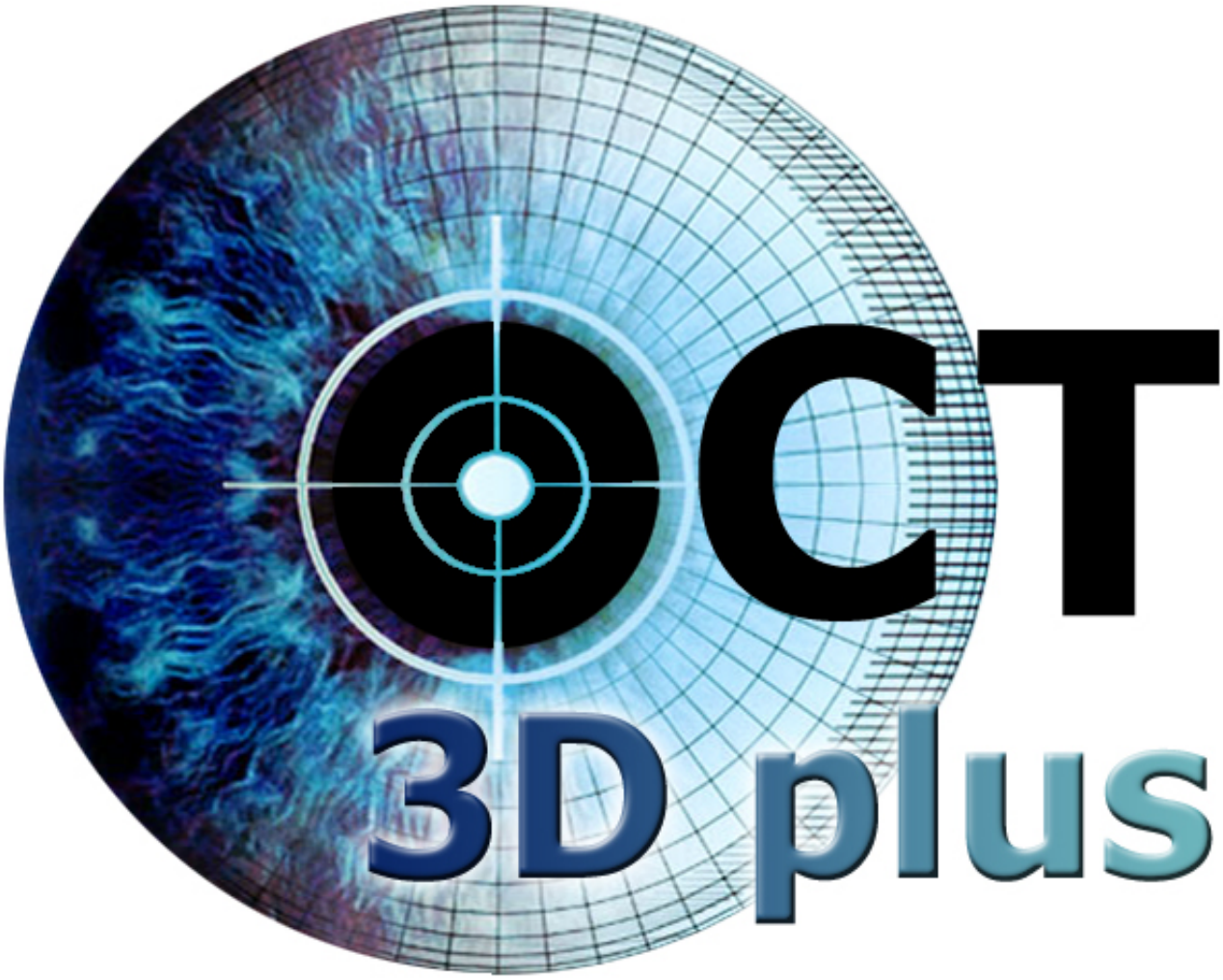
Berner Fachhochschule
Hochschule für Technik und Informatik HTI



Diplomarbeit OCT3D plus

Abschlussbericht V1.0

Autoren
Markus Fawer, Adrian Wyssmann (I4t)

Arbeitsplatz
N.552 (Rolex)

Diplomaufgabe

Optical Coherence Tomography (OCT) liefert Schnittbilder (Scans) durch die Retina des Auges. Es handelt sich um eine neuartige Untersuchungsmethode, für welche erst wenig Auswerte-Software zur Verfügung steht. In Zusammenarbeit mit dem Kantonsspital Luzern sind im letzten Jahr zwei Diplomarbeiten realisiert worden, welche einerseits die Visualisierung (3D und 2D) solcher Messungen und andererseits die Verbesserung der Messdaten (Denoising) zum Ziel hatten. Die vorliegende Diplomarbeit soll diese Arbeiten fusionieren und wie folgt erweitern:

1. Zusammenführung der beiden Arbeiten in eine Applikation.
2. Erweiterung und Verbesserung des Benutzerinterfaces gemass Wünschen der Aerzte.
3. Erweiterung der möglichen Eingabedaten auf Rotationsscans.
4. Einbau einer oder mehrerer Segmentierungs-Algorithmen, welche das Gewebe in seine Bestandteile segmentieren und 3-dimensional darstellen kann.

Abstract

The ophthalmic clinic of the canton of Lucerne owns special equipment that allows acquisition of retina detachment using a laser. This procedure is called Optical Coherence Tomography, OCT for short. The appliance scans slices of the eyes and creates a density profile of the tissues. Previously, two diploma theses had been working on this subject: "OCT Restoration", which was concerned with analyzing the images and the single layers of the eyes, and "OCT3D" which is a program that is capable of interpolate the pictures into a 3d model.

The objective of the diploma consists first to merge "OCT Restoration" and "OCT3D" and on the other hand, implementing one or more algorithms as extensions to segment the pictures. The doctors already working with OCT3D also have wishes concerning adaptations of the application.

Versionskontrolle

Version	Datum	Verantwortlich	Bemerkungen
0.1	01.12 2004	fawem/wyssa1	Erstellung
0.1	09.12.2004	fawem/wyssa1	Fertigstellung
1.0	10.12.2004	fawem/wyssa1	Freigegeben

Tabelle 1-1 Versionskontrolle

Konventionen

In diesem Dokument gelten die folgenden allgemeinen Formatierungskonventionen:

`Dateinamen`	für C++ Klassen/Dateien
`Methodenname()`	für C oder C++ Methodenname
`STRUCT_NAME` für	C Strukturen
`MACRO_NAMES()`	C Preprocessor Makros
OCT_TABELLENNAME	für Tabellennamen im OCT System

```
C++ Codeblock
```

```
Fehlermeldungen (Linking, Compiling)
```

«Original Zitat»

▶Abschnitt x.y	Verweis auf Abschnitt, Fig., Code, Tabelle
[CTRL]	Keyboard-Taste

Trademarks und Copyrights werden bei der ersten Verwendung im Dokument erwähnt. Möglicherweise werden geschützte Namen verwendet, die nicht explizit als geschützt vermerkt wurden.

I Inhaltsverzeichnis

1 Einführung

1.1 Aufgabenstellung

Ziel der Diplomarbeit ist es einerseits, die beiden Applikationen OCT-Restoration[1] und OCT3D[2] zusammenzuführen und andererseits, einen oder mehrere Segmentierungsalgorithmen (Extension) als Erweiterung zu implementieren. Des Weiteren haben auch die Ärzte noch Wünsche bezüglich Anpassungen der bestehenden Anwendung OCT3D. Konkret möchten wir folgende Ziele erreichen:

Ziel-Nr.	Beschreibung
1	**Zusammenführung OCT3D und OCT-Restoration** Zusammenführen der beiden Applikationen OCT3D und OCT-Restoration
2	**Implementation Segmentierungsalgorithmus** Implementierung des Pyramid-Linking Segmentierungsalgorithmus für 2D-Bilder.
3	**3D-Volumendarstellung für segmentierte Bilder** Implementierung einer 3D-Volumendarstellung für die segmentierten Bilder.
4	**2D-Betrachtung, Farbdarstellung** Es wäre praktisch, wenn via Mausbewegung die Farbdarstellung wie im 3D-Modul geändert werden konnte. So kann eine Läsion[3] auf einfache Weise kontrastreicher dargestellt werden.
5	**Export auf Harddisk** Es kann via Rechtsklick ein File, Scandaten eines ganzen Patienten oder einer ganzen Summe von Patienten angewählt und in einen Zug z.B. auf eine externe Harddisk exportiert werden. Falls ein File bereits besteht, fragt die Software nach, ob es wirklich überschrieben werden muss.
6	**Save-Funktion** Wird ein einzelnes Bild gespeichert, so erscheint ein grosser grauer "Bildschirmanteil", der überflüssig und unhandlich ist.
7	**Druckfunktion, Diagnoseblatt** Ein dargestelltes Bild kann per Rechtsklick oder via Menuauswahl gedruckt werden. Dabei kann die Bildschirmauswahl das ganze aktive Fenster oder aber nur einen beliebigen Ausschnitt (=Läsion) getroffen werden. Es gibt eine Druckvorschau, mit der man das Bild auch positionieren bzw. in der Grosse andern kann. Es gibt die Möglichkeit, eine individuelle Vorlage zu kreieren, wo neben dem Bild auch die Patientendaten und Diagnosen eingetippt werden können: Ein eigentliches OCT3D-Diagnoseblatt. Dieses Blatt kann ausgedruckt oder per Mail (z.B. PDF) verschickt werden.
8	**Erweiterung der möglichen Eingabedaten auf Radial-Scans** Die Applikation OCT3D solle dahingehend erweitert werden, dass 2D-Bilder von Radial Scans auch in der 3D-Volumendarstellung angeschaut werden können.

Tabelle 1-1 Ziele

[1] © 2003 Anita Sommer, Hanspeter Zimmermann, Hochschule für Technik und Informatik Biel

[2] © 2003 Yvonne Mettler, Ulrich Sigrist, Hochschule für Technik und Informatik Biel

[3] lat. laesio - eine Schädigung, Verletzung oder Störung einer anatomischen Struktur

1.2 Vorarbeit

Wir konnten in der Semesterarbeit einiges an Erfahrung in der C++ Programmierung sammeln. Diese beschränkt sich allerdings auf das programmieren einfacher Anwendungen.

Wir haben während unserer Semesterarbeit verschiedene Verfahren untersucht und Ihre Fähigkeiten zur Segmentierung der OCT Bilder getestet. Wir haben aber keine Anwendung implementiert die wir in die OCT3D Anwendung übernehmen konnten.

1.3 Vorgehen

Da wir nicht nur eine Extension machen, sondern direkt im Code der bestehende Anwendung OCT3D Änderungen vornehmen, genügt unsere Vorarbeit allerdings nicht. Wir müssen uns in einem ersten Schritt mit der Komplexität der OCT3D Anwendung auseinandersetzten. Auch das eingesetzte **Visual Toolkit VTK**[1], war für uns relativ neu. Wir hatten zwar während der Semesterarbeit den ersten Kontakt damit, doch das reicht bei einem so umfassenden Toolkit bei weitem nicht. Der Umfang von VTK ist gewaltig und die Übersicht über alle Möglichkeiten zu haben ist schier unmöglich. Es wird auch hier sicher am Anfang nicht ganz einfach sein, die Ideen so umzusetzen, wie man diese haben möchte.

Anhand der Aufgabenstellung (▶Abschnitt 1.1) können wir unsere Diplomarbeit grob in 3 Bereiche unterteilen:

1. Erstellen eines neuen Prozesses, der die 2D-Bilder segmentieren kann, sowie erstellen eines Prozesses, der das segmentierte Bild in 3D darstellen kann
2. Das Einbinden der bestehenden Filter aus OCT-Restoration als externe Prozesse ▶[EDK]
3. Änderungen/Anpassungen am bestehenden Quellcode

Für die ersten beiden Aufgaben haben wir eine relativ gute Beschreibung vom Extension Development Kit ▶[EDK]. Das Einbinden der Filter als externe Prozesse sollte kein allzu grosses Problem darstellen. Das Erstellen des Segmentierungsprozesses (Pyramid-Linking) wird wesentlich mehr Aufwand erfordern.

Die zweite Aufgabe sollte auch kein Problem darstellen. Dank der Dokumentation [EDK] sollten neue Prozesse einfach implementiert werden können. Die Filterklassen bestehen ja bereits aus dem OCT-Restoration, es muss also nichts mehr entwickelt werden.

Die letzte Aufgabe erscheint nicht ganz einfach. Einerseits handelt es sich bei der Applikation selber um eine recht komplexe und umfangreiche Applikation. Der Zusammenhang der einzelnen Klassen zu verstehen wird sicher einige Zeit in Anspruch nehmen. Des Weiteren fehlt uns die Erfahrungen, was **Microsoft Foundation Classes (MFC)**[2] oder **VTK** betrifft.

Es ist schwer für die in der Aufgabenstellung (▶Abschnitt 1.1) definierten Ziele eine detaillierte Zeitplanung zu machen. Wir werden trotz der oben genannten Gründe versuchen, eine optimale Zeitplanung zu machen. Durch möglichst strukturiertes Vorgehen, wollen wir versuchen, für alle Ziele der Diplomarbeit, eine optimale Lösung in Bezug auf Qualität und Aufwand zu finden.

1.4 Tests

Wir haben aufgrund der Tatsache, dass wir eine funktionierende und in der Praxis eingesetzte Applikation erhalten haben, auf ein ausformuliertes Testmanagement verzichtet. Vielmehr haben wir die eingebauten Funktionen verschiedenen Tests unterzogen, und die dabei aufgetretenen Fehler dann behoben. Viele dieser Fehler und Probleme sind im Abschlussbericht erwähnt. Ebenso wird auf deren Behebung eingegangen. Alle anderen behobenen Fehler und Probleme der Applikation sollen hier nicht explizit erwähnt werden.

---

[1] VTK ist ein eingetragenes Warenzeichen von Kitware, Inc.

[2] MFC ist ein eingetragenes Warenzeichen der Microsoft Corp.

2 Entwicklungsumgebung

Als Entwicklungsumgebung wählen wir wie unsere Vorgänger - das OCT3D-Team - Visual Studio[1] 6.0. Da wir in der Semesterarbeit mit dem Visual Studio .NET 2004 gearbeitet haben, wollten wir das Projekt eigentlich in der neuen Entwicklungsumgebung entwickeln. Wir stiessen dabei auf verschiedene Probleme, die wir nicht so einfach in den Griff bekamen ▶Abschnitt 2.3. Nachdem wir viel Zeit damit verloren haben, beim Versuch die Probleme zu lösen, entschieden wir uns doch wieder auf die alte Version des Visual Studio zurückzuwechseln, um hier nicht noch mehr Zeit zu verlieren.

2.1 Einstellungen

Pfade im Visual Studio und Windows-Umgebungsvariablen:

Pfad für…	Pfade
Include files	C:\OCT3\3RDPARTY\INTERBASE\INCLUDE C:\OCT3\3RDPARTY\VTK\INCLUDE C:\OCT3\SOURCE\INCLUDE C:\OCT3\3rdParty\OCT2 C:\OCT3\3rdParty\OCT2\CSC
Libraries files	C:\OCT3\3rdParty\interbase\lib_ms C:\OCT3\SOURCE\LIB

Tabelle 2-1 Pfade im Visual Studio

Umgebungsvariable	Wert
INTERBASE	C:\OCT3\3rdparty\interbase
OCT3D	C:\OCT3\source\oct3d
VTK	C:\OCT3\3rdParty\vtk

Tabelle 2-2 Umgebungsvariable

2.2 Die Verzeichnisstruktur

Da wir beide an unterschiedlichen Teilen der Applikation arbeiten und beide ihren Code testen wollen, haben wir eine einheitliche Verzeichnisstruktur für den Sourcecode auf beiden Rechnernvorgesehen. Dies wird es uns erleichtern, die Projekte (.dsw, .dsp) untereinander auszutauschen. Zudem gelten so die oben genannten Einstellungen:

```
C:\+ OCT3 -+- 3rdparty
          +- database
          +- source ----+- ext
                        +- include
                        +- lib
                        +- oct3d
```

Fig. 2-1 Verzeichnisstruktur

[1] Visual Studio ist ein eingetragenes Warenzeichen der Microsoft Corp.

2.3 Probleme Visual Studio .NET 2004

Wir haben zuerst das Projekt in Visual Studio .NET 2004 übernommen. Beim Kompilieren erhielten wir die folgende Fehlermeldung:

```
d:\Development\Toolkits\vtk42\include\vtk\vtkIOStream.h(71) : fatal error
C1083: Cannot open include file: 'iostream.h': No such file or directory
```

Aufgrund der unterschiedlichen Implementation der Klasse `iostream`, mussten wir die entsprechenden .NET-VTK-Klassen herunterladen ▶ http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore98/html/_core_differences_in_iostream_implementation.asp

Obwohl dadurch das Problem mit den VTK-Klassen behoben wurde, war es nicht möglich das Projekt OCT3D zu kompilieren. Aufgrund der zahlreichen Fehler, entschieden wir uns, zurück auf das Visual Studio 6 zu wechseln:

2.3.1 Lösung:

```
Die Header fstream.h und ios.h werden nicht gefunden.
```

Diese Headerdateien waren gemäss http://www.codeguru.com/forum/showthread.php?p=1000055#post1000055 nie nach Visual Studio Standard.

Da sich die Headerdateien in den beiden VS unterscheiden, muss `#include <fstream.h>` durch `#include <fstream>` ersetzten werden. Zusätzlich müssten auch noch weitere Methodenaufrufe geändert werden.

Aus

```
stream.open(strPath, ios::in|ios::nocreate|ios::binary, filebuf::sh_read);
```

wird

```
stream.open(strPath, ios::in|ios::binary);
```

Das Problem betrifft folgende Dateien:

- `SelectionDoc.cpp`
- `OCTExportReader.cpp`
- `FileWriter.cpp`
- `OCTRawData.h`

2.3.2 Probleme mit CFileException

```
Die Konstruktoren/Destruktoren bzw. CopyKonstruktoren für die CFileException
können nicht gefunden werden.
```

Das Problem betrifft folgende Dateien:

- `SelectionDoc.cpp`
- `OCTExportReader.cpp`
- `FileWriter.cpp`
- `OCTRawData.h`
- `oct3d.cpp`

2.3.3 Umgebungsvariabeln können nicht gefunden werden

```
Die Umgebungsvariabeln können nicht gefunden werden.
```

Entweder mit der Visual Studio Kommandozeile oder via Systemsteuerung die Umgebungsvariablen gemäss Abschnitt 2 setzen. Danach aber unbedingt Visual Studio neu starten.

2.3.4 Cannot convert parameter

```
O3DEditableModule.cpp: cannot convert parameter 1 from 'const CString' to
'CString &'
```

2.3.5 error C2668

```
c:\OCT3\source\oct3d\AVIWriterThread.cpp(175): error C2668: 'abs' : ambiguous
call to overloaded function
```

Dabei handelt es sich um folgende Zeile:

```
nFrameInc = abs(AfxGetApp()->GetProfileInt( T("AVI-Film"),
                "Frameincrement", O3D_VTK_AVI_FRAMEINCREMENT));
```

Details hierzu ▶http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore/html/C2668.asp

2.4 windows.h already included

```
windows.h already included
```

Die Datei stdafx.h muss so angepasst werden, so dass das System weiss, um welches Betriebssystem es sich handelt. Folgender Code sollte in der Datei stehen.

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__756B69C5_4678_49C6_88AC_E69E6D4D3A32__INCLUDED_)
#define AFX_STDAFX_H__756B69C5_4678_49C6_88AC_E69E6D4D3A32__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN        // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include "o3d extension.h"  // OCT3D extenison stuff


//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_STDAFX_H__756B69C5_4678_49C6_88AC_E69E6D4D3A32__INCLUDED_)
```

3 Ziel 1 - Einbinden OCT-Restoration

Autor	Adrian Wyssmann
Implementation	Adrian Wyssmann
Ziel	Zusammenführen der beiden Applikationen OCT3D und OCT-Restoration
Status	Ziel erreicht
Bemerkung	Alle Filter aus der Diplomarbeit OCT-Restoration sind als Extensions in die Applikation eingebunden und können in einer Prozesspipeline verwendet werden: • Anisotropic-Diffusion-Filter (AnisotropicDiffusionFilterExt) • Cross-Correlation-Filter (CrossCorrelationFilterExt) • CSC-Filter (CSCFilterExt) • Wavelet-Filter (WaveletFilterExt) • Wiener-Filter (WienerFilterExt)

Tabelle 3-1 Übersicht Ziel 1 - Einbinden "OCT Restoration"

3.1 Implementation

Unsere Vorgänger von der Diplomarbeit OCT3D haben ihre Applikation so gestaltet, dass man ihr auf einfache Weise weitere Prozesse als Erweiterung implementieren kann. Hierfür dient das **Extension Development Kit (EDK)[1].** Das EDK ist ein Set von Strukturen, Interface-Prozessen und Helper-Klassen, die es erlauben, ein Extension-Prozess für OCT3D als Windows[2] DLL zu entwickeln.

Für die Implementation der Filter aus dem OCT-Restoration, musste ich als erstes den Mechanismus verstehen, wie das EDK aufgebaut ist und wie man mit dessen Hilfe einen eigenen Prozess implementieren kann. Dank guter Dokumentation des [EDK] fiel mir das ziemlich einfach. Ich machte mich anschliessend daran machen, die Prozessklasse gemäss [EDK] zu erstellen.

Um eine Filterklasse aus dem OCT-Restoration nutzen zu können, mussten in der Headerdatei der Prozessklasse (`CO3DExtensionProcess`) die folgenden Dateien eingebunden werden:

```
#include "OCTRawData.h"
#include "WaveletFilter.h"
```

Code 3-1 WaveletFilterExtProc.h

Da die OCT-Restoration-Filter so implementiert sind, dass sie ein `OCTRawData` als Input erwarten, wird die entsprechende Klasse benötigt. Bei der zweiten Klasse handelt es sich um die Filterklasse aus der OCT-Restoration Anwendung.

3.1.1 Probleme

Ich hatte zuerst Probleme, die bestehenden Filterklassen des OCT-Restoration einzubinden, da ich versucht habe eine Klassenvariable vom entsprechenden Typ zu initialisieren, also etwa `CCrossCorrelateFilter myfilter;` Daraus resultierte folgender Fehler:

```
c:\oct3\source\ext\crosscorrelationfilterext\crosscorrelationfilterextproc.h(45
) : error C2259: 'CrossCorrelateFilter' : cannot instantiate abstract class due
to following members:
   c:\oct3\3rdparty\oct2\crosscorrelatefilter.h(32) : see declaration of
   'CrossCorrelateFilter'
```

[1] [EDK]

[2] Windows ist ein eingetragenes Markenzeichen der Microsoft.

Da es sich bei diesen Klassen um eine abstrakte Klasse handelt, ist es also nicht erlaubt eine Instanz dieser Klasse zu definieren. Ich verwende die Klassen folgendermassen:

```
CCrossCorrelateFilter::apply(myOCTRawData);
```

3.1.2 Verarbeitung der Input-Daten

Da die Filterklasse ein `OCTRawData` als Input erwartet, müssen wir als erstes die Daten die der Prozess erhält in ein `OCTRawData` konvertieren. Der Datenteil von `OCTRawData` ist vom Typ `unsigned short`. Wir casten die Inputdaten (`m_pDataIn->pImageData->pArrImageData[n]`) durch ein `reinterpret_cast` in einen Array vom Typ Pointer auf `unsigned short`. Den Pointer übergeben wir dann als Parameter für das `new OCTRawData()`:

```
data = reinterpret cast<unsigned short*>(m pDataIn->pImageData->
                                         pArrImageData[n]);

myOCTRawData = new OCTRawData(scanID,instanceNum,width,height,scanlength,
                              scanpattern,scantype,date time,
                              bitDepth,data);
```

Code 3-2 CWaveletFilterExtProc::GetOutput - Konvertierung in OCTRawData

Nun können die Daten durch die Filterklasse verarbeitet werden:

```
if (m lAdaptiveParam)
   WaveletFilter::setAdaptive(true);
else
   WaveletFilter::setAdaptive(false);

if (m lUseLogParam)
   WaveletFilter::setUseLog(true);
else
   WaveletFilter::setUseLog(false);

WaveletFilter::setLevel(m lWaveletLevelParam);
WaveletFilter::setAlpha(m dAlphaParam);
WaveletFilter::setBeta(m dBetaParam);

//apply WaveletFilter
WaveletFilter::apply(*myOCTRawData);
```

Code 3-3 CWaveletFilterExtProc::GetOutput() - Verarbeitung der Input-Daten

Da das OCT3D-Team Boolean-Werte als `Long` implementiert hat, verwenden wir folgendes Konstrukt, um der Filterklasse `Boolean`-Werte zu übergeben:

```
if (m lAdaptiveParam)
   WaveletFilter::setAdaptive(true);
else
   WaveletFilter::setAdaptive(false);
```

Dadurch wird folgende Fehlermeldung vermieden:

```
warning C4800: 'long' : forcing value to bool 'true' or 'false' (performance
warning)
```

3.1.3 Weitergabe der Daten

Für die Weitergabe verwenden wir den Code aus der Dokumentation [EDK]. Mit der Methode `myOCTRawData->getData()` erhalten wir den Pointer die Daten von `OCTRawData` (Pointer vom Typ `unsigned short`):

```
...
   psScalarsNew   = new unsigned short[nSize];
   pImageData->SetData(n, psScalarsNew);

   memcpy(psScalarsNew, myOCTRawData->getData(), nSize * sizeof(unsigned short));
   pTemp   = psScalarsNew ;
   pContext->feedback.nCurrent++; //feedback
}
SetModified(O3D CLEAN) ; //reset modified flag

*pData = m pDataOut ; //return output
return O3D_SUCCESS ;
```

Code 3-4 CWaveletFilterExtProc::GetOutput() - Weitergabe der Daten

3.2 Probleme

Das Erstellen der einzelnen Extensions verlief nicht ganz problemlos. Beim Kompilieren sind wir häufig auf Fehler gestossen. Wie sich herausgestellt hat, sind die meisten Fehler auf Kompatibilitätsproblem der verschiedenen Entwicklungsumgebungen zurückzuführen: Das OCT-Restoration Team verwendete für seine Klassen den Borland[1] Builder, während das OCT3D-Team - ebenso wie wir - das Visual Studio 6.0 von Microsoft verwendeten.

3.2.1 Kompatibilität Visual Studio ↔ Borland Builder

Um die Kompatibilität der Klassen zu gewährleisten, wäre es wahrscheinlich schlauer, `#inlcude` und `#pragma` Direktiven abhängig vom eingesetzten Compiler zu machen, anstatt die Direktiven einfach auszukommentieren:

```
#ifdef _VS_  //Visual Studio
#include xyz.h
#pragma xyz
#endif

#ifdef  BCC  //Borland
#include xyz.h
#pragma xyz
#endif
```

Code 3-5 Kompatibilität

Wir haben für das VS keine Pragmas eingeführt und ignorieren die Warnungen des Compilers.

3.2.1.1 Unterschiedliche Pragmas

Visual Studio und Borland Builder kennen beide Pragmas[2] (Compiler Direktiven). Allerdings sind diese vom Compiler festgelegt. Trifft der Compiler auf ein ihm unbekanntes Pragma, gibt er eine Warnung aus:

```
c:\oct3\3rdparty\oct2\waveletdenoiser.cpp(24) : warning C4068: unknown pragma
```

Die Fehlermeldung können wir, wie im Abschnitt 3.2.1 beschrieben, vermeiden.

[1] Borland ist ein eingetragenes Warenzeichen der Borland Software Corporation

[2] Compiler Direktiven: Damit kann ein Programmierer dem Compiler eine bestimmte Absicht mitteilen

3.2.1.2 Merfachdeklaration von Variablen

Grundsätzlich wäre folgende Deklaration nach ANSI-Standard erlaubt, da der Gültigkeitsbereich für die Laufvariable die Schlaufe selber ist:

```
for (int i = 0; i < 10; i++) { /*do something*/ }
for (int i = 0; i < 10; i++) { /*do something other*/ }
```

Code 3-6 Merfachdeklaration von Variablen

Leider quittiert der Microsoft Compiler dies mit einer Fehlermeldung, da er erkennt, dass dieselbe Variable mehr als einmal im gleichen Kontext definiert wird:

```
c:\oct3\3rdparty\oct2\wienerfilter.cpp(40) : error C2374: 'l' : redefinition;
multiple initialization
c:\oct3\3rdparty\oct2\wienerfilter.cpp(32) : see declaration of 'l'
```

Das Problem hatten wir oft, da die Filterklassen mit dem Borland Builder entwickelt wurden. Der Borland Compiler (BCC) hat im Gegensatz zu dem Microsoft Compiler kein Problem mit solchen Mehrfachdeklarationen.

Das Problem hängt offensichtlich mit der Language Extensions[1] des Visual Studios zusammen. Dabei handelt es sich um eine Erweiterung der Funktionen des ANSI-C-Standard von Microsoft. Diese hält sich aber anscheinend nicht mehr an alle gültigen ANSI-C-Standards. Wird die Language Extension ausgeschalten, so sind nur noch ANSI-C-Befehle gültig. Nach dem Ausschalten der Language Extension, bekommen wir diverse Fehlermeldungen, die sich auf die Datei `winnt.h` beziehen:

```
<d:\development\microsoft visual studio 6\vc98\include\winnt.h(357) : error
C2467: illegal declaration of anonymous 'struct'
d:\development\microsoft visual studio 6\vc98\include\winnt.h(376) : error
C2467: illegal declaration of anonymous 'struct'
d:\development\microsoft visual studio 6\vc98\include\winnt.h(1519) : error
C2146: syntax error : missing ';' before identifier 'PVOID'
d:\development\microsoft visual studio 6\vc98\include\winnt.h(1519) : error
C2501: ' inline' : missing storage-class or type specifiers
d:\development\microsoft visual studio 6\vc98\include\winnt.h(1519) : fatal
error C1004: unexpected end of file found
```

Die Suche im Internet hat mir hier nicht weitergeholfen. Da es mir wenig aussichtslos erscheint, diese Fehlermeldungen zu debuggen, schalten wir die Language Extensions wieder ein und passen die Filterklassen so an, dass keine Mehrfachdeklarationen mehr vorkommen:

```
int i;
for (i = 0; i < 10; i++) { /*do something*/ }
for (i = 0; i < 10; i++) { /*do something other*/ }
```

Code 3-7 Merfachdeklaration von Variablen entfernen

3.2.1.3 error C2457

```
Compiler Error C2467: illegal declaration of anonymous 'user-defined-type'
```

Diese Meldung erscheint, wenn die Language Extensions ausgeschaltet ist ▶ http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore98/html/c2467.asp.

[1] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore/html/_core_.2f.za.2c_2f.ze.asp

3.2.1.4 LNK2001: unresolved external symbol

```
AnisotropicDiffusionFilterExtProc.obj : error LNK2001: unresolved external
symbol "public: static void   cdecl AnisoDiffFilter::apply(class OCTRawData &)"
(?apply@AnisoDiffFilter@@SAXAAVOCTRawData@@@Z)
AnisotropicDiffusionFilterExtProc.obj : error LNK2001: unresolved external
symbol "public: static void   cdecl AnisoDiffFilter::setUseLog(bool)"
(?setUseLog@AnisoDiffFilter@@SAX_N@Z)
```

Laut Microsoft gibt es hier verschiedene Ursachen[1]. Ich habe das Problem gelöst, indem ich die entsprechende(n) CPP-Datei(en) dem Projekt hinzugefügt habe:

Projekt	Datei	Bemerkung
AnisotropicDiffusionFilterExt	AnisoDiffFilter.cpp	Klasse OCT-Restoration Filter
CrossCorrelationFilterExt	CrossCorrelateFilter.cpp	Klasse OCT-Restoration Filter
	Smooth.cpp	C:\OCT3\3rdparty\OCT2
CSCFilterExt	CSCFilter.cpp	Klasse OCT-Restoration Filter
	adverror.cpp	C:\OCT3\3rdparty\OCT2\csc\
	ipcislhierarchy.cpp	C:\OCT3\3rdparty\OCT2\csc\
WaveletFilterExt	WaveletFilter.cpp	Klasse OCT-Restoration Filter
	RedundantWT2D_MMX.cpp	C:\OCT3\3rdparty\OCT2\
	RedundantWT_MMX.cpp	C:\OCT3\3rdparty\OCT2\
	WaveletDenoiser.cpp	C:\OCT3\3rdparty\OCT2\
WienerFilterExt	WienerFilter.cpp	C:\OCT3\3rdparty\OCT2\

Tabelle 3-2 Zusätzliche Projektdateien für Filter-Extensions

Für alle diese Dateien muss die Option **"◉ Not Using Precompiled Headers"** ausgewählt werden, ansonsten erhalten wir folgende Fehlermeldung:

```
c:\oct3\3rdparty\oct2\CSCFilterExt.cpp(69) : fatal error C1010: unexpected end
of file while looking for precompiled header directive
```

3.2.1.5 LNK2005: already defined in LIBCMTD.lib

```
nafxcwd.lib(dllmodul.obj) : error LNK2005:  DllMain@12 already defined in
LIBCMTD.lib(dllmain.obj)
nafxcwd.lib(afxmem.obj) : error LNK2005: "void *   cdecl operator new(unsigned
int)" (??2@YAPAXI@Z) already defined in LIBCMTD.lib(new.obj)
nafxcwd.lib(afxmem.obj) : error LNK2005: "void   cdecl operator delete(void *)"
(??3@YAXPAX@Z) already defined in libcpmtd.lib(delop.obj)
nafxcwd.lib(dllmodul.obj) : warning LNK4006:  DllMain@12 already defined in
LIBCMTD.lib(dllmain.obj); second definition ignored
nafxcwd.lib(afxmem.obj) : warning LNK4006: "void *   cdecl operator
new(unsigned int)" (??2@YAPAXI@Z) already defined in LIBCMTD.lib(new.obj);
second definition ignored
nafxcwd.lib(afxmem.obj) : warning LNK4006: "void   cdecl operator delete(void
*)" (??3@YAXPAX@Z) already defined in libcpmtd.lib(delop.obj); second
definition ignored
```

Diese Fehlermeldung erhalten wir, wenn die C Run-Time Library (CRT) und die Microsoft Foundation Class Libraries (MFC) in der falschen Reihenfolge verlinkt werden.[2] Wenn MFC Libraries

[1] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore98/html/lnk2001.asp

[2] http://support.microsoft.com/default.aspx?scid=kb;en-us;148652

verwendet werden, müssen diese immer **vor** der CRT Library verlinkt werden. Das wird sichergestellt, indem in allen Dateien des Projekts, als erstes die Datei `Msdev\Mfc\Include\Afx.h` einbinden - entweder direkt (`#include <Afx.h>`) oder indirekt (`#include <Stdafx.h>`). Keine der bestehenden Klassen aus dem OCT-Restoration verwendet diese Anweisung.

Wir können dem Linker aber direkt mitteilen, dass er die richtige Reihenfolge einhalten soll, indem wir in den Projekteinstellungen folgende Einstellungen anpassen:

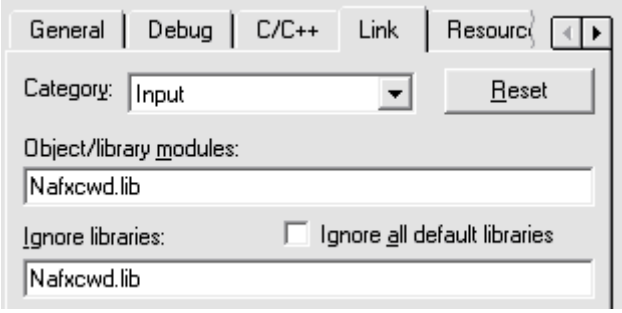


Fig. 3-1 Einstellungen VS6 Linker

3.2.2 Anisotropic-Diffusion-Filter

Der Anisotropic-Diffusion-Filter liess sich ohne Probleme implementieren.

3.2.3 Cross-Correllation-Filter

Folgende Fehler sind aufgetaucht und gemäss Beschreibung behoben worden:

3.2.3.1 fatal error C1083: mem.h

```
c:\oct3\3rdparty\oct2\smooth.cpp(3) : fatal error C1083: Cannot open include
file: 'mem.h': No such file or directory
```

`Mem.h` ist eine Headerdatei des Borland-Compilers. Sie definiert Methoden für die Speichermanipulation. Da wir aber mit dem Visual Studio arbeiten, müssen wir die entsprechende Klasse des Microsoft Compilers einbinden. Die entsprechende Datei heisst `Memory.h`:

```
#include "Smooth.h"
#include <math.h>
#include <memory.h> //mem.h
...
```

Code 3-8 Änderungen in c:\oct3\3rdparty\oct2\smooth.cpp

3.2.3.2 error C2065: M_PI

```
c:\oct3\3rdparty\oct2\crosscorrelatefilter.cpp(111) : error C2065: 'M_PI' :
undeclared identifier
```

Die Variable `M_PI` ist nicht definiert. Durch Deklaration der Variable lässt sich dieses Problem beheben:

```
#ifndef M_PI
   const double M_PI =  3.14159265358979323846 ;
#endif
```

Code 3-9 Änderungen in c:\oct3\3rdparty\oct2\crosscorrelatefilter.h

3.2.3.3 error C2782/C2784: class std::complex

Die Klasse `CrossCorrelateFilter` greift auf das Template für Komplexe Zahlen zu. Beim kompilieren im Visual Studio erhalten wir allerdings folgenden Fehler:

```
c:\oct3\3rdparty\oct2\crosscorrelatefilter.cpp(155) : error C2784: 'class
std::complex< Ty>   cdecl std::operator *(const  Ty &,const class
std::complex< Ty> &)' : could not deduce template argument for 'const  &' from
'class std::complex<double>'
c:\oct3\3rdparty\oct2\crosscorrelatefilter.cpp(155) : error C2782: 'class
std::complex< Ty>   cdecl std::operator *(const class std::complex< Ty> &,const
...
```

Eigentlich sollte der Microsoft Compiler die Anweisung `using std::complex` verstehen. Erst durch Änderung in `using namespace std` ist das Kompilieren ohne Fehler möglich:

```
//using std::complex;
using namespace std;
```

Code 3-10 Änderungen in c:\oct3\3rdparty\oct2\crosscorrelatefilter.h

Laut [MSDN] ist Unterschied hier, dass es sich bei der Anweisung `using namespace std` um eine using-Direktive handelt. Sie erlaubt es Namen eines anderen Namespace in diesem Namespace zu benutzen, ohne den Qualifier explizit anzugeben. Im Gegensatz zu der using-Deklaration (`using std::complex`) erlaubt es die using-Direktive, alle Namen des fremden Namespaces ohne Qualifier zu benutzen.

3.2.4 CSC-Filter

Abgesehen von den Kompatibilitätsproblemen (▶Abschnitt 3.2.1) sind hier keine Probleme aufgetaucht.

3.2.5 Wavelet-Filter

Folgende Fehler sind aufgetaucht und gemäss Beschreibung behoben worden:

3.2.5.1 fatal error C1083: values.h

```
c:\oct3\3rdparty\oct2\waveletdenoiser.cpp(21) : fatal error C1083: Cannot open
include file: values.h': No such file or directory
```

`Values.h` ist eine Headerdatei des Borland-Compilers. Sie definiert wichtige Konstanten, einschliesslich Maschinen Abhängigkeiten. Sie wird zur Verfügung gestellt für UNIX System V Kompatibilität[1]. Da wir die Klasse für eine Windows Anwendung benötigen, scheint diese Datei überflüssig.

```
#include "WaveletDenoiser.h"
#include <stdio.h>
//#include <values.h>
#include <math.h>
//#include "Globals.h"
...
```

Code 3-11 Änderungen in c:\oct3\3rdparty\oct2\WaveletDenoiser.h

[1] http://poli.cs.vsb.cz/c/help/allbcpp.htm

3.2.5.2 fatal error C1083: vcl.h

```
fatal error C1083: Cannot open include file: 'vcl.h': No such file or directory
```

VCL steht für **Visual Component Library** und wurde von Borland entwickelt. Es handelt sich dabei um ein Framework zum erstellen von Windows Anwendungen. Microsoft kennt VCL nicht, sondern verwendet analog dazu MFC. Beim Erstellen einer Anwendung mit dem Borland Builder wird diese Datei automatisch eingebunden. Sie wird in dieser Klasse aber sowieso nicht benötigt:

```
//#include <vcl.h>
#pragma hdrstop
...
```

Code 3-12 Änderungen in RedundantWT_MMX.cpp/RedundantWT_MMX.h

3.2.6 Wiener-Filter

Abgesehen von den Kompatibilitätsproblemen (▶Abschnitt 3.2.1) sind hier keine Probleme aufgetaucht.

4 Ziel 2 - Implementation Segmentierungsalgorithmus

Autor	Markus Fawer
Implementation	Markus Fawer
Ziel	Implementierung des Pyramid-Linking Segmentierungsalgorithmus für 2D-Bilder.
Status	Ziel erreicht
Bemerkung	Die Segmentierung basiert in einem ersten Schritt auf dem Pyramid-Linking Verfahren. Zusätzlich wurden noch ergänzende Massnahmen zur Unterscheidung der verschiedenen Schichten vorgenommen. Das gesamte Vorgehen für die Segmentierung ist im Abschnitt 4.1 beschrieben.

Tabelle 4-1 Übersicht Ziel 2 - Implementation Segmentierungsalgorithmus

4.1 Beschreibung des Pyramid-Linking Algorithmus

4.1.1 Segmentierung

4.1.1.1 Aufgaben einer Segementierung

Bei einer Segmentierung eines Bildes geht es darum zusammenhängende Teile eines Bildes zu finden und diese als eine Einheit zu markieren. Ein Beispiel eines Bildes vor und nach einer Segmentation.

Fig. 4-1 Bild vor…

Fig. 4-2 ..und nach einer Segmentierung

4.1.1.2 Eigenschaften einer Segmentierung

Das Segmentierte Bild kann gemäss der Tabelle 4-2 verschiedene Eigenschaften aufweisen. Diese Eigenschaften sind unabhängig voneinander, sodass ein Bild eine, zwei oder alle drei Eigenschaften aufweisen kann.

Art der Segmentierung	Eigenschaften
vollständige Segmentierung	Jedes Pixel wird (mindestens) einem Segment zugeordnet.
überdeckungsfreie Segmentierung	Kein Pixel wird mehr als einem Segment zugeordnet.
zusammenhängende Segmentierung	Jedes Segment bildet ein zusammenhängendes Gebiet.

Tabelle 4-2 Arten der Segmentierung

Wir unterscheiden deshalb auch nach der Qualität der Lösung nach den folgenden Gesichtspunkten:

Art der Lösung	Eigenschaften
unvollständige Lösung	Es wurden nicht alle Pixel die zu einem Segment gehören dem Segment zugeordnet.
vollständige Lösung	Es wurden alle Pixel die zu einem Segment gehören dem Segment zugeordnet. (Es ist aber auch möglich das zusätzliche Pixel zu dem Segment gezählt werden).
korrekte Lösung	Es wurden alle Segmente erkannt und alle Pixel wurden dem richtigen Segment zugeordnet. Die Anzahl der Pixel die dem Segment zu geordnet wurden ist korrekt.

Tabelle 4-3 Art der Lösung der Segmentierung

Für das ganze Bild unterscheiden wir zusätzlich noch die Anzahl der gefundenen Segmente nach:

Menge der Segmente	Eigenschaften
übersegmentierung	Es wurden zuviele Segmente identifiziert.
untersegmentierung	Es wurden nicht alle relevanten Segmente gefunden.

Tabelle 4-4 Menge der gefundenen Segmente

Um ein Bild zu segmentieren gibt es grundsätzlich verschiedene Ansätze:

Segmentierungsverfahren	Eigenschaften
Pixelorienterte Verfahren	Es werden Kritierien des Pixel gewertet um eine Zuweisung zu einem Segment zu erreichen. Dabei haben die anderen Pixel die im Bild vorkommen keinen Einfluss.
Kantenorientierte Verfahren	Es werden die Kanten die im Bild vorkommen, gesucht. Aufgrund dieses Ergebnisses werden Segmente definiert.
Regionenorientierte Verfahren	Regionen eines Bildes werden zusammengefasst, um die Segmente zu erhalten.
Modellorientierte Verfahren	Es wird die Kenntnisse über die Form des gesuchten Ergebnisses (Model) verwendet.
Texturorientierte Verfahren	Bei Texturorientierten Verfahren werden die Kenntnisse z.B. über die Struktur der gesuchten Oberfläche zum Suchen der Segmente verwendet.

Tabelle 4-5 Segmentierungsverfahren nach Kategorien

4.1.2 Verwendeter Algorithmus in der Diplomarbeit

Wir haben während unserer Semesterarbeit verschiedene Verfahren untersucht und Ihre Fähigkeiten zur Segmentierung der OCT Bilder getestet.

Die Eigenschaften der Bilder liessen grundsätzlich nur einige wenige Algorithmen für eine genaueren Untersuchung zu. Aufgrund der folgenden Eigenschaften der Bilder kamen für unseren Zweck weder die Pixel- noch die Kantenbasierten Verfahren in Frage:

1. Die Bilder sind stark verrauscht. Es gibt sehr viele Störungen in den Bildern die zum Teil nicht von der eigentlichen Information im Bild zu unterscheiden sind.
2. Der relevante Bereich des Bildes ist im Verhältnis zu der Grösse des Bildes sehr klein, und hat damit eine eher kleine Auflösung.
3. Die Bilder sind sehr unterschiedlich, was die Helligkeit der Aufnahme betrifft.
4. Die Aufnahmen sind sehr unterschiedlich was die Qualität der Ausrichtung betrifft.

Wir fanden vor allem den Pyramid-Linking Algorithmus für geeignet OCT Bilder zu segmentieren.

4.1.2.1 Pyramid-Linking Verfahren

Beim Pyramid-Linking Verfahren, wird eine so genannte Auflösungspyramide verwendet. Fig. 4-3 zeigt die Auflösungspyramide eines beliebigen Bildes. Die Auflösungspyramide besteht aus mehreren Ebenen mit einer Grundfläche, die identisch mit dem Originabild ist. Die restlichen Ebenen zeigen das Originalbild in jeweils einer kleineren Auflösung. Die Farbe der oberen Ebene werden aus einem Mittelwert bestimmter Pixel aus der unteren Ebene gerechnet.

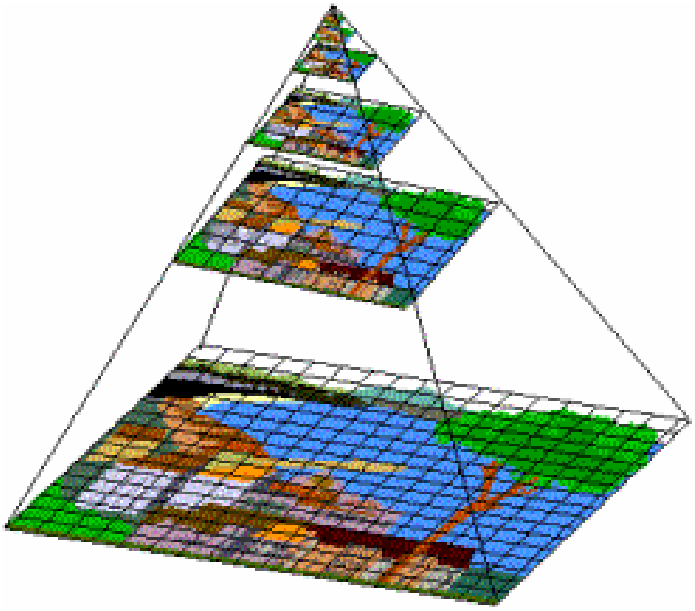


Fig. 4-3 Beispiel einer Auflösungspyramide

Ein Pixel in einer oberen Ebene nennt man Vaterpixel, während die Pixel in einer unteren Ebene als Kindpixel bezeichnet wird. Beim Pyramid-Linking Verfahren werden jeweils 16 Pixel in einem 4 x 4 Raster zusammengefasst davon dann der Mittelwert des Vaters berechnet. Dieses Raster, auch Kernel genannt, wird nun um 2 Pixel in der selben Zeile verschoben und erneut ein Mittelwert für den nächsten darüberliegenden Vaterpixel gerechnet. Nach dem Fertigstellen einer Zeile, wird der Kernel um zwei Zeilen verschoben. Der Kernel durchläuft nun diese Zeile und berechnet deren Vaterpixel. Durch dieses Verfahren, nimmt ein Kindpixel auf den Mittelwert von vier möglichen Vaterpixel Einfluss. Diese Tatsache kann mit Hilfe der Fig. 4-4 verifiziert werden.

Nun wird für jedes Kindpixel geschaut zu welchem von seinen 4 möglichen Vaterpixeln es die kleinste Differenz hat. Dieses Vaterpixel wird dann als aktuelles Vaterpixel gesetzt. Dieser Vorgang wird nun für jede Ebene wiederholt. In einem nächsten Schritt werden die Mittelwerte aller Vaterpixel berechnet. Nun wird neu mit einem gewichteten Mittelwert gearbeitet. Die Gewichtung des Mittelwertes erfolgt nun für jedes Pixel mit der Anzahl der Pixel in der Grundfläche die zum aktuellen Pixel über eine Verbindung verfügen. Die Formel hierzu lautet also wie folgt:

$$h_{current} = \sum_{(x,y)} \frac{h_{child} * c_{base}}{c_{possible}}$$

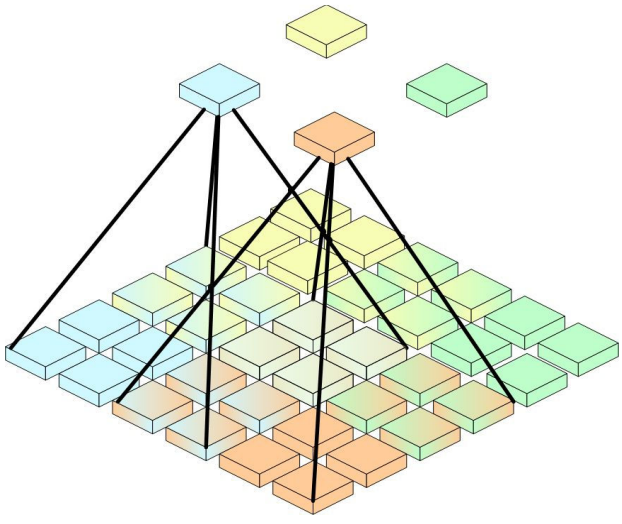


Fig. 4-4 Vater – Kind Beziehungen im Pyramid-Linking

wobei gilt:

$h_{current}$: Helligkeit des aktuellen Vaterpixels

h_{child} : Helligkeit des Kindpixels

c_{base} : Anzahl Kinder in der Grundfläche

$c_{possible}$: Anzahl möglicher Kinder in der Grundfläche

Aufgrund der Neuberechnung der Vaterwerte, kann es vorkommen, dass es für ein Kindpixel einen anderen möglichen Vater als den aktuell gesetzten gibt, welcher die kleinste Differenz zum Kindpixel besitzt. Diese Beziehungen werden wieder so angepasst, dass das Kindpixel wieder mit dem Vater verbunden ist zu dem es die kleinste Differenz hat. Aufgrund der Änderungen der Beziehungen, ist dann natürlich auch wieder eine Neuberechnung der Mittelwerte nötig. Diese zwei Schritte werden solange ausgeführt bis ein stabiles Ergebnis erzielt wird. Fig. 4-5 zeigt den Ablauf des Pyramid-Linking schematisch auf.

Fig. 4-5 Ablauf des Pyramid-Linking Algorithmus

4.1.2.2 Zusätzliche Bearbeitungsschritte

Der im Abschnitt 4.1.2.1 beschriebene Algorithmus funktioniert für viele Anwendungen einwandfrei. Mit diesem Algorithmus war es möglich die Bilder des OCT Gerätes zu segmentieren, dass die verschiedenen Gewebedichtheiten im Auge zu unterscheiden waren. Dies war allerdings für die von uns gesuchten Schichten im Auge noch nicht ausreichend, weil eine Schicht nicht nur durch die Dichtheit des Gewebes, sondern auch durch die Lage zu unterscheiden war. Zum Beispiel kann eine Nervenschicht dieselbe Dichtheit aufweisen wie eine Schicht mit Blutbahnen, diese sollten aber voneinander zu unterscheiden sein. Dazu mussten wir also zusätzliche Informationen über die Schichten, die wir haben verwenden. Wir haben uns dazu entschlossen, die Pigmentsschicht die normalerweise als die hellste Schicht in einem OCT Bild zu erkennen ist zu verwenden, um zwischen den Schichten zu unterscheiden.

Um die Pigmentschicht zu finden verwendeten wir ein Verfahren bei dem die hellsten Schichten in dem Bild benutzt wurden um Hilfslinien in das Bild zu zeichnen. Mithilfe dieser Linien war es uns dann möglich, eine erste Unterscheidung der Schichten zu machen. Wir können so eine erste Einfärbung der Segmente vornehmen, die sicher unterhalb der **Pigmentschicht** liegen. Da wir aber sehr unterschiedliche Arten von Bildern haben, und uns auch nicht darauf verlassen können, alle nach dem segmentieren noch möglichen Helligkeiten im Bild zu haben, mussten wir mit Hilfe der nun gefundenen Segmente eine neue Entscheidungslinie rechnen, und die Segmente noch einmal einfärben.

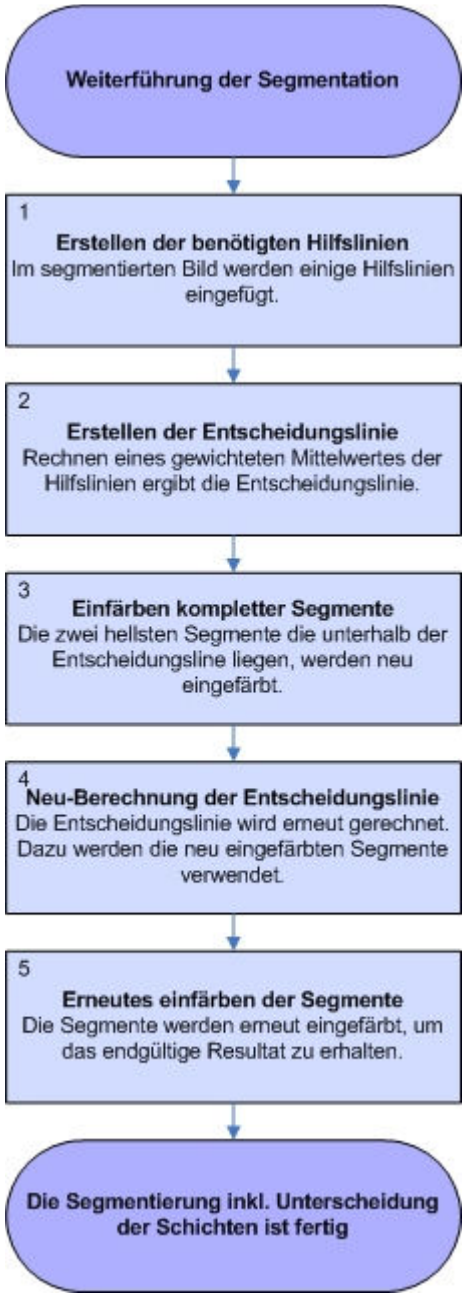


Fig. 4-6 Weiterführende Arbeiten nach der Segmentierung

4.2 Idee

Unsere Vorgänger von der Diplomarbeit OCT3D haben ihre Applikation so gestaltet, dass man ihr auf einfache Weise weitere Prozesse als Erweiterung implementieren kann. Hierfür dient das **Extension Development Kit (EDK).** Das EDK ist ein Set von Strukturen, Interface-Prozessen und Helper-Klassen, die es erlauben, ein Extension-Prozess für OCT3D als Windows DLL's zu entwickeln.

Zuerst erstellte ich die benötigten Header und Source Dateien gemäss [EDK], es handelt sich dabei um:

- `CPyramidLinkingApp.h`
- `CPyramidLinkingApp.cpp`
- `CPyramidLinkingProc.h`
- `CPyramidLinkingProc.cpp`

4.2.1 Registrieren des Prozesses

Da es sich bei unserem Segmentierungsprozess um einen Prozess handelt der sowohl eine Ausgabe erzeugt als auch eine "View" für die Ansicht benötigt, musste ich gemäss [EDK] die vier Methoden in der Klasse `CPyramidLinkingProc` implementieren.

- `virtual int GetOutput(CO3DprocConttext *pContext, O3D_DATA** pData);`
- `virtual int ResetParamValue( int nParam);`
- `virtual int Execute(CO3DprocCOntext* pContext);`
- `virtual int SetWindow(HWND hWnd, O3D_PROC_CONTEXT* pContext);`

Bei diesen Methoden handelt es sich um überladene Methoden der Klasse `CO3DExtensionProcess`. Für eine genaue Beschreibung verweise ich auf das Handbuch [EDK].

Ich verwendete ausserdem zusätzliche Klassen, um die Struktur einer Auflösungspyramide abzubilden. Der Aufbau einer Auflösungspyramide ist im Kapitel 4.1.2.1 beschrieben. Sie besteht zum einen Teil aus Ebenen (Levels) die eine bestimmte Menge von Knoten (Nodes) enthalten. Diese Knoten sind untereinander mit Kanten verbunden. Ich erstellte deshalb die Klasse CNode um einen Knoten, und die Klasse CLevel um eine Ebene in der Pyramide darzustellen. Ein Knoten verfügt über einen Wert der analog zu einem Pixel eine Helligkeit räpresentiert. Die Knoten gehören jeweils zu einem Level in der Pyramide. Dieses Level hat eine bestimmte Ausdehnung und eine Lage innerhalb der Pyramide.

Dies führte mich zur Definition der folgenden Dateien:

- `CNode.h`
- `CNode.cpp`
- `CLevel.h`
- `CLevel.cpp`

Allerdings musste ich noch zusätzliche Methoden einfügen, um die Bearbeitung der Nodes zu vereinfachen. Dies zusammen ergab die Deklaration der beiden Klassen `CNode` und `CLevel`.

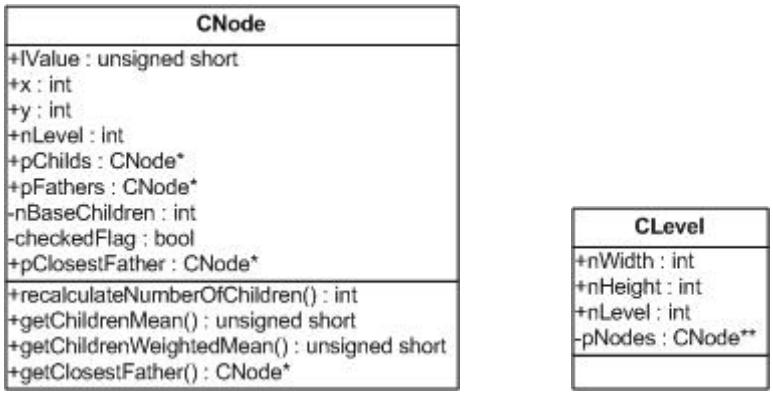


Fig. 4-7 UML Diagramm der Klassen CLevel und CNode

Diese zusätzlichen Methoden werden ausführlich im Kapitel 4.2.2 beschrieben. Wir wollen uns der Klasse `CPyramidLinkingProc` zuwenden und deren Implementation im Detail anschauen.

Zuerst definieren wir statische Prozessdefinition:

```
O3D_PROC_DESC  pyramidLinking = {
   1,                              // Prozess Handler
   "PyramidLinking Process",       // Prozess Name
   O3D_IO_2DANDVIDEO,              // benötigter Eingabe Typ
   O3D_IO_2DANDVIDEO,              // gelieferter Ausgabe Typ
   PARAMETER_COUNT(CPyramidLinkingProc),   // Anzahl der Parameter
   0,                              // Anzahl der Subprozesse
   NULL,                           // Liste der Subprozesse
};
```

Code 4-1 Statische Prozessdefinition des Pyramid-Linking Prozesses

In unserem Prozess verwenden wir einen Parameter, mit dem der Benutzer die Möglichkeit hat eine Segmentierung zu machen die eine Darstellung in der 3D Form ermöglicht, oder nur die Segmentierung des Bildes nach der Dichtheit des Gewebes durchführt.

Für diesen Parameter benötigen wir noch die die Aufrufe für die Makros zum Eintragen des Prozesses und dessen Beschreibungen.

