsen an, *sw* den für die Werteachse. Die Werte *fx*, *fy* und *fz* geben den Faktor an, um den das Operatorfenster größer als das jeweils zugehörige Sigma ist.

1 DIFF - Nichtlinearer Differenzenfilter. Hier gibt *sw* den Totbereich des Filters an und *fw* den Verstärkungsfaktor. Das Ergebnis wird zentriert auf 32768 ausgegeben.

2 DIFFADD - Wie 1, aber das Ergenis wird auf das Grauwertbild addiert.

3 POSDIFFADD - Wie 2, aber es werden nur positive Differenzen addiert.

4 NEGDIFFADD - Wie 2, aber es werden nur negative Differenzen addiert.

Hinweis: Quellpunkte mit Grauwert 65535 werden **ignoriert** (s. A.1.16). Hiermit ist eine große Beschleunigung des Filtervorganges möglich, wenn nur bestimmte Bereiche gefiltert werden müssen.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`NLG 0 2 2 2 32 2 2 2`

Führt eine nichtlineare Gaußfilterung durch, wobei die Ortssigmawerte alle 2 sind und das Wertesigma 32. Das Operatorfenster ersteckt sich jeweils 4 Pixel (2×2) vor bzw. hinter den aktuellen Pixel.

Anwendungen

Filterung verrauschter Grauwertdaten.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.
Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.8.2 NLGS

Synopsis

`NLGS [?|`*axis* σ σ_w *f*`]`

Beschreibung

Wendet eine separierte nichtlineare Gaußfilterung auf den geladenen Bildstapel an.
Dabei bedeutet *axis* die Achse, in der gefiltert wird (0=x-, 1=y- und 2=z-Achse), σ das Ortssigma und σ_w das Wertesigma.
Der Wert *f* gibt den Faktor an, um den das Operatorfenster größer als σ ist.
Hinweise:

- Quellpunkte mit Grauwert 65535 werden **ignoriert** (s. A.1.16). Hiermit ist eine große Beschleunigung des Filtervorganges möglich, wenn nur bestimmte Bereiche gefiltert werden müssen.
- Vor einer Anwendung von NLGS-Kette sollte das Kommando SAVEORI (s. A.8.3) aufgerufen worden sein, wenn Aufrufe zum Kommando MIXORI (s.

A.8.4) erfolgen sollen.

- Am Ende einer NLGS-Kette muss das Kommando DESTROYORI (s. A.8.5) aufgerufen werden, wenn zuvor SAVEORI benutzt wurde.
- Die Filterung sollte in verschiedenen Reihenfolgen bzgl. der Hintereinanderausführung der drei Einzelschritte (x/y/z) in den einzelnen Stufen erfolgen, um richtungsabhängige Artefakte zu vermeiden.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
NLGS 0 2 32 2
```

Führt eine separierte nichtlineare Gaußfilterung in der x-Achse durch, wobei das Ortssigma 2 und das Wertesigma 32 ist. Das Operatorfenster ersteckt sich jeweils 4 Pixel (2×2) vor bzw. hinter das aktuelle Voxxel.

Anwendungen

Filterung verrauschter Grauwertdaten.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.
Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.8.3 SAVEORI

Synopsis

```
SAVEORI [?]
```

Beschreibung

Speichert die Originalgrauwerte zwischen, bevor eine separierte nichtlineare Gaußfilterung (s. A.8.2) auf den geladenen Bildstapel angewendet werden kann.

Achtung: Diese Operation zerstört evtl. bereits getroffene Klassenzuordnungen und die GDA.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`SAVEORI`

Speichert die Originalgrauwerte zwischen.

Anwendungen

Filterung verrauschter Grauwertdaten - s. A.8.2.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.8.4 MIXORI

Synopsis

`MIXORI [?|`*rate*`]`

Beschreibung

Mischt die Originalgrauwerte zum bisherigen Zwischenergebnis der separierten nichtlinearen Gaußfilterung mit einer Gewichtung von *rate* (0-1) hinzu.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
MIXORI 0.3
```

Mischt die Originalgrauwerte mit 30% Gewicht zum bisherigen Zwischenergebnis hinzu.

Anwendungen

Filterung verrauschter Grauwertdaten - s. A.8.2.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.8.5 DESTROYORI

Synopsis

```
DESTROYORI [?]
```

Beschreibung

Bringt den Zwischenspeicher für die Originalgrauwerte (die Bereiche, die eigentlich für Klassenzuordnung und GDA reserviert sind) nach dem Ablauf einer NLGS-Kette (s. A.8.2) auf einen definierten Wert (GDA=1 und Klasse=0 auf allen Punkten).

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der

Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`DESTROYORI`

Setzt den Zwischenspeicher zurück.

Anwendungen

Filterung verrauschter Grauwertdaten - s. A.8.2.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9 Modul: move

Enthält alle Steuerbefehle zum Bewegen der Kamera.

A.9.1 STEP

Synopsis

`STEP [?|`*speed*`]`

Beschreibung

Ändert die Geschwindigkeit, mit der manuelle Bewegungen (Maus/Tastatur) und der Autopilot die Kamera bewegen. Standardwert ist 1.

Es wird nicht empfohlen, Werte größer 1.0 zu verwenden, wenn der Autopilot in engen Umgebungen navigiert, da die Gefahr besteht, dass Wände durchschritten werden.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`STEP 0.3`

Setzt die Schrittweite auf 0.3.

Anwendungen

Steuerung der Tastatur-/Mausempfindlichkeit.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.2 FWD

Synopsis

`FWD [?|`*howfar*`]`

Beschreibung

Bewegt die Kamera um *howfar* Pixel (in Kamerakoordinaten) vorwärts.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`FWD 1`

Bewegt die Kamera um 1 Pixel (in Kamerakoordinaten) vorwärts.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.3 REV

Synopsis

`REV [?|`*howfar*`]`

Beschreibung

Bewegt die Kamera um *howfar* Pixel (in Kamerakoordinaten) rückwärts.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`REV 1`

Bewegt die Kamera um 1 Pixel (in Kamerakoordinaten) rückwärts.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.4 RIGHT

Synopsis

`RIGHT [?|`*howfar*`]`

Beschreibung

Bewegt die Kamera um *howfar* Pixel (in Kamerakoordinaten) nach rechts.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
RIGHT 1
```

Bewegt die Kamera um 1 Pixel (in Kamerakoordinaten) nach rechts.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.5 LEFT

Synopsis

`LEFT [?|`*howfar*`]`

Beschreibung

Bewegt die Kamera um *howfar* Pixel (in Kamerakoordinaten) nach links.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`LEFT 1`

Bewegt die Kamera um 1 Pixel (in Kamerakoordinaten) nach links.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.6 UP

Synopsis

`UP [?|`*howfar*`]`

Beschreibung

Bewegt die Kamera um *howfar* Pixel (in Kamerakoordinaten) nach oben.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
UP 1
```

Bewegt die Kamera um 1 Pixel (in Kamerakoordinaten) nach oben.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.7 DOWN

Synopsis

`DOWN [?|`*howfar*`]`

Beschreibung

Bewegt die Kamera um *howfar* Pixel (in Kamerakoordinaten) nach unten.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`DOWN 1`

Bewegt die Kamera um 1 Pixel (in Kamerakoordinaten) nach unten.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.8 TMODE

Synopsis

`TMODE [?|`*mode*`]`

Beschreibung

Setzt den Drehpunkt, um den Drehungen ausgeführt werden. Gültige Modi sind zur Zeit:

Camera Drehungen erfolgen um die Kamera. Normalmodus.

VolCenter Drehungen erfolgen um das Zentrum des geladenen Volumens. Gut für Außenansichten.

Cutplane Drehungen erfolgen um den Mittelpunkt der vorderen Schnittebene — nützlich, um angeschnittene Objekte von allen Seiten zu betrachten.

PicCenter Drehungen erfolgen um das im Bildzentrum dargestellte Objekt.

Keep Behalte einen mit TPOINT gesetzten Drehpunkt bei.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`TMODE VolCenter`

Künftige Drehungen erfolgen um das Zentrum des geladenen Volumens.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.9 TPOINT

Synopsis

`TPOINT [?|`x y z`]`

Beschreibung

Setzt den Drehpunkt, um den Drehungen ausgeführt werden, explizit auf einen Punkt.

Achtung: Zuvor sollte `TMODE KEEP` (s. A.9.8) gesetzt werden.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`TPOINT 0 0 0`

Setzt den Drehpunkt auf den Koordinatenursprung.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.10 TRIGHT

Synopsis

`TRIGHT [?|`*degrees*`]`

Beschreibung

Dreht die Kamera um *degrees* Grad nach rechts. Der Drehpunkt (s. A.9.8 und A.9.9) wird beachtet.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`TRIGHT 5`

Dreht die Kamera um 5 Grad nach rechts.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.11 TLEFT

Synopsis

`TLEFT [?|`*degrees*`]`

Beschreibung

Dreht die Kamera um *degrees* Grad nach links. Der Drehpunkt (s. A.9.8 und A.9.9) wird beachtet.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`TLEFT 5`

Dreht die Kamera um 5 Grad nach links.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.12 TUP

Synopsis

`TUP [?|`*degrees*`]`

Beschreibung

Dreht die Kamera um *degrees* Grad nach oben. Der Drehpunkt (s. A.9.8 und A.9.9) wird beachtet.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`TUP 5`

Dreht die Kamera um 5 Grad nach oben.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.13 TDOWN

Synopsis

`TDOWN [?|`*degrees*`]`

Beschreibung

Dreht die Kamera um *degrees* Grad nach unten. Der Drehpunkt (s. A.9.8 und A.9.9) wird beachtet.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`TDOWN 5`

Dreht die Kamera um 5 Grad nach unten.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.14 RLEFT

Synopsis

`RLEFT [?|`*degrees*`]`

Beschreibung

Dreht die Kamera um *degrees* Grad längs ihrer Achse nach links (rollen).

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
RLEFT 5
```

Dreht die Kamera um 5 Grad längs ihrer Achse nach links.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.15 RRIGHT

Synopsis

`RRIGHT [?|`*degrees*`]`

Beschreibung

Dreht die Kamera um *degrees* Grad längs ihrer Achse nach rechts (rollen).

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`RRIGHT 5`

Dreht die Kamera um 5 Grad längs ihrer Achse nach rechts.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.16 UTURN

Synopsis

`UTURN [?]`

Beschreibung

Dreht die Kamera um 180°. Eine eventuell gesetzte vordere Schnitteben wird dabei beachtet, so dass die Kameraposition quasi an der Schnittebene gespiegelt wird.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
UTURN
```

Dreht die Kamera um.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.17 CTRIGHT

Synopsis

```
CTRIGHT [?]
```

Beschreibung

Dreht die Kamera um *degrees* Grad nach rechts. Im Gegensatz zum Kommando TRIGHT (s. A.9.10) liegt der Drehpunkt immer in der Kamera.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform

ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
CTRIGHT 5
```

Dreht die Kamera um 5 Grad nach rechts.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.18 CTLEFT

Synopsis

```
CTLEFT [?]
```

Beschreibung

Dreht die Kamera um *degrees* Grad nach links. Im Gegensatz zum Kommando TLEFT (s. A.9.11) liegt der Drehpunkt immer in der Kamera.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
CTLEFT 5
```

Dreht die Kamera um 5 Grad nach links.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.19 CTUP

Synopsis

```
CTUP [?]
```

Beschreibung

Dreht die Kamera um *degrees* Grad nach oben. Im Gegensatz zum Kommando TUP (s. A.9.12) liegt der Drehpunkt immer in der Kamera.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
CTUP 5
```

Dreht die Kamera um 5 Grad nach oben.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.20 CTDOWN

Synopsis

```
CTDOWN [?]
```

Beschreibung

Dreht die Kamera um *degrees* Grad nach unten. Im Gegensatz zum Kommando TDOWN (s. A.9.13) liegt der Drehpunkt immer in der Kamera.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
CTDOWN 5
```

Dreht die Kamera um 5 Grad nach unten.

