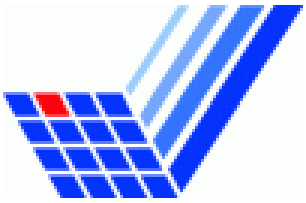
PG498 - 3DLaserNAV - Endbericht

Lehrstuhl für Graphische Systeme
Fachbereich Informatik
Universität Dortmund



Teilnehmer: Jan Bahlmann
Niko Brychcy
Dennis Endt
Leschek Fitzek
Andreas Grob
Matthias Heine
Martin Hüfner
Tim Kroener
Karsten Mark
Marcel Preukschat
Martin Rudel
Sheng Wang

Betreuer: Prof. Dr. Heinrich Müller
Dipl. Inform. Frank Weichert

Semester: Wintersemester 2006/2007
Sommersemester 2007

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
ATM	Asynchronous Transfer Mode
DCT	Diskrete Cosinus Transformation
DDI	Device Driver Interface
DDK	Driver Development Kit
DFT	Diskrete Fourier Transformation
DPCM	Differential-Puls-Code-Modulation
DX8	DirectX8
DX9	DirectX9
DX10	DirectX10
EDO	Extra Device Object
EOB	End of block
ES	Elementary Stream
FFTW	Fastest Fourier Transform in the West
FiDO	Filter Device Object
GDI	Graphics Device Interface
GoP	Group of Pictures
GUI	Graphical User Interface
HID	Human Interface Device
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IOCTL	Input/Output control
IP	Internet Protocol
IRP	Interrupt Request Package
LUT	LookUpTable
MFC	Microsoft Foundation Class
MPEG	Motion Picture Expert Group
MIL	Matrox Imaging Library
OGL	OpenGL
PG	Projektgruppe
PnP	Plug and Play
PS	Program Stream
RBF	Radiale Basisfunktion
RMS	Root-Mean-Square
RTP	Real Time Transport Protocol
RTSP	Real Time Streaming Protocol

SDK	Software Development Kit
SGML	Structured Generalized Markup Language
SNR	Signal-to-Noise-Ratio
TCP	Transmission Control Protocol
TS	Transport Stream
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VLC	VideoLAN Client
VLS	VideoLAN Server
VM	Virtual Machine
VR	Virtuelle Realität
WDM	Windows Driver Model

Notationen

`MBufClass`	Pseudocode	Pseudocode wird in dieser `Schriftart` gedruckt
a	Skalare	Ein Skalar wird mit Kleinbuchstaben dargestellt.
$**$	Faltung	Die Faltung wird in Formeln durch $**$ dargestellt.
$\mathbf{M}$	Matrix	Matrizen sind durch fett gedruckte Großbuchstaben gekennzeichnet.
$\mathbf{x}$	Vektor	Vektoren werden durch fett gedruckte Kleinbuchstaben signalisiert. Die Komponenten von $\mathbf{x}$ werden mit $x_1, ..., x_n$ bezeichnet
$\mathbf{x} \bullet \mathbf{y}$	Skalarprodukt	Der $\bullet$ signalisiert das Skalarprodukt zweier Vektoren
$\mathbf{x}^2$	Skalarprodukt	Das Skalarprodukt eines Vektors mit sich selbst wird durch das Quadrat ausgedrückt

1 Einleitung und Systemüberblick

1.1 Anforderung, Ziele und Motivation

In Zusammenarbeit mit dem Lehrstuhl für Mechanische Verfahrenstechnik (Fachbereich Bio- und Chemieingenieurwesen) werden in einem aktuellen Forschungsprojekt die Ursachen der Zerfallsprozesse von Flüssigkeitslamellen bei Hohlkegeldüsen im Hinblick auf praktische Anwendungsmöglichkeiten (z.B. Motoren, medizinische Sprühgeräte, Lackspritzanlagen) untersucht. Ein Teilaspekt dieses Projektes, die exakte dreidimensionale Visualisierung der Flüssigkeit in einer augmentierten Realität, ist das Thema der aktuellen Projektgruppe 484 am Lehrstuhl Informatik VII. Augmentierte Wirklichkeit bedeutet in diesem Zusammenhang, dass Videodaten (reale Daten) der ausströmenden Flüssigkeit mit virtuellen, rechnergenerierten Daten (z.B. Stromlinien) in einer „Welt“ dargestellt werden und mit ihnen interagiert wird.

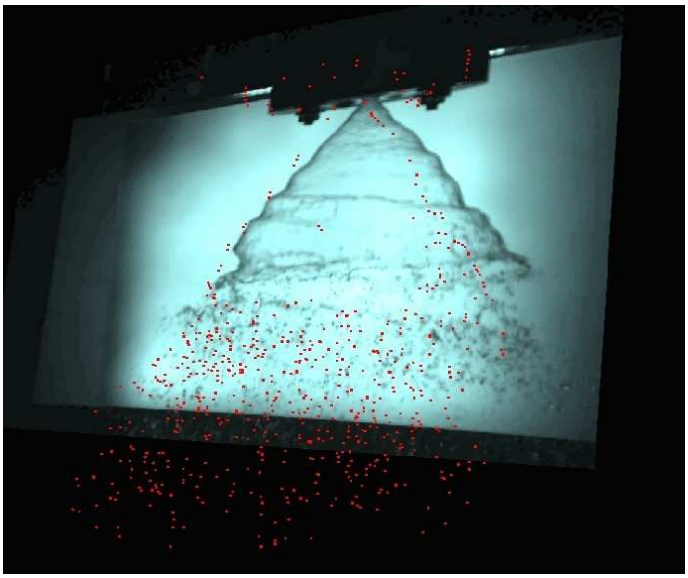


Abbildung 1: 3D-Sprühkegel

Obwohl es sich bei dem aufgezeigten Projekt um eine explizite Anwendung handelt, reflektiert sie doch die allgemeine Problematik, dass speziell die Visualisierung komplexer, in der Realität dreidimensionaler Versuchsszenarien, eine Kombination von immersiver, dreidimensionaler virtueller

Realität (VR) mit einer multimodalen Mensch-Maschine-Interaktion erfordert. Hier setzt die Aufgabe der Projektgruppe an. Innerhalb dieser soll eine VR-Interaktionsschnittstelle entworfen und realisiert werden, die es auch mehreren Benutzenden gleichzeitig ermöglicht, unter Verwendung handelsüblicher Laserpointer mit einer 3D-Szene zu interagieren.

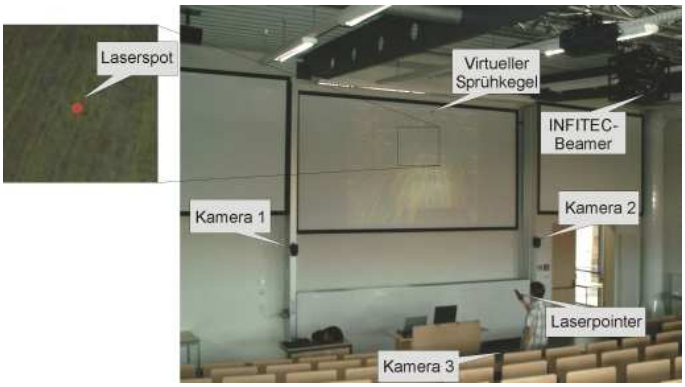


Abbildung 2: Darstellung des Informatik-Hörsaals mit seiner stereographischen Projektionswand und den zugehörigen Multimediakomponenten

Der Hörsaal innerhalb des neuen Informatikgebäudes an der OH-14 ermöglicht mittels einer modernen INFITEC-basierten stereographischen Projektion, die Integration der PG-Umsetzung in eine professionelle Multimedia-Umgebung (Abbildung 1). Abbildung 86 zeigt das Multimediaequipment schematisch und vermittelt eine mögliche Herangehensweise zur Realisierung der Aufgabenstellung. Mittels der beiden Domkameras eins und zwei kann der Laserpointer getrackt und seine Position im (3D-) Raum bestimmt werden. Ergänzend erlaubt Kamera drei die Positionsdetektion des Laserpunktes auf der Projektionsfläche. In Verbindung mit den Informationen zu den räumlichen Modalitäten kann der reale Laserstrahl L1 als virtueller Suchstrahl L2 in die visualisierte stereographische Szene „verlängert“ werden - zur Interaktion im virtuellen 3D-Raum.

1.2 Überblick über das Gesamtsystem

1. Akquisition: Bei einer Umsetzung innerhalb des Hörsaals kann auf die Videostreams der drei hochauflösenden Farb-Domkameras über die Netzwerk Remote Streamingsoftware „PView“ zugegriffen werden - zu realisieren über eine entsprechende Protokollanbindung.

2. Datenaufbereitung: Trotz der initial beschriebenen hochwertigen Aufnahmemodalitäten kann es zu geometrischen Verzerrungen und Diskontinuitäten (z.B. Rauschen, Reflektionen) innerhalb einzelner Sequenzen kommen. Diese Artefakte sind durch Methoden digitaler Bildverbesserung zu beheben. Zusätzlich sind Beleuchtungsveränderungen innerhalb der Videostreams zu kompensieren.
   Ergänzend kommt es bedingt durch die Kameraoptik zu Verzerrungen innerhalb der Aufnahmen, wodurch diese nicht direkt verwendbar sind. Unter Verwendung von eingebrachten Markierungsgittern kann zunächst eine Kalibrierung der Kameras erfolgen und basierend auf diesen Daten eine Entzerrung (Dewarping) der Streamingdaten.

3. Segmentierung und Tracking: Um die Segmentierung des Laserpunktes auch in zeitkritischen Situationen effizient innerhalb des Videostreams von Domkamera drei zu ermöglichen, muss der Laserpunkt automatisch mit geeigneten Algorithmen digitaler Bildverarbeitung vom Hintergrund getrennt werden, wobei auch Inhomogenitäten bzgl. Farbe und Kontrast zu berücksichtigen sind. Neben geometrischen und Farbassoziierten Merkmalen sind hierzu auch topologische Informationen über den vorhergehenden Bewegungspfad des Laserpunktes zu berücksichtigen. Ein Teilaspekt bei der Rekonstruktion des Bewegungspfades ist auch die Kompensierung des Handtremors, denkbar ist hierbei auch eine Spline-Approximation.
   Zur topologischen Positionsdetektion des Laserpointers muss dieser in den korrelierenden Aufnahmen der Kamera eins und zwei detektiert werden. Bei diesem Teilaspekt kommen primär die initial beschriebenen Segmentierungsansätze zum Tragen. Um sowohl das Tracking des Laserpointers robust zu halten, als auch Probleme durch Überdeckungen zu kompensieren, können modellbasierte Trackingverfahren, wie z.B. probabilistische Ansätze (Kalman-Filter oder Conditional Density Propagation), verwendet werden.