```
IMPLEMENT EXT PROC BASE(CPyramidLinkingProc,  pyramidLinking) ;
//we want default base implementation
IMPLEMENT EXT PARAM DESC(CPyramidLinkingProc,  pyramidLinking) ;
//we want default parameter handling
IMPLEMENT DEFAULT PROCDESC TEXT(CPyramidLinkingProc,  pyramidLinking,
IDS PROC SHORT, IDS PROC TOOLTIP) ;
BEGIN PARAM DESC TEXT MAP(CPyramidLinkingProc)
   PARAM DESC TEXT ENTRY(0, IDS PARAM SHORT 3DSEGMENT,
                         IDS PARAM TOOLTIP 3DSEGMENT)
END_PARAM_DESC_TEXT_MAP()
```

Code 4-2 Eintragen des Prozesses und dessen Beschreibungen

Für die Kurzbeschreibungen und den so genannten Tooltip erstellen wir eine Ressourcen Datei in der wir die Beschreibungen des Prozesses eintragen können.

Anschliessend beginnen wir die Implementation der Methoden. Zuerst die Getter- und Setter-Methoden für den Prozess Parameter, und die Methode zum zurücksetzen des Parameters auf seine Standard Einstellung. Ausserdem sollte noch der Austausch des Parameters implementiert werden. Dazu existiert ein Makro, dass wir verwenden können.

```
O3D_PARAM_EXCHANGE(0, nGet, pValue, m_lSegment);
```

Nun erfolgt die Implementation der Methode `GetOutput()`. In dieser Methode werden die Bearbeitungsschritte für das Segmentieren gemacht.

4.2.2 Beschreibung der `GetOutput()` Methode

Hier passiert das eigentliche Segmentieren des Bildes. Am Anfang der Methode wird zuerst geschaut, ob eine Modifikation der Daten überhaupt nötig ist. Falls dies nicht nötig ist wird `O3D_Succes` zurückgegeben. Dies konnte genauso aus dem `_sample` Prozess übernommen werden, der mit dem EDK mitgeliefert wird.

4.2.2.1 Einlesen der Daten

Nach der Initialisierung des Output Array für das Bild, lesen wir die Daten die wir vom Vorgängerprozess erhalten. Leider ist hier ein Fehler aufgetreten, an dem ich ziemlich lange suchen musste bis ich seine Ursache gefunden hatte. Der Fehler äusserte sich im Versuch zeilenweise durch das Bild zu gehen, und dabei die Auflösungspyramide zu erstellen. Dabei hatte ich immer Probleme mit den Indizes in den Arrays. Ich hatte schon zuvor gehört dass Probleme mit der Funktion zum Speichern der Bilder Probleme aufgetreten sind, und die Dimensionen des Bildes nicht richtig gesetzt wurden. Aus diesem Grund fügte ich ein `Assert()` beim einlesen der Daten ein um sicher zu sein, dass die Höhe und die Breite des gelieferten Bildes auch korrekt übergeben worden sind.

```
ASSERT (nBaseHeight <= nBaseWidth);
```

Da es sich um ein längliches Bild handelt, und dieses `Assert()` keine Fehlermeldung erzeugte, ging ich davon aus, dass der Array mit dem Bild ordnungsgemäss übergeben wurde. Nach längerer Suche merkte ich jedoch, dass die Höhe und die Breite des Bildes nicht dieselbe Auflösung haben, dies weil dass Bild vor dem Anzeigen in die Länge gezogen wird. Die Höhe also bei der Anzeige eine 4 Mal so hohe Auflösung hat wie die Breite. Wenn man in der OCT3D Anwendung ganz nahe an ein Pixel im Bild heranzoomt, kann man erkennen, dass dieses Pixel in der horizontalen und in der vertikalen Richtung eine andere Ausdehnung hat. Ich musste also die Höhe und Breite vertauschen, um das Bild in der richtigen Form speichern zu können.

```
// ATTENTION: height and width Parameters of the input data are perverted.
// Even if the picture has a bigger width than height. The number of pixels in
// y direction is higher than in x direction.
int nBaseHeight = m_pDataIn->pImageData->nWidth;
int nBaseWidth  = m_pDataIn->pImageData->nHeight;
```

Code 4-3 Einlesen der Daten

Das dieser Fehler in den anderen Prozessen nicht bemerkt wurde hängt wohl damit zusammen, dass die Anzahl der verarbeiteten Pixel gleich geblieben ist.
Da wir die Auflösung des Bildes kennen, können wir die Anzahl der Levels der Auflösungspyramide berechnen. Da ein Bild pro Ebene jeweils um die Hälfte kleiner ist, entspricht die Anzahl der Levels dem Zweierlogarithmus der Richtung mit den wenigsten Pixeln plus eins. In unserem Fall ist dies also die Breite (▶ Beginn Abschnitt 4.2.2.1).

```
int nMaxLevels   = 1 + (log((float) nBaseWidth)) / (log(2.f));
```

Code 4-4 Berechnung der Pyramidenhöhe

Anschliessend werden in einer `for()`-Schlaufe alle Bilder der Serie bearbeitet.

4.2.2.2 Definition der Farben die verwendet werden

Wir definieren die Farben die wir in dieser Segmentation verwenden wollen. Die Namen entsprechen ungefähr der Lookuptable welche vom OCT3D-Team definiert wurde.

```
// definition of all used Colors
unsigned short WHITE      = 2200;
unsigned short RED        = 1800;
unsigned short LIGHTRED   = 1700;
unsigned short ORANGE     = 1600;
unsigned short YELLOW     = 1550;
unsigned short GREEN      = 1400;
unsigned short LIGHTGREEN = 1300;
unsigned short LIGHTBLUE  = 1200;
unsigned short DARKBLUE   = 1100;
unsigned short PURPLE     = 1050;
unsigned short BLACK      = 0;
```

Code 4-5 Definition der verwendeten Farben

4.2.2.3 Erstellung der Auflösungspyramide

Die Erstellung der Auflösungspyramide erfolgt in mehreren Schritten. Zuerst werden die Levels für die Auflösungspyramide erstellt. Das erste Level entspricht der Grösse des Originalbildes. Für die darüberliegenden Levels benötigte ich die halbe Höhe und die halbe Breite des jeweils unteren Levels. Ausserdem müssen die Levels eine gerade Anzahl Pixel in der Breite und Höhe aufweisen, weil der Kernel jeweils um 2 Pixel verschoben wird. (▶Abschnitt 4.1.2.1)

```
for (int nLevel = 0; nLevel < nMaxLevels; nLevel++){
  pl_PyramidLevel[nLevel].nHeight  = nHeight;
  pl_PyramidLevel[nLevel].nWidth   = nWidth;
  pl_PyramidLevel[nLevel].nLevel    = nLevel;

  nWidth  = (nWidth  -1) / 2;
  nHeight = (nHeight -1) / 2;
}
```

Code 4-6 Bereitstellen der Levels für die Pyramide

Danach wird das Basislevel geschrieben, dieses Level entspricht dem Originalbild, deshalb besitzen wir alle Angaben die wir zur Erstellung der Knoten (`CNode`) benötigen.

```
CNode** pLevel 0 = new CNode*[nBaseHeight * nBaseWidth];
CNode* node;
for (int y = 0 ; y < nBaseHeight; y++){
  for (int x = 0; x < nBaseWidth; x++){
    nArrayPos       = getArrayPos(nBaseWidth, x, y);
    nImageArrayPos  = GetImageArrayPos(nBaseHeight, x, y);

    // create Node and set parameters
    node = new CNode();
    node->lValue         = pTemp[nImageArrayPos];
    node->nLevel         = 0;
    node->x              = x;
    node->y              = y;
    node->nBaseChildren  = 1;
    node->pClosestFather = NULL;
    node->checkedFlag    = false;
     ...
    pLevel 0[nArrayPos] = node;
  }// end x
}// end y
```

Code 4-7 Schreiben der Knoten in das Basislevel

Da, durch das vertauschen der Höhe und der Breite in den gelieferten Daten, die Positionen der Werte im Array unterschiedlich ist, verwende ich zwei verschiedene Methoden zur Berechnung der Position im Array. `CNode::GetImageArrayPos()` liefert die Position die das Pixel im übergebenen Array hat während `CNode::getArrayPos()` die Position in meinem erzeugten Array zurückgibt.

Diese beiden Methoden liessen mich relativ einfach aus den Koordinaten, die ich zum rechnen der Vater – Kind Beziehungen benötigte die Position der jeweiligen Knoten in den beiden Arrays bestimmen. Mit diesem erhaltenen `int` Wert, den ich als Index verwenden konnte, war es mir möglich sofort auf den entsprechenden Knoten zuzugreifen.

- `CNode::getArrayPos(nBaseWidth, x, y)`
- `CNode::GetImageArrayPos(nBaseHeight, x, y)`

Nachdem nun das gesamte unterste Level komplett erstellt wurde, kann es in die Pyramide übernommen werden. Danach kann, dank den Werten die in der untersten Ebene stehen, die weiteren Levels darüber geschrieben werden. Da wir die Zuweisungen von den Vätern zu den Kindern noch nicht für das gesamte Level komplett fertig gestellt haben, ist es uns noch nicht möglich zu sagen, wieviele Kinder jeder Vater auf der untersten Ebene hat. Diese Anzahl wird uns später als Gewicht für den Mittelwert dienen. Wir verwenden deshalb in einem ersten Durchgang einen Mittelwert der das Arithmetische Mittel aller 16 Kinder als Wert des Vaters setzt. Dieser Mittelwert wird in der Methode `CNode::getChildrenMean()` gerechnet. Sie summiert die Werte aller Kinder dieses Knotens und dividiert diesen Wert durch die Anzahl der Kinder die dieser Knoten besitzt. Da die Helligkeitswert in den OCT Bildern etwa zwischen 800 und 2200 liegen, ein Knoten im Maximum 16 Kinder gesetzt hat, und die Daten als `unsigned short` bearbeitet werden, müssen wir die Summe der Werte vor der Division in einer Variable des Typs `double` speichern, um keinen Datenüberlauf zu erhalten. Mein erster Ansatz die Werte zuerst zu dividieren, und dann erst zusammenzuzählen, habe ich verworfen, weil die Werte nach der Division abgerundet werden, und so Differenzen auftraten.

```
unsigned short CNode::getChildrenMean()
{
  int numberOfChildren   = 0;
  unsigned short us mean = 0;
  double f mean = 0.0;
  if (nLevel != 0){
    for (int i = 0; i < 16 ; i ++){
      if (pChilds[i] != NULL) numberOfChildren++;
    }
    for (int c = 0; c < 16; c ++){
      if (pChilds[c] != NULL){
        f mean += (double) (pChilds[c]->lValue / numberOfChildren);
      }
    }
    us mean = (unsigned short) f mean;
  } else {
    // in der untersten Ebene
    us mean = lValue;
  }
  return us mean;
}
```

Code 4-8 Methode zur Berechnung des Mittelwertes

Ebenfalls in diesem Durchgang setzen wir die Verbindungen zu allen möglichen Kindern, und setzen ebenfalls bei diesen Kindern die Referenz des aktuellen Knotens in den `pFathers` Array. Der `pFathers` Array enthält alle Pointer zu den möglichen Vaterknoten. Der `pChilds` Array enthält analog dazu die Pointer zu den Kindknoten.

```
// for all levels
for (nLevel = 1 ; nLevel < nMaxLevels; nLevel++){
  pContext->feedback.nCurrent++;
  nHeight = pl PyramidLevel[nLevel].nHeight;
  nWidth  = pl PyramidLevel[nLevel].nWidth;
  pLevel  = new CNode* [nHeight * nWidth];

  // for all nodes
  for (int y = 0; y < nHeight; y ++){
    for (int x = 0 ; x < nWidth; x ++){
     nArrayPos         = getArrayPos(nWidth, x, y);
     node              = new CNode;
     node->nLevel      = nLevel;
     node->x           = x;
     node->y           = y;
     node->nBaseChildren   = 0;
     node->pClosestFather  = NULL;
     downWidth  = pl PyramidLevel[nLevel-1].nWidth;
     downHeight = pl PyramidLevel[nLevel-1].nHeight;
     // Kindern suchen und setzen
     for (int j = 0; j < 4 ; j++){
       for (int i = 0; i < 4; i ++){
         // formula to get children in lower layer
      xChild = x * 2 + i;
          yChild = y * 2 + j;
      // position in child array
      nChildArrayPos = getArrayPos(4, i, j); // position in lower layer array
      int nChildLevelArrayPos  = getArrayPos(downWidth, xChild, yChild);

      if (xChild >= downWidth || yChild >= downHeight){
        node->pChilds[nChildArrayPos] = NULL;
      } else {
```

```
        node->pChilds[nChildArrayPos] =
                        pl PyramidLevel[nLevel-1].pNodes[nChildLevelArrayPos];
      }
       }// end i
      }// end j
      for (int father = 0; father < 4 ; father++){
         // definded starting conditions
         node->pFathers[father] = NULL;
      }
      // we tell the children nodes, that they have fathers
      for (int child = 0; child < 16; child++){
      for (int father = 0; father < 4 ; father++){
        if (node->pChilds[child] != NULL){
          if (node->pChilds[child]->pFathers[father] == NULL){
            node->pChilds[child]->pFathers[father] = node;
            // to set the father once is enough... we may leave
            break;
          }// end father == NULL
        }// end child != NULL
      }// end father
    }// end child
    node->lValue = (unsigned short) node->getChildrenMean();
    pLevel[nArrayPos] = node;
  }//end x
}// end y
pl_PyramidLevel[nLevel].pNodes = pLevel;
```

Code 4-9 Schreiben der übrigen Knoten in die Auflösungspyramide

4.2.2.4 Segmentierung des Bildes

Zu diesem Zeitpunkt ist die Auflösungspyramide erstellt worden, und die Beziehungen der Knoten untereinander gesetzt. Allerdings hat ein Knoten alle möglichen Väter gesetzt, die innerhalb seines Bereiches liegen, bzw. zu welchen er einen Einfluss auf den noch nicht gewichteten Mittelwert besitzt. Da wir jedoch den gewichteten Mittelwert rechnen wollen, geht es nun für jeden Knoten herauszufinden, welcher der vier möglichen Vaterknoten die kleinste Differenz in Bezug auf den Wert `pValue` besitzt. Dies geschieht indem die Methode `CNode::getClosestFather()` in der Klasse `CNode` aufgerufen wird. Diese Methode liefert einen Pointer zum ersten Vater, der die kleinste Differenz zum Kindknoten aufweist zurück. Man kann schon aufgrund dieser Formulierung erahnen, dass die stabile Auflösungspyramide nicht eindeutig sein kann. Ich werde im Abschnitt 4.2.2.5 auf Seite 30 auf dieses Problem und dessen Lösung eingehen.

Der zurückgelieferte Knoten wird als `pClosestFather` in der Klasse `CNode` gespeichert. In einem ersten Ansatz habe ich jeweils im `pFathers` Array alle Referenzen auf `NULL` gesetzt, und nur die Referenz zum Vater behalten die ich von der Methode `CNode():: getClosestFather()` zurück erhalten habe. Da die Zuweisungen innerhalb der Pyramide allerdings in der Fortsetzung des Pyramid-Linking Algorithmus ändern, machte dieser Ansatz jeweils eine Neuberechnung aller möglichen Väter nötig. Da dies ein nicht vertretbarer Aufwand darstellte, bin ich davon abgekommen, und fügte die Variable `CNode* pClosestFather` ein.

Zusätzlich zur Bestimmung des `pClosestFather` werden auch noch die Referenzen im `pChilds` Array gelöscht die diesen Knoten nicht mehr als den nächsten Vater gesetzt haben. Dies wird in der Berechnung des gewichteten Mittelwertes die Bestimmung des Gewichtes dieses Knotens erleichtern.

Nach der Bestimmung des Vaters mit der kleinsten Differenz, wird für jeden Knoten in der Pyramide definiert wieviele Kinder auf der untersten Ebene eine Beziehung zu diesem Knoten haben. Selbstverständlich kann sich diese Verbindung über mehrere Ebenen erstrecken. Der gefundene Wert dient uns dann später als Gewicht bei der Berechnung des gewichteten Mittel-

wertes. Die Summe der Kinder lässt sich recht einfach ermitteln. Dazu verwendete ich die Variable `nChildrenBaseLevel`. Diese Variable setzte ich im ersten Durchgang auf den Wert 1, damit alle Kinder auf der Basisebene dasselbe Gewicht haben.

```
// create Node and set parameters
node = new CNode();
node->lValue         = pTemp[nImageArrayPos];
node->nLevel         = 0;
node->x              = x;
node->y              = y;
node->nBaseChildren  = 1;
node->pClosestFather = NULL;
node->checkedFlag    = false;
```

Code 4-10 Parameter eines Nodes in der Basisebene

Nun wird für jeden Knoten geschaut ob er noch Referenzen in seinem `pChilds` Array hat die nicht auf `NULL` gesetzt sind. Dies bedeutet, dass diese Kinder den Knoten noch als nächsten Vater gesetzt haben. Falls es solche Kinder gibt, werden deren `nBaseChildren` Werte zusammengezählt, und wir erhalten das gesuchte Gewicht.

```
int number= 0;
for (int i = 0; i < 16; i++){
   if (pChilds[i] != NULL){
      if (pChilds[i]->pClosestFather == this){
         number += pChilds[i]->nBaseChildren;
      }
   }
}
```

Code 4-11 CNode::recalculateNumberOfChildren() - Berechnung der Gewichtung der Knoten

Die Methode `CNode::GetWeightedMean()` verwendet dieses Gewicht, um den gewichteten Mittelwert des Knotens zu bestimmen.

4.2.2.5 Definition einer Abbruchbedingung

Weil wir im ersten Durchgang durch die Pyramide, den arithmetischen Mittelwert für die Knoten gerechnet haben, und nun die gewichteten Mittelwerte verwenden ist zu erwarten, dass die Zuweisungen zum Vater `pClosestFather` angepasst werden müssen. Falls diese Zuweisungen sich nun ändern, werden sich logischerweise auch wieder die gewichteten Mittelwerte ändern, weil sich die Zuweisungen zu den Kindknoten verändert. Aus diesem Grund gilt es festzulegen, wie lange der Pyramid-Linking Algorithmus ausgeführt werden soll.

Ich versuchte es zuerst mit einer boolschen Variablen die bei jeder Veränderung eines Mittelwertes, oder bei einer Änderung einer Zuweisung zum nächsten Vater auf `true` gesetzt wird. Die Idee war diese beiden Arbeitschritte solange zu wiederholen, bis die Variable auf `false` bleiben würde. Ich hätte dann davon ausgehen können, dass die Pyramide in einen stabilen Zustand übergegangen wäre. Schnell konnte ich jedoch feststellen, dass dieser stabile Zustand niemals eintrat. Auch die Änderung der Abbruchbedingung, nur noch die Zuweisungen zu berücksichtigen, führte nicht zu Erfolg. Die `while`-Schlaufe wurde immer noch ununterbrochen ausgeführt. Ich stellte fest, dass es Verbindungen gab, die zwischen 2 oder mehreren Vätern hin und her gewechselt haben. Es gab also nur noch eine kleine Anzahl von Zuständen gab die von der Auflösungspyramide eingenommen wurde. Das brachte mich auf die Idee die Anzahl der Veränderungen pro Durchgang in der Pyramide zu zählen. Falls dieser Wert sich zum vorangegangenen Durchgang nicht mehr allzu stark verändert hat, konnte ich davon ausgehen ein "stabiles" Ergebnis erzielt zu haben.

```
CNode* oldFather;
CNode* newFather;

// variables abort condition
int changes0        = 0;                            // changes in level 0
int changesLastPass = nBaseHeight * nBaseWidth;   // changes0 last pass

// tolearance to abort is 0,1 promille of the pixels
int tolerance       = (nBaseHeight * nBaseWidth) * 0.0001;

// we interrupt the segmentation as soon the nodes
// in base level found the right connection to the
// father with the nearest mean value
int passes = 0;
while (abs (changesLastPass - changes0) > tolerance){
   changesLastPass = changes0;
   changes0 = 0;
   passes ++;
   …
   oldFather = node->pClosestFather;
   newFather = node->getClosestFather();
   if (oldFather != newFather){
      …
      changes0++;
   }
}
```

Code 4-12 Abbruchbedingung für die Segmentation

Nach dem ich diese Abbruchbedingung einmal so implementiert hatte, versuchte ich herauszufinden auf welchen Wert ich die Abbruchbedingung einzustellen hatte, so dass ich einerseits nicht unnötig viele Durchgänge und anderseits sicher sein konnte, dass die Abbruchbedingung auch wirklich erfüllt wird. Ich verwendete dazu einen Wert der sich aus der Anzahl der Pixel des Originalbildes rechnete. So wurde der Schwellwert automatisch der Grösse des Bildes angepasst. Der implementierte Schwellwert ist im Moment 0.0001 mal die Anzahl Pixel in der Grundfläche der Pyramide, was 0,1 Promille der Anzahl der Pixel entspricht. Dieser Wert wird bei einem 1024x512 grossen Bild in ungefähr 20 bis 25 Durchgängen erzielt. Um diesen Wert herauszufinden habe ich die Variable `passes` eingeführt.

Ich versuchte auch die den Algorithmus eine bestimmte Anzahl von Durchgängen durchlaufen zu lassen, um die Resultate der Segmentierung zu kontrollieren. Bei diesen Tests hat sich gezeigt dass es kaum mehr möglich ist Veränderungen, ab ungefähr dem 15-ten Durchgang festzustellen. Da es allerdings sein kann dass sich Helligkeiten leicht verändert haben, und dies am Bildschirm nicht festzustellen ist, bin ich sicher mit dem gefundenen Kompromiss ein gutes Resultat zu erhalten.

Ich möchte jetzt nochmals auf die Zuweisung der nächsten Väter eingehen. Diese Zuweisung wird wie schon erwähnt mit der Methode `CNode::getClosestFather()` ausgeführt. Allerdings reicht die alleinige Zuweisung des nächsten Vaters nicht, sondern es müssen noch weiter Arbeiten ausgeführt werden. Falls diese Zuweisung ändert müssen wir folgendes machen:

1. Der Pointer vom ehemaligen nächster Vater zum aktuellen Knoten muss gelöscht werden.
2. Die Anzahl der Kinder auf der untersten Ebene für den früheren nächsten Vater muss angepasst werden.
3. Der neue nächsten Vater muss ein Pointer zum aktuellen Knoten erhalten.
4. Die Anzahl der Kinder auf der untersten Ebene muss für den neuen nächsten Vater erhöht werden.
5. Der Pointer `pClosestFather` muss im aktuellen Knoten angepasst werden.
6. Die Variable für die Abbruchbedingung `changes0` wird inkrementiert

```
for (int nLevel = 0 ; nLevel < nMaxLevels ; nLevel++){
  int nHeight = pl PyramidLevel[nLevel].nHeight;
  int nWidth  = pl PyramidLevel[nLevel].nWidth;
  // for all nodes in the layer
  for (y = 0; y < nHeight; y ++){
    for (int x = 0 ; x < nWidth; x ++){
      nArrayPos = getArrayPos(nWidth, x, y);
      node = pl PyramidLevel[nLevel].pNodes[nArrayPos];

      oldFather = node->pClosestFather;
      newFather = node->getClosestFather();
      if (oldFather != newFather){
        // the father with the nearest mean has changed.
        // we have to do for :
        //      oldFather:
        //         - delete pointer to node
        //         - adapt number of children in base level
        //      newFathter:
        //         - create pointer to children
        //         - increase the number of children in base level
        //      node:
        //         - change pointer to pClosestFather
        //      general:
        //         - set changes0

        // delete pointer to node
        if (oldFather != NULL){
          for (int i = 0; i < 16; i ++){
            if (oldFather->pChilds[i] == node)
              oldFather->pChilds[i] = NULL;
      }
      // adapt number of children in base level
      oldFather->nBaseChildren -= (node->nBaseChildren);
        }
        // create pointer to node
        for (int i = 0; i < 16 ; i ++){
          if (newFather->pChilds[i] == NULL){
            newFather->pChilds[i] = node;
            break;
          }
        }
        // increase the number of children in base level
        newFather->nBaseChildren += (node->nBaseChildren);
        // change pointer to closest father
        node->pClosestFather = newFather;
        // set changes0
        changes0++;
      }
      pl PyramidLevel[nLevel].pNodes[nArrayPos] = node;
    }// end x
  }// end y
}// end nLevels
```

Code 4-13 Anpassen der Beziehung zum Vaterknoten mit der kleinsten Differenz

4.2.2.6 Einfärben der Segmente

Nach dem wir einen stabilen Zustand erreicht haben, geht es nun darum die Segmente des Bildes so zusammenzufassen, dass sie den einzelnen Gewebeschichten entsprechen. Es geht also in einem nächsten Schritt darum die Anzahl der Segmente die wir in dem Bild erkennen wollten zu definieren. In einem optimalen Bild ist es von blossen Auge möglich etwa 10 ver-

schiedene Schichten zu erkennen. Falls wir also Segmente einfärben wollten, müssten wir eine Ebene in der Pyramide haben die über 10 verschiedene Knoten verfügt. Wir könnten dann deren Werte in alle Knoten in der untersten Ebene schreiben die mit diesen Knoten verbunden sind, und wir hätten die 10 Segmente im Bild in den entsprechenden Farben eingefärbt. Da wir aber vor dem ausführen des Prozesses keinerlei Informationen über die Grösse des Originalbildes haben, bzw. der Algorithmus auch für beliebig geshrinkte Bilder funktionieren sollte, können wir schwer sagen auf welcher Ebene wir diese Bedingung erfüllen würden. Ich habe mich deshalb dazu entschlossen, die Anzahl der Knoten die auf den Ebenen liegen zu untersuchen. Dafür kontrolliere ich die oberste Ebene, danach die zweitoberste usw. Solange, bis ich eine Ebene mir mehr als 30 Knoten gefunden habe.

Die meisten Bilder besitzen nach des Anwendung des Wavelet Filters drei Bildbereiche. Oben und unten im Bild ist eine Schwarze Fläche entstanden die in der Mitte von einem farbigen Bereich unterbrochen wird. Ich kann mir auf diese Weise sicher sein für den farbigen Bereich mindestens 10 Segmente gefunden zu haben.

Ich nehme also in der Pyramide das oberste Level, dass mehr als 30 Knoten hat, und färbe alle Kinder in der Grundfläche, die via dem `pClosestFather` Pointer eine Verbindung haben in der Farbe dieses Knotens ein. Dadurch wird die Bedingung für eine vollständige und überdeckungsfreie Segmentierung erfüllt, da jedes Pixel nur einen nächsten Vater haben kann.