Anwendungen

Kamerabewegung per Script, VOXCONTROL.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.21 POS

Synopsis

`POS [?|`*x y z*`]`

Beschreibung

Positioniert die Kamera an der **Voxel**position *x y z*.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
POS 0 0 0
```

Positioniert die Kamera am Ursprung des Voxelkoordinatensystems.

Anwendungen

Kamerabewegung per Script, VOXCONTROL, Verbindung zu PLANEVIEW.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.22 CPOS

Synopsis

`CPOS [?|`$x\ y\ z$`]`

Beschreibung

Positioniert die Kamera an der Position $x\ y\ z$ im **Kamera**koordinatensystem.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
CPOS 0 0 0
```

Positioniert die Kamera am Ursprung des Kamerakoordinatensystems.

Anwendungen

Kamerabewegung per Script, VOXCONTROL, Verbindung zu PLANEVIEW.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.9.23 DIR

Synopsis

`DIR [?|`$dir_x\ dir_y\ dir_z\ right_x\ right_y\ right_z$`]`

Beschreibung

Setzt die Kameralage. Der Vector dir_x dir_y dir_z bestimmt die Blickrichtung der Kamera, der Vektor $right_x$ $right_y$ $right_z$ weist zur rechten Seite der Kamera.

Sind die beiden Vektoren nicht orthogonal, wird der Rechtsvektor automatisch orthogonalisiert.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
DIR 0 0 1 0 1 0
```

Positioniert die Kamera so, dass sie in Richtung der z-Achse blickt und dass ihre rechte Seite in Richtung der y-Achse weist.

Anwendungen

Kamerabewegung per Script, VOXCONTROL, Verbindung zu PLANEVIEW.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.10 Modul: remark

Beinhaltet diverse Algorithmen, um Voxeln nach bestimmten Kriterien neuen Klassen zuzuordnen.

A.10.1 FLIPSIDES

Synopsis

`FLIPSIDES [?|`*class*`]`

Beschreibung

Dreht die Normalen der Voxel mit Klasse *class* um 180°.

Anmerkung: Diese Operation wird auf den bereits quantisierten Daten durchgeführt und ist daher verlustfrei.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`FLIPSIDES 16`

Dreht die Normalen der Voxel mit Klasse 16.

Anwendungen

Anpassung invers segmentierter Oberflächen an existente Bilder bzw. die gewohnte Darstellung.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.10.2 REMARK

Synopsis

`REMARK [?|`*from to*`]`

Beschreibung

Ändert die Klasse aller Voxel, die bisher in Klasse *from* liegen, auf *to*.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
REMARK 16 17
```

Ändert die Klasse aller Voxel, die bisher zu Klasse 16 gehören, auf 17.

Anwendungen

Umfärben von segmentierten Daten zur Anpassung an die gewünschte Farbgebung, Sicherung von bereits segmentierten Bereichen gegen unerwünschte Ergänzung in weiteren Schritten.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.10.3 HULL

Synopsis

`HULL [?|`*from to dist target*`]`

Beschreibung

Generiert alle Verbindungslinien zwischen Voxeln der Klassen *from* und *to*, die nicht weiter als *dist* auseinanderliegen, und markiert sie als Klasse *target*.

Sind *from* und *to* identisch, so wird eine Art „partielle konvexe Hülle“ der Voxel in Klasse *from* gebildet.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
HULL 2 2 5 4
```

Berechnet die „partielle konvexe Hülle“ der Klasse 2 mit einer „Fangweite“ von 5. Das Ergebnis erhält Klasse 4.

Anwendungen

Verdichten dünner Markierungen, Setzen manueller Markierungen und lineare Ergänzung zu 3D-Gebilden.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.10.4 MKTRANSP

Synopsis

`MKTRANSP [?|`*class transparency*`]`

Beschreibung

Weist einer Voxelklasse (*class*) eine bestimmte Transparenz (*transparency*) zu.

Achtung: Dieses Kommando **zerstört** eine ggf. zuvor vorhandene Grauwert- oder Normaleninformation. Alle Voxel der angegebenen Klasse erscheinen fortan im XYZ-Modus von allen Seiten voll opak bzw. im XYT-Modus mit der angegebenen Transparenz.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
MKTRANSP 16 50
```

Weist allen Voxeln mit Klasse 16 die Transparenz 50% zu.

Anwendungen

Markierung semitransparenter Oberflächen, z.B. Flüssigkeitsoberflächen.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.10.5 FINDFORK

Synopsis

`FINDFORK [?|`*x y z newclass*`]`

Beschreibung

Findet Verzweigungspunkte.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`FINDFORK 0 0 0 17`

Findet Verzweigungen ab Punkt 0 0 0 - Verzweigungen werden mit Farbe 17 markiert.

Anwendungen

Zerlegung von sich verzweigenden Strukturen (z.B. Gefäßbäume).

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.10.6 FINDFORK2

Synopsis

`FINDFORK2 [?|`*old new radius*`]`

Beschreibung

Alternativer Algorithmus zum Finden von Verzweigungspunkten. *old* gibt die Voxelklasse an, die zerlegt werden soll, *new* die Farbe, in der Verzweigungen dargestellt werden sollen, und *radius* gibt den Radius der Gefäße an, die durchgehend erhalten bleiben sollen.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform

ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`FINDFORK2 16 17 20`

Findet Verzweigungen in Voxelklasse 16, markiert sie mit 17. Radiuseinstellung ist 20 Voxel.

Anwendungen

Zerlegung von sich verzweigenden Strukturen (z.B. Gefäßbäume).

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.10.7 FINDFORK3

Synopsis

`FINDFORK3 [?|`*old new [tolerance]*`]`

Beschreibung

Findet Verzweigungspunkte. Alternativer (in der Regel bester) Algorithmus. *old* gibt die Voxelklasse an, die zerlegt werden soll, *new* die Farbe, in der Verzweigungen dargestellt werden sollen, und *tolerance* gibt die Toleranz für kleine Abweichungen an.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`FINDFORK3 16 17 4`

Findet Verzweigungen in Voxelklasse 16, markiert sie mit 17. Toleranzeinstellung ist 4 Schritte.

Anwendungen

Zerlegung von sich verzweigenden Strukturen (z.B. Gefäßbäume).

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.10.8 KILLSIDE

Synopsis

`KILLSIDE [?|`*axis stopat outside border dist*`]`

Beschreibung

Markiert ausgehend von jeweils zwei gegenüberliegenden Seitenflächen (längs der Achse, die in *axis* angegeben ist) den „Außenbereich“ in der Farbe *outside*. Dabei wird ein Abstand von *dist* von der Farbe *stopat* gelassen. Der Bereich zwischen *outside* und *stopat* wird mit Farbe *border* gefüllt.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

`KILLSIDE 0 2 4 6 10`

Färbt den Außenraum in Richtung der x-Achse mit Klasse 4, bis der Algorithmus auf Voxel der Klasse 2 trifft. Die 10 Voxel vor dem Auftreffpunkt werden in Klasse 6 markiert.

Anwendungen

Schnelle grobe Entfernung großer Außenstrukturen (Luftraum).

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.
Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.10.9 REMARKNORM

Synopsis

`REMARKNORM [?|`*from to vector maxangle*`]`

Beschreibung

Überprüft die Normalen der Voxel der Klasse *from*. Liegt die Normale eines Voxels innerhalb von *maxangle* Grad um den Vektor *vector*, so wird es der Klasse *to* zugeordnet.
Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
REMARKNORM 16 17 0 1 0 10
```

Markiert Voxel der Klasse 16 um zu Klasse 17, falls deren Normalen bis auf 10° in Richtung der y-Achse zeigen.

Anwendungen

Klassifizierung von Voxeln mit bestimmten Normalen, z.B. als Ausschlussverfahren bei der Bestimmung von Flüssigkeitsoberflächen.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.10.10 CLASSMAX

Synopsis

`CLASSMAX [?|`*class*`]`

Beschreibung

Entfernt Trennlinien, wie sie z.B. von `FINDFORK3` erzeugt werden, wieder aus dem Volumen, indem die Klasse jedes Voxels, das zu *class* gehört, durch diejenige Klasse ersetzt wird, die in der unmittelbaren Umgebung die meisten Mitglieder hat.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
CLASSMAX 242
```

Entfernt Trennstriche, die die Klasse 242 besitzen.

Anwendungen

Entfernung der Trennungen nach einer mit Hilfe von `FINDFORK3` erstellten Segmentierung.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.11 Modul: polypen

A.11.1 MINDIST

Synopsis

`MINDIST [?|`*remark from to threshold mindist*`]`

Beschreibung

Markiert Voxel der Klasse *from* um zu *to*, falls sich innerhalb einer Umgebung von *mindist* ein Voxel findet, das einen Grauwert größer als *threshold* hat.

Ist *threshold* negativ, so wird nach einem Voxel gesucht, dessen Grauwert **kleiner** als der Betrag von *threshold* ist.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
MINDIST 242 241 250 4
```

Markiert Voxel der Klasse 242 um zu Klasse 241, falls sich in deren Umgebung innerhalb von 4 Pixeln ein Grauwert größer 250 findet.

Anwendungen

Klassifizierung von gefundenen Polypen als vermutliche „false positives“, die von Spiral-CT-Artefakten (Schlieren) herrühren.

Rückgabewert

Wird als Parameter `?` übergeben, werden die möglichen Parameter in Kurzform ausgegeben.
Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.11.2 MARKPOLY

Synopsis

`MARKPOLY [?|`*from to zmax rmax*`]`

Beschreibung

Sucht innerhalb der Voxel mit Klasse *from* nach Polypen gemäß dem Algorithmus in 7.1 mit den Größenparametern *zmax* und *rmax* (Bedeutung s. Algorithmusbeschreibung). Gefundene Polypen werden in Farbe *to* markiert.
Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem `?` als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Beispiele