4. Rekonstruktion der räumlichen Position des Laserpointers: Für den im vorhergehenden Schritt segmentierten Laserpointer ist nun die tatsächliche räumliche Lage zu bestimmen. Dieses ist möglich, da zu jedem Zeitpunkt zwei Aufnahmen unter einem vorgegebenen Winkel vorliegen und die korrelierenden Aufnahmen in einem geometrischen Zusammenhang (epipolare Geometrie) stehen. Aus den dreidimensionalen Informationen kann die Lokalisation im Raum bestimmt werden und in Kombination mit der Position des Laserpunktes auf der Projektionswand, die Orientierung des Laserstrahls.

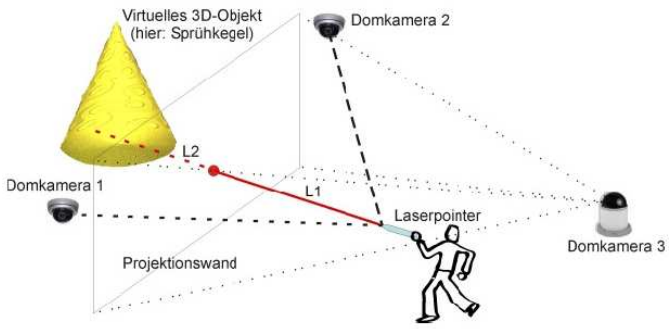


Abbildung 3: Schematischer Aufbau des Multimediaequipments im Informatikneubau

5. 3D-VR-Interaktionsschnittstelle: Der erste Teilaspekt ist hierbei die „Simulation“ der Mausinteraktion durch den Laserpointer. Da dieser prinzipiell nur die elementaren Operationen „Bewegung in x- und y-Richtung“, Laserspot „Ein“ oder „Aus“ erlaubt, sind durch geeignete Abfolgen dieser Elemtaroperationen Mausinteraktionen (z.B. Mausklicks) nachzubilden. Zu beachten ist hierbei, dass der Übergang von der 2D- zur 3D-Interaktion auch mit einer Anhebung auf sechs Freiheitsgrade (Degrees of Freedom) einhergeht. Realisierbar sind diese Anforderungen durch gesten-basierte Ansätze oder Kombinationen aus Verweilzeit und Position des Laserpointers, als auch Informationen zum Status (an/aus), welche nachfolgend der Interpretation durch einen „Action Generator“ für Maus-Events, konform in das Betriebssystem zu integrieren sind. Da eine „naive“ Nachbildung der normalen PC-Maus aber primär nur eine Interaktion in der 2D-Ebene erlaubt, ist sie um eine Penetration des Laserpunktes in die „3D-Welt“ zu erweitern („Virtual- Point-Metapher“). Bei diesem Raycasting-Ansatz (Abbildung 4) dringt von der Position des Laserpunktes auf der Projektionswand ein Sehstrahl in die virtuelle 3D-Szene ein und erlaubt aufgrund der Geometrieinformationen eine Kollisionsdetektion mit Objekten der virtuellen 3D-Welt („Selektion von Objekten“).

   Mit der Zunahme von Freiheitsgraden erscheint es opportun, ein erweitertes, 3D-konformes Graphical User Interface mittels 3D-Widget-Metapher zu etablieren. Über diese Interaktionselemente ist auch eine Inkooperation der z-Achse zur Handhabung der erweiterten Translations- und Rotationsbewegungen im 3D-Raum zu realisieren.

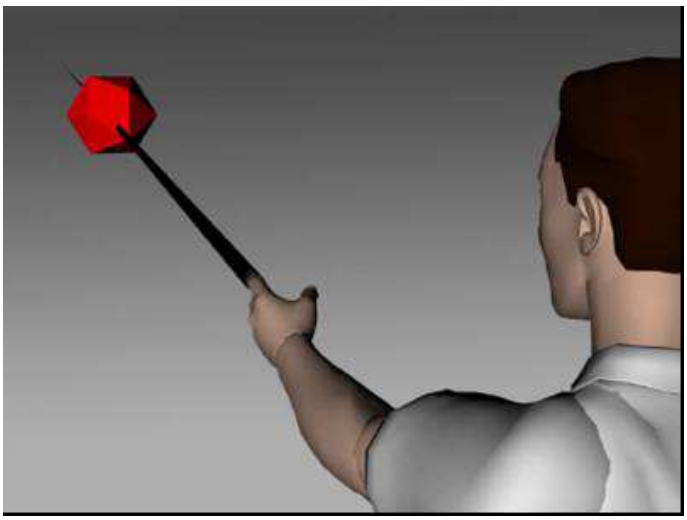


Abbildung 4: 3D-GUI mit dem virtuellen Laserpunkt

6. Visualisierung: Konform zur gegebenen dreidimensionalen Visualisierung soll der virtuelle Laserpunkt in die 3D-Szene eingefügt und die unter dem Punkt (5) beschriebenen Interaktions-Modi durch prägnante Piktogramme des virtuellen Pointers symbolisiert werden. Auch sind die 3D-Widgets geeignet zu generieren.

1.3 Realisierung

Aktuelle Realisierungen der oben genannten Problematik decken jeweils nur einen Teilbereich ab oder sind auf Grund ihres Designs vergleichsweise nicht oder nur eingeschränkt unter realen Anwendungsszenarien einsetzbar. Selbst unter Verwendung expliziter 3D-Eingabegeräte (z.B. 3D-Maus) ist vielfach keine intuitive oder kooperative Nutzung durch mehrere Personen gegeben.

Das von der Projektgruppe zu entwickelnde System soll alle Teilaspekte integriert bearbeiten und durch einen modularen Aufbau gewährleisten, dass sich zukünftige Anwendungen problemlos integrieren lassen. Daher ist auch die resultierende prototypische Umsetzung nicht auf die einleitend aufgezeigte Anwendung spezifiziert, sondern erlaubt eine generalisierte Nutzung und Erweiterung der Erkenntnisse der Projektgruppe. So findet sich das prinzipielle Anwendungsszenario in vielfältigen Bereichen wieder, die eine virtuelle oder augmentierte 3D-Visualisierung benötigen, sei es im Kontext einer komplexen Datenvisualisierung, 3D-Operationsplanung, bei

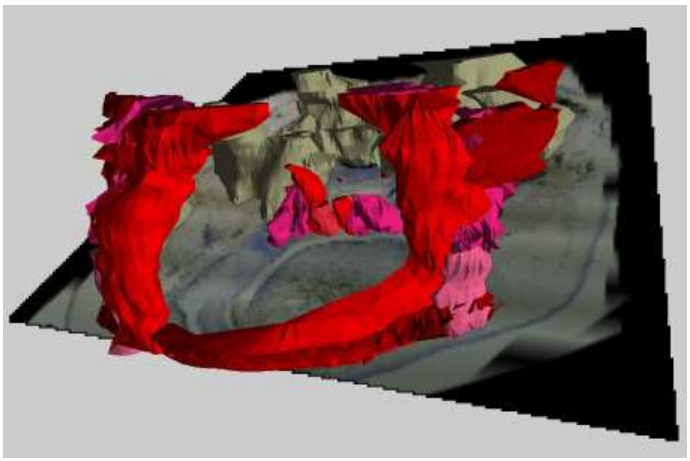


Abbildung 5: 3D-Modell zur Operationsplanung bei Lippen-Kiefer-Gaumenspalten

der Visualisierung von Fräs- und Zerspanungsprozessen, zur dreidimensionalen Repräsentierung mathematischer Simulationen oder zur Einbettung innerhalb einer erweiterten multimedialen Lernumgebung.

Da die Nutzung des Systems auch von Nicht-Informatikern und ohne umfangreiche Einarbeitung erfolgen soll, ist auf eine intuitive Benutzerführung zu achten. Bei der objektorientierten Umsetzung dieser Anforderungen in ein Softwaresystem sollen die Entwurfsphasen des Software-Engineering beachtet werden.

Obwohl die Implementierung des Systems primär unter WindowsXP erfolgen soll, ist eine mögliche Portierbarkeit auf Linux zu beachten. Das immense Datenvolumen, als auch die zeitkritischen Berechnungen in Echtzeit, können nur durch eine effiziente Software gewährleistet werden, welche sich auf Basis von C++ realisieren lässt. Die initiale Entwicklung soll sich hierbei an den MBone Videoconferencing-Tools3 und der 3D-Web GUI Vizmo4 orientieren.

1.4 Struktur dieses Berichts

Dieses Kapitel bot einen kurzen Überblick über das vorliegende Projekt. Die detaillierte Vorgehensweise wird in den Kapiteln 2 bis 8 beschrieben. Dabei stellt jedes Kapitel ein Modul des Softwaresystems vor. Im Einzelnen handelt es sich dabei um Datenakquisition, Kalibrierung und Dewarping, Bildverbes-

serung, Segmentierung, Schwellwert-Konfiguration, Tracking, Treiber sowie einem Kapitel zur 3D-Laserverlängerung. Die bei der Realisierung verwendeten Werkzeuge - wie zum Beispiel VMware oder Newmat - werden in Anhang C vorgestellt. Das abschließende Fazit befindet sich in Kapitel 12. Des Weiteren besteht der Anhang aus der Klassendokumentation, dem Pflichtenheft und dem Handbuch.