```
// write the values of the Nodes in the basel level
// we take the firstLevel with more than 30 nodes
for (nLevel = nMaxLevels; nLevel >= 0 ; nLevel--){
  nWidth = pl PyramidLevel[nLevel].nWidth;
  nHeight= pl PyramidLevel[nLevel].nHeight;
  if (nHeight * nWidth > 30){
    for (y = 0; y < nHeight; y ++){
      for (int x = 0; x < nWidth ; x++){
        nArrayPos = getArrayPos(nWidth, x, y);
        node = pl PyramidLevel[nLevel].pNodes[nArrayPos];
        for (int i = 0; i < 16; i++){
          if (node->pChilds[i] != NULL &&
              node->pChilds[i]->pClosestFather == node)
            node->pChilds[i]->lValue = node->lValue;
        }// end i
      }// end x
    }// end y
  }// end nHeight * nWidth
}//end nLevel
```

Code 4-14 Schreiben der Werte der Segmente in das Basislevel

Nun habe ich auf der Grundfläche eine Anzahl verschiedener Segmente. Diese Segmente können aber zur Erkennung der Schichten noch zusammengefasst werden, wenn diese einen ähnlichen Helligkeitswert aufweisen. Dazu verwende ich simple Schwellwerte die mir die Pixel im Bild auf die definierten Farben einfärben können. Somit kann ich alle Segmente, als eine Fläche die eine gemeinsame Farbe aufweisen, betrachten.

```
unsigned short value;
unsigned short scalar;
for (y = 0; y < nBaseHeight; y ++){
  for (int x = 0; x < nBaseWidth; x++){
    nArrayPos = getArrayPos(nBaseWidth, x, y);
    if ( &pl PyramidLevel[0].pNodes[nArrayPos] != NULL){
      value  = (unsigned short) pl PyramidLevel[0].pNodes[nArrayPos]->lValue;
      if (value <= DARKBLUE)                                scalar = 0;
      if (value >  DARKBLUE   &&   value <= LIGHTGREEN)     scalar = LIGHTBLUE;
      if (value >  LIGHTGREEN &&   value <= YELLOW)        scalar = GREEN;
      if (value >  YELLOW     &&   value <= LIGHTRED)       scalar = YELLOW;
      if (value >  LIGHTRED   &&   value <= WHITE)         scalar = RED;
      if (value >  WHITE      )                        scalar = WHITE;
      pl PyramidLevel[0].pNodes[nArrayPos]->lValue = scalar;
    } else {
      scalar = 0;
    }// end if Node == NULL
  }// end x
}// end y
```

Code 4-15 Erstes Einfärben der Segmente

Die bisher durchgeführte Segmentation definiert alle Bereiche die auch im Originalbild dieselbe Helligkeit aufweisen als Segment. Wie ich im Abschnitt 4.1.2 beschrieben habe, ist dies für unseren Zweck noch nicht ganz ausreichend.

Wir möchten gerne die Schichten wie sie im menschlichen Auge vorkommen, so gut wie möglich erkennen, und entsprechend einfärben.

4.2.2.7 Zusätzliche Segmentierung der Schichten im Auge

Ich verwende eine boolesche Variable die im Menü der Prozessparameter gesetzt werden kann. Mit dieser Variable wird entschieden ob der Prozess hier beendet wird, oder ob die Segmentation für die 3D-Ansicht weitergeführt werden soll. (▶Beginn Abschnitt 4.1.2.2) Diese Variable wird in der Methode `CPyramidLinkingProc::SetParamValue()` gesetzt. Falls die Variable auf true gesetzt ist, wird mit der zusätzlichen Segmentierung weitergearbeitet.

```
// the boolean to segment the picture for a 3D view ist set.
// we have to color the different layers in the picture to
// different colors, to represent the different layers in
// the human eye.
if (m l 3DSegment){
  // segment the picture for a 3DView
:
:
}
```

Code 4-16 Variable für die weitere Segmentierung

Unsere Hauptaufgabe ist es nun die Pigmentschicht, das heisst die hellste Schicht im Auge zu finden, diese Schicht ist normalerweise in der vorgängigen Segmentation auf den Wert 2200 gesetzt worden. Wir können uns aber auf keinen Fall darauf verlassen, dass diese Schicht durchgängig vorhanden ist. Es kann sein, dass die Schicht an den Rändern oder in der Mitte des Bildes Lücken aufweist. In extremen Fällen kann es auch sein, dass das Segment nicht auf den Wert 2200 gesetzt wurde, dies kann vor allem dann auftreten, wenn das Bild recht dunkel ist. Bei einem hellen Bild kann es auch vorkommen, dass Schichten oberhalb der eigentlichen Pigmentschicht auf den hellsten Wert segmentiert wurden. Ausserdem ist es nicht möglich Aussagen über den Verlauf der Pigmentschicht zu machen. Wir können trotz zuvor angewandter Filter zum Ausrichten des Bildes nicht sicher sagen ob die Pigmentschicht waagrecht im Bild liegt. Es kann zu extremen Abweichungen von der Waagrechten kommen, zum Beispiel, wenn der Patient während der Aufnahme das Auge bewegt hat.

Analyse des Bildes

Um mit einer gewissen Sicherheit Aussagen über die Lage, und den Verlauf der Pigmentschicht machen zu können, müssen wir in einem ersten Schritt das bisher erhaltene Bild analysieren. Dazu verwendete ich verschiedene Hilfslinien die Auskunft über den Verlauf der Pigmentschicht geben. Ich suche die waagrechte Linie, die am meisten Pixel mit dem hellsten Wert im Bild hat. Falls die Pigmentschicht waagrecht im Bild liegen sollte, gibt mir dieser Wert schon recht genau die Höhe an auf der die Pigmentschicht liegt.

Krümmungen im Verlauf der Pigmentschicht beeinflussen aber die Lage der Linie. Dies kommt vorallem dann vor wenn die Pigmentschicht leicht schräg im Bild liegt und an einem Punkt eine Krümmung aufweist. Dann kann man die Information dieser Linie nicht verwenden, weil diese genau waagrecht auf der Höhe der Krümmung zu liegen kommt.

Ich musste aus diesem Grund andere Entscheidungslinien einführen. Ich habe mir überlegt eine Linie einzuführen, die den Verlauf der Pigmentschicht genauer wiedergeben kann. Dazu messe ich auf beiden Seiten des Bildes den farbigen Bereiches. Um die Entscheidungslinie zu erhalten verbinde ich die beiden Mittelpunkte der Segmente. Dieser Ansatz lieferte mir, falls der farbige Bereich schräg abgebildet war, ebenfalls eine schräge Linie. Allerdings hatte diese Linie beim einfärben der Segmente wieder einen zu grossen Abstand von der Pigmentschicht. Ich musste aus diesem Grund auch diesen Ansatz fallen lassen. Als dritten Ansatz der mich schliesslich auch zum Ergebnis führte, nahm ich die Mitte des farbigen Bereichs für jede x-Koordinate und kreierte so eine Linie. Diese Linie sollte wegen der Annahme, das die Pigmentschicht ungefähr in der Mitte des farbigen Bereiches liegt ungefähr den Verlauf wiedergeben. Ausserdem musste ich noch den Verlauf der farbigen Schichten berücksichtigen.

```
for (x = 0; x < nBaseWidth ; x ++){
  // Look for the Layers of the Segments
  pRedLayerUp[x]      = getNorth(x, RED,   pl PyramidLevel);
  pRedLayerDown[x]    = getSouth(x, RED,   pl PyramidLevel);
  pYellowLayerUp[x]   = getNorth(x, YELLOW,   pl PyramidLevel);
  pYellowLayerDown[x] = getSouth(x, YELLOW,   pl PyramidLevel);
  pGreenLayerUp[x]    = getNorth(x, GREEN,   pl PyramidLevel);
  pGreenLayerDown[x]  = getSouth(x, GREEN,   pl PyramidLevel);
  pBlueLayerUp[x]     = getNorth(x, LIGHTBLUE,pl PyramidLevel);
  pBlueLayerDown[x]   = getSouth(x, LIGHTBLUE,pl PyramidLevel);
}

int CPyramidLinkingProc::getNorth(int x, unsigned short layerColor,
                                  CLevel* level){
  int nArrayPos  = getArrayPos(level->nWidth, x, 0);
  for (int i = 0; i < level->nHeight ; i++){
    if (level->pNodes[nArrayPos + (i * level->nWidth)]->lValue == layerColor){
       return i;
   }
  }
  return 0;
}

int CPyramidLinkingProc::getSouth(int x, unsigned short layerColor,
                                  CLevel* level){
  int nArrayPos  = getArrayPos(level->nWidth, x, level->nHeight-1);
  for (int i = level->nHeight-1; i >= 0 ; i--){
    if (level->pNodes[nArrayPos - (i * level->nWidth)]->lValue == layerColor){
      return i;
    }
  }
  return 0;
}
```

Code 4-17 Methoden getSouth() und getNorth() und ihre Anwendung

Die Erfahrung hat allerdings gezeigt, dass die Oberen Begrenzungen der Schicht wenig aussagekräftig sind, da Störungen hauptsächlich oberhalb der Pigmentschicht auftreten.

In den nun gefunden Arrays können noch Nullwerte auftreten, dass ist dann der Fall, wenn eine Farbe an einer x- Koordinate nicht gefunden wurde. Diese Nullwerte müssen entfernt werden, weil diese das Ergebnis zu stark stören würden. Dazu gibt es die Methode `CPyramidLinkingProc::RemoveAllGapsFromArray()` die einen Array und dessen Länge als Parameter übernimmt.

```
int CPyramidLinkingProc::RemoveAllGapsFromArray(int* array, int arrayLength){
  int x1 = 0;  int y1 = 0;  int x2 = 0;  int y2 = 0;
  bool foundGap = false;

  for (int x = 0; x < arrayLength; x++){
    if (array[x] == 0 && !foundGap){
      foundGap = true;
      x1 = x - 1;
      y1 = array[x1];
    }
    if (array[x] != 0 && foundGap){
      x2 = x;
      y2 = array[x];
      // Für alle Pixel am Anfang
      if (x1 < 0) {
        RemoveZerosFromArray(array, x2, y2);
      } else {
        RemoveGapFromArray(array, x1, y1, x2, y2);
      }
        foundGap = false;
    }
    // für alle Pixel am Schluss
    if (x == arrayLength-1 && foundGap){
      for (x = x1; x  < arrayLength; x++){
        array[x] = y1;
      }
    }
  }
  return 1;
}
```

Code 4-18 Vermeiden von Nullwerten in den Arrays

Diese Methode sucht nach einer Lücke mit Nullwerten im Array und gibt die Koordinaten der gefundenen Lücke an die Methode `CPyramidLinkingProc::RemoveGapFromArray()` weiter. Diese Methode macht nun die Verbindung zwischen den beiden Punkten am Rande der Lücke.

```
int CPyramidLinkingProc::RemoveGapFromArray(int* array, int x1, int y1,
                                            int x2, int y2){
  int changes = 0;
  int deltaY  = y2 - y1;
  int deltaX  = x2 - x1;
  float alpha = deltaY / deltaX;

  for (int x = x1;  x < x2 ; x++){
    array[x] = abs((x - x1) * alpha + y1);
  }
  return 1;
}
```

Code 4-19 Auffüllen von Lücken in Arrays

Falls die Lücken am Rande des Bildes auftreten, wird die Linie vom letzten gerechneten Punkt bis zum Bildrand waagrecht weitergeführt. Dazu dient mir die Methode `CPyramidLinkingProc ::RemoveZerosFromArray()` die ich bereits von früheren Versuchen implementiert hatte.

```
int CPyramidLinkingProc::RemoveZerosFromArray(int* array, int nArrayLength,
                                              int defaultValue){
  int changes = 0;
  for (int x= 0; x < nArrayLength ; x ++){
    if (array[x] == 0){
      if (x == 0){
        array[x] = defaultValue;
        changes ++;
      } else {
        array[x] = array[x-1];
        changes ++;
      }
    }
  }
  return changes;
}
```

Code 4-20 Vermeiden von Nullwerten am Bildrand

Nachdem wir die verschiedenen Linien gerechnet haben, können wir sie entsprechend ihrer Verlässlichkeit zu einer Linie zusammenbringen die uns die erste Entscheidungslinie ergibt. Dazu war einiges an Versuchen nötig um Ergebnisse zu erzielen, die uns eine in den meisten Fällen zuverlässige Entscheidungslinie liefert. Wir unterscheiden allerdings noch zwei Fälle von Entscheidungslinien. Der erste Fall tritt bei einem "normalen" Bild ein, die andere Gewichtung ist für den Fall gedacht, dass sich keine Roten Pixel (Wert 1800) im Bild befinden, dann ist die Gewichtung angepasst, und es wird die Linie benutzt die den ungefähre Ausrichtung des farbigen Bereiches im Bild widerspiegelt.

```
if (hasRed){
  // we have red pixels in the picture
  pDecisionLine[x] = ((2  * pBlueLayerDown[x])     +
            (7  * pYellowLayerDown[x])             +
            (3  * pMiddleLayer[x])                 +
            (10 * pRedLayerDown[x]))               /22;
  } else {
  // we dont't have read pixels in the picture
  pDecisionLine[x] = (   (2  * pBlueLayerDown[x])  +
            (7  * pYellowLayerDown[x])             +
            (3  * pMiddleLayer[x])                 +
            (2  * pLayerLine[x]))                  /14;
}
```

Code 4-21 Gewichtungen für die Entscheidungslinie

Die Tabelle 4-6 zeigt die verwendeten Linien, deren Bedeutung und die Gewichtung.

Array Name	Bedeutung	Gewicht	Wurde benutzt
pBlueLayerDown	Unterste Kante des blauen Segments.	2	In beiden Fällen
pYellowLayerDown	Unterste Kante des gelben Segments.	7	In beiden Fällen
pMiddleLayer	Mitte des farbigen Bereichs im Bild.	3	In beiden Fällen
pRedLayerDown	Unterste Kante des roten Segments	10	Falls Rote Pixel vorhanden sind.
pLayerLine	Verbindung der Mitte des farbigen Bereichs jeweils gemessen am linken und rechten Rand des Bildes.	2	Falls keine Roten Pixel vorhanden sind.

Tabelle 4-6 Definierte Linien und deren Gewichtung

Da es in speziellen Fällen vorkommen kann, dass die Linie Sprünge aufweist, habe ich noch eine Funktion geschrieben, die die Line am Schluss etwas glättet. Dies geschieht in der Methode `CPyramidLinkingProc::SmoothLine()` die als Parameter auch die Menge der Durchgänge also auch die Stärke der Glättung übernimmt.

```
int CPyramidLinkingProc::SmoothLine(int* array,
                    int nArrayLength,
                    int smoothValue){
  for (int smooth  = 0; smooth < smoothValue; smooth++){
    for (int x = 1; x < nArrayLength; x ++){
      if (abs(array[x] - array[x-1]) > 2){
        int newValue = (array[x] + array[x-1]) / 2;
        array[x] = newValue;
      }
    }
  }
  return 1;
}
```

Code 4-22 CPyramidLinkingProc ::SmoothLine() Methode

Nach dem Finden dieser Linie werden die beiden hellsten Bereiche des Bildes die unterhalb dieser Linie komplett eingefärbt, dass heisst es wird das ganze Segment eingefärbt, auch wenn Teile davon oberhalb der Entscheidungslinie liegen. Dies geschieht mit einem Verfahren, dass als **Flood Filling**[1] bekannt ist. Auf die Spezialitäten dieser Implementation komme ich im Abschnitt 4.2.2.7 auf Seite 34 noch einmal zu sprechen. Der Grund für dieses Vorgehen, liegt in unserem Vorhaben keine neuen Partitionen einzufügen, dass heisst wir belassen die gefundenen Segmente in Ihrer Grösse, und unterteilen sie nicht noch zusätzlich weiter.

Wir färben die bisher hellsten Segmente (RED) in weiss ein und die Segmente die bisher die zweithellste Farbe (YELLOW) gemäss der Definition im Pyramid-Linking Algorithmus hatten, erhalten die Farbe orange.

```
// we set the Color of the Segments in the South of the decision Line
// in another color to distinguish them.

// The red pixels in the North become WHITE
if (pNode->lValue == RED && pNode->y >  pDecisionLine[x]){
  pNode->lValue = WHITE;
  SetNeighboursColor(pNode, RED, pl PyramidLevel, pDecisionLine, 0);
}
// The yellow pixels become orange
if (pNode->lValue == YELLOW && pNode->y > pDecisionLine[x]){
  pNode->lValue = ORANGE;
  SetNeighboursColor(pNode, YELLOW, pl PyramidLevel, pDecisionLine,0);
}
```

Code 4-23 erstes Einfärben der Pigmentschicht

Nun können wir davon Ausgehen, die Pigmentschicht gefunden und einigermassen richtig eingefärbt zu haben. Wir können nun mit den eingefärbten Segmenten die Entscheidungslinie neu bestimmen, um die restlichen Segmente des Bildes einzufärben. Die Methode `CPyramidLinkingProc::RecalculatePigmentLayer()` übernimmt drei Farben. Jede x-Koordinate wird nun abgesucht. Kommt die erste Farbe in dieser x Position vor, wird die Mitte dieses Farbbereichs in den Array geschrieben, kommt die erste Farbe nicht vor, wird die Mitte des Bereichs der zweiten Farbe in den Array geschrieben, wird auch die zweite Farbe nicht gefunden, wird die Mitte des Bereichs mit der dritten Farbe in den Array geschrieben. Wird keine der drei Farben gefunden, wird 0 in den Array geschrieben. Diese Nullwerte können anschlies-

[1] Flood Filling bezeichnet das gesamte Einfärben eines Bildbereichs.

send mit dem Verfahren, dass auf der Seite 36 beschrieben wurde auch wieder weggebracht werden. Die nun enthaltene Entscheidungslinie führt in den meisten Fällen zu einem recht guten Ergebnis. Die Fig. 4-8 zeigt ein Beispiel eines Bildes bei dem die Entscheidungslinie als weisse Linie eingezeichnet wurde.

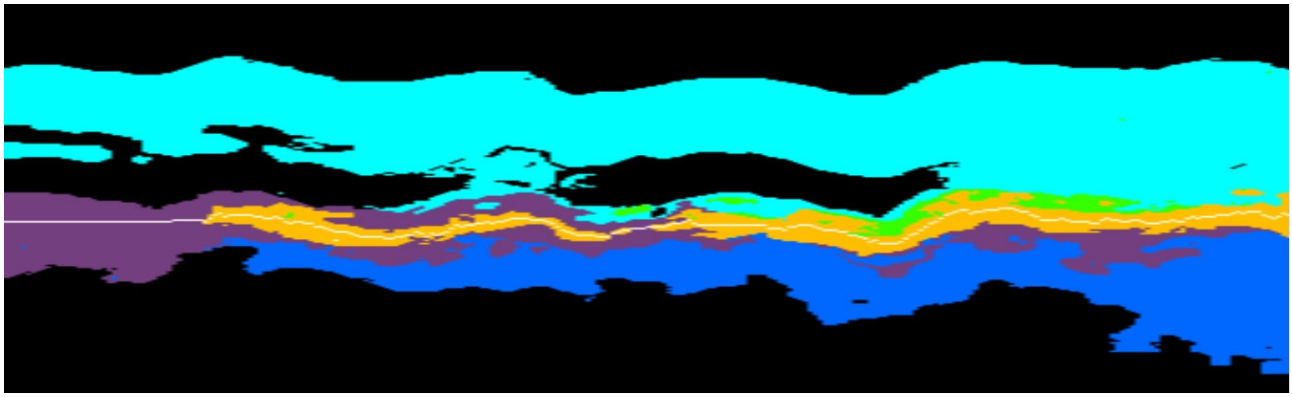


Fig. 4-8 Entscheidungslinie in einem OCT Bild

Nun können noch die restlichen Segmente eingefärbt werden.

```
for (y = 0 ; y < nBaseHeight; y ++){
  for (int x = 0; x < nBaseWidth; x ++){
    int nArrayPos = getArrayPos(nBaseWidth, x, y);
    CNode* pNode = pl PyramidLevel[0].pNodes[nArrayPos];
    // the red pixels become white
    if (pNode->lValue == RED && pNode->y > pDecisionLine[x]){
      pNode->lValue = WHITE;
      SetNeighboursColor(pNode, RED, pl PyramidLevel, pDecisionLine, 0);
    }
    if (pNode->lValue == YELLOW && pNode->y > pDecisionLine[x]){
      pNode->lValue = ORANGE;
      SetNeighboursColor(pNode, YELLOW, pl PyramidLevel, pDecisionLine,0);
    }
    // the green pixels become purple
    if (pNode->lValue == GREEN && pNode->y > pDecisionLine[x]){
      pNode->lValue = PURPLE;
      SetNeighboursColor(pNode, GREEN, pl PyramidLevel, pDecisionLine,0);
    }
    // the lightblue pixels become darkblue
    if (pNode->lValue == LIGHTBLUE && pNode->y > pDecisionLine[x]){
      pNode->lValue = DARKBLUE;
      SetNeighboursColor(pNode, LIGHTBLUE, pl PyramidLevel, pDecisionLine,0);
    }
  }//end x
}// end y
```

Code 4-24 Einfärben der übrigen Segmente

Dies geschieht ebenso wie beim ersten einfärben mit dem **Flood Filling** Verfahren. Das Verfahren ist in der Methode `CPyramidLinkingProc::SetNeighboursColor()` implementiert. Es handelte sich in einem ersten Ansatz um einen rekursiven Aufruf der Methode bei dem jedes Pixel in einer 8-er Nachbarschaft auf seine Farbe getestet wird, und bei Bedarf umgefärbt wird. Um sicherzustellen, dass wir so keine Endlosaufrufe generieren, habe ich

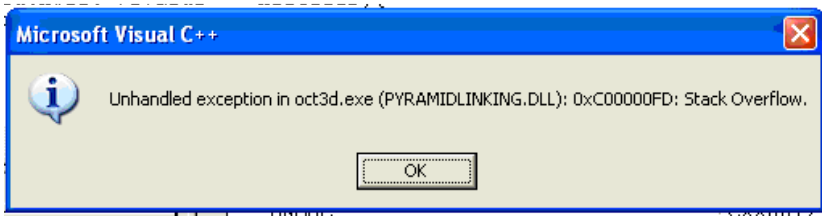


Fig. 4-9 Fehlermeldung bei rekusiven Methodenaufruf

ein Flag bei jedem Node vorgesehen, dass es uns ermöglichte, schon bearbeitet Knoten zu markieren. Trotz dieser Massnahme, erhielten wir Fehler für zu viele Funktionsaufrufe, was als Stack Overflow Fehler gemeldet wurde.

Ich versuchte daraufhin, festzustellen, ob es möglich wäre, die Anzahl der Richtung die wir untersuchten zu reduzieren. Normalerweise wird beim Flood Filling Verfahren, mit einer 8er Nachbarschaft[1] gearbeitet. Weil wir zeilenweise durch das Bild gingen beschränkte ich mich auf die vier Richtungen wie in Fig. 4-10 gezeigt, in denen die weiteren Aufrufe rekursiv fortgesetzt werden sollten.

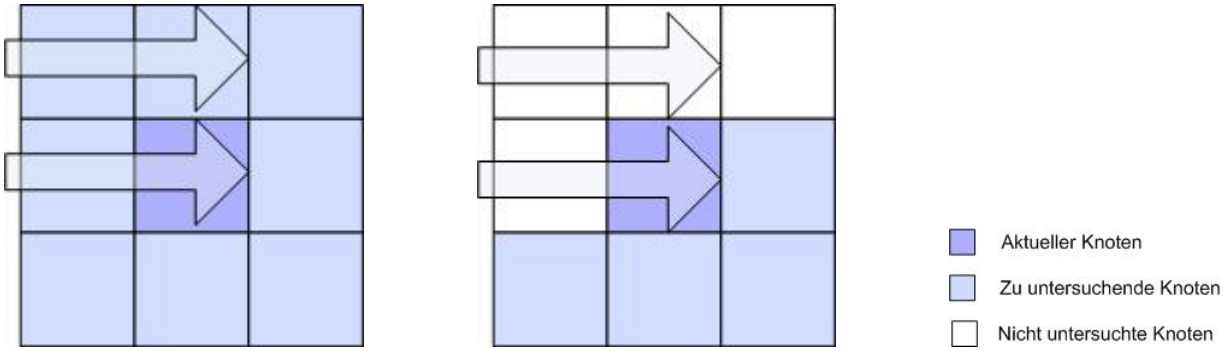


Fig. 4-10 Zu untersuchende Knoten im OCT Bild

Dies führte allerdings auch nicht zu gewünschten Ergebnis. Zum ersten erschien der Stack Overflow Fehler bei grossen Bildern immer noch, zum anderen gab es Formen von Segmenten die nicht sauber eingefärbt wurden.

Nach einer Rücksprache mit Herrn Cattin erhielten wir eine Lösung dieses Problems. Die Implementation erfolgt nun über eine Queue. Das heisst die Knoten in der Nachbarschaft werden in eine Queue gelegt und dann der Reihe nach abgearbeitet. Da die Queue nicht auf dem Stack sondern im Arbeitsspeicher liegt, tritt das Problem des Stack Overflows nicht mehr auf.

```
int CPyramidLinkingProc::SetNeighboursColor(   CNode* node,
                       unsigned short hisColor,
                       CLevel* level,
                       int* decisionLine,
                       int tolerance){
  // weCheck all neighbours
  int x = node->x;
  int y = node->y;
  unsigned short color = node->lValue;
  int nBaseWidth  = level->nWidth;
  int nBaseHeight = level->nHeight;
  // we detect the border of the level and leave
  if (x < 1 || y < 1 || x > nBaseWidth- 2 || y > nBaseHeight-2)
    return 0;
  int nArrayPos   = getArrayPos(nBaseWidth, x,   y   );
  // neighbour indices
  int nNorth      = getArrayPos(nBaseWidth, x,    y-1   );
  int nNorthEast  = getArrayPos(nBaseWidth, x+1,  y-1   );
  int nEast       = getArrayPos(nBaseWidth, x+1,  y   );
  int nSouthEast  = getArrayPos(nBaseWidth, x+1,  y+1   );
  int nSouth      = getArrayPos(nBaseWidth, x,    y+1   );
  int nSouthWest  = getArrayPos(nBaseWidth, x-1,  y+1   );
  int nWest       = getArrayPos(nBaseWidth, x-1,  y   );
  int nNorthWest  = getArrayPos(nBaseWidth, x-1,  y-1   );
  // neighbour nodes
  CNode* pNorth      = level->pNodes[nNorth];
```

[1] 8er Nachbarschaft: bezeichnet die 8 Pixel in der Nachbarschaft eines Pixels

```
  CNode* pNorthEast  = level->pNodes[nNorthEast];
  CNode* pEast       = level->pNodes[nEast];
  CNode* pSouthEast  = level->pNodes[nSouthEast];
  CNode* pSouth      = level->pNodes[nSouth];
  CNode* pSouthWest  = level->pNodes[nSouthWest];
  CNode* pWest       = level->pNodes[nWest];
  CNode* pNorthWest  = level->pNodes[nNorthWest];
  // we check all black pixels as visited
  // they are never interesting for us
  if (pNorth->lValue == 0){
    pNorth->checkedFlag = true;
  }
  :
  // all directions
  :
  if (pNorthWest->lValue == 0){
    pNorthWest->checkedFlag = true;
  }
```

Code 4-25 Bestimmen der Nachbar-Knoten

```
  // We could not use recursion.
  // Therefore we have to use a Queue
  // Queue Code snipplet provided by R.Cattin
  Queue queue;
  queue.push back(node);
  while (!queue.empty()){
    CNode* pNode = queue.front(); queue.pop front();
    if (!pNode->checkedFlag){
      pNode->checkedFlag = true;

      pNode->lValue = color;
      int nodeX = pNode->x;
      int nodeY = pNode->y;