```
MARKPOLY 16 241 -20 60
```

Findet Polypen in Voxelklasse 16, die mindestens 2.0 Voxel hoch sind und sich innerhalb dieser Höhe um nicht mehr als 6.0 Voxel ausbreiten. Gefundene Polypen werden rot (Klasse 241) markiert.

Anwendungen

Finden von Polypen und Divertikeln sowie geometrisch analogen Strukturen.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

A.11.3 OLDMARKPOLY

Synopsis

`OLDMARKPOLY [?|`*newcol cupradius cupheight*`]`

Beschreibung

Veraltet. Bitte nicht mehr benutzen.

Ohne Parameter erfolgt eine Fehlermeldung, da dieses Kommando Parameter erfordert. Mit einem ? als Parameter werden die möglichen Parameter in Kurzform ausgegeben. Ansonsten werden die Parameter überprüft und im Erfolgsfall wird der Befehl ausgeführt und mit einer Erfolgsmeldung (3xx) beendet oder ansonsten eine Fehlermeldung (5xx) erzeugt.

Rückgabewert

Wird als Parameter ? übergeben, werden die möglichen Parameter in Kurzform ausgegeben.

Ansonsten erhalten Sie bei korrekter Ausführung des Befehls eine INFO (3xx) Meldung über das Ergebnis, bei Fehlschlag einen entsprechenden Fehlercode (5xx).

Anhang B

Tastenreferenz zu Voxren

B.1 Interne Kommandos

Das Hauptprogramm von VOXREN implementiert einige Tastenbelegungen, die aufgrund ihrer Natur (z.B. Interaktion mit der Renderingengine) in recht direktem Zusammenhang mit dem Programm stehen und nicht austauschbar sind.

Taste	**Bedeutung**	**Merkhilfe**
R	View zurücksetzen	**R**eset
SPACE BAR	Aktuelles Bild erneut rendern	
M	Format des abzuspeichernden Bildes/Films wählen	**m**ovie
⇑ Shift + M	Starte/Stoppe Aufzeichung von Bildern/Film.	**M**ovie
Strg + M	Schreibe 25 identische Bilder (1 Sekunde Pause beim Abspielen)	**m**ovie
⇑ Shift + Q	Beende Programm	**Q**uit
⇑ Shift + S	Speichere aktuelles Volumen	**S**ave
C	Berechne die unoptimierte GDA neu	**C**alculate
⇑ Shift + C	Berechne die **optimierte** GDA aus der aktuellen	**C**alculate
Strg + C	Setze die unoptimierte GDA aufgrund der aktuellen Sichtbarkeitsinformation (versteckt zzt. unsichtbare Klassen)	**C**alculate
I	Finde den kleinsten Würfel, der Oberflächen enthält, und zeige dessen Koordinaten an	**I**nner cube
Strg + A	3D-Smoothing an/aus	**A**liasfilter
X	Stoppe das Abspielen einer Positionsdatei	e**x**it playing
G	Hole Informationen zum aktuellen Pixel	**g**et Pixelinfo
H	Mache eine Voxelklasse unsichtbar	**H**ide

B.2 Modul: marker

Taste	Bedeutung	Merkhilfe
V	Schalte Sichtbarkeit von Markern um	**v**isibility
⇑ Shift + G	Messe Pixelabstände	**g**et pixel distances

B.3 Modul: multiview

Taste	Bedeutung	Merkhilfe
N	Nächster View	**N**ext view
⇑ Shift + V	Neuen View erzeugen	**V**iew create
Strg + V	View zerstören	**V**iew destroy

B.4 Modul: surgery

Taste	Bedeutung	Merkhilfe
⇑ Shift + H	Färbe Zusammenhangskomponente um	vgl. **H**ide
“	Schneide vertikal	
$	Schneide manuell - kleines Skalpell	
#	Schneide in der GDA - 1 Pixel	
⇑ Shift + I	Demarkiere - 1 Pixel	
⇑ Shift + J	Demarkiere - 5x5 Feld	

B.5 Modul: parms

Taste	Bedeutung	Merkhilfe
Pos1 oder 7	Bewege vordere Schnittebene vorwärts	
Ende oder 1	Bewege vordere Schnittebene rückwärts	
Einfg	Schalte Sichtbarkeit der Schnittebenenkante	
+	Zoom in - kleinerer Öffnungswinkel	
-	Zoom out - größerer Öffnungswinkel	
L	Licht heller	**L**ight
⇑ Shift + L	Licht dunkler	**L**ight
Strg + L	Lichtmodus umschalten	**L**ight
Strg + O	Schalte exakte Kantenberechnung ein/aus	**O**ptions
O	Schalte Kantenglättung ein/aus	**O**ptions
⇑ Shift + O	Schalte Oberflächenglättung ein/aus	**O**ptions
>	Vergrößere View	>
<	Verkleinere View	<
Y	Verringere Bildauflösung	
⇑ Shift + Y	Erhöhe Bildauflösung	
P	Schalte symmetrische Palette ein/aus	**P**alette
⇑ Shift + P	Schalte inverse Palette ein/aus	**P**alette
⇑ Shift + R	Schalte Rendermodus um	**R**endermode

B.6 Modul: stereo

Taste	**Bedeutung**	**Merkhilfe**
S	Schalte Stereomodus um	**S**tereo
E	Erhöhe Augenabstand	**E**ye
⇑ Shift + E	Verringere Augenabstand	**E**ye
D	Erhöhe Schielwinkel	**D**egree
⇑ Shift + D	Verringere Schielwinkel	**D**egree

B.7 Modul: move

Taste	Bedeutung	Merkhilfe
[5]	Schalte 90° bzw Schnellbewegungsmodus an	Keypad
[Bild ↑] oder [9]	Gehe vorwärts	Keypad
[Bild ↓] oder [3]	Gehe rückwärts	Keypad
[T]	Schalte Drehpunkt um	**T**urn
[⇑ Shift] + [T]	Setze festen Drehpunkt	**T**urn
[↑]	Drehe aufwärts	Cursor
[↓]	Drehe abwärts	Cursor
[→]	Drehe nach rechts	Cursor
[←]	Drehe nach links	Cursor
[/]	Rolle links	Keypad
[*]	Rolle rechts	Keypad
[0]	Umdrehen	Keypad
[6] oder [.]	Verschiebe nach rechts	Keypad
[4] oder [,]	Verschiebe nach links	Keypad
[8]	Verschiebe nach oben	Keypad
[2]	Verschiebe nach unten	Keypad
[Entf] oder [A]	Schalte Autopilotmodus um	Keypad

B.8 Modul: remark

Taste		Bedeutung	Merkhilfe
F9	oder	Fülle mit Multiskalen-Füllalgorithmus (6.2)	**B**FS
⇑ Shift + B			
F11	oder	Fixiere Klasse 6 (gelb) nach 2 (grün)	**F**ixate
⇑ Shift + F			
F12	oder	Undo - ändere Klasse 6 (gelb) nach 0 (grau)	**U**ndo
⇑ Shift + U			

Anhang C

Mausreferenz zu VOXREN

Bestimmte Funktionen wie die Bewegung im Volumen sowie Bearbeitungsfunktionen, die es erfordern, dass man ein Startvoxel angibt, sind bei VOXREN aus naheliegenden Gründen mit der Maus bedienbar.

C.1 Modul: marker

Mit der Kombination [⇑ Shift] + [Rechte Maustaste] können Abstände gemessen werden. Dazu klickt man die beiden zu vermessenden Punkte einfach mit der rechten Maustaste an, während man auf der Tastatur die linke Shifttaste festhält. Ein dritter Klick entfernt die Entfernungsmarker anschließend wieder.

C.2 Modul: surgery

Die virtuellen Chirurgiefunktionen stehen durch Drücken der Maustasten zusammen mit der Steuerungstaste (Control) auf der Tastatur zur Verfügung.

Drei Funktionen sind implementiert:

[Strg] + [Linke Maustaste] implementiert ein frei führbares Skalpell mit einer Klinge, deren Größe mit `CUTSIZE` eingestellt werden kann. Das Skalpell ist sensitiv auf die berührte Voxelklasse - andere Klassen werden nicht verletzt.

[Strg] + [Mittlere Maustaste] implementiert ein Werkzeug, das zusammenhangsgesteuerte Schnitte ermöglicht. Es führt einen bezüglich der aktuellen Kameraposition senkrechten Schnitt auf der Zusammenhangskomponente des angeklickten Punktes

innerhalb einer schmalen Ebene durch (s. 6.7 zur Funktionsweise).

[Strg] + [Rechte Maustaste] erlaubt die Umfärbung der angeklickten Zusammenhangskomponente auf die aktuelle mit `COLORIZE` eingestellte Farbe.

C.3 Modul: move

Dieses Modul implementiert die Bewegungsfunktionen. Die linke Maustaste erlaubt eine Bewegung vorwärts und rückwärts sowie Drehungen um die Kameraachse nach links und rechts.

Die mittlere Maustaste bewirkt Drehungen um den aktuellen Drehpunkt sowohl nach links und rechts als auch nach oben und unten.

Die rechte Maustaste wird verwendet, um nach rechts, links, oben und unten „auszuweichen“ (engl. strafe).

Des Weiteren kann durch gleichzeitiges Drücken von linker und mittlerer Taste der Autopilot ein- und ausgeschaltet werden sowie mit [⇑ Shift] + [Mittlere Maustaste] der Drehpunkt gesetzt werden.

Anhang D

Schnittstellen

D.1 Dateiformate

$\epsilon CC\epsilon\mathcal{T}$ verwendet eine Reihe einfacher Dateiformate, um Daten einzulesen oder auszugeben. In allen Fällen ist es möglich, die Dateien zusätzlich mit `gzip` zu komprimieren.

D.1.1 P5-Stapel

Die Importroutinen beherrschen das Einlesen von Portable Greymaps im Format `P5`, das in der Unixwelt weit verbreitet ist und in den manpages des netpbm Paketes von Jef Poskanzer dokumentiert ist.

Der Parser ist allerdings nicht 100Spezifikation in Übereinstimmung, liest aber alle von üblichen Tools geschriebenen `raw format pgms`.

Die Annahme ist, dass die Dateien folgenden Aufbau aufweisen:
`P5\n`
[`optionale Kommentare`]
sizex sizey`\n`
maxval`\n`
data

Dabei sind die Optionalen Kommentare von der Form „#[Zeichenfolge]\n“. *sizex sizey* geben an, wie groß das Bild in Pixeln ist, und *maxval* entscheidet, ob es sich um 16 oder 8 Bit/Pixel handelt.

Aus diesen Daten bestimmt sich das Format von *data*:

- Ist $maxval \geq 256$, so handelt es sich um 16 Bit/Pixel und ein Pixel belegt 2 Byte, die in der Folge Highbyte/Lowbyte in der Datei abgelegt sind.
- Die Daten sind in Zeilen zu *sizex* Pixel organisiert, die jeweils die einzelnen Pixel einer Zeile von links nach rechts enthalten.
- *data* besteht aus *sizey* solchen Zeilen, die ohne jegliche Trennzeichen hintereinander folgen. *data* hat also in C-Notation die Länge ($maxval \geq 256$? 2:1)**sizex***sizey*.

Nach der Spezifikation erlaubte folgende Bilder nach dem ersten Bild werden ignoriert.
VOXREN wertet beim Einlesen von P5-Dateien folgende Kommentare speziell aus:

- `# Dicom/(0028,0030)`
  Wird diese Sequenz am Beginn einer Kommentarzeile gefunden, so wird in der Zeile nach einem ':' (Trennzeichen) gesucht und zwei beliebige dahinter befindliche Fließkommazahlen, die durch ein nicht als Fließkommazahl interpretierbares Zeichen getrennt sind, eingelesen. Diese Zahlen werden als „PixelSpacing“ aufgefasst und gehen in die Darstellung der Voxel als Seitenverhältnis ein.
- `# Dicom/(0020,0032)`
  Diese Sequenz wird analog als Zahlentriplet interpretiert, das den Wert „ImagePositionPatient“ enthält. Aus dem Abstand der so gewonnen Positionen im ersten und jedem folgenden Bild werden die Bildabstände berechnet. Treten dabei mehrere verschiedene Abstände auf, so wird der zuletzt gefundene angenommen (da er vermutlich am genauesten ist) und eine entsprechende Warnung ausgegeben.

Alle weiteren Kommentare werden lediglich eingelesen und bei Speicheroperationen reproduziert.

D.1.2 16HL-Stapel

Aus historischen Gründen (P5 ist erst seit April 2000 in der Lage, 16-bittige Files zu verwalten) liest $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ auch Dateien, die als erste Zeile den Eintrag `16HL\n` haben.

Aufgrund von Fehlern in den vorhandenen Programmen wird in diesem Fall *maxval* ignoriert und es werden immer 16-bittige Daten angenommen.

D.1.3 3d32-Volumendateien

VOXREN liest zusätzlich ein proprietäres Format, in dem komplette Bildstapel in einer Datei abgelegt werden. Dies vereinfacht die Handhabung erheblich gegenüber Stapeln.

Das Dateiformat ist sehr einfach aufgebaut: `3D32\n`
sizex sizey sizez`\n`
[`Optionale Dateioptionen in einem MIME-ähnlichen Format`]
`\n`
data

Folgende Dateioptionen werden erkannt und ausgewertet:

- `Data/Representation: quantized`
  Muss immer gesetzt sein - sie dient der Unterscheidung zu einem alten Dateiformat mit anderem Voxelformat.

- `Data/Signature:`
  Kann gesetzt sein und erlaubt es, einzelne Dateien so zu signieren, dass sie auch auf nicht allgemein freigeschalteten Installationen von VOXREN betrieben werden können.

- `Voxel/Physicalsize:`
  Gibt die physische Größe von Voxel in mm in den drei Achsen an. Wird zur Berechnung von Abständen verwendet.

- `Image/Offset:`
  Gibt die Position des Nullpunktes des Volumens relativ zum Kamerakoordinatensystem wieder. Nützlich für abgeschnittene Volumina und Multivolume-Darstellung.

- `Image/Scale:`
  Gibt die Skalierung der drei Voxelachsen relativ zum Kamerakoordinatensystem an. Implementiert die korrekte Darstellung bei nichtisotropen Volumina (Voxel sind dann Quader, keine Würfel).

- Image/Turn:
  Gibt die Drehwinkel längs der drei primären Voxelachsen (in der Reihenfolge x, y, z ausgeführt) für das Volumen gegenüber dem Kamerakoordinatensystem an. Nützlich zum Matchen verschiedener Volumina bei Multivolumedarstellung.

- `Image/Physicalsize:`
  Veraltete Option, die `Voxel/Physicalsize:` und `Image/Scale:` vereint.

- `View/Position:`
  Gibt die Position der Kamera auf dem primären View an.

- `View/Direction:`
  Gibt die Blickrichtung Kamera an.

- `View/Topvector:`
  Gibt die Richtung des Vektors an, der bei der Kamera nach oben zeigt.

- `View/Rightvector:`
  Gibt die Richtung des Vektors an, der bei der Kamera nach rechts zeigt.

- `View/Zoom:`
  Gibt den Öffnungswinkel der Kamera codiert als $(\tan \frac{\alpha}{2})$ an.

- `View/Intersectionplane:`
  Gibt die Entfernung der aktiven Schnittebene von der Kamera an.

- `Voxren/Marker:`
  Definiert einen Marker. Syntax: `Voxren/Marker:` *Markername posx posy posz color*, wobei color ein Triplet von 16-bittigen hexadezimal codierten RGB-Werten ist.

Nach den Dateioptionen folgt eine Leerzeile als Trennmarkierung und danach folgen die eigentlichen Daten.

Diese sind in *sizez* z-Schichten angeordnet, die jeweils *sizey* Zeilen mit *sizex* Voxel enthalten. Einzelne Voxel belegen jeweils vier Byte, deren Inhalt an Tabelle D.1 ersichtlich wird.

Byte 1	Byte 2	Byte 3	Byte 4
RGB-Modus			
GDA. Eine GDA von 0 bezeichnet ein Oberflächenvoxel.	Rotanteil der Voxelfarbe	Grünanteil der Voxelfarbe	Blauanteil der Voxelfarbe
andere Modi			
GDA wie auch im RGB-Mode	Grauwertklasse **0-31**	Low-Byte des Grauwertes	High-Byte des Grauwertes
	Oberflächenvoxelklasse **32-255**	Low-Byte der quantisierten Normalen	High-Byte der quantisierten Normalen

Tabelle D.1: Aufbau eines Voxels in einer 3d32-Datei

D.2 Relevante Datenstrukturen

D.2.1 Zugriff auf die Volumendaten

Die C Includedatei voxel.h beschreibt die internen Speicherstrukturen für Volumendaten und bietet einige gängige Zugriffsmakros.

Im Folgenden werden die wichtigen Deklarationen in dieser Datei detailliert beschrieben:

```
typedef unsigned char        voxeldepth;
typedef unsigned char        voxelclass;
typedef unsigned short int   voxelnorm;
```

Die GDA belegt mit dem Typ unsigned char jeweils 1 Byte, ebenso wie die Voxelklasse. Der Grauwert bzw. die Normale belegt als unsigned short int zwei Byte.

```
typedef struct fullvox {
    voxeldepth vox;
    voxelclass class;
    voxelnorm  norm;
} voxelfull;
```

Diese Struktur findet Verwendung beim Schreiben und Lesen von 3d32-Daten auf/von Festplatte. Dort werden Voxel als Einheit dargestellt, um das Dateiformat und dessen Verständlichkeit zu vereinfachen und Operationen diekt auf den Dateien zu vereinfachen.

```
struct voxelfield {
```

Diese Struktur stellt ein Voxelvolumen im Speicher dar.

```
    struct voxelfield *next;
    char   name[32];
```

VOXREN ist in der Lage, mehrere Volumina gleichzeitig im Speicher zu halten (für bewegte Sequenzen oder multimodale Darstellungen). Daher ist die Struktur als verkettete Liste ausgelegt.

```
    int size[3];
    int mult[3];
```

Die Größe des Voxelvolumens ist in `size[3]` festgehalten, wobei `size[0]` die Ausdehnung in x-Richtung, 1 in y und 2 in z angibt. Das Hilfsfeld `mult[3]` dient der Berechnung von offsets in den eigentlichen Datenfeldern. Es enthält der Reihe nach size[0], size[0]· size[1] und size[0]· size[1]· size[2].

```
    voxeldepth  *depth;
    voxelclass  *class;
    voxelnorm   *norm;
```

Intern werden die einzelnen Felder getrennt gespeichert, da sich hierdurch oft bessere Zugriffsmuster für den Cache ergeben.

```
    int *multy,*multz;
```

Hilfsfelder zum schnellen Zugriff auf Ebenen und Zeilen. Der Zugriff selbst wird virtualisiert durch das Macro getvoxelindex().

```
volume_metadata *metadata;
```

Zusatzdaten zum Volumen (Kommentare etc.)

```
vector offset;
vector scale;
vector turn;
```

Transformation gegenüber dem Kamerakoordinatensystem. Virtualisiert durch die inline-Funktionen in localsys.h.

```
vector matrix[3],imatrix[3],tmatrix[3],itmatrix[3],nsmatrix[3];
```

Hilfsmatrizen für die Übergänge zwischen den Koordinatensystemen.

```
};
```

Von diesem Typ werden zwei Zeiger global deklariert:

```
extern struct voxelfield *curvoxelfield,*voxelfieldroot;
```

voxelfieldroot dient als Startpointer für die verkettete Liste. curvoxelfield gibt das „aktive“ Feld an, also das, auf das Operationen wirken.

```
#define getvoxelindex(x,y,z) ((x)+curvoxelfield->multy[(y)]+curvoxelfield->mult
#define getvoxeldepth(ind) (curvoxelfield->depth+(ind))
#define getvoxelclass(ind) (curvoxelfield->class+(ind))
#define getvoxelnorm(ind)  (curvoxelfield->norm~+(ind))
```

Virtualisierungsmacros zum Zugriff auf das Voxelfeld. Hierdurch wird die exakte Implementation vor der Anwendung verborgen, so dass leicht z.B. für andere Architekturen auf andere interne Darstellungen übergegangen werden kann.

```
void voxel_make_matrix(struct~voxelfield *vf);
```

Erstelle Hilfsmatrizen zum Koordinatenübergang. Muss aufgerufen werden, wenn die Parameter offset, scale oder turn verändert werden.

```
int voxelfield_create(char *name);
int voxelfield_select(char *name);
int voxelfield_destroy(char *name);
```

Erzeuge oder zerstöre ein Voxelfeld, bzw. setzte das angegebene Voxelfeld als aktuelles Feld.

Es folgen einige Macros, die das Erstellen von Routinen unterstützen, in denen SMP per Threads unterstützt werden soll. Insbesondere erlauben die Routinen den transparenten Übergang zu anderen Threadingmechanismen.

```
#define TH_SETUP_VARS(type)
```

Erzeuge Verwaltungsvariablen für die „Setup-Schleife“ einer Operation, die Threads benutzt. Dieses Macro muss direkt am Anfang einer solchen Funktion aufgerufen werden. Es erhält als Parameter einen Datentyp, der die Kommunikationsstruktur zu den Thread-Routinen darstellt. Dieser muss mindestens die Felder min, max und barofs als integer enthalten und darf ansonsten beliebige weitere Felder beherbergen. Dieser Aufruf erzeugt die Variablen x,step, thread[] und rargs. Diese Variablennamen dürfen nicht zusätzlich selbst deklariert werden.

```
#define TH_RUN_THREAD(name)
```

Erzeugt einen Thread. Das Argument gibt den aufzurufenden Funktionsnamen an.

```
#define TH_JOIN_THREADS
```

Warte auf alle bisher abgespalteten Threads.

`#define TH_SETUP_FOR`

Start einer Standard-Setuproutine. Die Threads werden in Z-Richtung verteilt.

`#define TH_SETUP_FOR_Y`

Alternative Variante mit Verteilung in Y-Richtung.

Innerhalb der gethreadeten Routine helfen folgende Macros bei der Ausführung:

`#define TH_FORZYX_SHOW(text)`

Durchlaufe das gesamte aktuelle Volumen in z-, y- und x-Richtung (innerste Schleife läuft in x-Richtung - cachefreundlichste Sequenz). Die Aufteilung auf die Threads wird automatisch berücksichtigt. Der Threadparameter muss in der Zeigervariablen `parm` verfügbar sein.

`#define TH_FORZY_X_SHOW(text)`

Analog mit rückwärts laufendem x.

`#define TH_FORZXY_SHOW(text)`

Analog mit Schleifenfolge zxy, alle Schleifen laufen aufwärts.

`#define TH_FORZX_Y_SHOW(text)`

Analog mit Schleifenfolge zxy, y läuft rückwärts.

`#define TH_FORYXZ_SHOW(text)`

Analog mit Schleifenfolge yxz, alle laufen vorwärts. Achtung - dies erfordert threadsetup mit `TH_SETUP_FOR_Y`.

```
#define TH_FORYX_Z_SHOW(text)
```

Analog mit Schleifenfolge yxz, z läuft negativ. Achtung - dies erfordert threadsetup mit `TH_SETUP_FOR_Y`.

```
#define TH_FORZYX_END
```

Beendet einen solchen Schleifenblock wieder. Alle Schleifenmacros laufen auf den (vom Benutzer zu deklarierenden) Variablen x, y und z.

D.2.2 Zusatzdaten zu Volumina - volume_metadata.h

```
typedef struct volume_metadata {
    vector image_physicalsize;
```

Physikalische Größe eines Voxels in mm.