2 Datenakquisition

Die Arbeitsgruppe „Datenakquisition“ hatte die Aufgabe, die eingehenden Bilddaten, die von einer beliebigen Anzahl von Kameras aufgenommen werden, in geeigneter Weise aufzubereiten und sie für die anderen Arbeitsgruppen in Form einer Datenstruktur zur Verfügung zu stellen und den Zugriff auf diese Struktur zu ermöglichen. Für eine geeignete und performante Datenstruktur erfolgte anfangs eine Absprache mit den beteiligten Arbeitsgruppen. Die dabei festgelegte Form wird als die Pic Datenstruktur bezeichnet.

Da der Hörsaal für eine anfängliche Implementierung aufgrund von Umbau-/Reparaturarbeiten nicht zur Verfügung stand, entschied sich die Projektgruppe 498 für zwei verschiedene Umgebungen, das Test- und das Liveszenario. Das Testszenario, aufgebaut und stets erreichbar in den Räumen der Projektgruppe, besteht aus zwei Projektoren, die auf eine Rückprojektionsfläche ausgerichtet sind. Drei Kameras werden im Raum verteilt und an einen PC mit einer Framegrabberkarte angeschlossen. Eine Kamera soll dabei den Laserpointer auf der Rückprojektionsfläche erfassen, die anderen zwei sollen die Positionierung des Laserpointerbenutzers bestimmen. Das anfängliche Testszenario wird in Abbildung 6 schematisch erläutert.
Das Liveszenario, befindet sich im Hörsaal, bietet ebenso zwei Projektoren und drei, diesmal fest installierte Kameras. Der hauptsächliche Unterschied zwischen dem Test- und Liveszenario ist der, dass eine Darstellung nicht über eine Rückprojektionswand erfolgt.

Damit eine Implementierung schnell voranschreiten und andere Arbeitsgruppen ihre Arbeit aufnehmen konnten, wurde auf bestehende Techniken zurückgegriffen. Das bedeutet, das für das Testszenario das Softwarepaket MIL der Matrox Meteor-II/Multi-Channel verwendet wird. Diese Software wird im Kapitel C.16 eingehend erläutert. Analog wird im Liveszenario die bereits bestehende Verbindung von den Kameras, auf die Dallmeier DIS-1/S genutzt und von einer Neuimplementierung abgesehen. Die Kameradaten werden hier von dem Dallmeier DIS-1/S über einen VLC media player in das Netzwerk verteilt. Der VLC media player wird ausführlich in Kapitel C.1 beschrieben.

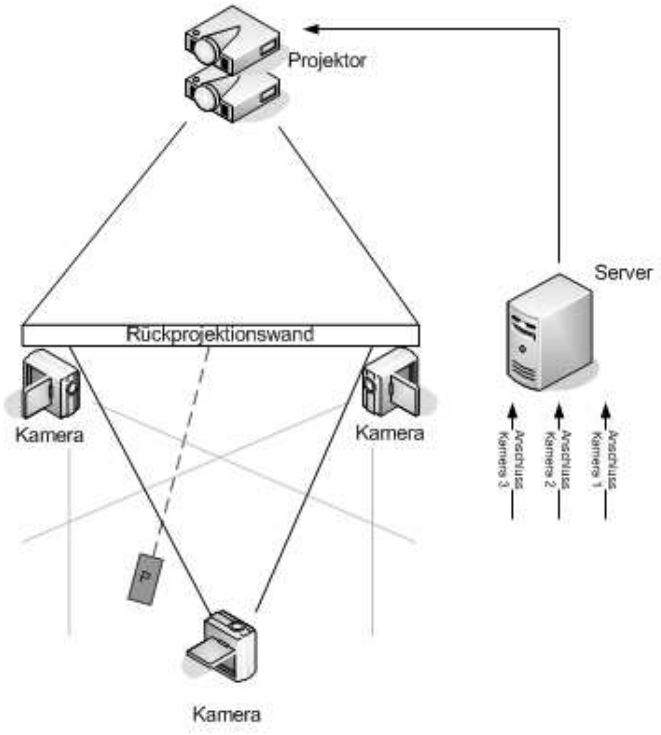


Abbildung 6: Schematischer Überblick des Testszenarios

2.1 Testszenario

Im Testszenario wird die Matrox Meteor-II/Multi-Channel (Abbildung 7) als Framegrabberkarte eingesetzt und mit den drei vorhandenen Kameras verbunden. Erste Versuche, die Kameradaten über ein selbst implementiertes Programm abzufragen, gestalteten sich schwierig, da das richtige Treiberpaket in Form der Matrox Image Library (MIL) nicht aufgespielt war. Auf dem Entwicklungsrechner befand sich Version 7.1, die schon seit November 2002 erhältlich ist und eine unzureichende Unterstützung für Visual Studio 2005 bot. Die mitgelieferten Beispiele ließen sich mit der Version nicht kompilieren, so dass auf die Version 7.5 gewechselt werden musste. Die mit der Version 7.5 gelieferten Beispiele ließen sich kompilieren und die Arbeitsgruppe erhielt einen ersten Eindruck über die Verwendung und Einbindung der Bibliotheken.

Damit für das Test- und Liveszenario nicht zuviele individuelle Anwendungen implementiert werden mussten, entschied die Arbeitsgruppe, einen naiven

Abbildung 7: Matrox Meteor-II/Multi-Channel

Ansatz zu verfolgen. Dieser Ansatz sah folgendes vor: Erfassen der Kameradaten über die MIL und Ablage dieser Daten im MPEG-2 Format auf der Festplatte in einer Datei. Ein VLC media player sollte dann diese Datei aufrufen und sie über das Netzwerk in Form von MPEG-TS Paketen streamen. Damit wäre dann das Testszenario dem Liveszenario angeglichen. Eine weitere Anwendung, die für beide Szenarien gelten würde, sollte dann den Netzwerkstream vom VLC media player empfangen, die MPEG-TS Pakete decodieren, in Frames zusammenlegen und diese dann in die Pic Datenstruktur übertragen. Somit ergeben sich insgesamt zwei Anwendungen die implementiert werden müssen.

2.1.1 Ansatz Linux

Nach einer ersten Testimplementierung unter Windows XP stellte sich heraus, dass es nicht möglich war, über die MIL gleichzeitig in eine Datei zu schreiben und diese für ein anderes Programm verfügbar zu machen. Die Funktion `MBufExportSequence` schreibt mit exklusiven Zugriffsrechten in der Datei und verhindert so den Lesezugriff von einer anderen Anwendung, sprich dem VLC media player. Ein Anruf bei der Matrox-Hotline ergab keine Veränderung. Die Implementierung der `MBufExportSequence` Funktion mit ihrem exklusiven Zugriff kann nicht verändert werden.

Parallel wurde nach einer Möglichkeit recherchiert, direkt vom VLC media player auf die Matrox Meteor-II Framegrabberkarte zuzugreifen. Damit würde die Implementierung einer Anwendung, wie die Eingangs beschriebene Testimplementierung hinfällig. Diese Möglichkeit wurde gefunden, aber für eine Realisierung wäre ein Wechsel des Betriebssystems notwendig

geworden. Video4Linux (v4l) ist die Portierung des VLC media player für Linux und die orginale Video capture/overlay Application Programming Interface (API) des Linux Kernels und wurde ab dem 2.1.x Linux Kernel unterstützt. V4l2 ist die zweite Generation der v4l API, die eine Anzahl von Designschwächen der ersten Version behebt. Diese Version wurde auch in den Standard-Kernel von 2.5.x übernommen. Die Arbeitsgruppe entschied sich die Linuxdistribution Debian 3.1 (Sarge, 2.4.x) lokal aufzusetzen und diesen Weg weiter zu verfolgen.

Nach der erfolgten Installation von Debian musste die Framegrabberkarte eingebunden und eingerichtet werden. Der Support für Linuxtreiber erfolgt laut Hersteller Matrox nur von Drittanbietern und ohne Gewähr. Die tatsächliche Verfügbarkeit der Matrox Meteor-II Treiber kann in der Tabelle 1 eingesehen werden.

Amerinex	15 Tage Trial
Bildanalysesystem AB	Anderes Angebot, kein Treibersupport
Clemex	Anderes Angebot, kein Treibersupport
Compix Imaging	Treiber Win98, Xp, 2k und NT
Ebbosoft	Treiber Win98, Xp, 2k und NT
emlix	Treiber vorhanden (Open Source)
Explora Nova	Anderes Angebot, kein Treibersupport
FDS Research	Anderes Angebot, kein Treibersupport
I.O. Industries	FTP nicht verfügbar
Impuls	30 Tage Trial
Kinetic Imaging	Anderes Angebot, kein Treibersupport
The MathWorks	Anderes Angebot, kein Treibersupport
Media Cybernetics	Treiber für Win98, XP, 2K und NT
MicroMotion	Anderes Angebot, kein Treibersupport
MVTec Software GmbH	Keine Treiber hinterlegt
National Instruments	Keine Treiber hinterlegt
Noesis	Anderes Angebot, kein Treibersupport
Norpix	Zwei Wochen Trial
Soft Imaging System	Anderes Angebot, kein Treibersupport
Visiopharm	Anderes Angebot, kein Treibersupport

Tabelle 1: Verfügbare Treiberpakete der Matrox Meteor-II

Nach Auswahl eines geeigneten und kostenlosen Treiberpakets konnte der Kernel neu gebaut werden. Jedoch traten noch die verschiedensten Probleme und Inkompatibilitäten auf. Das Betriebssystem stürzte mitten in der Startroutine ab und ein Wechsel von Debian 3.1 auf Debian 4.0 (Etch, 2.6.x)

konnte nicht realisiert werden, da das vorliegende Treiberpaket für eine andere Version gedacht war. Die Treiber waren für den Kernel 2.4.x gedacht und konnten mit dem vorhandenen Wissen nicht auf 2.6.x portiert werden.

Aufgrund dieser lückenhaften Unterstützung anderer Betriebssysteme wurde die Recherche und weitere Verfolgung an dieser Stelle abgebrochen und Alternativen für Windows gesucht.