      // neighbour indices
      nNorth      = getArrayPos(nBaseWidth, nodeX,   nodeY-1   );
      nNorthEast   = getArrayPos(nBaseWidth, nodeX+1,   nodeY-1   );
      :
      // all directions
      :
      pNorthWest   = level->pNodes[nNorthWest];
      // Nodes with the searched Color and
      // not checked yet has to be pushed in the Queue
      if((!pNorth->checkedFlag)
        && pNorth->lValue == hisColor
        && pNorth->y > decisionLine[pNorth->x])
        queue.push back(pNorth);
      :
      //all directions
      :
    }
  }
  return 1;
}
```

Code 4-26 Implementation der Queue

Als letzter Arbeitsschritt wird noch das Bild in den Ausgabe Array geschrieben, und der Speicher soweit als möglich wieder freigegeben. Es war mir nicht möglich den Speicher sauber freizugeben, da wir verschiedene Anzahlen von Pointern haben die zu den Knoten zeigen. Man müsste eine Art Garbage Collector implementieren, der die Anzahl Referenzen die auf einen

Knoten zeigt kennt, und diesen dann, wenn dieser Zähler auf Null ist löscht. Ich habe aber darauf verzichtet und lösche implizit alle Objekte die ich zuvor auch erstellt habe. Es ist deshalb nicht mehr möglich einen Pointer zu benutzen der auf ein gelöschtes Objekt zeigt.

5 Ziel 3 - 3D-Volumendarstellung für segmentierte Bilder

Autor	Markus Fawer
Implementation	Markus Fawer
Ziel	Implementierung einer 3D-Volumendarstellung für die segmentierten Bilder.
Status	Ziel erreicht
Bemerkung	Die 3DSegment Ansicht wurde implementiert, um eine geeignete Ansicht zu erstellen, die das Betrachten der Schichten die man zuvor im Pyramid-Linking erhalten hat anzuzeigen. Es ist möglich, die segmentierten Schichten aus dem Pyramid-Linking so darzustellen, dass man diese mit einem Mausklick wegklicken kann, um die anderen Schichten unterhalb sehen zu können. Es können insgesamt 8 verschiedene Konturen wahlweise angezeigt werden.

Tabelle 5-1 Übersicht Ziel 3 - 3D-Volumendarstellung für segmentierte Bilder

5.1 Idee

Die Idee war nun einen ähnlichen Ansatz wie des 3D Surface View zu verfolgen, es sollte aber möglich sein zusätzliche Konturen anzuzeigen, und mit der Maus mit den Konturen interagieren zu können. Ausserdem sollte es möglich sein die Konturen besser als Flächen darstellen zu können. Das Aussehen der Flächen sollte geschmeidiger sein. Ausserdem sollten die Konturen als Objekte und nicht als einzelne Flächen dargestellt werden.

Ich entschloss mich die Ansicht ebenfalls innerhalb der OCT3D Anwendung zu implementieren. So konnte ich die bereits implementierten Ansichten als Ausgangslage verwenden.

5.2 Realisierung

5.2.1 Vorbereitung

Ich nahm aus diesem Grund die Klassen des 3DSegmentView aus Ausgangslage und ergänzte diese mit der Möglichkeit acht verschiedene Konturen anzuzeigen.

- `CDSegmentPanel`
- `C3DSegmentView`
- `C3DSegmentViewProc`

Dazu musste ich neben der Implementation der 3 Klassen für die Ansicht auch noch neue Ressourcen eintragen und die Ansicht der Parameterauswahl erstellen.

Ich übernahm die Farben die ich im Pyramid-Linking Prozess definiert hatte (▶Fig. 5-1). Dadurch wird es möglich die erhaltenen Bilder aus der Segmentation direkt mit den Objekten die in der 3-Dimensionalen Ansicht erstellt werden zu vergleichen.

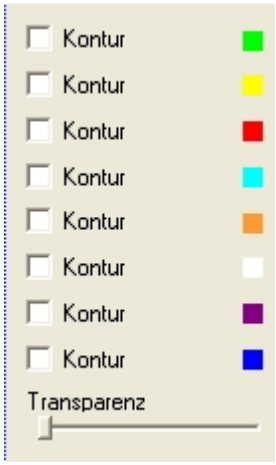


Fig. 5-1 Farbreihenfolge C3DSegmentView

```
//   //Surface 1 ?
if (m p3DSegmentViewProc->GetPolyData1() != NULL){
  :
  m pActorVolume1->GetProperty()->SetColor(0.0, 1.0, 0.0); // GREEN
  :
}
//Surface 2 ?
if (m p3DSegmentViewProc->GetPolyData2() != NULL){
  :
  m pActorVolume2->GetProperty()->SetColor(1.0, 1.0, 0.0); // YELLOW
  :
}
//Surface 3 ?
if (m p3DSegmentViewProc->GetPolyData3() != NULL){
  :
  m pActorVolume3->GetProperty()->SetColor(1.0, 0.0, 0.0); // RED
  :
//Surface 4 ?
if (m p3DSegmentViewProc->GetPolyData4() != NULL){
  :
  m pActorVolume4->GetProperty()->SetColor(0.0, 1.0, 1.0); //LIGHTBLUE
  :
}
//Surface 5 ?
if (m p3DSegmentViewProc->GetPolyData5() != NULL){
  :
  m pActorVolume5->GetProperty()->SetColor(1.0, 0.6, 0.2); //ORANGE
  :
}
//Surface 6 ?
if (m p3DSegmentViewProc->GetPolyData6() != NULL){
  :
  m pActorVolume6->GetProperty()->SetColor(1.0, 1.0, 1.0); //WHITE
  :
}
//Surface 7 ?
if (m p3DSegmentViewProc->GetPolyData7() != NULL){
  :
  m pActorVolume7->GetProperty()->SetColor(0.5, 0.0, 0.5); //PURPLE
  :
}
//Surface 8 ?
if (m p3DSegmentViewProc->GetPolyData8() != NULL){
  :
  m pActorVolume8->GetProperty()->SetColor(0.0, 0.0, 1.0); //DARKBLUE
  :
}
```

Code 5-1 Farben der Konturen in der 3D Ansicht

Ebenfalls nahm ich als Default-Werte für die Konturen, die Werte die im Pyramid-Linking definiert werden. Ich habe aber die Möglichkeit gelassen diese Defaultwerte anzupassen, falls die Ansicht später auch für andere Prozesse verwendet werden sollte. Die Reihenfolge der Werte habe ich so gesetzt, wie die Farben in den segmentierten Pyramid-Linking Bildern vorkommen sollten. Das heisst falls es Rote Pixel geben sollte, sind diese sicher oberhalb der Pigmentschicht zu finden. Die Pigmentschicht sollte mit den Farben weiss und orange (ev. violett) zu erkennen sein.

```
/**
 * Resets a values to its default
 *
 * @param nParamid of parameter to reset
 *
 * @return int        O3D E INVALIDINDEX      if index out of bounds<br>
 *                          O3D SUCCESS
 */
int C3DSegmentViewProc::ResetParamValue(int nParam)
{
  // Colors defined in Pyramid-Linking
  int WHITE        = 2200;
  int RED          = 1800;
  int LIGHTRED     = 1700;
  int ORANGE       = 1600;
  int YELLOW       = 1550;
  int GREEN        = 1400;
  int LIGHTGREEN   = 1300;
  int LIGHTBLUE    = 1200;
  int DARKBLUE     = 1100;
  int PURPLE       = 1050;
  int BLACK        = 0;

  switch (nParam)
  {
    case 0:
      m nKontur1 = GREEN ;
      break;
    case 1:
      m nKontur2 = YELLOW ;
      break;
    case 2:
      m nKontur3 = RED ;
      break;
    case 3:
      m nKontur4 = LIGHTBLUE;
      break;
    case 4:
      m nKontur5 = ORANGE;
      break;
    case 5:
      m nKontur6 = WHITE;
      break;
    case 6:
      m nKontur7 = PURPLE;
      break;
    case 7:
      m nKontur8 = DARKBLUE;
      break;
    default:
      return O3D E INVALIDINDEX ;
      break;
  }
  return O3D SUCCESS ;
}
```

Code 5-2 Definition der Farbenwerte innerhalb des Prozesses

5.2.2 Darstellung der Kontur mit VTK

Die Bilder die vom OCT Gerät geliefert werden liegen in der Distanz recht weit auseinander, sodass es kaum möglich ist zu bestimmen, welche Konturen miteinander verbunden werden müssen. Da die Bilder von einem Bild zum nächsten meistens eine grosse Abweichung der Form aufweisen ist es nicht möglich, zu bestimmen ob die einzelnen Ränder der Konturen überhaupt verbunden sind oder nicht. Zudem macht es wenig Sinn generell alle Konturen die in Frage kommen zusammen zu verbinden, und so zu suggerieren, dass eine Fläche komplett verbunden sei.

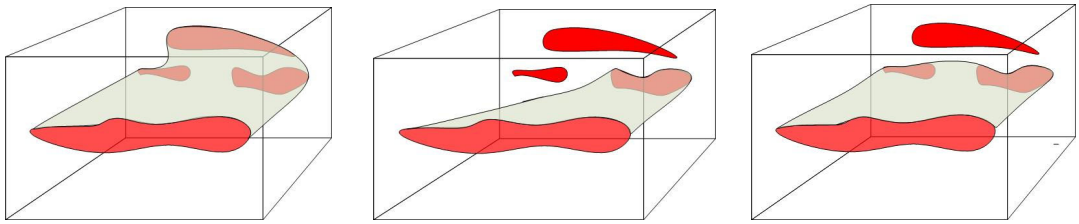


Fig. 5-2 Beispiel von verschiedenen Verbindungen zwischen Konturen.

Eine zufrieden stellende Darstellung der Flächen wird aus diesem Grund kaum je möglich sein. Auch eine Interpolation der Wert zwischen den einzelnen Konturen in die Richtung in der die Schnitte liegen wird nicht einfach möglich sein.

Ich übernahm die Implementation der Klasse C3DSurfaceView, um eine funktionierende Basis Pipeline für die Verarbeitung der Daten mit VTK zu haben. Diese Pipeline ist in einem Beispiel in [VTK] beschrieben, als eine Pipieline die sich für die Darstellung von segmentierten Bilder eignen würde (▶Fig. 5-3).

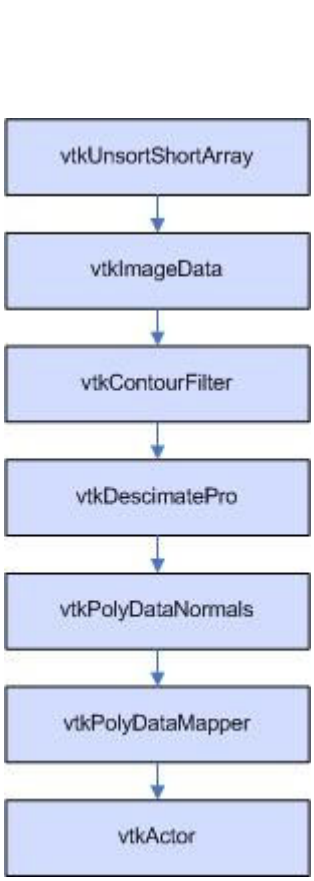


Fig. 5-3 Original Pipeline

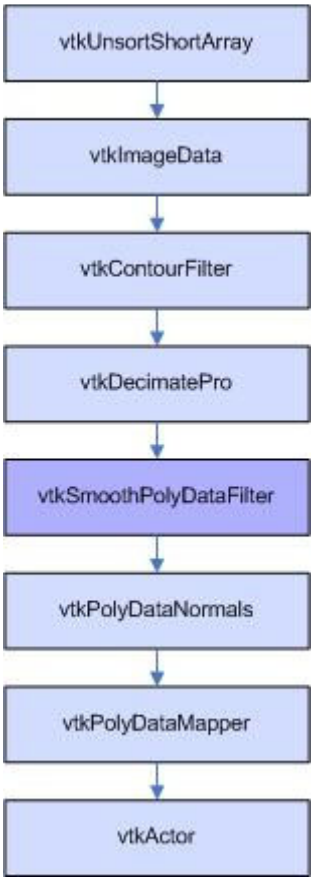


Fig. 5-4 VTK Pipeline mit dem vtkSmoothPolyData-Filter

Das Aussehen der mit dieser Pipeline gelieferten Bilder ist nicht ganz den Wünschen gemäss der Aufgabenstellung entsprechend, da die Kanten nicht so schön abgerundet waren, wie wir uns das vorgestellt haben. Nach längerem suchen fand ich den den `vtkSmoothPolyDataFilter` der versprach die Flächen die dargestellt werden, etwas geschmeidiger zu machen und so die Kanten etwas abzurunden. Es war nicht ganz einfach die nötigen Informationen über die Parameter die einen Sinn ergeben würden zu erhalten [VTKDOC42]. Es blieb mir nicht viel anderes übrig als Versuche zu unternehmen und die verschiedenen Methoden zu testen. Die neue VTK-Pipeline sah nach dem Einbau des Smooth-Filters wie in Fig. 5-4 aus.

Der `vtkSmoothPolyDataFilter` erzeugte die Flächen nun etwas feiner und man erkennt den Verlauf der Fläche etwas besser. Aufgrund fehlender Informationen wie ich die Ausgabe noch besser gestalten könnte, postete ich eine Nachricht in [VTKUSER]. Ich erhielt auch prompt eine Antwort:

«You could use:
- vtkImageGaussianSmooth filter before vtkContourFilter
- vtkWindowedSincPolyDataFilter or vtkSmoothPolyDataFilter before vtkPolyDataNormalshth Goodwin»

by Goodwin Lawlor

Zum einen wurde mir der `vtkSmoothPolyDataFilter` empfohlen, den ich in der Zwischenzeit ja bereis implementiert hatte.

Der zweite Vorschlag den `vtkWindowedSincPolyDataFilter` zu benutzen, habe ich implementiert, ohne allerdings einen Vorteil zum `vtkSmoothPolyDataFilter` zu bemerken.

```
vtkWindowedSincPolyDataFilter* pWinSincPolyDataFilter =
vtkWindowedSincPolyDataFilter::New();
pWinSincPolyDataFilter->SetInput(pDecimatePro->GetOutput());
pWinSincPolyDataFilter->SetFeatureAngle(pSmooth ANGLE);
pWinSincPolyDataFilter->SetFeatureEdgeSmoothing(TRUE);
pWinSincPolyDataFilter->SetBoundarySmoothing(TRUE);
pWinSincPolyDataFilter->SetGenerateErrorScalars(TRUE);
pWinSincPolyDataFilter->SetNonManifoldSmoothing(TRUE);
pWinSincPolyDataFilter->SetNumberOfIterations(ITERATIONS);

m_pPolyDataNormals1->SetInput(pWinSincPolyDataFilter->GetOutput());
```

Code 5-3 Beispiel eines Versuchs mit dem `vtkWindowedSincPolyDataFilter`

Ich habe mich entschlossen den `vtkSmoothPolyDataFilter` implementiert zu belassen.

Der Vorschlag betreffend der Implementation eines Filter habe ich getestet. Ich konnte jedoch mit den verschiedenen Parameter keine zufrieden stellende Lösung erzeugen. Folgend Parameter können beim `vtkImageGaussianSmooth` Filter eingestellt werden.

Name	Bedeutung
Dimensionality	Anzahl der Dimensionen in denen der Filter arbeitet.
StandardDeviations	Definiert die Standard Abweichung. Kann für alle Richtungen einzeln gesetz werden.
Radius Factors	Distanz die besagt in welcher Distanz Werte noch gerechnet werden, bevor diese auf 0 abgerundet werden.

Tabelle 5-2 Parameter des `vtkImageGaussianSmooth` Filter

In der jetzigen Ansicht werden die Konturen als Fläche dargestellt, die in derselben x und y Position in Ihrer Farbe übereinstimmen. Wenn sowohl im vorderen Bild als auch im hinteren Bild die Farbe des Pixels nicht übereinstimmt, wird das Segment als 2 Dimensionale Fläche dargestellt.

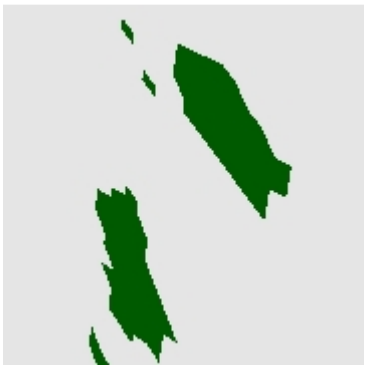


Fig. 5-5 Darstellung von Konturen die nur in einem Bild vorhanden sind

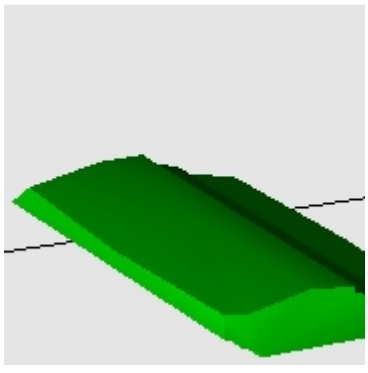


Fig. 5-6 Darstellung von Konturen die in mehreren Bildern vorhanden sind

5.2.3 Implementation der Interaktion mit den Konturen

Ich übernahm die Implementation der Möglichkeit die Konturen mit einem Schieber die Transparenz zu setzen. Allerdings sollte es auch möglich sein, die Transparenz durch das anklicken der Fläche zu verändern. Ich versuchte als erstes jedem Actor in der Pipeline einen Observer hinzuzufügen, der auf einen Mausklick reagiert.

```
m pActorVolume1 = vtkActor::New();
:
m pActorVolume1->SetPickable(TRUE);
:
//here I want to caught the action
m pActorVolume1->AddObserver(vtkCommand::LeftButtonPressEvent,
p3DSegmentCallback);
```

Code 5-4 Versuch den `Observer` dem `Actor` anzufügen

Dieser Ansatz hat aber nicht funktioniert. Der Grund dafür wurde mir in [VTKUSER] mitgeteilt:

«This is what I understand regarding events and callbacks. The vtkActor (actually the base class vtkProp) specifies only one event - PickEvent. So adding an observer on vtkActor for mouse events will have no effect. What you can do is leave the mouse event observer on interactor itself. When the event occurs, get the click position, simulate a pick and see if picked property is desired property.»

Vidyadhar

Nach dem anfügen des Observers an den `vtkRenderWindowInteractor`, funktionierte das setzen der Transparenz wie gewünscht.

```
vtkRenderWindowInteractor *pInteractor =
                      m pRenderWindow->MakeRenderWindowInteractor();
:
pInteractor->AddObserver(vtkCommand::EndPickEvent, p3DSegmentCallback);
```

Code 5-5 3DSegmentView::GetPolyData 1()

```
virtual void Execute(vtkObject *caller, unsigned
                                        long eventId, void* callData){

if (eventId == vtkCommand::LeftButtonPressEvent){
  vtkRenderWindowInteractor *interactor =
                reinterpret cast<vtkRenderWindowInteractor *>(caller);
  int nX = interactor->GetEventPosition()[0];
  int nY = interactor->GetEventPosition()[1];
  int nZ = interactor->GetEventPosition()[2];

  vtkRenderer* pRenderer = interactor->FindPokedRenderer(nX, nY);
  if (pRenderer){
    vtkPropPicker* picker = vtkPropPicker::New();
    pRenderer->GetProps()->InitTraversal();
    int picked = picker->Pick(nX,nY,nZ,pRenderer);
    if (picker->GetActor() != NULL){
      vtkProp* pProp = picker->GetProp();
      vtkProperty* pickedProp = picker->GetActor()->GetProperty();
      if (pickedProp->GetOpacity() == 0.05f)
        pickedProp->SetOpacity(1.00f);
      else
        pickedProp->SetOpacity(0.05f);
      }
    }
  }
}
```

Code 5-6 C3DSegmentView::Execute() - abfangen von Events vom Observer

Mit der Maus kann man nun zusätzlich noch mit einem Klick auf die Kontur die Helligkeit dieser Kontur auf 5% Prozent setzen. Das heisst ich kann innere Schicht durch die transparente Schicht betrachten. Falls ich Schichten ganz ausblenden möchte kann ich die Checkboxen im Panel verwenden. Das setzen aller Konturen auf einen bestimmten Durchsichtigkeits-Wert ist mit dem Schieberegler möglich.

Alle anderen Interaktion die sich mit dem Volumen in der 3D Ansicht machen lassen sind geblieben. Dazu gehören die beliebigen Drehungen, Drehungen um die Blickrichtung der Kamera **[CTRL]** und das Zoomen des Volumens rechte Maustaste, sowie die Verschiebung in der X-Y Ebene der Kamera mit der mittleren Maustaste.

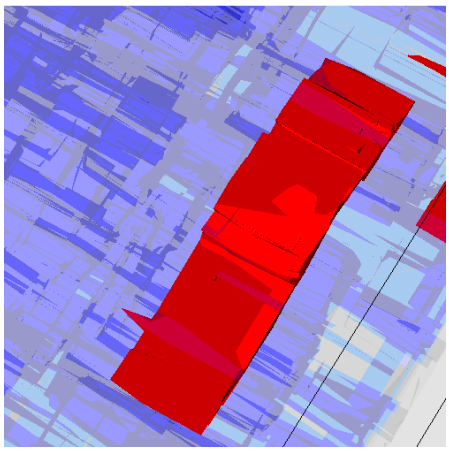


Fig. 5-7 Beispiel einer transparenten Fläche

6 Ziel 4 - 2D-Betrachtung, Farbdarstellung

Autor	Adrian Wyssmann
Implementation	Adrian Wyssmann
Ziel	Es wäre praktisch, wenn via Mausbewegung die Farbdarstellung wie im 3D-Modul geändert werden konnte. So kann eine Läsion[1] auf einfache Weise kontrastreicher dargestellt werden.
Status	Ziel teilweise erreicht
Bemerkung	Das Color-Window (▶Fig. 6-1) lässt sich durch drücken der Kombination **[ALT]+ [linke Maustaste]** anpassen. Allerdings verschwinden die Farben, d.h. die Anpassungen am Color-Window werden auf einem S/W-Bild gemacht.

Tabelle 6-1 Übersicht Ziel 4 - 2D-Betrachtung, Farbdarstellung

6.1 Idee

6.1.1 vtkImagePlaneWidget

Mein erster Ansatz, die 2D-Ansicht um die Methode zu erweitern, den Kontrast anpassen zu können, nahm ich aus der 3DCutView Ansicht. Hier wurde ein `vtkImagePlaneWidget` verwendet, welches standardmässig eine solche Methode eingebaut hat. Eine gedrückte rechte Maustaste bewirkt in Zusammenhang mit einer Mausbewegung folgende Änderung des Color-Window:

- Die Bewegung von **links nach rechts** erhöht den Level-Wert des Color-Window und verdunkelt das Bild. Ein tiefer Wert bewirkt dabei ein helleres Bild (Tendenz zu weiss)
- Die Bewegung von **oben nach unten** erhöht die Breite des Fensters des Color-Window und vermindert den Kontrast. Ein hoher Wert bewirkt dabei, dass das Bild grau wird.

Aufgrund des Programmablaufs innerhalb der 2D-View, habe ich das `vtkImagePlaneWidget` als globale Klassenvariable definiert. In der Methode `CreateVtkPipeline()` wird das `vtkImagePlaneWidget` initialisiert. Dabei wurde mit der Methode `SetPlaneOrientation()`, die Orientierung des PlaneWidgets auf die Z-Achse gesetzt (2 = Z-Achse). Das PlaneWidget zeigt also die Fläche der X-Y Achsen an.

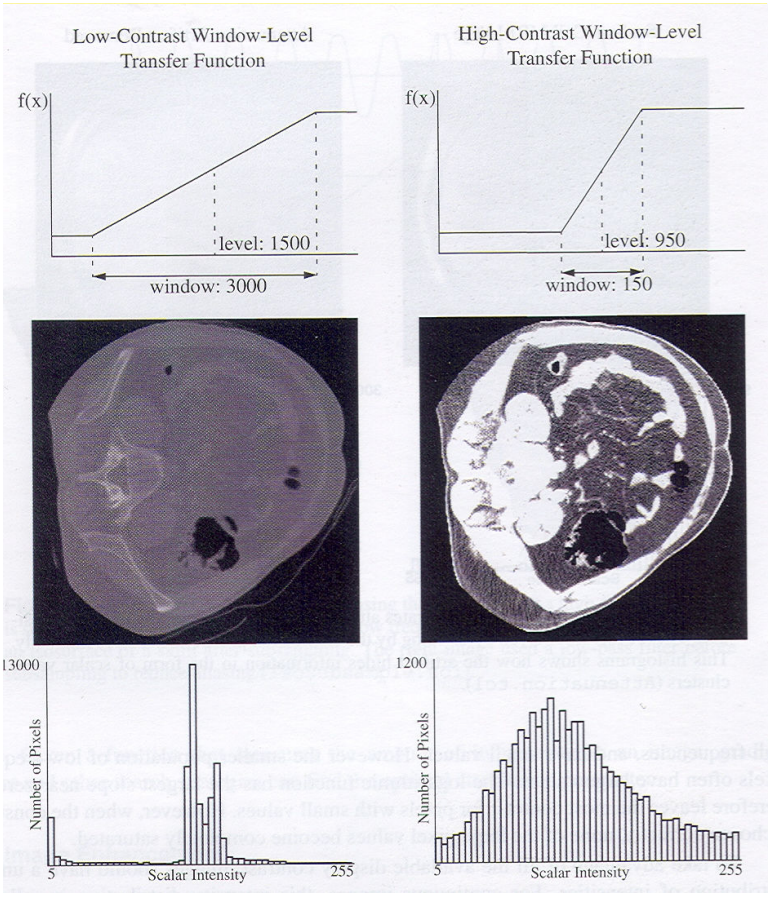


Fig. 6-1 Color-Window

[1] lat. laesio - eine Schädigung, Verletzung oder Störung einer anatomischen Struktur

```
m_pImagePlaneWidget = vtkImagePlaneWidget::New();
m_pImagePlaneWidget->DisplayTextOn();
m_pImagePlaneWidget->GetTextProperty()->SetColor(O3D_VTK_FORGROUND_TEXT);
m_pImagePlaneWidget->SetInput(pImageData);
m_pImagePlaneWidget->RestrictPlaneToVolumeOn();
m_pImagePlaneWidget->GetPlaneProperty()->SetColor(O3D_VTK_FORGROUND_ZOOMBOX);
m_pImagePlaneWidget->SetPlaneOrientation(2);
m_pImagePlaneWidget->SetInteractor(m_pInteractor);

m_pImagePlaneWidget->SetSlicePosition(fBounds[4]+(fBounds[5]-fBounds[4]+1)/2);
m_pImagePlaneWidget->Modified();
m_pImagePlaneWidget->UpdatePlacement();
m_pImagePlaneWidget->GetLookupTable()->DeepCopy(m_pLookupTable);
```

Code 6-1 2DView.cpp::CreateVtkPipeline()

Ein `vtkImagePlaneWidget` wird üblicherweise für 3D-Volumen verwendet - so wie in der 3DCutView Ansicht. Die Idee, es auch für 2D-Objekte zu verwenden, indem man dem Plane einfach sagt, sie solle zuvorderst auf der Z-Achse liegen (`SetSlicePosition`), hat nicht funktioniert.

Das 1. Bild der Serie (1. Bild in der Video-Selection) war von Anfang an sehr dunkel ▶Fig. 6-2. Wieso das so ist, kann ich mir nicht erklären. Es war hier auch nicht möglich, die Originalfarben wieder anzuzeigen. Alle weiteren Bilder hatten dann das Problem, dass sie vom 1. Bild überlagert wurden.

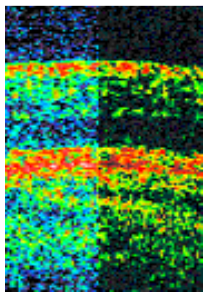


Fig. 6-2 ImagePlane Widget Problem

Ein Posting im [VTKUSER] erklärte man mir, dass `vtkImagePlaneWidget` nur für 3D-Bilder gebraucht werden sollen, für 2D-Bilder eigenen sich entweder `vtkImageViewer` oder `vtkImageViewer2`:

«tyry using class vtkImageViewer2, as it was meant for 2D image display and has built in window-level functionality controlled by the mouse. vtkImagePlaneWidget is really designed to work with 3D image data.»

by Dean Inglis
http://www.vtk.org/pipermail/vtkusers/2004-October/077094.html

6.1.2 vtkImageViewer/vtkImageViewer2

In der [VTKDOC42] habe ich keine Beschreibung zum vtkImageViewer2 gefunden. Es ist lediglich eine kurze Beschreibung für den `vtkImageViewer` zu finden:

«vtkImageViewer is a convenience class for displaying a 2d image. It packages up the functionality found in vtkImageWindow, vtkImager, vtkActor2D and vtkImageMapper into a single easy to use class. Behind the scenes these four classes are actually used to to provide the required functionality. vtkImageViewer is simply a wrapper around them.»

http://www.vtk.org/doc/release/4.2/html/classvtkImageViewer.html

6.1.2.1 vtkImageViewer

`vtkImageViewer` ist eine Klasse um 2D-Bilder anzuzeigen. Es handelt sich dabei um eine Wrapper-Klasse, die die Klassen `vtkImageWindow`, `vtkImager`, `vtkActor2D` und `vtkImageMapper` umhüllt. `vtkImageViewer` bietet denselben Funktionsumfang wie die von ihm umhüllten Klassen. Das beinhaltet beispielsweise auch das Einstellen des Color-Window, wie in der Klasse `vtkImageMapper`:

```
m_pImageViewer->SetInput(pImageData);
m_pImageViewer->SetupInteractor(m_pInteractor);

m_pImageViewer->GetImageMapper()->RenderToRectangleOn();
m_pImageViewer->SetParentId(m_hWndData);
m_pImageViewer->SetSize(m_pRenderWindow->GetSize());
m_pImageViewer->GetRenderer()->SetBackground(O3D_VTK_BACKGROUND);
m_pImageViewer->GetRenderer()->SetActiveCamera(pCamera);
```

Code 6-2 C2DView::CreateVtkPipeline() - Initialization vtkImageViewer

Die Implementation mit dem `vtkImageViewer` hat allerdings nicht wunschgemäss funktioniert. Es war zwar möglich, das Color-Window zu verändern und das sogar in Farbe ▶Abschnitt 6.1.2.2. Dafür gab es aber folgende Probleme:

- Das Bild wurde in den Originial-Dimensionen angezeigt, d.h. es beanspruchte auf dem Bildschirm die 512x1024 Pixel ▶Abschnitt 4.2.2.1.
- Trotz dem korrekten Zuweisen des Interactors, waren keine Interaktionen mehr möglich. Weder ein verschieben des Bildes noch ein Zoomen war möglich.

Da auch im [VTKUSER] niemand eine Lösung für das Problem hatte, ist die Lösung mit dem `vtkImageViewer` **nicht brauchbar.**

6.1.2.2 vtkImageViewer2

Zwar ist hier im Gegensatz zum `vtkImageViewer` eine Interaktion möglich, dennoch funktioniert die Implementation ebenfalls nicht wunschgemäss:

- Sobald das Color-Window verändert wir, verschwinden die Farben aus dem Bild, wir haben nur noch ein Graustufenbild. Beim Zurücksetzen des Color-Window mit dem Code 6-3 erscheinen die original Farben wieder.
- Wir haben eine Doppelbelegung, wie ich es in Abschnitt 8.3 erkläre. Der Interaktor, den ich dem vtkImageViewer2 zuweise, führt bei einer Mausbewegung mit gedrückter linker Maustaste ein Zeichnen der Zoom-Box durch. Gleichzeitig wird aber auch das Color-Window verändert.