```
    int commentsize;
    unsigned char *comments;
```

Kommentare, getrennt durch\n, abgeschlossen durch \0.

```
} volume_metadata;
```

Zur Bearbeitung der Kommentare (enthalten evtl. Einstellungen, Parameter etc.) stehen zwei einfache Funktionen zur Verfügung:

```
int volume_metadata_add_comment(volume_metadata *md,unsigned char *comment);
```

Füge Kommentar hinzu.

```
int volume_metadata_change_comment(volume_metadata *md,
                                   unsigned char *comment,int matchlen);
```

Ändere Kommentar.

D.2.3 Virtuelle Kameras - view.h

VOXREN ist in der Lage, mehrere virtuelle Kameras (sog. Views) zu verwalten. Die zugehörigen Datenstrukturen und Routinen sind in view.h definiert.

```
extern char *rendermodes[RMODE_MAX+1];
```

Feld mit Strings, die den numerischen Rendermode in Text umsetzen. Terminiert mit `NULL`.

```
extern char *stereomodes[SM\_MAX+1];
```

Analog für die Stereomodi,

```
extern char *lightmodes[LM_MAX+1];
```

für die Lichtmodi,

```
extern char *turnmodes[TO_MAX+1];
```

und die Drehpunkte.

```
typedef struct inboundconnection {
[...]
} IBC;
```

Datenstruktur zur Verwaltung von TCP/IP-Verbindungen.

```
typedef struct marker3d {
    struct marker3d *next;
    [...]
} marker3d_t;
```

Liste aller gesetzten Marker.

```
typedef struct global_state {\\
    [...]
} global_state_t;
```

Enthält einige globale Daten, die nicht viewbezogen sind (Marker, TCP-verbindungen, bisher allozierte Views, aktuelle Bildnummer, Gammakorrekturwert).

```
typedef struct view {
    struct view *next,*master;    /* multi-view */
    char         name[30];        /* view Name  */
    vector       mastermove,masterturn; /* turn and move master */
```

VOXREN kann mehrere Ansichten gleichzeitig darstellen - sie werden in Form einer verketteten Liste verwaltet. Einzelne Views können einem „Masterview“ zugeordnet werden, wodurch sie diesem automatisch folgen (um `mastermove` verschoben und `masterturn` verdreht). Zur Identifikation hat jeder View einen Namen.

```
    ggi_visual_t vis;             /* associated visual */
```

Jedem View ist ein Fenster zugeordnet, das als LibGGI Visual repräsentiert wird.

```
    vector pos;                   /* location of the eye. */
    vector dir;                   /* direction we look.   */
    vector right;                 /* "right" vector for the camera */
    vector top;                   /* "top" vector */
    double zoom;                  /* zoom factor  */
    double jumpstart;             /* distance to intersection plane*/
    double maxdepth;              /* maximum render distance       */
```

Parameter der virtuellen Kamera, die mit dem View assoziiert ist.

```
    int    lightpower;            /* light power     */
    int    glob_opacity;          /* global opacity  */
    enum light light;             /* lighting scheme */
```

Verwendetes Beleuchtungsmodell und seine Parameter.

```
enum rmode rendermode;        /* Rendering mode  */
int renderflags;
int smoothout;
int finestepping;          /* Get better edges by finestepping */
```

Verwendeter Rendermodus und -optionen.

```
int renderwidth_x;
int renderwidth_y;
```

Gerenderte Bildgröße.

```
int scalemeup;                /* Scale me up */
```

Vergrößerungsfaktor.

```
ggi_color bgcolor;            /* Background color */\\
int greycenter,greywidth;     /* maps for handling the grey
                                 components in the gaps */
greyrange greymark[NUM_GREY_RANGES];    /* grey markers */
```

Hintergrundfarbe, Grauwertfenster und markierte Grauwertbereiche.

```
renderfield    rfield,lfield;
```

Zielstrukturen für das Rendering - das gerenderte Bild wird dort abgespeichert.

```
enum stereomode stereomode;   /* stereo mode */
int stereobase;               /* base size for stereo mode */
double stereodeg;/* angle at which to turn the eyes inwards */
```

Aktiver Stereomodus und zugehörige Parameter (Augenabstand und Konvergenzwinkel).

```
double stepconst;
int fastmove;    /* do turns at 90 degrees, move 10x faster. */
int turnonobject;/* turns operate on object */
vector currturnpoint;
```

Aktuelle Bewegungsparameter (Drehpunkt, Geschwindigkeit).

```
    int colorizeto;  /* colorize all hit voxels with that color. */
```

Aktuelle Zielfarbe für Umfärbungen.

```
    FILE *playfile;  /* The file we currently play back */
    int     playdir; /* Direction we play the movie */
    int    playpos;  /* Position within file */
```

Aktueller Abspielstatus für Positionsdateien.

```
    int    autonavigate;        /* Autopilot */
```

Autopilotenmodus.

```
    enum markershow markershow;
```

Markermodus.

```
    int  dodisp;
    char caption[256];
```

Flags zum Anzeigen von Änderungen und Kommentaren im Bild.

```
    int   movietype; /* type to write */
    void *moviehand; /* handle while writing */
```

Aktuelle Einstellungen zum Abspeichern von Bildern.

```
} view_t;
```

Von diesen Strukturen werden zwei Pointer definiert:

```
extern view_t *viewroot,*curview;
```

Startpunkt für die verkettete Liste der Views und aktuell aktiver View.

```
void view_normalize(view_t *myview);
```

Richtet das Kameradreibein neu aus, falls es durch viele Operationen ggf. leicht verzogen wurde.

```
int     view_create_view      (char *name);
view_t *view_find_view_by_name(char *name);
int     view_destroy_view     (view_t *which);
void    view_destroy_all_views(void);
```

Erzeuge, zerstöre und finde Views.

```
int string2rm(char *string,int defrm);
int string2lm(char *string,int defrm);
int string2to(char *string,int defto);
```

Abstrahierter Zugriff auf die Rendermodi, Lichtmodi und Drehpunktsoptionen.

```
int         add_marker_byref    (vector *vec,char *label,ggi_color *col);
int         add_marker_byvalue  (vector *vec,char *label,ggi_color *col);
marker3d_t *get_marker_byname   (char *label);
marker3d_t *get_marker_bynumber (int x);
int         delete_marker_byname(char *label);
void        delete_all_markers  (void);
void        delete_all_nonsys_markers(void);
```

Erzeugen, Zerstören und Zugriff auf Marker. Marker, die „byref“ angelegt werden, folgen dem angegebenen Vektor automatisch (dieser Vektor muss dann solange Gültigkeit behalten, bis der Marker wieder gelöscht wird!). Marker, die „byvalue“ angelegt werden, erstellen eine lokale Kopie des Vektors und folgen dem Vektor nicht.

D.3 Die Schnittstelle für Kommandomodule

VOXREN kann durch Hinzufügen von Kommandomodulen in seiner Funktionalität erweitert werden. Der Aufbau eines solchen Moduls wird in den nächsten Abschnitten am Beispiel eines sehr einfachen Moduls aufgezeigt.

D.3.1 Dateistruktur

Die empfohlene Dateistruktur für ein VOXREN -Modul besteht aus einem Verzeichnis, das alle für das Modul notwendigen Dateien enthält. Ggf. können für komplizierte Module Unterverzeichnisse darunter angelegt werden.
Das Verzeichnis sollte mindestens enthalten:

- Ein Makefile
- Eine Modulinit/-exitroutine (typ. in main.c)
- Eine Headerdatei zur Deklaration extern definierter Funktionen, die in den Sprungtabellen der Initroutine benutzt werden.
- Eine oder mehrere .c-Dateien, die die eigentlichen Funktionen implementieren. Triviale Module können dies gleich in main.c tun.

D.3.2 Makefile

```
CFLAGS+=-fPIC -Wall -I"../../.."
```

Die Option `-fPIC` ist notwendig, um Position Independent Code zu erzeugen, also letztlich ein ELF-Style Shared Library binden zu können. `-Wall` sollte ohnehin immer verwendet werden (alle Warnungen). Der Includepath muss so angepasst werden, dass die VOXREN -Includedateien dort gefunden werden können.

```
SRCFILES=main.c flip.c remark.c hull.c bif.c killside.c mktransp.c \
remarknorm.c classmax.c
```

Zusätzlich zum Init-/Exitcode in main.c werden in diesem Beispiel 8 weitere Quelldateien mit Funktionen eingebunden.

```
OBJFILES=$(patsubst %.c,%.o,$(SRCFILES))
```

Die entsprechenden .o-Dateien (Objectcode) werden generiert, bevor die Library zusammengelinkt wird. Die Namen der Dateien werden hier von GNU Make automatisch erzeugt.

```
SOFILE=remark.so
```

Das Modul erhält einen eindeutigen Namen. Die letztlich erzeugte Library **muss** MODULNAME.so heißen. Analog gibt es weitere Abhängigkeiten von diesem Modulnamen (s. folgende Abschnitte).

```
include ../Make.rules
```

Die eigentlichen Erstellungsregeln fasst man sinnvollerweise an zentraler Stelle zusammen.

Make.rules

Die eben erwähnte Datei `Make.rules` enthält typischerweise folgende Regeln:

```
all:    $(SOFILE)
```

Hauptziel ist die Erstellung des Moduls.

```
$(SOFILE):      $(OBJFILES)
        $(LD) --whole-archive -shared -soname=$@ -o $@ $^
```

Erzeuge ein Shared-Library aus den Objectcodedateien.

```
install:        all
        -install -g $(GROUP) -m 664 $(SOFILE) $(LIBDIR)/modules/command/
```

Installiere im VOXREN Kommandomodul-Directory.

D.3.3 main.c

Dort befinden sich die init/exit-Routinen, die von VOXREN beim Programmstart bzw. -ende aufgerufen werden und die zu diesem Zeitpunkt die vom Modul implementierte Zusatzfunktionalität bei VOXREN an- bzw. abmelden müssen.

Dazu ruft VOXREN folgende Funktionen auf:

mod_command_*modulname*_init

Diese Routine wird direkt nach dem Laden des Moduls (beim Start von VOXREN) aufgerufen.

```
static void *selfhandle=NULL;
int mod_command_remark_init(void *shand) {
    selfhandle=shand;
```

Um ggf. dynamisch Routinen aus dem eigenen Library aufzurufen, kann es nötig sein, sich das „selfhandle“ merken zu müssen. Es handelt sich dabei um ein von dlopen() geliefertes Handle auf das gerade geladene Modul. Diese Referenz kann auch benutzt werden, um bei generischem Laderanteil das aktuell zu initialisierende Modul festzustellen.

```
    command_register_module("remark",remark_cmdlist);
    keys_register("remark",remark_keys);
```

Üblicherweise registriert ein Modul Tastenkombinationen, Kommandos oder Mausaktionen. Dazu stehen die Funktionen command_register_module, keys_register und mouse_register zur Verfügung. All diese Funktionen nehmen als Parameter den Namen des Moduls, das die Funktionen registriert, sowie einen Zeiger auf ein Array von Kommandos, Tastenbelegungen bzw. Mausbelegungen.
Ggf. können noch weitere Initialisierungsaktionen durchgeführt werden, bevor die Funktion mit 0 (Erfolg) oder Fehlercode terminiert.
Im Fehlerfall muss die Funktion zuvor alle ggf. angeforderten Ressourcen wieder freigegeben haben.