2.1.2 Ansatz Windows

Eine Sequenz von Frames soll mit einer eigenen Anwendung in eine Datei geschrieben werden. Dabei soll lesender Fremdzugriff gestattet werden und über einen aufgesetzen VLC media player auf diese Datei zugegriffen, bzw. in das Netzwerk gestreamed werden. Folgende Funktion stellt die MIL zur Verfügung:

`MBufInquire M_HOST_ADDRESS`

Sie liefert die Position des Frames im Speicher. Dabei muss man beachten, dass es zwei grundsätzliche Zugriffsmodi für Farbbilder gibt:

- Packed: Erster Zeiger zeigt auf den roten Buffer, danach folgt grün und blau.
- Planar: Drei getrennte Speicher für die Farbkanäle, Zeiger auf die drei MBuf-Child-Buffer.

Nach einer umfassenden Recherche und weiteren Versuchen mittels QuEnc und FFmpeg, die Kamerabilddaten in ein gängiges MPEG-2 Format zu codieren, beschloss die Arbeitsgruppe unter angestiegenem Zeitdruck, das Testszenario abzuändern. Die zu realisierende Server-Anwendung befindet sich nicht mehr entfernt auf einem anderen PC, sondern direkt auf dem PC, an dem die Matrox Meteor-II Framegrabberkarte angeschlossen ist. Die Kamerabilddaten werden nun direkt formatiert und in die Pic-Datenstruktur eingetragen. Dieser Ansatz konnte erfolgreich realisiert werden. Kapitel 2.1.4 erläutert die verwendete Implementierung.

2.1.3 Trennung von Client und Server

Aus Leistungsgründen wird die Anwendung wie in Abbildung 8 verteilt eingesetzt. Der Client besitzt neben der GUI (Grafischen Benutzeroberfläche) nur den Treiber für die Mausansteuerung und der 3D-Hook (siehe Kapitel 9).

An den Client wird zusätzlich noch die Projektionshardware angeschlossen bzw. angesteuert. Der Server wird mit den Modulen Dewarping, Bildverbesserung, Segmentierung und Tracking ausgerüstet und verfügt über die Matrox Meteor-II Framegrabberkarte. Die Kommunikation zwischen Client und Server erfolgt über das Netzwerk in Form von UDP Nachrichten. Nachrichten können Informationen über die Mauskoordinaten und Servereinstellungen beinhalten.

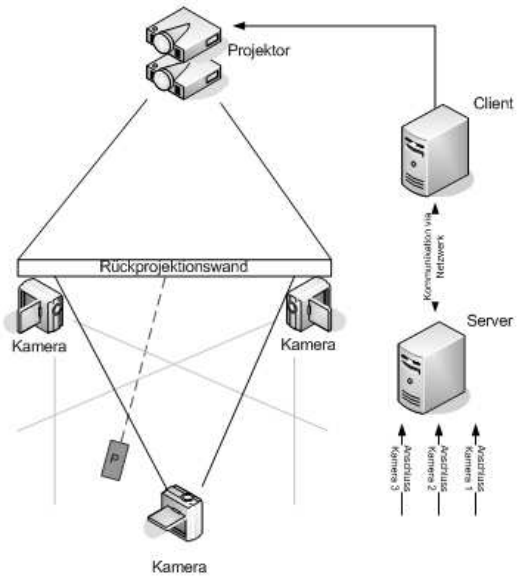


Abbildung 8: Trennung von Client und Server

2.1.4 Lokale Simulation

Der nachstehende Programmcode ist dem Framegrabbing Modul im Gesamtsystem entnommen. Er wurde erfolgreich für das Testszenario eingesetzt und besteht aus einem Konstruktor, der Prozedur SingleStep und dem abschließenden Destruktor.

Die Prozedur SingleStep ermittelt die Systemzeit, stellt den zu betrachtenden Kamerakanal ein und ruft über die Prozedur `MbufGetColor2d` die Bilddaten ab. Dabei werden die Bilddaten in das Array PictureRight, PictureLeft und PictureBack eingetragen und definieren damit ein weiteres Array, das Frame. Abschließend wird dem Frame die Systemzeit als Zeitstempel zugeordnet.

```
void Framegrabbing::SingleStep()
{
GetSystemTime(&SysTime);

MdigChannel(MilDigitizer, Channel[0]);
MdigGrab(MilDigitizer, MilImageDisp);
MdigGrab(MilDigitizer, MilImageDisp);
PictureRight = m_Frame->frameset[2];
MbufGetColor2d(MilImageDisp, M_PACKED+M_BGR24 , M_ALL_BAND, 0, 0, x,
y,PictureRight[0][0]);

MdigChannel(MilDigitizer, Channel[1]);
MdigGrab(MilDigitizer, MilImageDisp);
MdigGrab(MilDigitizer, MilImageDisp);
PictureBack = m_Frame->frameset[0];
MbufGetColor2d(MilImageDisp, M_PACKED+M_BGR24 , M_ALL_BAND, 0, 0, x,
y, PictureBack[0][0]);

MdigChannel(MilDigitizer, Channel[2]);
MdigGrab(MilDigitizer, MilImageDisp);
MdigGrab(MilDigitizer, MilImageDisp);
PictureLeft = m_Frame->frameset[1];
MbufGetColor2d(MilImageDisp, M_PACKED+M_BGR24 , M_ALL_BAND, 0, 0, x,
y, PictureLeft[0][0]);

m_Frame->timeStamp = timeGetTime();
}
```

Der Destruktor `~Framegrabbing()` führt die Speicherbereinigung durch, indem er den Buffer, Digitizerspeicher, Displayspeicher, Systemspeicher und anschließend die Anwendung löscht.

2.2 Anwendungsszenario

Die Domkameras im Hörsaal ermöglichen es, den Laserpointer auf der Leinwand und den Vortragenden aufzuzeichnen. Aus ihren Bilddaten werden in den Dallmeier Video Streaming Servern für jede Kamera ein MPEG-2-Videostream erzeugt und per Multicast in das Hörsaalnetz verteilt. Um die Bilddaten zur weiteren Verarbeitung bereit zu stellen, müssen diese aus dem Netzwerk empfangen und decodiert werden. Da die Implementierung eines

eigenen MPEG2-Decoders den Aufwand des Projektes übersteigt, wurden verschiedene Quellen bereits vorhandener Decoder in Betracht gezogen.

2.2.1 MPEG-2-Codec

Das analoge Bildsignal der Domkameras benötigt eine Bitrate von rund 216 Mbit/s. Um die Videodaten effizient zu archivieren und über das Netzwerk zu streamen, wird eine MPEG2-Codec Codierung vorgenommen. So ist es möglich die Bitrate bis auf rund 1 Mbit/s zu reduzieren. Dabei werden aktuelle Methoden der Bilddatenkompression angewandt, wie sie in der angegebenen Literatur unter [Str05] beschrieben werden. MPEG2 ist ein generischer Standard der Moving Picture Experts Group zur verlustbehafteten Video- und Audiocodierung. Es werden also nur Datenformate und Decodierverfahren explizit spezifiziert. Des Weiteren bietet der Codec die Wahlmöglichkeit weiterer Parameter, wie z.B. der Auflösung. Ein Beispiel für eine mögliche Konfiguration ist das sogenannte „Main Profile @ Main Level“, welches unter anderem auf Video-DVDs verwendet wird. Die Auflösung beträgt hierbei 720×576 Bildpunkte bei einer vertikale Framerate von 50 Hertz. Das Codierverfahren wird auch hybride Diskrete-Cosinus-Transformation (DCT) genannt. Es handelt sich dabei um eine Kombination aus einer diskreten Cosinus-Transformation und einer zeitlichen Differential-Puls-Code-Modulation (DPCM) mit Bewegungskompensation. Der genaue Aufbau lässt sich aus Abbildung 9 entnehmen.

Diskrete Cosinus-Transformation

Die diskrete Cosinus-Transformation transformiert ein zeitdiskretes Signal, $f : \{(m,n)|m = 0,1,\ldots,M-1, n = 0,1,\ldots,N-1\} \rightarrow \mathbb{R}$, vom Orts- in den Frequenzbereich. Sie wird genutzt, um Redundanzen in Bildsignalen zu reduzieren.

$$\begin{aligned} C(k,l) =& \alpha_M(k) \cdot \alpha_N(l) \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n) \\ & \cdot cos\left(\frac{(2 \cdot m + 1) \cdot k \cdot \pi}{2 \cdot M}\right) \cdot cos\left(\frac{(2 \cdot n + 1) \cdot l \cdot \pi}{2 \cdot N}\right), \\ & \text{mit } k \in \{0,\ldots,M-1\}, l \in \{0,\ldots,N-1\} \end{aligned} \tag{1}$$

mit $\alpha_X(p) = \sqrt{\frac{1}{X}}$ für $p = 0$ und $\alpha_X(p) = \sqrt{\frac{2}{X}}$ für $p \neq 0$. Die $C(k,l)$ bezeichnen die $M \times N$ DCT-Koeffizienten. Diese Transformation bietet einige vorteilhafte Eigenschaften für die Bildcodierung. Zum einen dekorreliert sie

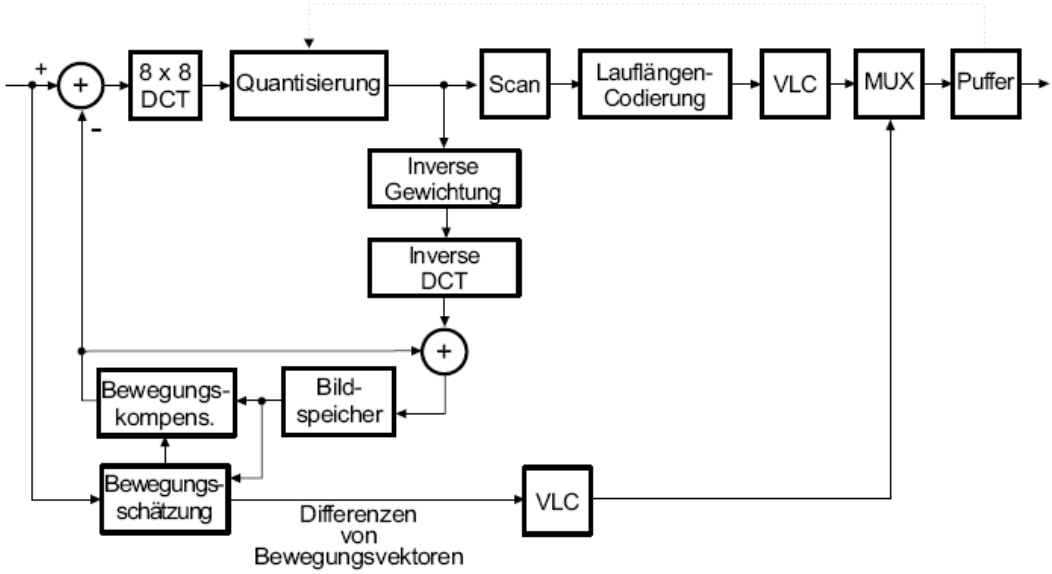


Abbildung 9: MPEG2-Coder [Str05]

die Pixel der Eingabebilder. Zudem ist sie separierbar und ermöglicht somit eine effiziente Berechnung ihrer Koeffizienten. Außerdem werden für ihre Berechnung im Gegensatz zur diskreten Fourier-Transformation keine komplexen Zahlen benötigt.