```
pImageData->GetBounds(fBounds) ;
int nWidth = fBounds[1]-fBounds[0];
int nHeight = fBounds[3]-fBounds[2];
m_pImageMapToWindowLevelColors->SetLevel(0.5*(range[1] + range[0]));
m_pImageMapToWindowLevelColors->SetWindow(range[1] - range[0]);
```

Code 6-3 Zurücksetzen des Color-Window

Da auch im [VTKUSER] niemand eine Lösung für das Problem hatte, ist die Lösung mit dem `vtkImageViewer2` ebenfalls **nicht brauchbar**.

6.2 Implementation

Da ja bereits eine Callback-Klasse existiert (▶2DView.h, Klasse CvtkCallbackData), welche benutzerdefinierte Aktionen ausführt, wie beispielsweise das Einzeichnen einer Zoom-Box, verwende ich diese Klasse um das Color-Window zu setzen. Details zum Unterschied Interactor und Observer siehe Abschnitt 8.3.

Dazu führe ich in der Callback-Klasse eine neue Variable ein. Die wird dann innerhalb der Klasse C2DView initialisiert:

```
vtkImageMapToWindowLevelColors* m_pImageMapToWindowLevelColors
```

Abhängig von der Richtung der Mausbewegung - ΔnX, ΔnY - verändere ich die Grösse oder den Level des Color-Window:

```
case vtkCommand::MouseMoveEvent:

   if ((GetAsyncKeyState(VK_MENU) & 0x8000) && m_bWindowLevelEnabled) {
      if (nX > m_nX) {
         m_pImageMapToWindowLevelColors->SetWindow(
                  m_pImageMapToWindowLevelColors->GetWindow()+abs(m_nX - nX));
      } else {
         m_pImageMapToWindowLevelColors->SetWindow(
                  m_pImageMapToWindowLevelColors->GetWindow()-abs(m_nX - nX));
      }
      if (nY > m_nY) {
         m_pImageMapToWindowLevelColors->SetLevel(
                  m_pImageMapToWindowLevelColors->GetLevel()+abs(m_nY - nY));
      } else {
         m_pImageMapToWindowLevelColors->SetLevel(
                  m_pImageMapToWindowLevelColors->GetLevel()-abs(m_nY - nY));
      }
      m_nX = nX;
      m_nY = nY;
         m_pRenderer->GetRenderWindow()->Render();
      return;
   }
```

Code 6-4 CvtkCallbackData::Execute()

Nun brauche ich in der Klasse C2DView eine globale Variable:

```
vtkImageMapToWindowLevelColors* m_pImageMapToWindowLevelColors
```

Diese weise ich bei der Initialisierung von CvtkCallbackData der Callback-Klasse zu:

```
pICallback->m_pImageMapToWindowLevelColors = m_pImageMapToWindowLevelColors;
```

Code 6-5 C2DView::CreateVtkPipeline() - Zuweisen des vtkImageMapToWindowLevelColors

Damit das Color-Window angepasst wird, muss ich das Bild aktualisieren. Aufgrund des Datenfluss innerhalb der Klasse, mache ich das in der Methode C2DView::OnChangeVideoSelection():

```
m_pImageMapToWindowLevelColors->SetOutputFormatToRGB();
m_pImageMapToWindowLevelColors->SetInput(pImageData);
m_pImageMapToWindowLevelColors->Update();
pImageData = m_pImageMapToWindowLevelColors->GetOutput();
SetWindowLevelText();
```

Code 6-6 C2DView::OnChangeVideoSelection()

6.2.1.1 Anzeigen der aktuellen Werte des Color-Window

Um wie in der 3D-Ansicht die Werte des Color-Window anzeigen zu können, habe ich einen `TextActor` eingebaut, der die entsprechenden Werte anzeigt:

```
pTextActor = vtkTextActor::New();
pTextActor->ScaledTextOn();
pTextActor->SetPosition(0,5);
//doesn't work: pTextActor->SetTextProperty(pTextProp);
pTextActor->GetTextProperty()->SetFontFamilyToArial();
pTextActor->GetTextProperty()->BoldOff();
pTextActor->GetTextProperty()->SetColor(O3D VTK FORGROUND TEXT);
pTextActor->GetTextProperty()->ShadowOn();
pTextActor->SetHeight(0.03);
```

Code 6-7 Text-Actor zum Anzeigen der Werte des Color-Window

Damit ich den `TextActor` nicht auch noch der Callback-Klasse übergeben muss, werde ich den Text in der Methode `C2DView::PreTranslateMessage()` anpassen. Diese Default-Methode dient zum Filtern von Windows-Nachrichten:

```
BOOL C2DView::PreTranslateMessage(MSG* pMsg) {
   SetWindowLevelText();
```

Code 6-8 C2DView::PreTranslateMessage() - Setzen des Textes

Die Methode `C2DView::SetWindowLevelText()` setzt den Text des `TextActors`:

```
void C2DView::SetWindowLevelText() {

   CString strWindowLevel, strColorLevel;
   strWindowLevel.Format(TEXT("Window, Level ( %.0f , %.0f )"),
                         m pImageMapToWindowLevelColors->GetWindow(),
                         m pImageMapToWindowLevelColors->GetLevel());
   m pTextActor->SetInput(strWindowLevel);
   m pRenderer->AddActor(m pTextActor);
}
```

Code 6-9 C2DView::SetWindowLevelText()

6.3 Probleme

Leider besteht auch hier das Problem, dass durch verändern des Color-Window unser Farbbild nur noch in Graustufen angezeigt wird. [VTKUSER] konnte mir auch hier nicht weiterhelfen. Wenn ich das Color-Window gemäss *Code 6-10* initialisiert habe, erhalte ich für die Fensterbreite den Wert 255 und für den Level den Wert 127.5:

```
float* range = pImageData->GetScalarRange();
m pImageMapToWindowLevelColors->SetLevel(0.5*(range[1] + range[0]));
m_pImageMapToWindowLevelColors->SetWindow(range[1] - range[0]);
```

Code 6-10 Initialisierung des Color-Window

Daraus schliesse ich, dass das Problem könnte etwas mit der eigentlichen Anzahl Farben (je 256 für R,G,B) und den Scalarwerten zu tun haben. Ich habe nun versucht, dem `vtkImageMapToWindowLevelColors` die Lookuptable, zuzuweisen. Obwohl es sich um dieselbe Lookuptable handelt, die auch für das einfärben der Bilder verwendet wurde, resultiert daraus ein schwarzes Bild.

```
m_pImageMapToWindowLevelColors->SetLookupTable(m_pLookupTableOriginal);
```

Code 6-11 Zuweisen einer Lookuptable

Im Gegensatz zu der Implementation mit dem `vtkImageViewer2` (▶Abschnitt 6.1.2.2), haben wir aber keine Probleme mit der Interaktion. Eine Mausbewegung bei gedrückter linker Maustaste ermöglicht es die Zoom-Box zu zeichnen. Bei gleichzeitigem drücken der Taste **[ALT]** wird das Color-Window verändert.

7 Ziel 5 - Export auf Harddisk

Autor	Adrian Wyssmann
Implementation	Adrian Wyssmann
Ziel	Es kann via Rechtsklick ein File, Scandaten eines ganzen Patienten oder einer ganzen Summe von Patienten angewählt und in einen Zug z.B. auf eine externe Harddisk exportiert werden. Falls ein File bereits besteht, fragt die Software nach, ob es wirklich überschrieben werden muss.
Status	Ziel erreicht
Bemerkung	In der Ansicht Datenselektion lassen sich via Standard-Menü oder Kontextmenü alle Scanserien eines Patienten oder die gesamte Datenbank als o3p-Dateien[1] auf die Festplatte exportieren. Der Zielordner kann selber ausgewählt werden. Wahlweise lässt sich auch pro Patient ein eigener Ordner erstellen. Der Dateinamen wird automatisch erstellt und ist eindeutig. Alte Dateien werden automatisch überschrieben. Dies entspricht nicht exakt den Wünschen der Ärzte.

Tabelle 7-1 Übersicht Ziel 5 - Export auf Harddisk

7.1 Idee

Als erstes habe ich mir Gedanken darüber gemacht, wie wir die Export-Methode einbinden wollen. Mir erschien es am einfachsten, dies zunächst als Menü-Einträge im Standardmenü einzubinden. Wenn dies dann funktioniert, kann man immer noch ein Kontext-Menu kreieren.

In der OCT3D Applikation sind 3 Standard-Prozess bereits implementiert: "2D Ansicht", "2D Ansicht (Datei)" und "Scanserie exportieren". Diese Prozesse können in jedem Fall auf eine Scanserie ausgeführt werden, da sie nicht in eine Pipeline eingebunden werden müssen. Das brachte mich natürlich auf die Idee, den bereits implementierten Exportprozess für die zusätzliche Exportmöglichkeit zu verwenden.

Die Standardprozesse sind in der Datei `O3DStandardModule.cpp` definiert:

```
const char* CO3DStandardModule::GetNameForAggregate(O3D HPROC hProc) {
   if (hProc == AGGREG PROC FIRST) {
      return "2D Ansicht"
   }
   else if (hProc == AGGREG PROC FIRST+1) {
      return "2D Ansicht (Datei)"
   }
   else if (hProc == AGGREG PROC FIRST+2) {
      return "Scanserie exportieren"
   }
   return NULL ;
}
```

Code 7-1 O3DStandardModule::GetNameForAggregate()

[1] Exportformat der Gruppe OCT3D

Registriert werden die Standardprozesse in der Klasse `CO3DStandardModule`, beim initialisieren:

```
int CO3DStandardModule::Init() {
...
   REGISTER STANDARD AGGREG PROC(AGGREG PROC FIRST,
    sizeof( sub 2d)/sizeof(  alloc subproc),  sub 2d) ;
   REGISTER STANDARD AGGREG PROC(AGGREG PROC FIRST+1,
    sizeof( sub 2dDatei)/sizeof(  alloc subproc),  sub 2dDatei) ;
   REGISTER STANDARD AGGREG PROC(AGGREG PROC FIRST+2,
    sizeof( sub 2dFileWriter)/sizeof(  alloc subproc),  sub 2dFileWriter) ;

   return O3D SUCCESS ;
}
```

Code 7-2 CO3DStandardModule::Init()

Das Makro `REGISTER_STANDARD_AGGREG_PROC` ruft dafür die Methode `CO3DStandardModule::RegisterProcess()` auf und registriert den Standardprozess:

```
#define REGISTER STANDARD PROC( hproc,  class) \
    if (O3D SUCCESS > CO3DStandardModule::GetInstance()->
    RegisterProcess( hproc,  class::GetProcDesc,  class::FreeProcDesc,
     class::GetParamDesc,  class::CreateInstance,  class::GetProcDescText))\
      return O3D E NOSUCHPROC ; \

#define REGISTER STANDARD AGGREG PROC( hproc,  nr,  def) \
   if (O3D_SUCCESS > CO3DStandardModule::GetInstance()->
    RegisterProcess(_hproc, NULL, NULL, NULL, NULL, NULL, _nr, _def)) \
      return O3D_E_NOSUCHPROC ; \
```

Code 7-3 CO3DStandardModule: Makros

7.1.1 Wie wird ein Standardprozess aufgerufen?

Angesetzt habe ich dort, wo es für mich am offensichtlichsten erschien: Beim Erstellen des Kontextmenüs für die Scanserie. Das Kontextmenü mit den verfügbaren Prozessen wird erstellt, sobald man auf den Hyperlink einer Scanserie klickt. Diese Aktion führt die Methode `ScanSelectioView::OnHyperlink()` aus. Die beiden Parameter `(PARAM) wParam` und `lParam` entscheiden, welcher Patient und welche Scanserie ausgewählt wird.

```
LRESULT CScanSelectionView::OnHyperLink(WPARAM wParam, LPARAM lParam) {
...
   //what selection
   long lPatient = wParam, lSerie = lParam;

   long lLast2D, lLast3D ;
   CProcDescArr* pArrayProcDesc = GetDocument()->GetProcDescArr(lPatient,
      lSerie, &lLast2D, &lLast3D);
   if (pArrayProcDesc != NULL) {
...
   menu.AppendMenu(MF STRING, ID PROCESS START+nIndex,
      pProcessDescription->GetName());
...
   return 1;
}
```

Code 7-4 CScanSelectionView::OnHyperLink()

Durch die ID_PROCESS_START+nIndex, die der Methode AppendMenu() mitgegeben wird, kann der Prozess, der sich hinter dem entsprechenden Menüeintrag verbirgt, eindeutig identifiziert werden. ID_PROCESS_START ist in der Klasse CO3DStandardModule definiert:

```
//process handle for first aggregate process
#define AGGREG_PROC_FIRST   1000
```

Code 7-5 O3DStandardModule.cpp

nIndex wird von der Grösse des Array pArrayProcDesc definiert. Dieses ist vom Typ CProcDescArr. Dieser Typ wird in der Datei ProcessDecription.h definiert:

```
typedef CArray<CO3DProcessDescription*, CO3DProcessDescription*>
CProcDescArr ;
```

Code 7-6 ProcessDecription.h::CProcDescArr

Der Array selber wird von der Methode CSelectionDoc::GetProcDescArr() zurückgeliefert. Diese Methode betrachtet eine Scanserie, und füllt den Array mit den möglichen Prozessen für diese Serie. Die Informationen für die Prozesse holt die Methode CO3DProcessManager::GetProcDescArr(), indem sie die Methode CO3DProcessManager::GetProcDescByType() aufruft:

```
...
CO3DProcessManager::GetInstance()->GetProcDescByType(O3D_IO_DATASOURCE,
                                                     O3D_IO_DATASINK_2D,
                                                     O3D_FI_IN_AND|
                                                     O3D_FI_OUT_AND,
                                                     &arrTemp);
...
```

Code 7-7 CO3DProcessManager::GetProcDescByType()

CO3DProcessManager::GetProcDescByType() holt dabei die Prozessbeschreibungen aus m_mapProcsByName. m_mapProcsByName ist eine Hash-Map[1], in der alle Prozesse, resp. Pipelines die im OCT3D zur Verfügung stehen, gespeichert sind.

```
...
while (pos) {
   m_mapProcsByName.GetNextAssoc(pos, pszKey, pProcDesc) ;
   if (NULL != pProcDesc) {
      ...
      if (bInMatch && bOutMatch )
         pArr->Add(pProcDesc) ;
   }
}
```

Code 7-8 CO3DProcessManager::GetProcDescByType()

Wird ein Menüeintrag aus dem Kontextmenü angeklickt, wird eine Nachricht verschickt. Durch ON_COMMAND_RANGE wird die Nachricht abgefangen. Die ID der Nachricht liegt hierbei zwischen ID_PROCESS_START und ID_PROCESS_END. Dies hat wiederum zur Folge, dass die Methode CSelectionDoc::OnProcess() aufgerufen wird, welche den Prozesskontext (pctx) initialisiert und dann den eine PostThreadMessage an die Anwendung schickt, die den Prozess ausführt.

[1] CMap<O3D_HPROC, O3D_HPROC&, __proc_info*, __proc_info*> m_mapProcInfo ;

```
ON COMMAND RANGE(ID PROCESS START, ID PROCESS END, OnProcess)

void CSelectionDoc::OnProcess(UINT nId)
{
...
   CO3DProcessDescription *pProcessDescription =
    m arrayProcDesc.GetAt((int)uiIndex);
   if (!pProcessDescription)
      return;

   CO3DProcessContext* pCtx =  CO3DProcessManager::GetInstance()->
    CreateContext() ;
   if (NULL != pCtx) {
      pCtx->InitForExecution(new CPatientData(m pPatientData),
                  new COctScanSerieData(m pOctScanSerieData),
                  pProcessDescription->GetName()) ;
      AfxGetApp()->PostThreadMessage(WM_APPDOCPROCESS, (UINT)0,
        (long)pCtx);
   }
}
```

Code 7-9 CSelectionDoc::OnProcess()

Der Methode `CO3DProcessContext::InitForExecution()` wird neben Patienten- und Scanseriedaten auch der Name des Prozesses übergeben. Danach wir die Methode `PostThreadMessage()`[1] aufgerufen, welche die Nachricht `WM_APPDOCPROCESS` in eine Message-Queue schreibt. Diese Queue wird von der Main-Klasse (`COct3dApp`) abgearbeitet:

```
ON THREAD MESSAGE(WM APPDOCPROCESS, OnAppDocProcess)
...
void COct3dApp::OnAppDocProcess(UINT ui, LONG l) {
   CProcessDoc* pProcessDoc = (CProcessDoc*)m pProcessDocTemplate->
   CreateNewDocument() ;

   pProcessDoc->SetContext((CO3DProcessContext*)l);
   pProcessDoc->Execute() ;
}
```

Code 7-10 COct3dApp::OnAppDocProcess()

Nachdem nun geklärt ist, wie der Mechanismus funktioniert, kann ich nun mit der Implementation beginnen.

7.2 Implementation

Wie ich am Ende des vorherigen Abschnitts beschrieben habe, muss ich der Methode `CO3DProcessContext::InitForExecution()` u.a. den Name des Prozesses übergeben. Dieser Name muss eindeutig sein. Ich definiere für die 3 Standardprozesse in der Stringtable 3 IDs, denen ich den entsprechenden Namen zuweise. Die Änderung fliesst in die Methode `O3DStandardModule::GetNameForAggregate()` ein:

```
return PROC2DVIEW; //replace return "2D Ansicht"
return PROC2DFILE; //replace return "2D Ansicht (Datei)"
return PROCEXPORT; //replace return "Scanserie exportieren"
```

Code 7-11 Änderungen an O3DStandardModule.cpp::GetNameForAggregate()

Nach dieser Änderung kann ich der Methode `InitForExecution()` direkt die ID für den Standardprozess übergeben, den ich ausführen will.

[1] http://www.flounder.com/messaging.htm#PostThreadMessage

7.2.1 Menü definieren

Als nächstes erstelle ich mit dem Resource-Editor die Menüeintrag für den Export und zwar im Menü `IDR_SELECTIONTYPE`. Das ist dasjenige Menü, das dem Benutzer zur Verfügung steht, sobald er sich in der Ansicht "Datenselektion" befindet. Der Menüpunkt **[Export]** umfasst 3 Punkt: **[Alle Patienten]**, **[Ausgewählter Patient]** und **[Einstellungen...]**.

Fig. 7-1 Export-Menü

Nun müssen wir den Event abfangen, wenn der Benutzer einen der Menüeinträge auswählt. Wir fangen das durch das Message-Mapping in der Klasse `CSelectionDoc` ab:

```
ON_COMMAND(ID_EXPORT_THIS, OnExportThis)
ON_COMMAND(ID_EXPORT_FULLDB, OnExportFullDB)
ON_COMMAND(ID_EXPORT_SETTING, OnExportSetting)
```

Code 7-12 CSelectionDoc::MESSAGE_MAP

7.2.2 Ausgewählter Patient exportieren - OnExportThis()

Die Methode `CSelectionDoc::OnExportThis()` führt dabei den Export für alle Scanserien eines bestimmten Patienten durch. Hierfür muss anhand der Patientendaten, die Scanserien ausgelesen und in eine Liste geschrieben werden:

```
CInterbaseData::GetInstance()->ReadSeriesPatient(m_pPatientData->GetPatient(),
m_listOctScanSerie))
```

Code 7-13 Scanserien eines Patienten in eine Liste abfüllen

Die Daten des aktuellen Patienten bekommen wir durch Klassenvariable, `m_pPatientData`. Diese wird immer dann gesetzt, wenn in der Ansicht "Datenselektion" ein Patient markiert wird. Für jede Scanserie aus der Liste `m_listOctScanSerie` wird nun die Methode `DoExport()` ausgeführt:

```
void CSelectionDoc::OnExportThis() {
...
   CInterbaseData::GetInstance()->ReadSeriesPatient(m_pPatientData->
      GetPatient(), m_listOctScanSerie))

...
   POSITION pos = m_listOctScanSerie.GetHeadPosition();

   while(pos)
   {
      pDataScan = m_listOctScanSerie.GetNext(pos);
      if (O3D_SUCCESS != DoExport(m_pPatientData, pDataScan)) {
         AfxMessageBox( SLOAD(IDS_ERROR_EXPORT));
         return;
      }
   }
   return;
}
```

Code 7-14 CSelectionDoc::OnExportThis()

Das `CSelectionDoc::DoExport()` macht nichts anderes, wie in Abschnitt 7.1.1 beschrieben, als den Prozesskontext zu initialisieren und via `PostThreadMessage`, den Prozess auszuführen:

```
int CSelectionDoc::DoExport(CPatientData* pPatientData, COctScanSerieData*
 pDataScan) {
   if (pDataScan) {
      CO3DProcessContext* pCtx =  CO3DProcessManager::GetInstance()->
                              CreateContext() ;
      if (NULL != pCtx) {
         pCtx->InitForExecution(new CPatientData(pPatientData),
                        new COctScanSerieData(pDataScan),
                        PROCEXPORT) ;
         if ( AfxGetApp()->PostThreadMessage(WM APPDOCPROCESS, (UINT)0,
            (long)pCtx))
            return O3D SUCCESS;
      }

   }
   return O3D E GENERAL;
}
```

Code 7-15 CSelectionDoc::DoExport()

7.2.3 Alle Patienten exportieren - OnExportFullDB()

Um alle Patienten einer Datenbank zu exportieren, müssen wir für jeden Patienten der Datenbank dasselbe machen, wie im Prozess `CSelectionDoc::OnExportThis()`. Alle Patienten sind in der Liste der Ansicht "Datenselektion" gespeichert. Wir müssen nun einfach diese Liste abarbeiten:

```
void CSelectionDoc::OnExportFullDB() {
...
   POSITION pos = m listPatient.GetHeadPosition();

   while(pos) {
      if (pPatientData) {
         //get scandata of current patient
         if (IDOK != CInterbaseData::GetInstance()->
            ReadSeriesPatient(pPatientData->GetPatient(),
            m listOctScanSerie))
            AfxMessageBox(CInterbaseData::GetInstance()->
                        GetError());

         if (!m listOctScanSerie.IsEmpty()) {
            POSITION pos2 = m listOctScanSerie.GetHeadPosition();
            while(pos2)
            {
               pDataScan = m listOctScanSerie.GetNext(pos2);
               if (O3D SUCCESS != DoExport(pPatientData,
                              pDataScan)) {
                  AfxMessageBox( SLOAD(IDS ERROR EXPORT));
                  return;
               }
            }
         }
      }
   }
}
```

Code 7-16 CSelectionDoc::OnExportFullDB()

7.2.4 Export-Einstellungen

Diese Methode ruft ein Dialogfenster auf. In diesem, besteht die Möglichkeit die Exporteinstellungen festzulegen (**[Export▸Einstellungen]**):

```
int CSelectionDoc::OnExportSetting()
{
   CDlgExportSettings exportSettingsDlg;
   exportSettingsDlg.DoModal();

   return O3D SUCCESS;
}
```

Code 7-17 CSelectionDoc::OnExportSetting()

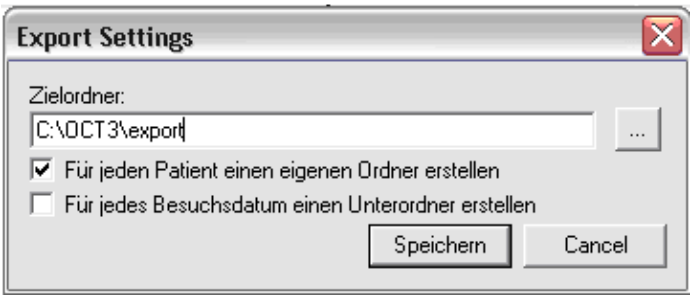


Fig. 7-2 Dialog "Export Settings"

Im Dialog "Export Settings" lassen sich unter anderem der Zielordner für den Export festlegen. Mit den beiden Checkboxen kann man weitere Exporteinstellungen festlegen:

Einerseits kann man festlegen, ob für jeden Patienten ein separater Unterordner erstellt werden soll. Der Name des Unterordners setzt sich aus dem Vor- und Nachname des Patienten zusammen. Die jeweiligen Exportdateien eines Patienten werden in dem jeweiligen Unterordner erstellt. Die zweite Option ist nur verfügbar, wenn die erste Option ausgewählt ist. Sie ermöglicht es, innerhalb des Patientenordners einen weiteren Ordner für jedes Besuchsdatum zu erstellt. Alle Einstellungen werden mit dem Klick auf **[Speichern]** in die Registry geschrieben. Die Einstellungen sind also auch nach dem Neustart der Anwendung wieder verfügbar.

Fig. 7-3 Dialog "Browse for Folder"

Zielordner auswählen

Um einen Zielordner auszuwählen, hat es einen Schaltfläche "Browse" **[...]**. Mit einem Klick auf diese Schaltfläche öffnet sich ein Dialogfenster, mit dem man im Dateisystem browsen kann, um einen bestimmten Ordner zu finden. Das MFC bietet eine Klasse, für ein Dialog-Fenster zum öffnen von Dateien: `CFileDialog`[1].

Diese eignet sich allerdings nur, um Dateien zu öffnen. Leider bietet das MFC keine Klasse für ein Dialog-Fenster zum Öffnen von Ordnern. Auf [MSND] bin ich auf die Seite über "Windows Shell[2]" gelangt. Es gibt eine Methode, die nennt sich `SHBrowseForFolder`. Ich habe den Code des Beispiels auf der Seite[2] kopiert und entsprechend angepasst:

[1] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclib/html/_MFC_CFileDialog.asp

[2] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/anch_WinShell.asp

```
void CDlgExportSettings::OnButtonExportBrowse()  {

   BROWSEINFO m bi = {0};
   TCHAR strPath[MAX PATH];

   //without that, I can't use BIF NEWDIALOGSTYLE (64)
   OleInitialize(NULL);

   CString strTitle =  SLOAD(ID EXPORT BROWSEFOLDER);
   m bi.lpszTitle    = strTitle;
   m bi.ulFlags    = BIF VALIDATE | 64; //64, otherwise BIF NEWDIALOGSTYLE
                              will be undeclared

   LPITEMIDLIST pItemIDList = SHBrowseForFolder(&m bi);

   if (pItemIDList != 0)
    {
        // get the name of the folder
        if ( SHGetPathFromIDList(pItemIDList, strPath))
        {
         SetCurrentDirectory(strPath);
         m strExportFolder = strPath;
         //set folder text
         GetDlgItem(IDC EXPORT FOLDER)->SetWindowText(strPath);
        }
        // free memory used
        IMalloc * imalloc = 0;
        if ( SUCCEEDED( SHGetMalloc(&imalloc)))
        {
            imalloc->Free(pItemIDList);
            imalloc->Release();
        }
    }
}
```

Code 7-18 CDlgExportSettings::OnButtonExportBrowse()

7.2.4.1 Dateiname der Export-Datei(en)

Da während des Exports keine Benutzereingabe möglich ist (▶Abschnitt 7.2.5) kann wird der Dateinamen für die Export-Dateien automatisch gewählt. Der Dateiname setzt sich folgendermassen zusammen:

<Patientennamen>_<Seite>_<Besuchsdatum>_<Scantyp>.o3d

▶ Beispiel: Max Muster_OD_11.11.2011_Line.o3d

Bestehende Dateinamen werden ohne Rückfrage überschrieben

7.2.5 Probleme

Der Export mit der dem Standardprozess "Scanserie Exportieren" hat nicht wie erwartet funktioniert. Ich habe dasselbe mit dem Standardprozesse "2D Ansicht" probiert, indem ich dem der Methode `InitForExecution()` einfach die ID `PROC2DVIEW` anstelle von `PROCEXPORT` übergeben habe. Jede Scanserie wurde geöffnet und die Bilder angezeigt.

Nach langem Suchen und Ausprobieren habe ich die Ursache des Problems gefunden. Sobald eine Benutzereingabe verlangt wird, oder eine MessageBox angezeigt wird, wird nur jeweils 1 Prozess ausgeführt. Das Problem liegt am Informationsaustausch via `PostThreadMessage`:

```
AfxGetApp()->PostThreadMessage(WM_APPDOCPROCESS, (UINT)0, (long)pCtx);
```

Das `PostThreadMessage()` wird zwar ordnungsgemäss ausgeführt und die Nachricht wird anscheinend in der dafür vorgesehenen Queue gespeichert. Allerdings wird nur eine Nachricht, die in der Queue ist, auch wirklich ausgeführt.

Da also keine Benutzereingabe während des Exportvorgangs möglich ist, musste ich die Klasse `CFileWriter` so anpassen, dass der Dateinamen automatisch generiert wird ▶Abschnitt 7.2.4.1:

```
int CFileWriter::Execute(CO3DProcessContext *pCtx) {
   if (!pCtx || !(pCtx->pPatientInfo) ||
      !m pDataIn || !(m pDataIn->pImageData)) {
      pCtx->SetError( SLOAD(IDS ERROR INVALID DATA));
      return O3D E INVALIDINPUT;
   }