```
    return0;
}
```

mod_command_*modulname*_exit

Diese Funktion wird kurz vor Beendigung von VOXREN aufgerufen.

```
int mod_command_remark_exit(void *shand) {
    selfhandle=NULL;
```

Setze selfhandle auf NULL, um ggf. spätere nicht mehr gültige Zugriffe hierauf zu bemerken.

```
    keys_unregister("remark");
    command_unregister_module("remark");
```

Deregistriere die zuvor registrierten Kommandos und Tasten.

```
    return 0;
}
```

Kehre mit Status Erfolg zurück.

Benötigte #include-Dateien

Typischerweise wird ein VOXREN-Modul folgende Headerdateien von VOXREN benötigen:

```
#include "view.h"
#include "voxel.h"
#include "command.h"
#include "keys.h"
```

sowie seine eigene Headerdatei für nicht in main.c implementierte Funktionen:

```
#include "mod_command_remark.h"
```

Aus Gründen der Übersichtlichkeit wird dringend empfohlen, dieser Namenskonvention (mod_command_*modulname*.h) zu folgen.

Registrierstruktur für Tasten *modulname*_keys

Diese Struktur wird in der Funktion mod_command_*modulname*_init an den Registrieraufruf keys_register(" *modulname*" ,*modulname*_keys); übergeben.

Es handelt sich dabei um ein Array mit Einträgen vom Typ keyentry_p (deklariert in keys.h), das durch einen NULL-Eintrag im Feld handler terminiert wird.

Ein Beispiel:

```
/* List of keys provided by this module. */

static keyentry_p remark_keys[] = {
    { GIIK_F9 , " Do multiscale BFS."     , remark_key_bfs},
    { 'B'     , " Do multiscale BFS."     , remark_key_bfs},
    { GIIK_F11, " Fixate yellow to green.", remark_key_fixate},
    { 'F'     , " Fixate yellow to green.", remark_key_fixate},
    { GIIK_F12, " Undo   yellow to grey." , remark_key_undo},
    { 'U'     , " Undo   yellow to grey." , remark_key_undo},
    { GIIK_VOID, NULL, NULL}
};
```

Dies registriert 6 Tasten bei VOXREN , die die Funktionen remark_key_bfs, remark_key_fixate bzw. remark_key_undo aufrufen, wenn die entsprechenden Tasten gedrückt werden. Wie man sieht, ist es möglich, eine Funktion auf mehrere verschiedene Tasten zu legen. Da die Funktionen beim Aufruf den auslösenden Tabelleneintrag als Parameter übergeben bekommen, ist es somit möglich, ähnliche Aufgaben in einer C-Funktion zusammenzufassen und sie anhand des Eintrages in der Tastentabelle zu unterscheiden. Der Eintrag „hlp" ist lediglich ein Hilfetext für den Benutzer. Er erscheint bei der Eingabe von HELP COMMAND *modulname*.

Registrierstruktur für Kommandos *modulname*_cmdlist

Diese Struktur wird in der Funktion mod_command_*modulname*_init an den Registrieraufruf command_register_module(„*modulname*", *modulname*_cmdlist); übergeben.

```
/* List of commands. See command.h for details. */
```

```
static struct commands remark_cmdlist[] = {
    { "FLIPSIDES", "flip sides of a class",
        mod_command_remark_flipsides,
        NULL,DIRECT_PTR,0.0,0.0,
        DD_DISP},
    { "REMARK", "Remark a class (from to).",
        mod_command_remark_remark,
        NULL,DIRECT_PTR,0.0,0.0,
        DD_DISP},
[...]
    { NULL,NULL,NULL,NULL,0,0.0,0.0,0 }
};
```

D.3.4 Implementierung eines Tastaturkommandos

```
static int remark_key_fixate(keyentry_t entry,mousecoord *effmouse) {
    mod_command_remark_doremark(COLOR_FILLCOLOR,COLOR_FIXEDCOLOR);
    curview->dodisp|=DD_DISP;
    return 1;
}
```

Ein Tastaturkommando wird aufgerufen, wenn die Hauptschleife von VOXREN die zu diesem Kommando registrierte Taste von einem Eingabegerät gelesen hat. In der Regel genügt es in der dann aufgerufenen handler-Funktion einfach die dazugehörige Aktion auszuführen. Die Funktion muss 1 zurückgeben, wenn die betreffende Taste als „erledigt“ angesehen werden soll. Gibt sie 0 zurück, so wird nach weiteren passenden Tasten gesucht.

Falls als Ergebnis einer Taste sich ein verändertes Bild ergeben sollte, so sollte in der Variablen curview->dodisp das Bit DD_DISP gesetzt werden, um ein baldmöglichstes Neurendern zu erzwingen.

D.3.5 Implementierung eines Scriptkommandos

Kommandozeilenparser

Findet der zentrale Befehlsparser einen registrierten Befehl, so ruft er die in der entsprechenden `struct commands` definierte Funktion auf.
Im Folgenden wird eine typische solche Routine genauer erklärt:

```
void mod_command_remark_flipsides(cmd_p self,IBC *ptr,unsigned char *parms) {
```

Die Parameter bedeuten dabei Folgendes:

self ist ein „self-pointer“, also ein Zeiger auf die Struktur, die den Aufruf der Funktion verursacht hat. Damit wird es möglich, mehrere ähnliche Funktionen zusammen zu implementieren (quasi zu überladen), indem man entweder lediglich durch Auswertung von Parametern in der Struktur entscheidet, worauf die Funktion anzuwenden ist (viele Funktionen, die z.B. nur einen Integer-Wert ändern, sind so implementiert), oder sogar die auszuführende Funktion selbst von diesem Zeiger abhängig macht.

ptr erlaubt die Nutzung des Kanals, auf dem das Kommando empfangen wurde, zur Rückgabe von Statusmeldungen. Dazu kann ptr an z.B. tcp_printf übergeben werden.

parms enthält einen Zeiger auf die Parameter, die dem Kommando übergeben wurden, oder NULL, falls keine Parameter übergeben wurden.

```
    int from;
```

Hilfsvariable, die den Übergabewert aufnimmt.

```
    if (parms==NULL||*parms=='?') {
        tcp_printf(ptr,"500 Command requires parameter.\n");
```

Prüfe, ob Parameter angegeben wurden. Ist der Parameter `?`, so sollte möglichst eine kurze Hilfe mit den möglichen Parametern ausgegeben werden.

```
    } else {
        from=-1;
        sscanf(parms,"%d",&from);
```

Lies die übergebenen Parameter ein.

```
        if (from   >=     0&&from   <=  255 ) {
```

Überprüfe sie auf Gültigkeit.

```
            voxel_flipsides(from);
```

Führe die Funktion aus.

```
            tcp_printf(ptr,"303 OK. Flipping class %d.\n",from);
```

Teile dem Benutzer mit, dass das Kommando korrekt beendet wurde.

```
            curview->dodisp|=self->dispchanges;
```

Sorge dafür, dass ggf. ein neues Bild gezeichnet wird. Es wird empfohlen, hier grundsätzlich den Wert aus dem Selfpointer zu verwenden, so dass man Änderungen in der Logik zum Neuzeichnen des Bildes an zentraler Stelle machen kann.

```
        } else {
            tcp_printf(ptr,"402 Parameters illegal.\n");
        }
    }
}
```

Falls Parameter nicht in Ordnung, Benutzer benachrichtigen.

Die Arbeitsfunktion

Wie oben gesehen, ruft der Kommandozeilenparser eine Funktion auf, um die letztlich erwünschte Aktion auszuführen. Das hat den Vorteil, dass man leicht diese Funktion in mehreren Zusammenhängen nutzen kann (z.B. auch per Tastatur oder durch ein anderes Kommando, das die Funktion dann anders parametriert). Diese Vorgehensweise ist daher für nichttriviale Funktionen empfohlen. Lediglich bei sehr einfachen Funktionen kann die Aktion auch einfach direkt im Kommandozeilenparser erfolgen.

Funktionen mit nennenswerter Rechenzeit wird man, sofern parallelisierbar, durch den Einsatz von Threads beschleunigen wollen. Das folgende Beispiel zeigt einen solchen Fall. Dort ruft der Kommandozeilenparser zunächst eine Thread-Setuproutine auf, die die eigentlichen Arbeitsprozesse (Threads) startet, auf die Ergebnisse wartet und ggf. selbige zurückmeldet.

Thread-Setuproutine

Threads erhalten beim Start nur einen `void *`-Zeiger. Ihre Parameter müssen daher über eine Struktur übergeben werden, die durch diesen Zeiger angesprochen wird.

Im konkreten Fall benötigen wir nur einen Parameter, die Voxelklasse, deren Innen- und Außenseiten vertauscht werden sollen.

```
struct flip_parm {
        int min,max,barofs; /* Always needed - Thread-info. */
        int class;
};
```

Die drei weiteren Parameter werden benötigt, um die Verteilung der Arbeit auf die einzelnen Threads zu regeln. Per default verteilt VOXREN die Last, indem es n Threads startet, von denen jeder $\frac{1}{n}$ der Bildschichten zur Bearbeitung erhält. Diese Information wird in den Feldern `min` und `max` übertragen. Dort wird die erste und die letzte zu bearbeitende Schicht (+1) eingetragen.

Die Variable `barofs` dient der Visualisierung des Fortschrittes der einzelnen Komponenten. Jeder Thread hat dazu eine eigene Balkenanzeige, deren Position auf dem Bildschirm so bestimmt wird.

Doch nun zur Thread-Setuproutine:

```
static int voxel_flipsides(int class) {

    TH_SETUP_VARS(flip_parm)
```

Dieses Macro erstellt die notwendigen Variablen, um die anderen Thread-Setup-Macros nutzen zu können. Dabei handelt es sich um die Integer `x` und `step`, sowie ein Feld `rargs` vom übergebenen struct-Typ (dem Threadparameter).

```
    TH_SETUP_FOR
```

Dieses Macro leitet eine Schleife ein, die die Last durch Aufteilung des Volumens in z-Richtung verteilt.

Anschließend müssen alle Thread-Parameter (außer `min, max` und `barofs`, die werden bereits von TH_SETUP_FOR mitverwaltet) ausgefüllt werden. In unserem Beispiel ist das nur die zu flippende Klasse.

```
rargs[x].class=class;
```

Nachdem somit die Parameter alle gesetzt sind, kann der Thread bereits gestartet werden:

```
TH_RUN_THREAD(thread_flipsides)
```

ACHTUNG: Wenn Threads deaktiviert wurden (zum Debugging oder für geringfügig bessere Performance auf Singleprozessormaschinen), so werden die Funktionen von `TH_RUN_THREAD` direkt aufgerufen. Es darf also **kein** internes Locking verwendet werden, das explizit auf einen anderen Thread wartet (ausschließendes Locking ist erlaubt). Es ist also **nicht** garantiert, dass mehrere Threads parallel laufen.

Nun muss noch die von TH_SETUP_FOR geöffnete Schleife wieder geschlossen werden, und anschließend wird auf die Beendigung aller Threads gewartet.