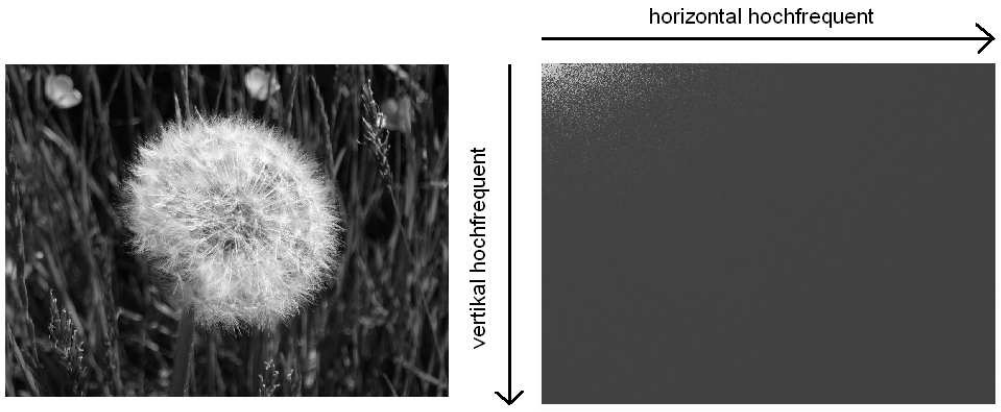


Abbildung 10: diskrete Cosinus-Transformation

Die Koeffizienten der DCT werden zur weiteren Codierung quantisiert. Dies hat meist zur Folge, dass nur wenige hochfrequente Koeffizienten von

Null verschieden sind, wie man in Abbildung 10 erkennen kann. Anschließend wird eine Laufl\"angencodierung vorgenommen. Dazu bildet man Tupel $< Run, Level >$, wobei Level den Wert des quantisierten Koeffizienten und Run die Anzahl der vorangegangenen Nullen angibt. Zusätzlich verwendet man eine Sequenz namens EOB, die signalisiert, dass nur noch Null-Koeffizienten folgen. Um möglichst lange Läufe von Nullen zu erhalten, nutzt man die dreiecksförmige Anordnung der von Null verschiedenen Koeffizienten durch eine spezielle Auslesereihenfolge (Zig-Zag-Scan), wie sie in Abbildung 11 dargestellt wird. Eine anschließende variable Längencodierung der Tupel ermöglicht eine weitere Reduzierung der Bandbreite.

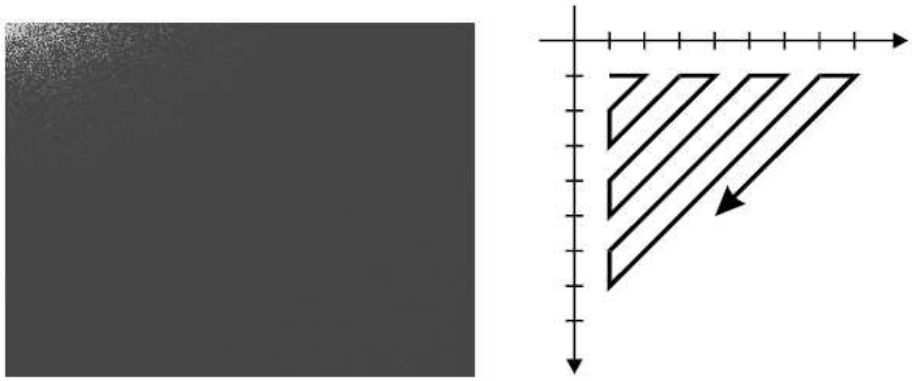


Abbildung 11: Zig-Zag-Scan [Str05]

Zeitliche Differential-Pulse-Code-Modulation

Bei einer Differential-Pulse-Code-Modulation (DPCM) wird anstelle eines benötigten Wertes die Differenz zwischen dem Wert und einem Schätzer übertragen. Somit lässt sich im Allgemeinen Bandbreite einsparen. Eine Folge von Bildern $B_k, k \in \{1, 2, 3, \ldots\}$, wie sie in einem Videostream vorliegt, ermöglicht eine Anwendung der DPCM, wie sie in Abbildung 12 beschrieben wird. Sei $s_k(m, n)$ der Farbwert des Bildpunktes an der Position (m, n) des Bildes B_k zum Zeitpunkt k, dann ist $s_k(m, n)$ Prädiktor für den Farbwert $s_{k+1}(m, n)$. Die Differenz zwischen Schätzer und dem eigentlichen Wert, auch Prädiktionsfehler genannt, errechnet sich also aus $s_{k+1}(m, n) - s_k(m, n)$. Es gilt zu beachten, dass für Bewegungen im Bild, die schneller als ein Pixel pro Frame sind, eine örtliche DPCM zu einer geringeren Bandbreite führen würde als die zeitliche DPCM.

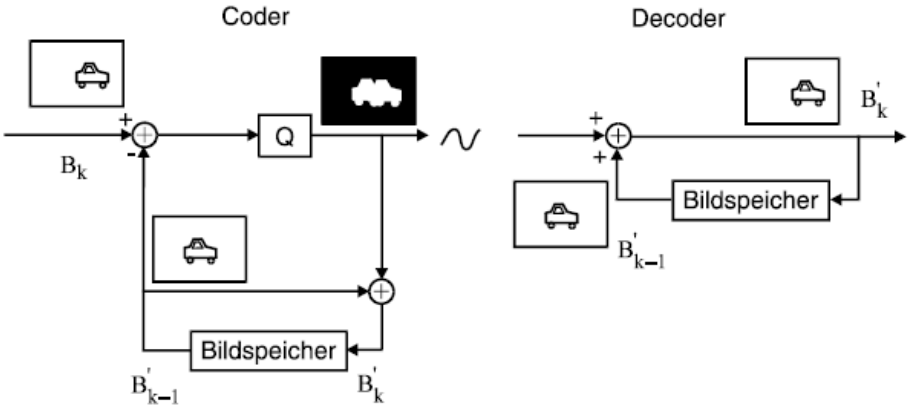


Abbildung 12: zeitliche Differential-Pulse-Code-Modulation [Str05]

Um diesem Effekt entgegenzuwirken, führt man eine Bewegungskompensation mit Hilfe des sogenannten Block-Matchings durch. Dabei wird das Bild in $M \times N$ quadratische Segmente $P(u, v)$ aufgeteilt. Zu jedem dieser Unterblöcke $P(u, v)$ wird die Menge M der optimalen Verschiebungsvektoren bestimmt und ein Vektor $(\delta_x(u, v), \delta_y(u, v)) \in M$ ausgewählt, so dass sich das zu codierende Bild aus Segmenten des letzten Bildes ergibt, wie es in Abbildung 13 zu entnehmen ist.
Es gilt dabei:

$$M = \left\{ (\delta_x, \delta_y) \mid \eta(\delta_x, \delta_y) = \min_{\forall \delta_x, \delta_y} \{\eta(\delta_x, \delta_y)\} \right\} \tag{2}$$

mit

$$\eta(\delta_x, \delta_y) = \sum_{s_k(m,n) \in P(u,v)} [s_k(m, n) - s_{k-1}(m - \delta_x, n - \delta_y)]^2 \tag{3}$$

Die Funktion $\eta(\delta_x, \delta_y)$ berechnet dabei die Summe der Farbwertabweichung aller Pixel des Segmentes des alten Frames und des um (δ_x, δ_y) Bildpunkte versetzten Segmentes des neuen Frames. Die (δ_x, δ_y), für die diese Summe minimal wird, werden als Vektor zur Bewegungskompensation gewählt. In der Praxis verwendet man meist Segmente von 16×16 Pixel (MPEG2-Makroblöcke) und beschränkt sich bei der Helligkeitsdifferenz auf Absolutwerte. Zudem sucht man die Verschiebungsvektoren nicht im ganzen Frame, sondern nur in einem Bereich um den Block, also z.B. im Bereich

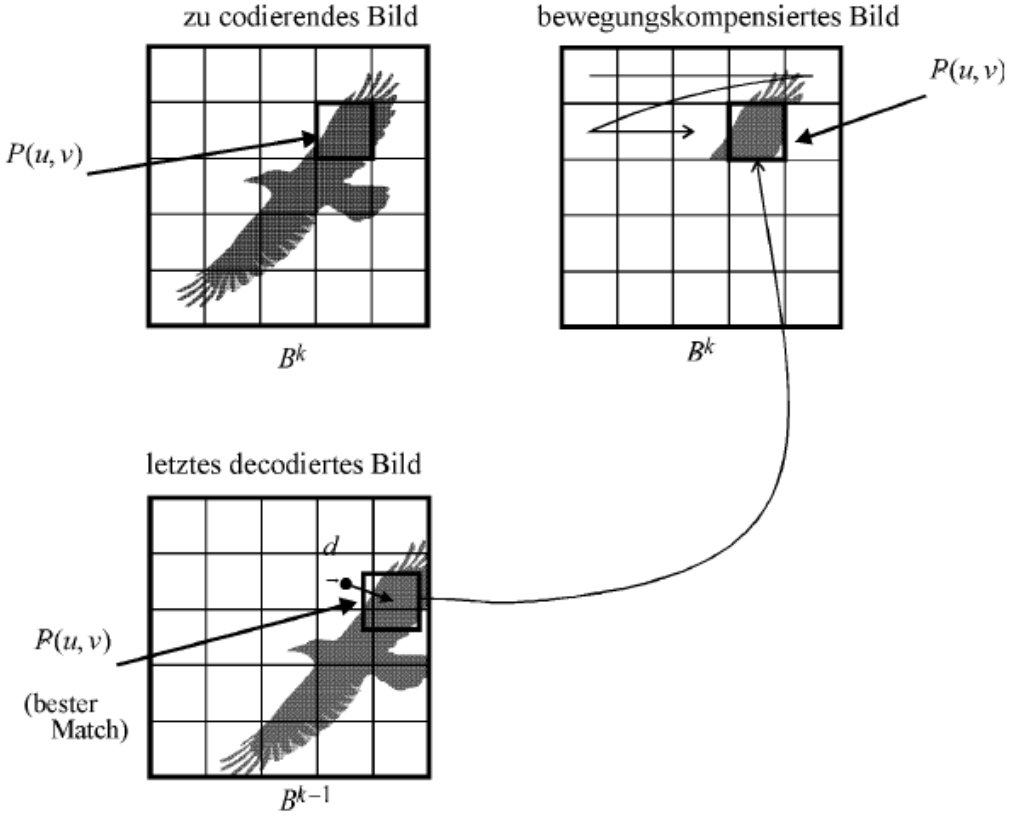


Abbildung 13: Bewegungskompensation [Str05]