   CString strPath = COct3dApp::GetExportDirectory();

   //used to get the strings strScanType and strDateVisit
   COctScanSerieData* pDataScan = reinterpret cast<COctScanSerieData*>
                        (pCtx->pDataInfo);

   //set strings
   CString strPatientName = pCtx->GetPatientName();
   CString strScanType = pDataScan->GetDisplayname();
   CString strDateVisit = pDataScan->GetVisit();
   CString strSite = pCtx->pDataInfo->pszSite;

   CString strFileName  = strPatientName + " " + strSite;

   //get folder options from registry
   if (AfxGetApp()->GetProfileInt( T("Export"), "FolderPerPatient", 0)) {
      strPath =  strPath+ "\\" + strPatientName;
      CreateDirectory(strPath, NULL);

      if (AfxGetApp()->GetProfileInt( T("Export"), "FolderPerVisitdate",
                        0))
      {
         strPath =  strPath+ "\\" + strDateVisit;
         CreateDirectory(strPath, NULL);
      } else {
         strFileName += " " + strDateVisit;
      }
   }

   strFileName += " " + strScanType + ".o3d";
   CString strFile = strPath + "\\" + strFileName;
   return Write(pCtx, strFileName, strFile);
}
```

Code 7-19 Änderung in CFileWriter::Execute()

Das hat natürlich auch Auswirkungen auf den Export einer einzelnen Scanserie: "Scanserie exportieren". Der Dateinamen wird auch hier automatisch generiert und im Exportverzeichnis gespeichert.

7.2.6 Implementation eines Kontextmenü

Nachdem der Export über das Standard-Menü ordnungsgemäss funktioniert hat, konnte ich noch das Kontextmenü einbauen. Das ganze muss ich in die Klasse `CDBSelectionPanel` einbauen. Dazu muss ich ein Message-Mapping machen, dass mir einen Rechtsklick auf das Panel abfängt. Hierfür dient die Windows-Nachricht `NM_RCLICK`. Zudem muss ich noch die Nachrichten abfangen, die durch das Kontextmenü ausgelöst werden:

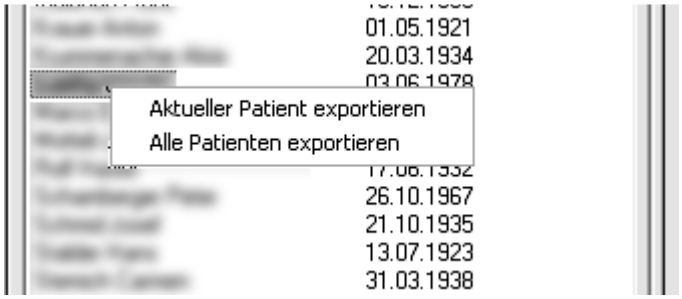


Fig. 7-4 Kontextmenü für den Export

```
ON_NOTIFY(NM_RCLICK, IDC_LIST_PATIENT, OnRclickListPatient)
ON_COMMAND(ID_EXPORT_THIS, OnExportThis)
ON_COMMAND(ID_EXPORT_FULLDB, OnExportFullDB)
```

Code 7-20 CDBSelectionPanel - Message Mapping

Die Methode `CDBSelectionPanel::OnRclickListPatient()` erstellt das Kontextmenü, während die Methoden `CDBSelectionPanel::OnExportThis()` resp. `CDBSelectionPanel::OnExportFullDB()` die Exportfunktion ausführen:

```
void CDBSelectionPanel::OnRclickListPatient(NMHDR* pNMHDR, LRESULT* pResult) {
   NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;
   if (pNMListView->uChanged & LVIF_STATE && pNMListView->uNewState &
   LVIS_SELECTED &&
      m_pDocument)
      m_pDocument->SetPatient(m_lcPatient.GetItemData(pNMListView->
                  iItem));
   *pResult = 0;

   CMenu ctxmenu; ctxmenu.CreatePopupMenu();

   CPoint point; GetCursorPos(&point);

   ctxmenu.AppendMenu(MF_STRING, ID_EXPORT_THIS,
             SLOAD(ID_EXPORT_THIS_SHORT));
   ctxmenu.AppendMenu(MF_STRING, ID_EXPORT_FULLDB,
             SLOAD(ID_EXPORT_FULLDB_SHORT));

   ctxmenu.TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON, point.x, point.y,
               this);
}

void CDBSelectionPanel::OnExportThis() {
   m_pDocument->OnExportThis();
}

void CDBSelectionPanel::OnExportFullDB() {
   m_pDocument->OnExportFullDB();
}
```

Code 7-21 Änderung in der Klasse CDBSelectionPanel

8 Ziel 7 - Save-Funktion

Autor	Adrian Wyssmann
Implementation	Adrian Wyssmann
Ziel	Wird ein einzelnes Fotos gespeichert, so erscheint ein grosser grauer "Bildschirmanteil", der überflüssig und unhandlich ist.
Status	Ziel erreicht
Bemerkung	Es war uns nicht möglich, nur den grauen Bildschirmanteil zu entfernen. Man kann aber nun den gewünschten Bildausschnitt, der gespeichert werden soll markieren und abspeichern.

Tabelle 8-1 Übersicht Ziel 7 - Save-Funktion

8.1 Idee

8.1.1 Speichern der Bilddaten

Durch folgende Zeile wäre es möglich die Bilddaten abzuspeichern:

```
COct3dApp::SaveImageData(m p2DViewProc->m pImageData[m nActiveImage],
strFilename);
```

Code 8-1 C2DView::OnFileSave() - Bilddaten speichern

Auf diese Art und Weise werden nur die ursprünglichen Bilddaten gespeichert. Das bringt zwei Nachteile mit sich:

- Da wir in Abschnit 4.2.2.1 gesehen haben, dass das Bild vertauschte Dimensionen hat, scheint das Bild verzerrt ►Fig. 8-1
- Änderungen am Bild, wie beispielsweise das verändern des Color-Window zum erkennen einer Läsion oder ein vergrösserter Ausschnitt können nicht abgespeichert werden.

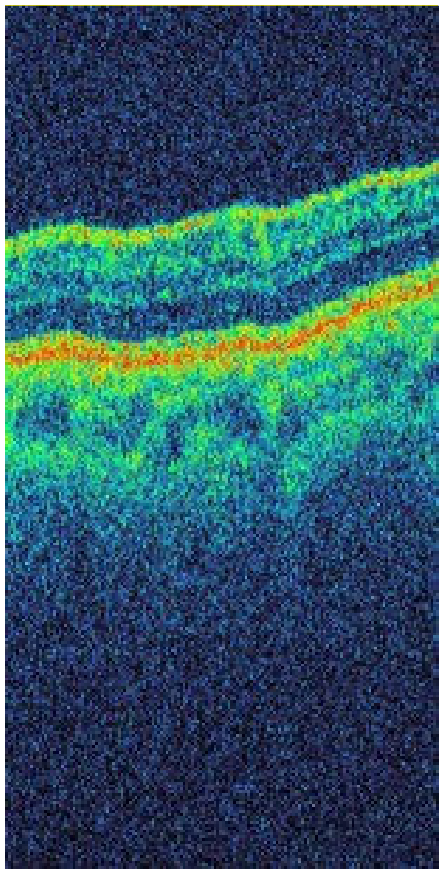


Fig. 8-1 Original-Dimensionen eines OCT-Bildes

8.1.2 Speichern des Render-Window

Die Gruppe OCT3D hat diese Möglichkeit implementiert. Der Inhalt des Render-Window `m_pRenderWindow` wird dabei durch einen `vtkWindowToImageFilter` in `vtkImageData` umgewandelt. Die `vtkImageData` werden dann der Applikation zum speichern übergeben:

```
if (m p2DViewProc == NULL || m p2DViewProc-
>m pImageData == NULL ||
   m p2DViewProc->m nImageData-1 <
m nActiveImage)
   return;

CString strFilename;

vtkWindowToImageFilter *pWindowToImageFilter =
vtkWindowToImageFilter::New();
...
pWindowToImageFilter->SetInput(m_pRenderWindow);
COct3dApp::SaveImageData(pWindowToImageFilter-
>GetOutput(), strFilename);
```

Code 8-2 C2DView::OnFileSave() - Render-Window speichern

Der Vorteil hier liegt klar darin, dass Bilder mit Änderungen (gezoomter Ausschnitt, veränderter Kontrast) so abgespeichert werden können.

Der Nachteil hier ist, dass nicht nur die eigentlichen Bilddaten, sondern auch der Rand zwischen dem eigentlichen Bild und dem Rand des Render-Window als Bilddaten betrachtet werden.

8.2 Implementation

In der ursprünglichen OCT3D Anwendung ist die Funktion implementiert, die es erlaubt, in der 2D-Ansicht ein Rechteck einzuzeichnen, um einen gewünschten Ausschnitt zu zoomen. Dieses Rechteck will ich nun auch dazu verwenden, um einen bestimmten Ausschnitt zu speichern. Dieselbe Methode in den 3D-Ansichten zu implementieren scheint mir da wesentlich schwerer, da durch den IntercatorStyle die Methode der Maus bereits gegeben ist (Zoomen, Verschieben der Kamera, …). Es kann also gut zu einer Doppelbelegung der Maustasten kommen ▶Abschnitt 8.3. Aus diesem Grunde verzichten wir auf diese Möglichkeit und konzentrieren uns darauf, die Methode in der 2D-Ansicht seriös zu implementieren.

8.2.1 Speichern eines Bildausschnittes

In der ursprünglichen OCT3D-Applikation wurde das Bild automatisch auf den Ausschnitt gezoomt, sobald man das Rechteck fertig gezeichnet hat. Ich möchte nun, dass das Rechteck bestehen bleibt. Der Ausschnitt wird erst gezoomt, wenn wir innerhalb des Rechtecks klicken. Klicken wir ausserhalb, wird die Auswahl (Rechteck) aufgehoben. Das dies überhaupt möglich ist müssen alle VTK-Objekte für die Zoom-Box als globale Variablen in der Klasse `C2DView` definiert werden:

```
vtkLookupTable*                    m_pLookupTableOriginal;
vtkImageMapToWindowLevelColors*    m_pImageMapToWindowLevelColors;
vtkTextActor*                      m_pTextActor;
vtkPoints*                         m_pPoints;
vtkActor2D*                        m_pBoxActor;
double m_dColorWindow,             m_bColorLevel;
```

Code 8-3 Neue Klassenvariablen in der Klasse C2DView

Die meisten davon sind ursprünglich innerhalb der Methode `C2DView::CreateVtkPipeline()` lokal definiert worden. Bei der Initialisierung des Observers fügen wir noch 2 weitere `vtkCommands` hinzu:

```
CvtkCallbackData *pICallback = CvtkCallbackData::New();
pICallback->m_pBoxActor = m_pBoxActor;
pICallback->m_pPoints   = m_pPoints;
pICallback->m_pRenderer = m_pRenderer;
pICallback->m_pImageMapToWindowLevelColors = m_pImageMapToWindowLevelColors;
m_pInteractor->AddObserver(vtkCommand::LeftButtonPressEvent, pICallback);
m_pInteractor->AddObserver(vtkCommand::MouseMoveEvent, pICallback);
m_pInteractor->AddObserver(vtkCommand::LeftButtonReleaseEvent, pICallback);
m_pInteractor->AddObserver(vtkCommand::MiddleButtonPressEvent, pICallback);
m_pInteractor->AddObserver(vtkCommand::RightButtonPressEvent, pICallback);
```

Code 8-4 C2DView::CreateVtkPipeline() - Observer initialisieren

Da die Zoom-Box-Funktionalität bereits in der Callback-Klasse implementiert ist, muss ich sie nur noch entsprechend anpassen. Wichtig dabei sind folgende Punke:

1. Ein Klick mit der linken Maustaste ausserhalb des markierten Bereichs bewirkt, dass die Zoom-Box wieder deaktiviert wird.
2. Ein Klick mit der mittleren der der rechten Maustaste bewirkt ebenfalls, dass die Zoom-Box wieder deaktiviert wird.
3. Ein Klick innerhalb des markierten Bereichs bewirkt, dass der Ausschnitt gezoomt wird.
4. Über das Menü **[Datei▶Speichern]** soll der gewählte Ausschnitt gespeichert werden.

Punkt 1 bis 3 sind ziemlich einfach: Ich muss einfach bestimmen, ob sich der Mauszeiger innerhalb oder ausserhalb der des markierten Bereichs, sprich innerhalb oder ausserhalb des `BoxActors` befindet. Es darf dabei keine Rolle spielen, in welcher Richtung die Zoom-Box gezeichnet wurde (Ursprung und Endpunkt). Ich bestimme als erstes die linke Obere und die rechte untere Ecke der Zoom-Box:

```
float p0[4];
float p2[4];
m pPoints->GetPoint(0, p0);
m pPoints->GetPoint(2, p2);

float xMin, xMax, yMin, yMax;
if (p0[0]<p2[0]) {
     xMin = p0[0]; xMax = p2[0];
} else {
   xMin = p2[0]; xMax = p0[0];
}
if (p0[1]<p2[1]) {
   yMin = p0[1]; yMax = p2[1];
} else {
   yMin = p2[1]; yMax = p0[1];
}
```

Code 8-5 Bestimmen der Zoom-Box

Es gilt zu beachten, dass der Ursprung für die Koordinaten in VTK in der unteren linken Ecke liegt. Nun wird überprüft ob die Mauskoordinaten (nX,nY) innerhalb der Zoom-Box liegen:

```
if ((nX > xMin) && (nX < xMax) && (nY > yMin) && (nY < yMax)) {
   //do zooming
}
```

Code 8-6 Ausschnitt zoomen

8.2.2 Speichern

Zum Speichern muss ich der Methode `COct3dApp::SaveImageData()` das zu speichernde Bild als `vtkImageData` übergeben. Bis anhin war das einfach das gesamte Render-Window:

```
 vtkWindowToImageFilter *pWindowToImageFilter = vtkWindowToImageFilter::New();
pWindowToImageFilter->SetInput(pRenderWindow);
```

Code 8-7 Inhalt des Render-Window in vtkImageData umwandeln

Ich möchte ja nun nur einen bestimmten Ausschnitt innerhalb des Render-Window speichern. Ich habe lange gesucht, bis ich eine Klasse gefunden habe, die mir dabei hilft. VTK bietet hierfür eine Möglichkeit: `vtkImageClip`. Mit Hilfe diesem Filter kann ich bestehende Daten (`vtkImageData`) beschneiden.

```
vtkImageClip* pImageClip = vtkImageClip::New();
pImageClip->SetInput(pWindowToImageFilter->GetOutput());

if (m pBoxActor->GetVisibility()) {
   // +/-1 and +/-2 are to remove the box frame
   pImageClip->SetOutputWholeExtent((int)p0[0]+1, (int)p2[0]-2,
                                    (int)p2[1]+1, (int)p0[1]-2,0,0);
   }
```

Code 8-8 Bild beschneiden

In der Methode `C2DView::OnFileSave()` rufe ich die Methode `C2DView::GetScreenContent()` auf. Diese gibt den gewünschten Bildausschnitt zurück:

```
vtkImageData* pImageData = GetScreenContent(m pRenderWindow);
COct3dApp::SaveImageData(pImageData, strFilename);
```

Code 8-9 C2DView::OnFileSave() - Implementation

In `C2DView::GetScreenContent()` muss nur noch folgendes gemacht werden: Solange kein Ausschnitt auf dem Bild markiert ist, wird das gesamte Render-Window zurückgegeben. Ist ein Bereich auf dem Bild markiert, so wird nur dieser Bereich zurückgegeben:

```
vtkImageData* C2DView::GetScreenContent(vtkRenderWindow* pRenderWindow) {
  vtkWindowToImageFilter *pWindowToImageFilter =
                        vtkWindowToImageFilter::New();
  ...
  pWindowToImageFilter->SetInput(pRenderWindow);

  float p0[4]; float p2[4];
  m pPoints->GetPoint(0, p0);
  m pPoints->GetPoint(2, p2);

  vtkImageClip* pImageClip = vtkImageClip::New();
  pImageClip->SetInput(pWindowToImageFilter->GetOutput());

  if (m_pBoxActor->GetVisibility()) {
     pImageClip->SetOutputWholeExtent((int)p0[0]+1, (int)p2[0]-2,
                                      (int)p2[1]+1, (int)p0[1]-2,0,0);
  }

  return pImageClip->GetOutput();
}
```

Code 8-10 C2DView::OnFileSave() - Implementation

8.3 Probleme mit der Interaktion in VTK

Zur Veranschaulichung des Problems, ein Beispiel der Klasse `vtkInteractorStyleTrackbalCamera`:

Mit einer Mausbewegung und gedrückter linker Maustaste lässt sich die die Kamera um die 3D-Ansicht drehen. Eine Mausbewegung in Kombination mit einer gehaltenen **[CTRL]**-, **[ALT]**-Taste o.ä. bewirkt ebenfalls eine Interaktion, falls dafür eine Interaktion definiert ist. Für eine Mausbewegung mit gleichzeitigem Halten der Taste **[CTRL]** ist im *vtkInteractorStyleTrackbal-Camera* keine Interaktion definiert. Da aber für die für Bewegung der Maus mit gedrückter linker Maustaste eine Interaktion definiert ist, bewirkt dies in Kombination mit **[CTRL]** ebenfalls eine Drehung der Kamera um die 3D-Ansicht.

Es besteht nun die Möglichkeit selber Interaktionen zu definieren. Jedem vtkObjekt kann man einen Observer zuweisen. Der Observer fängt bestimmte Kommandos (*vtkCommand*) ab:

```
m pInteractor->AddObserver(vtkCommand::LeftButtonPressEvent, pICallback);
m pInteractor->AddObserver(vtkCommand::MouseMoveEvent, pICallback);
m_pInteractor->AddObserver(vtkCommand::LeftButtonReleaseEvent, pICallback);
```

Code 8-11 Beispiel eines vtkObservers

Das Beispiel in Code 8-11 zeigt, wie man eine Mausbewegung und/oder ein Klicken mit der linken Maustaste auf einem bestimmten Objekt abfangen kann. Man braucht nur noch eine Callback-Klasse - abgeleitet von `vtkCommand` - welche die gewünschten Aktion(en) ausführt ▶2DView.h, Klasse `CvtkCallbackData`. Um der Mausbewegung mit gehaltener **[CTRL]**-Taste eine bestimmte Aktion zuzuweisen, also beispielsweise das einzeichnen des Rechtecks, macht man das wie folgt:

```
case vtkCommand::MouseMoveEvent:
   if ((GetAsyncKeyState(VK CTRL) & 0x8000)) {
      //draw box
   }
```

Code 8-12 CvtkCallbackData - Benutzerdefinierte Interaktion der Mausbewegung

Da diese benutzerdefinierte Interaktion aber via Observer geschieht, wird dennoch auch die Interaktion ausgeführt, die im `vtkInteractorStyleTrackbalCamera` definiert ist. Das bedeutet, dass sich beim Einzeichnen des Rechtecks die Kamera rotiert, was das ganze **unbrauchbar** macht.

Es wäre grundsätzlich möglich einen eigenen InteractorStyle zu implementieren, der dieselben Methoden hat, wie der *vtkInteractorStyleTrackbalCamera*. Das ist aber dank der schlechten [VTKDOC42] schwierig und zeitintensiv.

9 Ziel 7 - Druckfunktion, Diagnoseblatt

Autor	Adrian Wyssmann
Implementation	Adrian Wyssmann
Status	Ziel teilweise erreicht
Ziel	Ein dargestelltes Bild kann per Rechtsklick oder via Menuauswahl gedruckt werden. Dabei kann die Bildschirmauswahl das ganze aktive Fenster oder aber nur einen beliebigen Ausschnitt (=Läsion) getroffen werden. Es gibt eine Druckvorschau, mit der man das Bild auch positionieren bzw. in der Grosse andern kann. Es gibt die Möglichkeit, eine individuelle Vorlage zu kreieren, wo neben dem Bild auch die Patientendaten und Diagnosen eingetippt werden können: Ein eigentliches OCT3D-Diagnoseblatt. Dieses Blatt kann ausgedruckt oder per Mail (z.B. PDF) verschickt werden.
Bemerkung	Das Ausdrucken des ganzen Bildes oder eines Ausschnittes ist möglich. Es werden auch die Benutzerdaten nach eine fix definierten Header ausgedruckt. Allerdings ist es nicht möglich, wie gewünscht, den Ausschnitt zu platzieren.

Tabelle 9-1 Übersicht Ziel 7 - Druckfunktion, Diagnoseblatt

9.1 Idee

*«It's common knowledge that printing is one of the hardest things for properly implementing in your Win32 program. You have probably heard or even experienced how hard printing is with Win32 API. The good news is that we are **using MFC which greatly simplifies this task**. You get print preview, standartized dialogs(print setup, page setup), print job interruption dialog and OS management for free. If this doesn't look much to you just have a look at the MFC source files involved in printing and print preview.»*

http://www.codersource.net/mfc_print_tutorial_1.html

Da die Anwendung OCT3D das MFC-Framework als Grundaufbau benutzt, wollte ich eigentlich die Print-Architektur des MFC-Frameworks gebrauchen, um die Bilder auszudrucken. Die Klasse `CFormView` implementiert bereits Member-Methoden für das Drucken und die Druckvorschau. Folgende Member-Methoden sind implementiert:

Methode	Beschreibung
OnPreparePrinting()	Um Werte in das Druck-Dialogfenster zu schreiben, z.B. Länge des Dokuments
OnBeginPrinting()	Zuweisen von Fonts und GDI Ressourcen
OnPrepareDC()	Anpassen des Device Context für eine bestimmte Seite
OnPrint()	Ausdrucken einer bestimmten Seite
OnEndPrinting()	Zuweisung von GDI Ressourcen aufgeben

Tabelle 9-2 Member-Methoden für das Drucken in der Klasse CFormView

9.1.1 Probleme

Laut [MSDN] implementiert Microsoft Windows Device-unabhängige Displays. In MFC bedeutet das, dass dieselben Aufrufe zum Zeichnen (`OnDraw`) für das Zeichnen auf dem Bildschirm oder einem anderen Device (bsp. Drucker) verantwortlich sind. Konkret heisst dass, ich kann wie folgt einen Text auf den Device Context `pDC` zeichnen:

```
void CProcessView::OnDraw(CDC* pDC) {
   pDC->TextOut(0,0,"2D Bilddaten - Cattin Roger - Raster Lines OD");
}
```

Code 9-1 CProcessView::OnDraw()

Der Text wird im entsprechenden Kontext gezeichnet. Somit funktioniert auch die Druckvorschau, wie in Fig. 9-1 zu sehen ist:

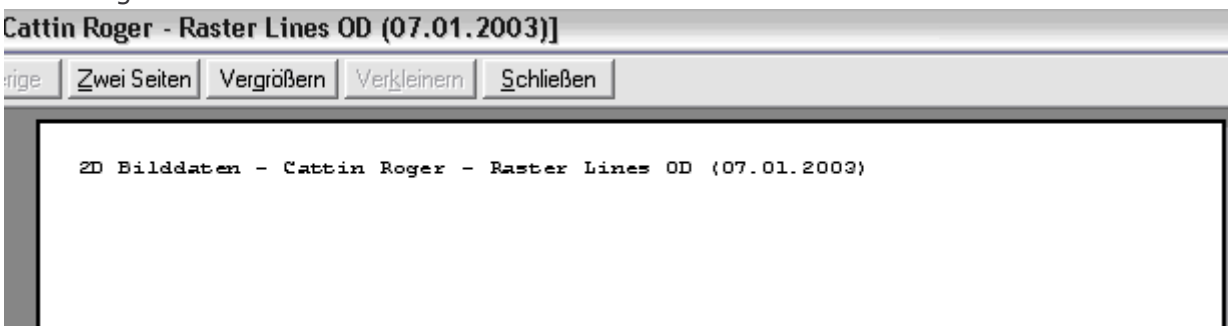


Fig. 9-1 Druckvorschau mit MFC Framework

Child-Window

Beim Initialisieren der Prozess-Ansicht wird für die Ansicht mit der Methode `CreateWindowEx()` ein Fenster erzeugt. Diese Methode mach nicht anderes als ein neues Kind-Fenster zu erzeugen. In diesem Fenster werden die Bilddaten dargestellt (`C2DView, C3DView, …`)

```
void CProcessView::OnInitialUpdate() {
   CFormView::OnInitialUpdate();
     ...
   m_hWndSubView = ::CreateWindowEx(WS_EX_CONTROLPARENT, OCT3DCLASS,
            "OCT3DProcessWnd", WS_CHILD|WS_VISIBLE|WS_TABSTOP,
            rWnd.left, rWnd.top, rWnd.Width(), rWnd.Height(),
            m_hWnd, NULL, AfxGetInstanceHandle(), NULL);
}
```

Code 9-2 CProcessView::OnInitialUpdate() und CProcessView::OnFilePrint()

Dadurch habe ich nun ein Fenster, dass zwar zur Klasse `CProcessView` gehört, aber anscheinend ist der Inhalt dieses Fensters nicht Bestandteil des Kontexts der Prozess-Ansicht. Der Code 9-1 schreibt den Inhalt zwar in den Kontext, der Text wird aber teilweise verdeckt ▶ Fig. 9-2.

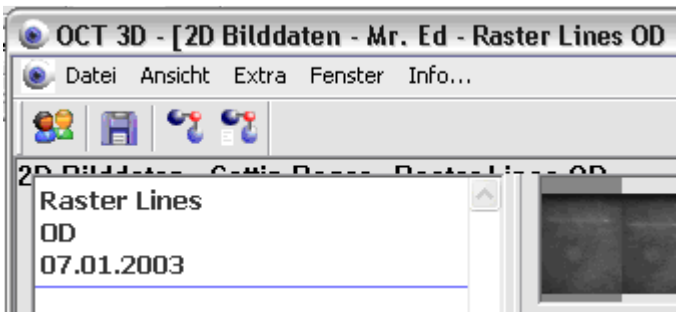


Fig. 9-2 OnDraw() auf den Kontext

Zur besseren Veranschaulichung habe ich in Fig. 9-3 versucht, die einzelnen Kontexte und Fenster zu markieren:

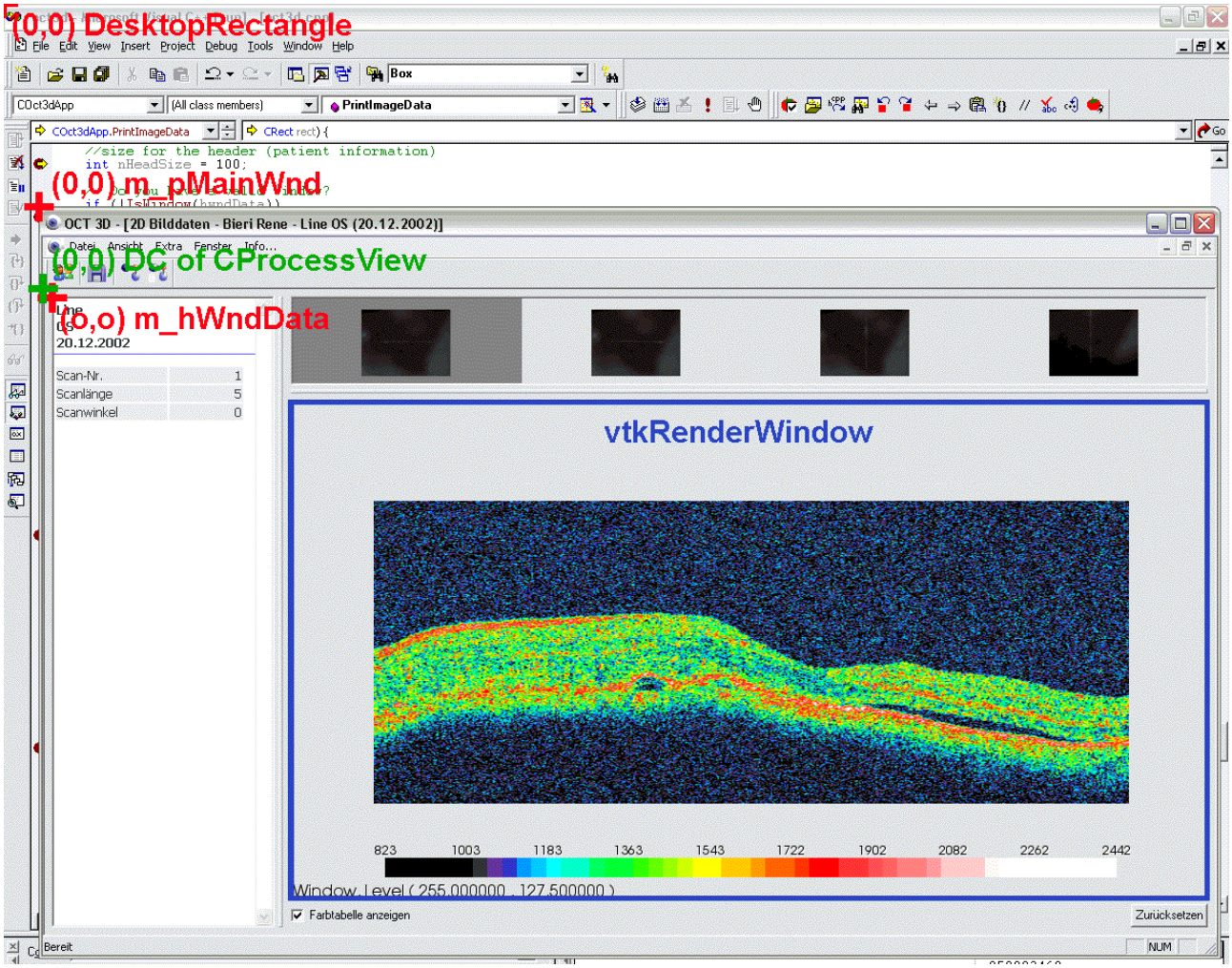


Fig. 9-3 Die verschiedenen Window-Rectangles

Wie in Fig. 9-3 zu sehen ist, möchte ich den Inhalt vom Fenster `m_hWndData` - oder besser gesagt, den Inhalt des vtkRenderWindow - ausdrucken. Da die Print-Architektur des MFC-Frameworks mit Device Kontexten arbeitet (Klasse `CDC`), benötige ich irgendwie den Kontext des Fensters `m_hWndData`.

In der Klasse CDC existiert eine Member-Methode `Attach()` mit der ich ein HWND einem Device Kontext anfügen kann. Dies geschieht folgendermassen:

```
pDC->Attach(::GetWindowDC(m_hWndSubView));
```

Die einzige Möglichkeit das `Attach()` auszuführen sehe ich in einer der overriden Member-Methoden für das Drucken:

```
void CProcessView::OnPrepareDC(CDC *pDC, CPrintInfo* pInfo){
   pDC->Attach(::GetWindowDC(m_hWndSubView));
   CFormView::OnPrepareDC(pDC, pInfo);
}
```

Code 9-3 CProcessView::OnPrepareDC() - Attach(HDC)

Wie in Fig. 9-4 zu sehen ist, scheint das `Attach()` grundsätzlich zu funktionieren:

Fig. 9-4 Fenster nach pDC->Attatch()

Daraus resultiert aber jedesmal ein **"Debug Assertion Failed" in der Datei wingdi.cpp, Zeile 109**. Im schlimmsten Fall stürzt die Applikation sogar ab ▶Fig. 9-5. Das Problem tritt genau so auf, wenn ich das `Attach()` in einer anderen Methode implementiere:

CProcessView::OnBeginPrinting()
CProcessView::OnPrint()
CProcessView::OnDraw()

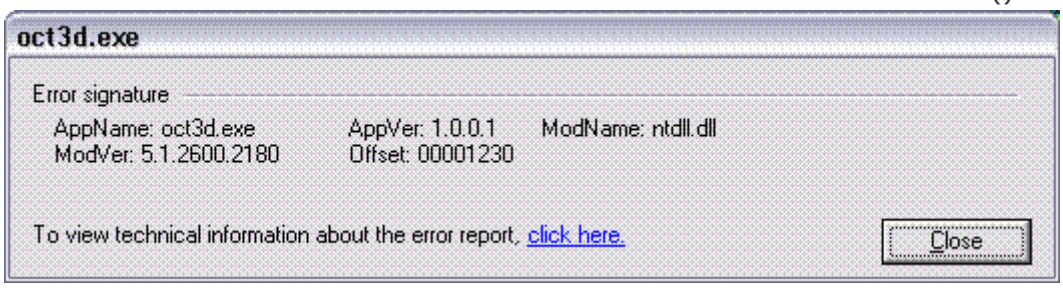


Fig. 9-5 Absturz der Applikation

Ich habe hier ziemlich viel Zeit verloren, eine Ursache, resp. eine Lösung für das Problem zu finden. Schlussendlich fand ich eine Lösung im Artikel [Q186736] der Knowledge-Base von Microsoft:

9.2 Implementation

Der Artikel [Q186736] zeigt eine Technik für das Capturing und Ausdrucken des gesamten Inhalts eines Fensters (einschliesslich Non-Client Bereich). Hierbei wird anhand des Window-Handle von einem bestimmten Fenster, ein Bitmap generiert. Dieses wird dann auf dem Drucker ausgegeben. Nachteile dieser Methode:

- Man muss auf die die Methoden des MFC Frameworks (bsp. PrintPreview) verzichten.
- Ein Header/Footer muss "gezeichnet werden".
- Man hat keine PrintInfo-Struktur (ideal zum einstellen von Rändern, …)

Wenn in der Menü-Bar auf **[Datei▶Drucken]** geklickt wird, generiert das MFC-Framework eine Meldung `ID_FILE_PRINT`. In der Klasse `CProcessView` definieren wir das Message-Mapping für diese ID. Wie bei der Methode für das Speichern eines Bildes, leiten wir auch hier die Nachricht an die Instanz des Fensters `m_hWndSubView` weiter:

```
ON_COMMAND(ID_FILE_PRINT, OnFilePrint)
...
void CProcessView::OnFilePrint() {
   ::SendMessage(m_hWndSubView, WM_COMMAND, MAKEWPARAM(ID_FILE_PRINT, 0),
            NULL);
}
```

Code 9-4 CProcessView::OnInitialUpdate() und CProcessView::OnFilePrint()

Damit die Meldung dort abgefangen und die entsprechenden Aktionen ausgeführt werden, muss auch in den Outputklassen ein Message-Mapping mit der dazugehörigen Methode definiert sein. Details dazu im nächsten Abschnitt.

9.2.1 Implementation in `C2DView`

Da wir für die Druckfunktion Informationen aus dem Prozesskontext brauchen. müssen wir in den Wnd-Klassen der Output-Ansichten eine zusätzliche Variable einführen:

```
CO3DProcessContext*         m_pCtx;
```

Den Wert von `m_pCtx` weisen wir bei der Methode `C2DView::InitWnd()` zu. Somit können wir auch in der Methode `C2DView::OnFilePrint()` darauf zugreifen.

```
int C2DView::InitWnd(CO3DProcessContext *pCtx) {
   CWaitCursor wait;

   m_pCtx = pCtx;
...
```

Code 9-5 C2DView::InitWnd - Prozess-Kontext

Wie bereits im vorherigen Abschnitt erwähnt, muss die Nachricht, welche die Klasse `CProcessView` an die entsprechende Output-Klasse schickt, abgefangen und die dazugehörige Methode aufgerufen werden:

```
ON_COMMAND(ID_FILE_PRINT, OnFilePrint)
```

Die Methode `C2DView::OnFilePrint()` holt den Inhalt des Render-Window. Hierfür verwenden wir, einen `vtkWindowToImageFilter`:

```
void C2DView::OnFilePrint() {
   vtkWindowToImageFilter *pWindowToImageFilter =
                                          vtkWindowToImageFilter::New();
   pWindowToImageFilter->AddObserver(vtkCommand::ErrorEvent, m_p2DViewProc->
                                     m_pO3DvtkCallbackMsg);
   pWindowToImageFilter->AddObserver(vtkCommand::WarningEvent,
                                     m_p2DViewProc->m_pO3DvtkCallbackMsg);
   pWindowToImageFilter->SetInput(m_pRenderWindow);

   COct3dApp::PrintImageData(m_hWndData, m_pCtx,
           (CScanData*)m_p2DViewProc->m_pDataIn->pArrScanData[m_nActiveImage],
           m_p2DViewProc->GetName());
   pWindowToImageFilter->Delete();
}
```

Code 9-6 C2DView::OnFilePrint()

In den 3D-Output-Klassen übergeben wir einfach die Scandaten des 1. Bildes, also

```
(CScanData*)m_p2DViewProc->m_pDataIn->pArrScanData[0]
```

Das Drucken selber wird durch den Aufruf der Methode `COct3dApp::PrintImageData()` ausgeführt. Dieser Methode müssen wir noch die Scan- und Patientendaten mitgeben.

Drucken eines Ausschnitts

Damit wir auch einen Ausschnitt ausdrucken können, verwende ich die Zoom-Box. Nach den Erweiterungen, wie ich sie in Abschnitt 8.2 beschrieben, implementiert habe, steht dem nichts im Weg. Aufgrund der Implementation der von `COct3dApp::PrintImageData()` (▶Abschnitt 9.2.2) muss ich eine `RECT`-Struktur verwenden, um der Methode die Grösse des Ausschnitts zu übergeben. Die Methode wird entsprechend angepasst.

Die Dimensionen des Rechtecks bestimme ich wie beim Speichern des Bildes anhand der Koordinaten der Zoombox. Ich muss dabei beachten, dass das die VTK im Gegensatz zu Windows, den Koordinaten-Ursprung in der linken unteren Ecke definiert hat.

```
void C2DView::OnFilePrint() {
   vtkWindowToImageFilter *pWindowToImageFilter =
                                          vtkWindowToImageFilter::New();
   ...
   pWindowToImageFilter->SetInput(m pRenderWindow);

   CRect rect;
   rect.top = 0; rect.left = 0; rect.bottom = 0; rect.right = 0;

   if (m pBoxActor->GetVisibility()) {
      float p0[4]; float p2[4];
      m pPoints->GetPoint(0, p0);
      m pPoints->GetPoint(2, p2);

      LONG xMin, xMax, yMin, yMax;
      if (p0[0]<p2[0]) {
         xMin = (LONG) p0[0];
         xMax = (LONG) p2[0];
      } else {
         xMin = (LONG) p2[0];
         xMax = (LONG) p0[0];
      }

      if (p0[1]<p2[1]) {
         yMin = (LONG) p0[1];
         yMax = (LONG) p2[1];
      } else {
         yMin = (LONG) p2[1];
         yMax = (LONG) p0[1];
      }

      ::GetWindowRect(m hWndData, &rect);
      int iHeight = rect.bottom-rect.top;
      rect.bottom = iHeight-yMin+rect.top;
      rect.top    = iHeight-yMax+rect.top;
      rect.right  = rect.left + xMax;
      rect.left += xMin;
   }
   COct3dApp::PrintImageData(m hWndData, m pCtx,
                             (CScanData*)m_p2DViewProc->m_pDataIn->
                                         pArrScanData[m nActiveImage],
                             m p2DViewProc->GetName(), rect);
   pWindowToImageFilter->Delete();
}
```

Code 9-7 C2DView::OnFilePrint()

9.2.2 Implementation `COct3dApp::PrintImageData()`

Wie beim Speichern eines Bildes, ist die Implementation der Druckfunktion in der Klasse `COct3dApp` gemacht. Folgende Helper-Methoden habe ich aus dem Artikel [Q186736] übernommen:

Methode	Beschreibung
HBITMAP Create24BPPDIBSection()	Erstellt ein 24-Bit-Pixel Bitmap
HDC GetPrinterDC()	Erstellt einen Drucker Kontext [Q186736]
void CreateHeader()	zeichnet den Header in den Drucker Kontext. Details zum Header siehe Abschnitt 9.2.3

Tabelle 9-3 Helper-Methoden für die Druckfunktion

In der Methode `COct3dApp::PrintImageData()` muss ich nun anhand des übergebenen Window-Handle und des RECT den Ausdruck auf Papier machen. Das Vorgehen ist etwa folgendermassen:

5. Druckerdialog anzeigen (Anzahl Kopien, Drucker, …)
6. Leeres Bitmap erstellen
7. Den definierten Bereich des Fensters in den Speicher schreiben
8. Den Bildausschnitt in den Speicherbereich kopieren
9. Das Bitmap auf den Drucker Kontext schreiben
10. Den Header (Patientendaten) auf den Drucker Kontext schreiben
11. Ausdruck starten

9.2.2.1 Unterschiedliche Auflösung Drucker ↔ Bildschirm

Ich führe folgende Variablen ein, um auf dem Patientenblatt Ränder zu definieren:

```
float fMarginLeft  = 2.0; //cm
float fMarginRight = 1.0; //cm
float fMarginTop   = 1.5; //cm
float fHeadSize    = 4.0; //cm
```

Code 9-8 COct3dApp::PrintImageData() -

Da der der Drucker und der Bildschirm unterschiedliche Auflösung haben. Ich muss deshalb die Möglichkeit haben, die Werte auf den entsprechenden Kontext umzurechnen. Damit habe ich auch die Möglichkeit ein Bitmap sinvoll auf dem Drucker auszugeben. Ein Bitmap von 100x100 Pixeln wird auf einem Drucker von 600dpi sehr klein ausgedruckt. Auf einem Bildschirm mit normalerweise 96dpi, wirkt es wesentlich Grösser. Deshalb muss das Bitmap gestreckt werden ▶Abschnitt 9.2.2.2. Ich führe weitere Variablen ein:

```
int iDpiPrinter = GetDeviceCaps(hdcPrinter, LOGPIXELSX); //dpi of printer
int iDpiScreen  = GetDeviceCaps(hdcWindow, LOGPIXELSX);  //dpi of screen
float fRatioPrinter = iDpiPrinter/fInchToCM;             //dpcm of printer
float fRatioScreen  = iDpiScreen/fInchToCM;              //dpcm of screen
int iWidthToStretch;
int iHeightToStretch;
```

Code 9-9 COct3dApp::PrintImageData() -

Nun kann ich die Grösse des Drucker Kontext bestimmen: Von der ursprünglichen Grösse ziehe ich die entsprechenden Ränder ab:

```
// Get the resolution of the printer device minus margins.
iPrtWidth  = GetDeviceCaps(hdcPrinter, HORZRES)-
                          (fRatioPrinter*(fMarginLeft+fMarginRight));
iPrtHeight = GetDeviceCaps(hdcPrinter, VERTRES)-2*(fRatioPrinter*fMarginTop);
```

Code 9-10 COct3dApp::PrintImageData() - Grösse des Drucker Kontext bestimmen

9.2.2.2 Bestimmen des zu druckenden Bereichs

Der Unterschied zum Beispiel in [Q186736], muss ich noch den korrekten Bereich auswählen. Wenn für die übergebenen RECT-Struktur `rect`, keine Dimensionen definiert sind (`rect.bottom` und `rect.right` sind 0), verwende ich wie in [Q186736] den Rectangle, welcher mir die Funktion `::GetWindowRect(hwndData, &rc)` zurückliefert. Sonst muss ich anhand von `rect`, den korrekten Bereich innerhalb von `rc` auswählen.

```
::GetWindowRect(hwndData, &rc);
iWndWidth  = rc.right-rc.left;
iWndHeight = rc.bottom-rc.top;

// Get the rectangle bounding the window.
if ((rect.bottom == 0) && (rect.right==0)) {
     iXsrc = 0;
     iYsrc = 0;
} else {
     iXsrc = rect.left-rc.left;
     iYsrc = rect.top-rc.top;
     rc = rect;
}
```

Code 9-11 COct3dApp::PrintImageData() - bestimmen des zu druckenden Bereichs

Bevor ich das Bitmap auf dem Drucker ausgeben kann, muss ich sicherstellen, dass das Bild vollständig auf den Druckerkontext passt und nicht abgeschnitten wird. Hierfür bestimme ich einen **Scalingfactor**.

```
fScalingFactor = 1000;
//image width > printer width
fTemp = ((iPrtWidth/fRatioPrinter)/(rc.right/fRatioScreen));
if (fTemp < fScalingFactor) fScalingFactor = fTemp;

//image height > printer height
fTemp = ((iPrtHeight/fRatioPrinter)/(rc.bottom/fRatioScreen));
if(fTemp < fScalingFactor) fScalingFactor = fTemp;
```

Code 9-12 COct3dApp::PrintImageData() - bestimmen des ScalingFactors

Ist der Scalingfactor grosser als 1, ist das Bild auf jeden Fall kleiner als der Druckbereich. Das Bild wird nicht gestreckt:

```
iWidthToStretch  = fRatioPrinter/fRatioScreen*rc.right;
iHeightToStretch = fRatioPrinter/fRatioScreen*rc.bottom;
```

Code 9-13 COct3dApp::PrintImageData() - Bild wird nicht gestreckt

Ist der Scalingfactor kleiner als 1 ist mindesten eine der beiden Seite des Bildes grösser als der Druckbereich. Es wird so verkleinert, dass es vollständig auf den Druckbereich passt:

```
iWidthToStretch  = fRatioPrinter/fRatioScreen*rc.right*fScalingFactor;
iiHeightToStretch = fRatioPrinter/fRatioScreen*rc.bottom*fScalingFactor;
```

Code 9-14 COct3dApp::PrintImageData() - Bild wird gestrekct

Das Kopieren des Bildausschnitts in den Speicherbereich (▶Punkt 4) wird mit der Funktion `BitBlt()` gemacht:

```
BitBlt(hdcMemory, 0, 0,  rc.right, rc.bottom, hdcWindow,
       iXsrc, iYsrc, SRCCOPY);
```

Das Ausgeben des Bitmaps auf den Drucker Kontext (▶Punkt 5) wird mit der Funktion `StretchDIBits()` gemacht. Ich muss dabei der Fuktion den Offset für den linken Rand und den oberen Rand plus Headergrösse mitgeben:

```
StretchDIBits(hdcPrinter, iMarginLeft,
              (fMarginTop+fHeadSize)*fRatioPrinter, iPrtWidth, iPrtHeight,
              rc.left, rc.top, rc.right, rc.bottom,
              ds.dsBm.bmBits,
              (LPBITMAPINFO)&ds.dsBmih,
              DIB RGB COLORS,
              SRCCOPY);
```

Code 9-15 COct3dApp::PrintImageData() - StretchDIBits()

Nun muss ich noch den Header ausgeben. Hierfür habe ich eine eigene Methode definiert: `COct3dApp::CreateHeader()`. Ihr muss ich unter anderem die Grösse des Headers und die Breite des verfügbaren Drucker Kontext angeben.

9.2.3 Der Header

Aus dem Header sollte ersichtlich sein, um welchen Patienten, welche Scanserie des Patienten und um welches Bild der Scanserie es sich handelt. Ich stelle mir vor, dass er ungefähr folgendermassen aussehen soll:

```
OCT 3D - Patientenblatt                                          Dateiname.o3d
-------------------------------------------------------------------------------
Patientenname: Max Muster (11.11.1911)
Scanart:       Raster Lines
Seite:         OD
Ansicht:       2D Ansicht
Besuchsdatum:  11.11.2002
-------------------------------------------------------------------------------
Scan-Nr.: 1, Scanlänge: 5, Scanhöhe: 5, Scanwinkel: 90°
-------------------------------------------------------------------------------
```

Code 9-16 Header mit Patientendaten

Den Header "zeichnen" wir auf den Printer Kontext. Da wir eigentlich nur Text zeichnen, reichen die Funktionen `TextOut()` und `GetTextExtentPoint32()`.

9.2.4 Implementation in 3D-Ansichten

Wie ich bereits zu Beginn von Abschnitt 8.2 erklärt habe, ist die Verwendung einer Zoom-Box in der 3D-Ansichten schwer zu implementieren. Da wir keine Zoom-Box haben, müssen wir der Methode `COct3dApp::PrintImageData()` einfach als zu druckenden Bereich, das gesamte Fenster `m_hWndData` übergeben:

```
void C3DCutView::OnFilePrint() {
...
   CRect rect;
   ::GetWindowRect(m hWndData, &rect);
   COct3dApp::PrintImageData(m hWndData, m pCtx,
         (CScanData*)m p3DCutViewProc->m pDataIn->pArrScanData[0],
         m p3DCutViewProc->GetName(), rect);
...
}
```

Code 9-17 OnFilePrint() in 3D-Ansichten

10 Erweiterung der möglichen Eingabedaten auf Radial-Scans

Autor	Markus Fawer
Implementation	Markus Fawer
Status	Ziel nicht erreicht
Bemerkung	Aus zeitlichen Gründen war es uns allerdings nicht mehr möglich die Implementation innerhalb der Diplomarbeit zu realisieren. Die Lösung sollte mit dem berechnen von neuen Bildebenen, die mithilfe der linearen Interpolation erstellt werden realisierbar sein.

Tabelle 10-1 Übersicht Ziel 8 - Erweiterung der möglichen Eingabedaten auf Radial-Scans

10.1 Idee

Die Idee wäre eine Erweiterung der 3-Dimensionalen Ansichten, die erlauben würde, um eine zusätzliche Funktion zu erweitern. Es geht darum die Radial Scans als Ausgangsdaten zu verwenden. Radial Scans sind Aufnahmen, welche Radial um ein Zentrum gemacht wurden ▶Fig. 10-1.

Dazu müssten die Werte welche mit den Aufnahmen linear interpoliert[1] werden, um die richtigen Werte zu erhalten, die im Volumen benutzt werden könnten.

Die lineare Interpolationen würde für jeden Punkt den man im Volumen rechnen möchte durchgeführt werden müssen.

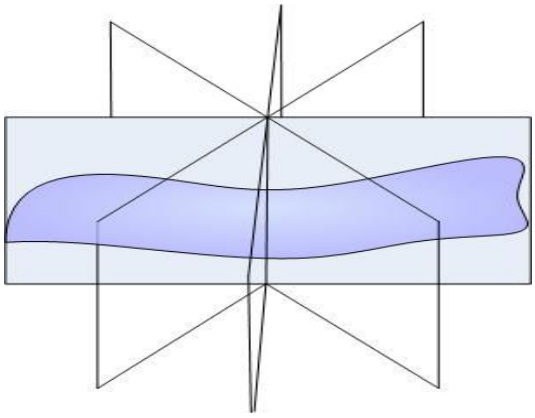


Fig. 10-1 Lage der Aufnahmen eines Radial Scans

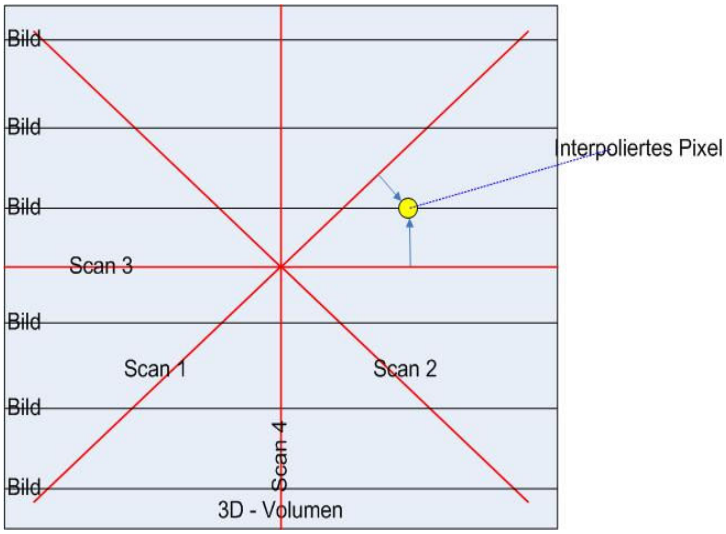


Fig. 10-2 Interpolation der Pixelwerte auf die Bildebenen

Die Interpolation müsste für alle Pixel der Scans auf die Bildebenen erfolgen. Die Bildebenen würden dann, analog zu den Raster Scan Bildern zu dem Volumen zusammengesetzt. L

[1] Lineare Interpolation: Errechnen eines Mittelwertes nach, Wert = Distanz * (Value1) + (1 - Distanz) *(Value2)

11 Schlussfolgerungen

Wir hatten zwar auf der einen Seite eine gute Dokumentation [OCT3D], konnten aber viele Details nicht nachvollziehen, da der Code selber wenig erklärenden Kommentar enthielt. Erschwerend für uns kam hinzu, dass wir von den engesetzten Techniken wie **Microsoft Foundation Classes (MFC)**, keine Ahnung hatten. Auch das eingesetzte **Visual Toolkit VTK**, war für uns relativ neu. Wir hatten zwar während der Semesterarbeit den ersten Kontakt damit, doch das reicht bei einem so umfassenden Toolkit bei weitem nicht. Der Umfang von VTK ist gewaltig und die Übersicht über alle Möglichkeiten zu haben ist schier unmöglich. Trotz verschiedner guter Quellen, die uns zur Verfügung standen, war es bei Detailfragen ziemlich frustrierend. Die Dokumentation [VTKDOC42] für die einzelnen VTK-Objekte ist sehr schlecht. Man ist für viele Fragen und Probleme auf andere VTK-Benutzer angewiesen ▶[VTKUSER].

11.1 Effizienz steigern

Wer mit VTK arbeiten will, muss berücksichtigen, dass die Dokumentation sehr schlecht ist. Insbesondere die Beschreibung der Methoden der einzelnen VTK-Objekte sind sehr schlecht beschrieben. Für VTK-Newbie's ist es ganz klar ein Vorteil, wenn er sich zuerst anhand einfachen Beispielen mit VTK beschäftigt, bevor er sich an komplexere Probleme wagt. Das gibt ihm auch das Verständnis, wie VTK arbeitet.

Wer Änderungen an Anwendungen macht, ist gut damit beraten, nebst dem Grundwissen der jeweiligen Programmiersprache, auch die grundlegenden Techniken zu kennen. Das reicht allerdings nicht. Es existieren mittlerweile viele Frameworks, die das Ziel haben, dem Programmierer ein Werkzeug zu geben, mit dem er Windows-GUI-Anwendungen möglichst einfach implementieren kann, ohne die detaillierten Kenntnisse des darunter liegenden Betriebssystems zu kennen. Das **Microsoft Foundation Classes (MFC)** beispielsweise ist ein Framework von Microsoft, das so umfassend wie komplex ist. Es erleichtert einem aber die Implementation vieler Ideen. Doch wie bei VTK, wäre es auch hier ganz klar ein Vorteil, wenn man schon einmal damit in Berührung gekommen ist. Die Dokumentation der **Microsoft Foundation Classes (MFC)** in [MSDN] ist aber massiv besser als die Dokumentation für VTK.

11.2 Arbeitsaufwand einschätzen

In der Aufgabe, wie wir sie gehabt haben, ist es schwierig den Arbeitsaufwand für die einzelnen Schritte richtig einzuschätzen. Wenn man den Quellcode zuvor nicht kennt und bsp. nicht welches Framework der jeweilige Autor der Applikation verwendet hat, kann man schwierig abschätzen, was für einen Zeitaufwand für die Einarbeitung notwendig ist.

11.3 Fazit

Den Aufwand der wir in unsere Diplomarbeit gesteckt haben, war sehr gross. Trotzdem sind wir ein wenig enttäuscht vom Endergebnis. Wir erhofften uns in den 8 Wochen, die uns zur Verfügung standen für die jeweiligen Ziele eine optimale Lösung zu finden. Das gelang uns nicht überall. Bei einigen Zielen ist es an den oben genannten Problemen gescheitert, bei anderen Zielen sind es einfach kleiner Details, die uns persönlich stören. Beispielsweise wäre es schön für die Druckfunktion eine Druckvorschau zu haben. Aus zeitlichen Gründen mussten wir aber einen Schlussstrich ziehen.

Hinzu kam auch noch der, aus unserer Sicht, hohe Massstab, den die Diplomarbeit OCT3D setzt. Dennoch sind wir überzeugt, durch unsere Arbeit eine Verbesserung der Anwendung OCT3D erzielt zu haben. Wir haben versucht unsere Ziele so gut wie möglich zu erreichen, was uns im grossen und ganzen auch gelang. Zudem haben wir in unserer Diplomarbeit sehr viel gelernt.

A.1 Quellenangaben

Quelle	Detaillierte Quellangaben
[EDK]	**OCT3D Extension Development Kit**; Version 1.01, 10.12.2003; von Yvonne Mettler, Ulrich Sigrist; © 2003 Yvonne Mettler, Ulrich Sigrist, Hochschule für Technik und Informatik Biel
[HEROLD]	**C Kompakt Referenz**; von Helmut Herold; ISBN 3-8273-1984-6; © 2002 Addison-Wesley Verlag
[KITWARE]	**Website of Kitware Inc.;** www.kitware.com
[MSDN]	**Microsoft Developer Network**; http://www.msdn.microsoft.com
[OCT3D]	**OCT3D Systemhandbuch**; Version 1.1, 11.12.2003; von Yvonne Mettler, Ulrich Sigrist; © 2003 Yvonne Mettler, Ulrich Sigrist, Hochschule für Technik und Informatik Biel
[PITAS]	**Digital Image Processing Algorithms**; von Ionnais Pitas; ISBN 0-13-145814-0; © 1993 Prentice Hall International (UK) Ltd
[Q186736]	**Knowledge-Base Artikel Q186736**; http://support.microsoft.com/kb/q186736/
[STROU]	**Die C++ Programmiersprache** - 4., aktualisierte Auflage; von Bjarne Stroustrup; ISBN 3-8273-1660-X; © 2000 Addison-Wesley Verlag
[VTK]	**The Visualization Toolkit, 3rd Edition**; von Willi Schroeder, Ken Martin, Bill Lorsen; ISBN 1-930934-07-6; © 2002 Kitware, Inc (http://www.kitware.com)
[VTKDOC42]	**VTK 4.2.1 Documentation;** Revision 1.1495, 2003/02/18 03:45:04; http://www.vtk.org/doc/release/4.2/html/
[VTKUG]	**The VTK User's Guide VTK 4.2**; von Kitware, Inc; ISBN 1-930934-08-4; © 2003 Kitware, Inc, (http://www.kitware.com)
[VTKUSERS]	**VTK-User Forum;** http://public.kitware.com/mailman/listinfo/vtkusers

Tabelle 11-1 Quellenangaben

A.2 Tabellenverzeichnis

A.3 Codeverzeichnis

A.4 Bildverzeichnis

A.5 Stichwortverzeichnis

E

F

G

I

K

L

M

N

O

P

Q

S

T

W

Z