```
}
TH_JOIN_THREADS
```

Danach könnten ggf. noch Rückgabewerte (die man am besten über die rargs- Struktur abwickelt) zusammengesammelt werden. Die Funktion terminiert anschließend:

```
return 0;
}
```

Die mit Threads implementierte Arbeitsroutine

Threads müssen einen Prototypen der Form `void *(*func)(void *parms)` haben. Parameter werden daher wie oben erwähnt über eine in `void *parms` übergebene Struktur zugänglich gemacht.

```
static void *thread_flipsides(void *p) {
    int x,y,z,idx;
    voxelclass *vc;
    voxelnorm *vn;
```

Diverse Variablen. Die Variablen x,y und z müssen deklariert werden, wenn man die Threadmacros TH_FOR[ZYX_]*_SHOW nutzen möchte.

```
    struct flip_parm *parm=p;
```

Zuerst müssen die übergebenen Parameter in eine lesbare Form gebracht werden. Nutzt man die Threadmacros, so muss der entsprechende Zeiger auf die struct mit dem Namen `parm` deklariert werden. Die Macros greifen hierauf zu, um die Schleifen richtig zu initialisieren und die Fortschrittsbalken korrekt anzuzeigen.

Am einfachsten verwendet man nun eines der TH_FOR[ZYX_]*_SHOW Macros, die automatisch mit den Variablen x, y und z durch den relevanten Bereich des Voxelfeldes laufen.

```
    TH_FORZYX_SHOW("Flipping.")
```

Die konkrete Aufgabe verlangt nun einen Zugriff auf das Voxelfeld, die Prüfung, ob das aktuelle Voxel von der gewünschten Klasse ist, und wenn ja, ein Umdrehen der Normalen.

```
        idx=getvoxelindex(x,y,z);
```

Hole den Index eines Voxels.

ACHTUNG: getvoxelindex ist aus Geschwindigkeitsgründen ein Macro! Keine nicht makrosicheren Konstrukte übergeben.

```
        vc=getvoxelclass(idx);
```

Lies tdie Voxelklasse aus. Beachten Sie, dass die Zugriffsmakros `getvoxelclass|norm|depth` immer einen Zeiger liefern. Das vereinfacht den Schreibzugriff und erlaubt Zeigerartithmetik.

```
        if (*vc==parm->class) {
            vn=getvoxelnorm(idx);
            *vn=vecindex_flip(*vn);
        }
```

Falls es sich um ein Voxel der gewünschten Klasse handelt, lies die aktuelle Normale, drehe sie um 180° und speichere das Ergebnis.

```
    TH_FORZYX_END
    return NULL;
}
```

Schließe die mit TH_FORZYX_SHOW geöffneten Schleifen wieder und beende die Threadroutine. Rückgabewerte handhabt man besser über die Parameterfelder als über den zurückgebbaren Zeiger.

Zu erstellende Headerdatei

Falls man nicht alle Funktionen eines Moduls in der main.c-Datei unterbringen kann bzw. will, so wird eine Headerdatei benötigt, damit die Kommandoparsingfunktionen in die Arrays in main.c eingetragen werden können.

Eine typische solche Datei sieht so aus:

```
/*
 * Header file containing prototypes for all the command functions.
 * Required to allow main.c to insert the pointers into the commandlist.
 */

void mod_command_remark_remarknorm(cmd_p vec2,IBC *ptr,unsigned char *parms);
void mod_command_remark_flipsides(cmd_p unused,IBC *ptr,unsigned char *parms);
void mod_command_remark_mktransp(cmd_p vec2,IBC *ptr,unsigned char *parms);
void mod_command_remark_remark(cmd_p unused,IBC *ptr,unsigned char *parms);
void mod_command_remark_hull(cmd_p unused,IBC *ptr,unsigned char *parms);
void mod_command_remark_filltofork(cmd_p vec2,IBC *ptr,unsigned char *parms);
void mod_command_remark_filltofork2(cmd_p vec2,IBC *ptr,unsigned char *parms);
void mod_command_remark_filltofork3(cmd_p vec2,IBC *ptr,unsigned char *parms);
void mod_command_remark_killside(cmd_p vec2,IBC *ptr,unsigned char *parms);
void mod_command_remark_classmax(cmd_p vec2,IBC *ptr,unsigned char *parms);
/* Other symbols that are called across modules
 */
int mod_command_remark_doremark(int from, int to);
```

Es wird empfohlen, sich an die hier gezeigte Benennungskonvention zu halten, also Funktionen, soweit sie nicht static deklariert werden können, in der Form mod_command_*modulname_funktionsname* zu benennen.

D.3.6 Nützliche Headerdateien und Funktionen

Bei der Erstellung von Key-/Kommandomodulen sind folgende Headerdateien zusätzlich zu den bereits genannten mit den globalen Datenstrukturen nützlich:

command.h

```
typedef struct commands *cmd_p;
struct commands {
char *cmd,*help;
void (*cmdfunc)(cmd_p param,IBC *ptr,unsigned char *parms);
void *param;
int  offset;
double min,max;
int dispchanges;
};
```

Zunächst werden die Einträge eines Kommandodeskriptors deklariert. Zu einem Kommando gehören folgende Angaben:

cmd Der Name des Kommandos. Damit das Kommando erkannt wird, muss es bezüglich eines via `strcasecmp` ausgeführten Vergleiches mit diesem Feld übereinstimmen.

help Wird eine Hilfe zu diesem Kommando bzw. zu den Kommandos des Moduls, das es enthält, angefordert, so erscheint dieser Text als Erklärung neben dem Kommando.

cmdfunc Wird das Kommando erkannt, so wird die Funktion cmdfunc aufgerufen und erhält als Parameter den gefundenen Kommandodeskriptor, einen Zeiger auf die Kommunikationsstruktur der auslösenden Verbindung sowie einen Zeiger auf die Parameter des Kommandos.

param ist ein Zeiger auf einen vom Kommandodeskriptor abhängenden Parameter. Er wird in der Regel von generischen Kommandofunktionen benutzt, die eine allgemeine Aufgabe haben (z.B. einen Integer einzulesen).
Allerdings besitzt VOXREN nun viele solche Variablen innerhalb der `view_t`-Struktur, die dynamisch erzeugt und zerstört wird. Man kann also zu Compilerlaufzeit keinen entsprechenden Zeiger eintragen. Daher gibt es im Zusammenhang mit

offset folgende Möglichkeit:
Wird in offset die Konstante DIRECT_PTR eingetragen, so handelt es sich bei dem Inhalt von param um einen echten Zeiger auf einen Wert mit fester

Speicheradresse. Ansonsten enthält offset die Relativposition des zu lesenden Feldes bezüglich des Anfangs eines Views. Dieser Wert ist mit Hilfe des Macros

```
#define VIEW_OFF(x) \
  ((unsigned char *)&command_dummyview->x-\
   (unsigned char *)command_dummyview)
```

leicht zu ermitteln.
In der aufgerufenen Funktion nutzt man das Macro

```
#define GETPTR(x) \
  ( (x->offset==DIRECT_PTR) ? \
x->param :            \
(void *)(*((unsigned char **)x->param)+x->offset) )
```

um die gemeinte Adresse zu erhalten.

min/max enthalten Grenzen für generische Funktionen, z.B. die minimal und maximal zulässigen Werte für einen gelesenen Integer.

dispchanges enthält die zu setzenden Displaybits, zeigt also an, ob die Ausführung einer Funktion ein Neurendern (DD_DISP) oder gar eine Größenänderung des Views (DD_RESIZE) zur Folge hat.

Einige der eben angesprochenen universellen Kommandos sind ebenfalls in command.h deklariert:

```
void tcp_cmd_integer(cmd_p vec2,IBC *ptr,unsigned char *parms);
void tcp_cmd_double(cmd_p vec2,IBC *ptr,unsigned char *parms);
void tcp_cmd_vector(cmd_p vec2,IBC *ptr,unsigned char *parms);
void tcp_cmd_color(cmd_p vec2,IBC *ptr,unsigned char *parms);
```

Sie lesen Integer, Doublewerte, Vektoren (Datentyp vector, deklariert in vector.h) und GGI-Farbtripel (RGB als 16bit Hex-Tripel) ein.

Außerdem sind dort die Kommandoregistrierfunktionen deklariert, so dass ein Kommandomodul command.h immer `#includen` muss.

```
int command_register_module(const char *modname,cmd_p cmdlist);
int command_unregister_module(const char *modname);


int command_init(void);
int command_exit(void);
```

keys.h

Analog ist die Datei `keys.h` wichtig für Module, die Tastenkommandos zur Verfügung stellen wollen.

Um den genauen Mechanismus zu verstehen, wird empfohlen, sich mit dem Eventhandling von LibGGI auseinanderzusetzen . Die relevanten Deklarationen finden sich in

```
#include <ggi/types.h>
#include <ggi/keyboard.h>
```

Analog zum Kommandodeskriptor definiert diese Headerdatei einen Tastendeskriptor:

```
typedef struct keyentry {
uint32 key;
char *hlp;
int (*handler)(struct keyentry *self, mousecoord *effmouse);
} keyentry_p;
typedef keyentry_p *keyentry_t;
```

Er ist etwas einfacher aufgebaut und enthält nur

key den Tastencode des LibGGI-Keysymbols, das die Aktion auslösen soll. Diese Symbole sind für druckbare Zeichen schlicht die entsprechenden Zeichenkonstanten (also z.B. 'f' für die Taste Klein-F), für die Sondertasten existieren #define s, die leicht in ggi/keyboard.h nachgeschlagen werden können.

hlp einen Hilfetext.

handler die aufzurufende Funktion. Diese erhält außer einem zum Kommandomodul analogen Selfpointer auch eine Struktur, die die aktuelle Mausposition enthält. Diese ist unabhängig von eventuell eingestellten Zoomfaktoren etc. jeweils bezüglich der Renderfelder in der view-Struktur, so dass damit festgestellt werden kann, „auf welchen Voxel“ der Mauscursor zeigt.

Genau wie bei command.h deklariert keys.h auch die Registrierfunktionen, so dass ein Modul, das Tastenkommandos implementiert keys.h `#includen` muss.