$-16 < \delta_x \leq 16, -16 < \delta_y \leq 16$. Es hat sich weiterhin als vorteilhaft herausgestellt, die Messung von δ_x und δ_y mit der Genauigkeit eines halben Bildpunktes (half-pel) vorzunehmen. Dazu müsssen allerdings erst einmal die Subpixel interpoliert werden, was einen nicht zu vernachlässigenden Rechenaufwand bedeutet.

Mehrere Frames werden zu einer Group of Pictures (GoP) zusammengefasst, wie sie in der Abbildung 14 veranschaulicht wird. Es gibt drei verschiedene Arten von Frames, welche sich in der Weise mit der sie prädiziert werden unterscheiden. Die sogenannten I-Frames (Intra-Mode) werden nicht zeitlich prädiziert. Sie bieten einen Einstiegspunkt im Falle eines Decodierversagens. Dieser kann z.B. durch einen Übertragungsfehler oder durch das Umschalten zwischen mehreren Videostreams auftreten. Es wird ca. jedes zwölfte Bild im Intra-Mode übertragen. Bei einer vertikalen Framerate von 25 Hertz erhält man also alle halbe Sekunde ein eigenständig decodierbares Bild. Die

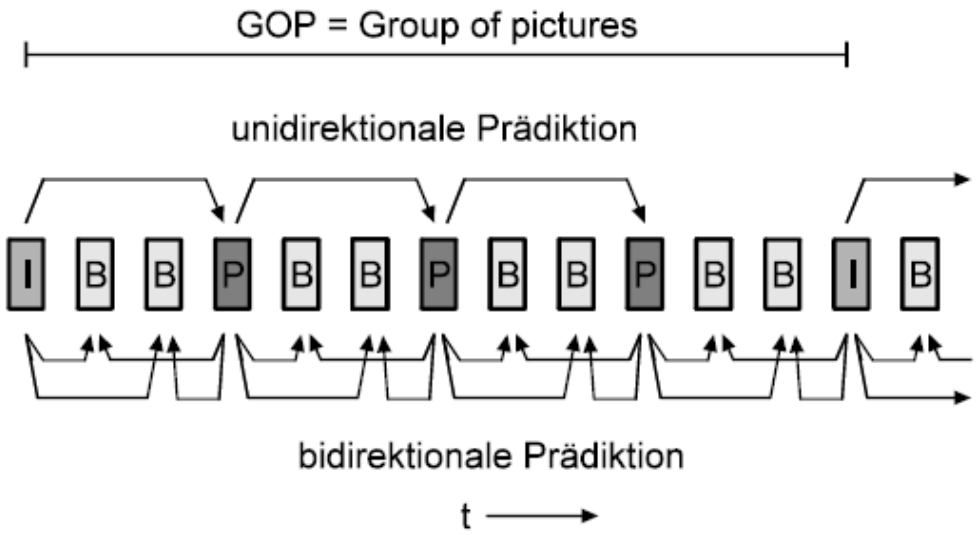


Abbildung 14: Group of Pictures [Str05]

P-Frames hingegen sind prädiktiv codierte Bilder. Für ihre Prädiktion werden Makroblöcke aus den vorhergegangenen I- und P-Frames benötigt. Bei B-Frames handelt es sich um bidirektional prädizierte Bilder, dazu werden sowohl Makroblöcke aus vorhergegangenen als auch aus nachfolgenden I- und P-Frames genutzt. Oft ist es möglich den Frame vollständig aus vergangenen und zukünftigen Frames zu interpolieren ohne ein Differenzbild zu übertragen. Durch diesen Effizienzgewinn bei den B-Frames ergibt sich ein Bitratenverhältnis von etwa I:P:B = 9:3:1. Da in der Regel konstante Bitraten erwünscht sind, benötigt man eine Pufferregelung. Diese ermöglicht es auch die Genauigkeit der Quantisierung abhängig vom Füllstand des Puffers zu modifizieren. Die Größe des Puffers sollte jedoch begrenzt werden, um zu lange Verzögerungen bei Live-Bildern zu verhindern. Weiterhin ist zu beachten, dass die Wiedergabereihenfolge der Frames IBBP ungleich der Übertragungsreihenfolge IPBB ist. Nur so können die B-Frames bidirektional prädiziert werden.

Datenstrom

Die kleinste Einheit des MPEG-2 Videostreams ist der Block. Es handelt sich um einen 8×8 Pixel großes Segment des Bildes und stellt das Eingangsformat der DCT dar. Die nächstgrößere Einheit ist der Makroblock,

er besteht aus vier Blöcken. Es handelt sich um das Eingangsformat der Bewegungskompensation. Zu jedem Makroblock wird in einem Header angegeben, ob er im I-, P- oder B-Mode übertragen wird. So können Teile eines Frames, für die kein adequater vorheriger Schätzer vorhanden ist, z.B. beim Aufdecken von Bildbereichen, im Intra-Mode übertragen werden. Eine Zeile von Makroblöcken bildet eine Slice. Sie bietet durch ein bestimmtes Bitmuster ($0^{23}1^1$) die Möglichkeit zur horizontalen Resynchronisation. In ihrem Header können zusätzlich Informationen über die vertikale Position übertragen werden. Ein Picture besteht aus einer auflösungsabhängigen Anzahl von Slices. In ihrem Header wird unter anderem der Prädiktions-Mode des Frames vermerkt. Die Abfolge von mehreren Pictures nennt man eine Sequence, ihr Header enthält Informationen über die Bildgröße, die Farbcodierung, das Bildseitenverhältnis und das Zeilensprungverfahren.

Für Medien mit geringer Bitfehlerhäufigkeit, wie z.B. Festplatten oder DVDs bietet sich der MPEG2-Programmstrom als Multiplexformat an. Es besteht dabei die Möglichkeit Einzelbilder entweder in einzelnen Blöcken (bis zu $64kB$) oder in Blöcke fester Länge aufzuteilen. Der Umgang mit B-Frames wird erleichtert, in dem man den Frames Timestamps zuordnet, einen für den Decodierzeitpunkt („decode time stamp“) und einen für den Wiedergabezeitpunkt („presentation time stamp“). Diese Konventionen erleichtern das Streaming von Videodaten. Mit dem MPEG2-Transportstrom steht zusätzlich ein Multiplexformat für Känale mit hoher Bitfehlerhäufigkeit zur Verfügung. Dazu werden die Programmstrompakte in 188 Byte kurze Teilpakete unterteilt. Diese Maßnahme verringert die Auswirkungen eines einzelnen Paketverlustes. Die Länge von 188 Byte wurde aus der ATM-Übertragungstechnik abgeleitet, es handelt sich um die Länge von vier ATM-Zellen.

2.2.2 DirectShow Filtergraph

Die DirectShow ist ein Bestandteil von Microsoft DirectX [dir04] und basiert auf einer Filter-Architektur. Es gibt drei Grundarten von Filtern: Source-Filter zur Ausgabe von Video- und Audiodaten, Transform-Filter zur Bearbeitung von Eingaben und Ausgabe von einem oder mehrerer Resultate und Rendering-Filter zum Darstellen oder Speichern von Mediendaten. Diese Architektur vereinfacht die Verarbeitung von Bildsequenzen zu einer Verkettung von Filtern, wobei die Ausgabe eines Filters als Eingabe des nächsten Filters genutzt wird. Zudem gibt es die Möglichkeit, Filter mit mehreren Ausgaben einzusetzen. Üblicherweise findet man am Anfang dieses sogenannten Filtergraphen einen Source-Filter und am Ende einen Rendering-Filter.