```
int keys_register(char *name,keyentry_t entry);
int keys_unregister(char *name);
```

Literaturverzeichnis

[BHM] J. E. Bresenham, *Algorithm for Computer Control of a Digital Plotter*, IBM Systems Journal, 4(1):25-30, 1965

[BVMD] V. Aurich, A. Beck, *ECCET: Ein System zur 3D-Visualisierung von Volumendaten mit Echtzeitnavigation*, Proceedings Bildverarbeitung für die Medizin 2002, S389, Springer 2002, verfügbar unter *http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-56/117.ps.gz*

[BVMS] V. Aurich, A. Beck, M. Cohnen, C. Vogt, *Segmentierung von Hohlkörpervolumina in verrauschten CT-Daten und automatische Detektion von Polypen und Divertikeln*, Proceedings Bildverarbeitung für die Medizin 2002, S181, Springer 2002, verfügbar unter *http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-56/116.ps.gz*

[D3D] *http://www.dresden-3d.de/*

[D4C] *http://sourceforge.net/projects/dcm4che/*

[EcMan] *http://www.eccet.de/manual/*

[FP2] *http://www.d6.com/users/checker/pdfs/gdmfp.pdf*

[FPO] *http://www.stereopsis.com/FPU.html*

[GE] *http://www.gemedicalsystems.com/splash.html*

[GGI] *http://www.ggi-project.org/*

[HPV2] H. Bourquain, A. Schenk, F. Link, B. Preim, H.-O. Peitgen, *HepaVision2 - a software assistant for preoperative planning in LRLT and oncologic liver surgery*, Computer Assisted Radiology and Surgery (CARS), June 2002, pp. 341-346

[IArt] *http://www.i-Art.com.tw/*

[JDA] *http://www.tiani.com/JDicom/JDICOM_manual_V1_0.pdf*

[JDC] *http://www.tiani.com/JDicom/index.html*

[KPX] *http://www.knoppix.de/*

[MDC] *http://www9.medica.de/cipp/md_medica/custom/pub/content,lang,1/ ticket,g_a_s_t/oid,3120*

[MPG] *http://www.dimdi.de/de/mpg/recht/index.htm*

[NLG2] V. Aurich, E. Mühlhaus, S. Grundmann, *Kantenerhaltende Glättung von Volumendaten bei sehr geringem Signal-Rausch-Verhältnis*, Zweiter Aachener Workshop über Bildverarbeitung in der Medizin 1998

[NLG] V. Aurich, G. Winkler, K. Hahn, A. Martin, K. Rodenacker, *Noise Reduction in Images: Some Recent Edge-Preserving Methods*, Pattern Recognition and Image Analysis, Vol.9, No. 4, 1999, 749-766. Auch als Preprint des Instituts für Biomathematik and Biometrie, GSF - Forschungszentrum für Umwelt und Gesundheit, Neuherberg, Germany, 1998. http://www.gsf.de/ibb

[PBM] *http://netpbm.sourceforge.net/*

[PSP] *http://www.perspektrum.de/*

[RALS] *http://www.cevis.uni-bremen.de/ bernhard/papers/riskAnalysis.pdf*

[RöFo] M. Cohnen, C. Vogt, V. Aurich, A. Beck, D. Häussinger, U. Mödder, *Multi-Slice CT-Colonography in Low Dose Technique - Preliminary Results*, Fortschr Röntgenstr 2002; 174:835-838

[RöKoD] V. Aurich, A. Beck, *ECCET: Ein System zur Visualisierung von Volumendaten mit Echtzeitnavigation*, Fortschr Röntgenstr 2002; 174:309

[RöKoV] V. Aurich, A. Beck, M. Cohnen, C. Vogt, *Automatische Detektion von Polypen und Divertikeln bei der CT-Kolonographie*, Fortschr Röntgenstr 2002; 174:267

[SGR] G. Schäfer, *Normalenschätzung an einer segmentierten Voxelmenge*, Diplomarbeit, Düsseldorf 2001

[SMB] *http://www.samba.org/*

[SPB] *http://www.3dconnexions.com/products/4000/*

[SER] *http://www.m-ww.de/gesund_leben/elektrosmog/kuenst_strahlenexposition.html*

[TNI] *http://www.tiani.com/*

[VCS2] *http://www.radiologie.uni-duesseldorf.de/cohnen/ct-kolo.htm*

[VCSA] C. Vogt, *private Mitteilungen*, Dez. 2002

[VCS] *http://www.uni-duesseldorf.de/WWW/gahepinf/virt_kolo/ct_colo.htm*

[VH] *http://www.nlm.nih.gov/research/visible/visible_human.html*

[VOLS] W. Lamadé, G. Glombitza, A.M. Demiris, C. Cárdenas, H.P. Meinzer, G. Richter, T. Lehnert, C. Herfarth, *Virtuelle Operationsplanung in der Leberchirurgie*, Chirurg 70, S239-245, Springer 1999

Abbildungsverzeichnis

2.1 Anordnung der Ansichten bei 3-Schnitt-Darstellung 10

3.1 VOXREN im Einsatz 30
3.2 Die graphische Benutzeroberfläche VOXCONTROL 31
3.3 PLANEVIEW als virtuelles Koloskop 33
3.4 Die graphische Shell von $\epsilon\mathcal{CC}\epsilon\mathcal{T}$ 35
3.5 Aufzeichnen von Kamerapfaden 43
3.6 Mit PLANEVIEW erreichbare Renderingqualität 54
3.7 Mit VOXREN erreichbare Renderingqualität 55
3.8 Komplexe Szenen in VOXREN 56

4.1 Die 2D-Darstellung in Planeview 60
4.2 Einblendung von Overlays 63
4.3 Gammakorrektur 64
4.4 Kameramodell beim Raycasting 65
4.5 Mögliche Bildfehler durch diskrete Schrittweite 66
4.6 Optimierungen mit Bounding Box und GDA 68
4.7 Exakte Schnittpunktsberechnung 69
4.8 Nutzung einer Schnittebene zur Entfernung unerwünschter Vordergrundstrukturen 70
4.9 Nutzung der hinteren Schnittebene 71
4.10 Beliebig orientierte Schnittebene in Grauwertvolumina 72
4.11 Frei definierbare Schnittebenen 73
4.12 RGB-Modus von VOXREN 74
4.13 ZSH-Modus von VOXREN und PLANEVIEW 75

4.14 Z-Shading 76
4.15 Rendermodi unter Nutzung vorberechneter Normalen 77
4.16 Kugelkoordinaten 78
4.17 Glanzlichter 79
4.18 Veränderung der Position der Lichtquelle 80
4.19 Simulierter Kontrasteinlauf - Berechnung 81
4.20 simulierter Kontrasteinlauf 82
4.21 Kombinierte Darstellungen 84
4.22 Trilineare Interpolation 85
4.23 Planeview: Maximumsprojektion und Z-Shading 87
4.24 Normalenschätzung mit verschiedenen Abtastgittern 88
4.25 Gradientenschätzung mit kleiner und großer Reichweite 89
4.26 Kontrasteinlaufsimulation bei PLANEVIEW 90
4.27 Verwendung einer Rendermaske bei Planeview 91
4.28 Interpolation 92
4.29 Funktionsprinzip Aliasfilter 93
4.30 Kopplung der separierten Stufen 94
4.31 Mögliche Artefakte beim Aliasfilter 95
4.32 Erzeugung stereographischer Ansichten 96
4.33 Lineblanking-Verfahren 97
4.34 Tiefendarstellung mit Nebel-/Plasmaeffekten 100

5.1 Filterung mit dem nichtlinearen Gaußfilter 101
5.2 Filterung mit dem nichtlinearen Gaußfilter - 3D-Ansicht 102

6.1 Einfache Schwellwertentscheidung 110
6.2 Mehrstufen–Füllalgorithmus - Vorbereitung 111
6.3 Mehrstufen–Füllalgorithmus - Einzelschritte 112
6.4 Mehrstufen–Füllalgorithmus - Taschenfüllung 113
6.5 Notwendigkeit der NLG-Filterung 119
6.6 Vollautomatische Vorverarbeitung - Entfernen der Außenluft 120
6.7 Vollautomatische Vorverarbeitung - Entfernen des Außenraumes . . . 121

6.8 Vollautomatische Vorverarbeitung - Entfernen von Lunge und Hautoberfläche . . . 122
6.9 Vollautomatische Vorverarbeitung - Rekonstruktion und Polypensuche 124
6.10 virtuelle Koloskopie mit Kontrastmittel . . . 126
6.11 Manuell unterstützte Segmentierung (Zusammenhangskomponenten) . 128
6.12 Finden von Verzweigungen und ihre Nutzung zur Segmentierung . . . 129
6.13 Finden von Verzweigungen - Verfahren 1 . . . 130
6.14 Finden von Verzweigungen - Verfahren 2 . . . 131
6.15 Trennung von Aorta und Wirbelsäule mit dem Verzweigungsfinder . . 132
6.16 Trennung von Aorta und Wirbelsäule mit dem Verzweigungsfinder . . 133
6.17 Schließen von Lücken durch den Hüllenoperator . . . 135
6.18 Trennung von Kleinhirn und Hirnstamm - Markierung . . . 136
6.19 Trennung von Kleinhirn und Hirnstamm - Hüllenoperator . . . 137
6.20 Trennung von Kleinhirn und Hirnstamm - Ergebnis . . . 138
6.21 Manuelles Schneiden von Oberflächen . . . 140
6.22 Manuelle Segmentierung in der Grauwertansicht . . . 141
6.23 Nutzung zusammenhangsgesteuerter Schnitte . . . 143
6.24 Nutzung zusammenhangsgesteuerter Schnitte II . . . 144
6.25 Visualisierung gesehener Bereiche . . . 145

7.1 Polypenfinder - Dosenwandverfahren . . . 148
7.2 Polypenfinder - großer Polyp (10 mm) . . . 149
7.3 Polypenfinder - kleiner, flacher Polyp (3 mm) . . . 150
7.4 Polypenfinder - Inselverfahren . . . 151
7.5 Polypenfinder - Stuhlrest. In der 2D-Ansicht erkennbar am Lufteinschluss . . . 152
7.6 Polypenfinder - Strahlenartefakt. Vom Artefakterkenner rot markiert (vgl. Abbildungen 7.2,7.3 und 7.5). . . . 153
7.7 Messungen und Marker . . . 156
7.8 2D-Suchlauf . . . 157

8.1 Der „Tiefenblick“ des Autopiloten . . . 161
8.2 Vom Autopiloten durchflogener Pfad . . . 162

8.3 Durch Interpolation gewonnener Pfad 173

9.1 Einfache Erfassung von Reliefs mit Textur mit 3DSCAN 176
9.2 Mit 3DSCAN erzeugtes Volumen 178
9.3 Kontrastmittelgestützte virtuelle Koloskopie 184
9.4 Die Leber und ihr Gefäßsystem 185
9.5 Segmentierungsergebnisse Leber 186

10.1 Dichtefeldvisualisierung eines MEG-Datensatzes 189

Tabellenverzeichnis

9.1 Vergleich virtueller Normaldosis-Koloskopie mit hochauflösender Videokoloskopie 183
9.2 Vergleich virtueller Niedrigdosis-Koloskopie mit hochauflösender Videokoloskopie 183

D.1 Aufbau eines Voxels in einer 3d32-Datei 311

Danksagung

Mein besonderer Dank gilt Herrn Prof. Dr. Aurich.
Durch ständige Ansprechbereitschaft, fruchtbare Anregungen und Diskussionen und Unterstützung mit Rat und Tat trug er wesentlich zum Gelingen dieser Arbeit bei.

Ebenfalls besonders danken möchte ich Herrn Dr. Cohnen und Herrn Dr. Vogt (Uniklinik Düsseldorf), die zur Entwicklung viele nützliche Hinweise aus medizinischer Anwendersicht beisteuerten. Sie haben sich tief in das System eingearbeitet, uns immer hervorragend mit klinischen Testdaten und Erfahrunsberichten versorgt und somit wesentlich dazu beigetragen, die Praxistauglichkeit des Systems zu verbessern.
Mein Dank gilt weiterhin den Herren Dr. Hackländer, Prof. Cramer und den anderen Mitarbeitern des Klinikums Wuppertal für überlassenes Datenmaterial zur Erkundung neuer Anwendungen.
Ebenso möchte ich mich bei Frau Dr. Timmann-Braun und ihrer Arbeitsgruppe an der Uniklinik Essen bedanken, die nützliches Datenmaterial und Impulse zur Entwicklung komplexerer Segmentierungsmethoden im Rahmen ihrer Arbeit an der Volumetrie des Kleinhirns lieferten.

Schließlich danke ich den Herren Prof. Dr. Wanke und Priv. Doz. Dr. Hackländer für ihre Tätigkeit als Zweitgutachter.