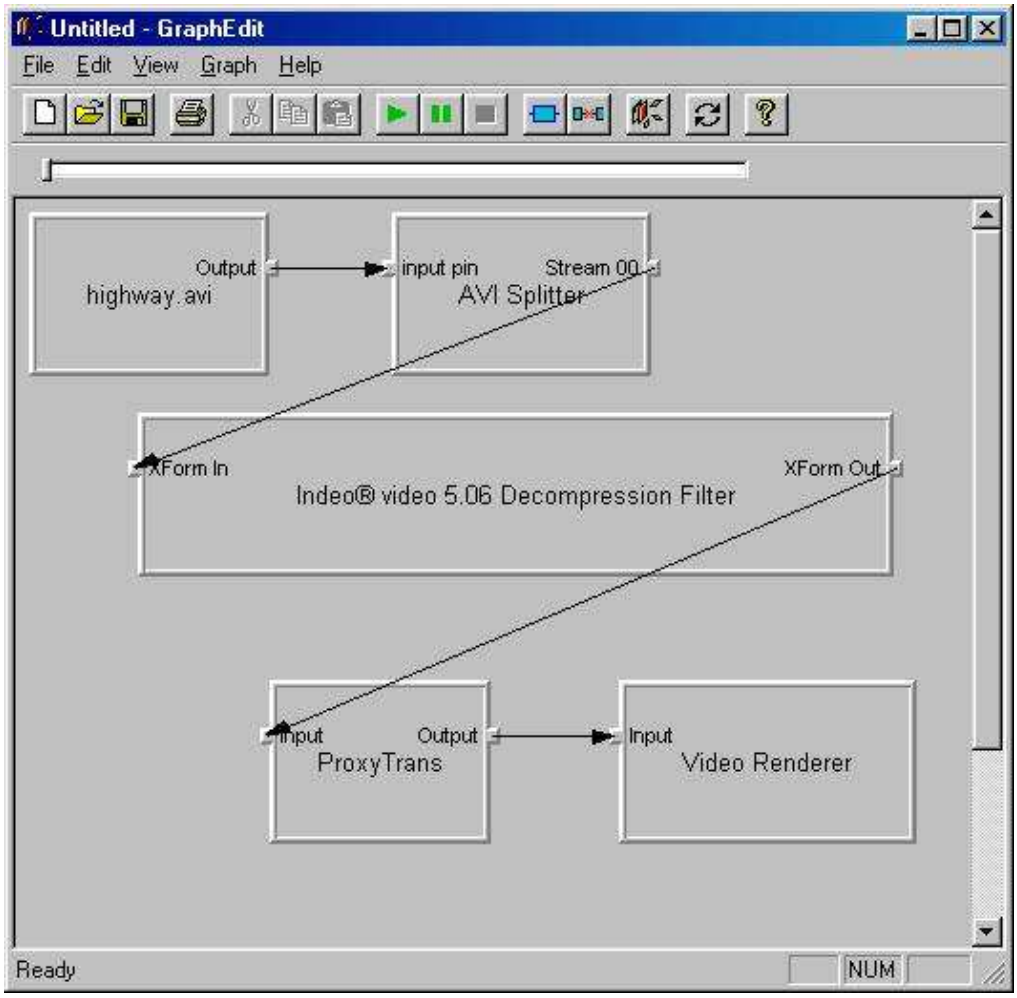


Abbildung 15: Filtergraph in GraphEdit

Das Erstellen von Filtergraphen wird durch das Programm GraphEdit, welches im DirectX 9.0 SDK enthalten ist, erleichtert. Mit Hilfe dieses Tools lassen sich die benötigten Filter zusammenstellen und verbinden. Ein exemplarischer Filtergraph ist in Abbildung 15 dargestellt. Dabei stellt DirectShow unter anderem Filter zum Splitten, also dem Separieren von Ton und Bilddaten in einem Videostream, und zum Decodieren der Daten bereit. Die Möglichkeit einen Source-Filter mit einer Videodatei als Quelle zu erstellen, ist ebenfalls vorhanden. Jedoch findet sich kein Source-Filter, um Videostreams aus dem Netzwerk einzulesen. Da das Schreiben kleiner Teile des Streams in eine Datei und das anschließende Einlesen mit Hilfe des üblichen Source-Filters zu einer beträchtlichen Verschlechterung der Lauf-

zeit des Systems führt, wurde der Ansatz, die Daten mittels DirectShow zu decodieren, verworfen.

2.2.3 VLS-Projekt

Der VideoLAN Server ist ein dedizierter Streaming Server von den Entwicklern des VideoLAN-Projektes [vlc07]. Er ermöglicht das Übertragen von MPEG-Streams sowohl im Multicast als auch im Unicast. Beim Unicast wird explizit die Adresse des Empfängers in den versandten Paketen eingetragen. Multicast-Pakete enthalten zwar auch eine Adresse, diese ist allerdings nicht einem eindeutigen Empfänger zugeordnet und befindet sich im IP-Class-D-Adressbereich. Alle Clienten, die den Stream empfangen wollen, können Pakete mit dieser Adresse annehmen und verarbeiten. Der VideoLAN-Server wurde in C++ mit einem gänzlich selbst erstellten Framework programmiert. VLS nutzt also keine Standard-Template-Bibliotheken wie `iostreams` oder `vectors`.

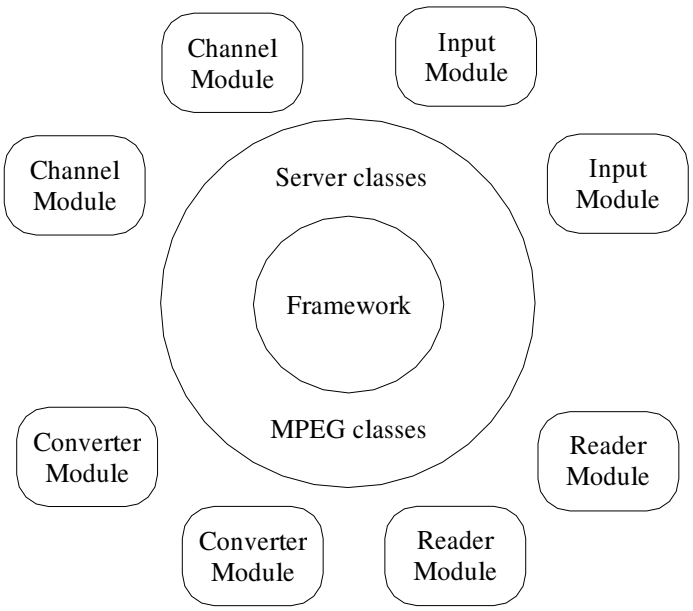


Abbildung 16: Struktur des VLS-Projektes

Der Server besteht aus drei Komponenten: Framework, allgemeinen (common) und optionalen (modul) Klassen. Bei den Modulen handelt es sich um

Klassen, die von allgemeinen Klassen erben. Eingebaute Module sind statisch verlinkt. Hingegen werden „plug-in“-Module dynamisch geladen. Die Struktur des VLS ist in Abbildung 16 dargestellt.

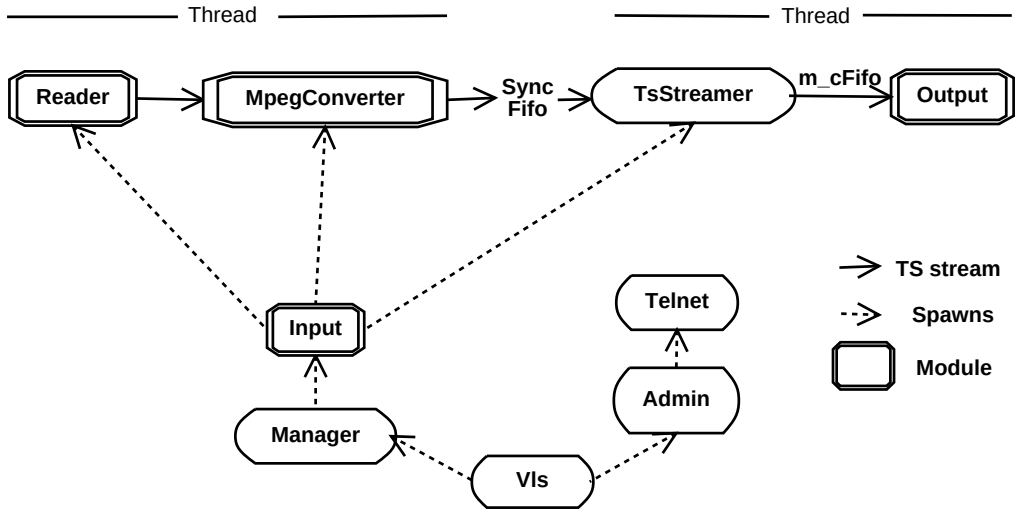


Abbildung 17: Architektur des VLS-Projektes

Die Architektur des Servers ist aus drei Hauptkomponenten zusammgesetzt: dem Source-Thread (`Reader` und `MpegKonverter`), dem Consumer-Thread (`TsStreamer` und `Output`) und der Managementkomponente (`Input`, `Admin` und `Manager`). Die Managementkomponente ruft alle benötigten Module auf, wenn VLS oder ein neuer Stream gestartet wird. Der Source-Thread liest die Daten schnellstmöglich durch den `Reader` ein, um den `SyncFifo`-Buffer zu füllen. Danach holt sich der Consumer-Thread die Pakete aus dem Buffer und streamt sie. Für das Timing des Streams ist der `TsStreamer` verantwortlich, er steuert das Output-Modul an. Die Variable `m_cFifo` gibt an, wieviele Ts-Pakete zusammen in einem UDP-Paket versendet werden. Abbildung 17 verdeutlicht die Architektur des VideoLAN-Servers.

Um die Effizienz zu verbessern und zu viele Speicherallokationen zu vermeiden, wird einmalig ein Pool von Ts-Paketen im Speicher bereitgestellt. Diese Pakete werden vom Ts-Provider verwaltet. Zu Beginn wird ein neues Ts-Paket durch den Konverter angefordert. Es wird mit den Bytes, die der `Reader` übergibt, gefüllt und durchläuft den `TsStreamer` und das `Output`-Modul. Nachdem es versandt wurde, wird es durch den Ts-Provider wieder für den Konverter verfügbar gemacht. Der Ablauf wird in Abblildung 18 illustriert. Die Kapazität des Paketpools wird so kalkuliert, dass sie ca. drei Sekunden des Streams aufnehmen kann.

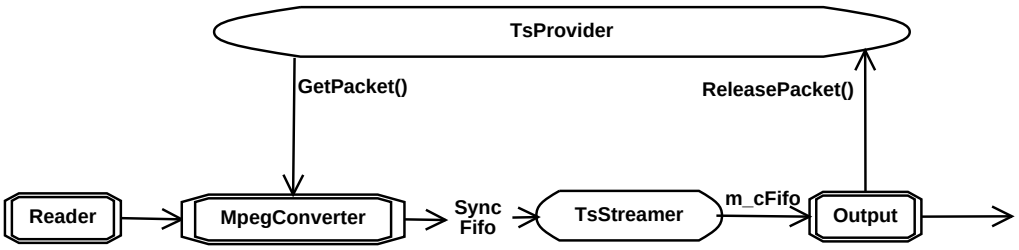


Abbildung 18: TsStream-Provider

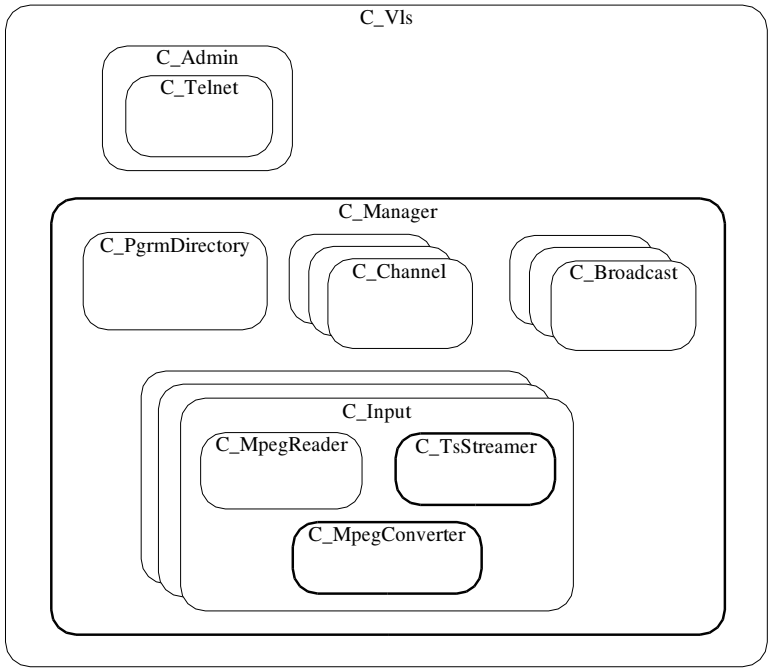


Abbildung 19: Klassenübersicht des VLS-Projektes

Die Abbildung 19 ermöglicht einen Überblick über einige der allgemeinen Klassen des VLS. Die Main-Klasse des VideoLAN-Servers heißt `C\_Vls`. Sie startet Admin und Manager, lädt die Module und übermittelt Nachrichten zwischen ihnen. `C\_Admin` ist die Administrationsklasse. Zu ihren Aufgaben gehört die Verwaltung der verschiedenen Administrations-Schnittstellen, sowie die Analyse und die Umsetzung der Kommandos, die zu diesen Schnittstellen gesendet werden. Zu diesen Schnittstellen gehört unter anderem das Telnet-Interface `C_Telnet`. Der Manager `C_Manager` gehört zu

den wichtigsten Klassen im VLS. Er startet die unterschiedlichen Eingaben und Kanäle, die in der Datei „vls.cfg“ angegeben sind. Diese Config-Datei enthält eine Liste, in der alle verfügbaren und alle momentan gestreamten Programme verzeichnet sind. Diese Klasse ist auch der Interpreter für die Kommandos, die an den Administrator gesendet werden. Sie kann Streams starten, stoppen, unterbrechen und wiederaufnehmen.

`C_Channel` ist die Parent-Klasse aller Channel-Module, z.B. der Netzwerk- und Datei-Output-Module. Das Programm-Verzeichnis `C_PgrmDirectory` enthält eine Liste aller als Eingabe zur Verfügung stehenden Programme. Die Klasse `C_Broadcast` beschreibt Broadcasts als eine Kombination von Input, Programm und Kanal. So lassen sich Streams repräsentieren, die momentan durch den VLS versendet werden. `C_Input` ist die Parent-Klasse aller Input-Module. Grundsätzlich erstellt ein Input einen MPEG-Stream aus einer Quelle und übergibt ihn an den Konverter. Jeder Input stellt eine Liste der verfügbaren Programme bereit.

Die Klasse `C_MpegReader` beschreibt einen Reader, der Daten aus einer Quelle (z.B. Datei oder DVD) ausliest und einen kontinuierlichen MPEG-Stream (Ts oder Ps) liefert. Die Klasse `C_MpegConverter` ist dafür zuständig, aus dem Stream, den der Reader bereit stellt, einen gültigen MPEG-Ts-Stream zu erstellen. Dieser Stream wird von der Klasse `C_TsStreamer` auf einem Kanal mit einer bestimmten Übertragungsgeschwindgkeit übermittelt. Somit ist VLS in der Lage beliebige MPEG-Streams über ein Netzwerk zu streamen. Da ein selbstentwickeltes Framework genutzt wird, ist die Portierung in das Visual Studio verhältnismäßig einfach. Allerdings ist VLS, da es nur serverseitig genutzt werden kann, nicht in der Lage MPEG2-Streams zu decodieren. Daher ist es für das Framegrabbing sinnvoller den Client des VideoLAN-Projekts zu nutzen. Dieser beinhaltet mittlerweile auch die Funktionen des Servers. Aus diesem Grund wurde die Entwicklung am VLS eingestellt.

2.2.4 VLC-Projekt

Der VLC Media Player [vlc07] ist eine vollständige Softwarelösung für Video-Streaming und Wiedergabe. Das ursprüngliche VideoLAN-Projekt wurde von Studenten der Ecole Central Paris entwickelt. Mittlerweile beteiligen sich Entwickler aus aller Welt unter der GNU General Public License an dem Projekt. VideoLAN wurde zu Beginn als reine Netzwerk-Streaming-Lösung entworfen. Aber der VLC hat sich zu einem ausgereiften, plattformübergreifenden Media Player entwickelt. VLC läuft auf folgenden

Plattformen: Linux, Windows, Mac OS X, BeOS, BSD, Solaris, Familiar Linux, Yopy/Linupy und QNX. Er eignet sich zur Wiedergabe von MPEG-Dateien, DVDs, VCDs und Audio CDs. Zusätzlich kann er Streams über UDP Unicast, UDP Multicast, HTTP und RTP/RTSP öffnen. Der VideoLAN Client kann mittlerweile sogar als Streaming Server verwendet werden.

Einen Überblick über das VideoLAN-Projekt ermöglicht die Abbildung 20. Um die Module des VLC, die es ermöglichen MPEG-Streams über das Netzwerk zu empfangen und zu decodieren, zu nutzen, muss der Quellcode des Projektes in die vorgegebene Entwicklungsumgebung Visual Studio 2005 eingebunden werden. Dabei stellt sich das Problem, dass beim Framework des VLC Bibliotheken verwendet wurden, die für das Visual Studio nicht zur Verfügung stehen. Darum ist es nicht möglich die VLC-Quellen in der Entwicklungsumgebung zu verändern und zu erstellen. Obwohl VLC ein mächtiges Werkzeug zum Streamen und Decodieren von Videos darstellt, lässt es sich nicht in das Gesamtsystem integrieren. Das Verwenden eines Ausgabefilters im VLC, der Einzelbilder als Dateien auf die Festplatte schreibt, und das anschließende Einlesen dieser Dateien würde zu einer großen Verzögerung zwischen der Aufnahme und der Bearbeitung der Bilddaten führen.

2.2.5 VLC External API

Der VLC-Player bietet eine Möglichkeit, um Videostreams über ein Netzwerk zu empfangen und zu decodieren. Die Entwickler von VLC streben die Definition einer `MediaControlAPI` [vlc07] an, einer generischen Programmierschnittstelle für Media Player. Diese Schnittstelle soll auch von anderen Media Playern verwendet werden können. Sie soll für VLC mit Hilfe der minimalen externen API `libvlc` implementiert werden. Die API soll grundlegende Wiedergabefunktionen wie Play, Pause und Stop ermöglichen. Ferner soll die Funktionalität zum Suchen in Streams und die Ausgabe von Streaminformationen gewährleistet werden.
Für die Bearbeitung von Bildmaterial soll eine Anweisung zum Erstellen von Snapshots in der API enthalten sein. Somit stellt die Verwendung der `MediaControlAPI` ein mögliches Konzept zum Empfang der Videodaten aus dem Hörsaal, dem Decodieren und der Ausgabe von Einzelbildern für die weitere Verarbeitung dar. Wie es sich bei dem Versuch die API zu diesem Zwecke zu nutzen herausgestellt hat, konnte bisher nur die Definition der API und nicht ihre Implementierung ein fortgeschrittenes Stadium erreichen. Sie kann daher im Rahmen des Projektes nicht verwendet werden.

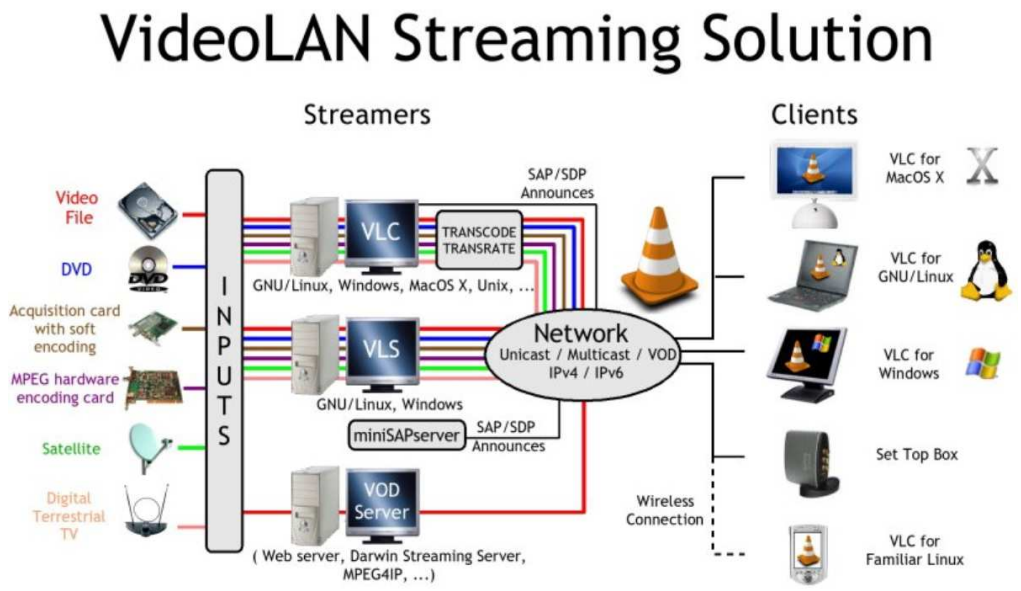


Abbildung 20: Streaming mit VLC

Zudem ist zur Zeit keine ausführliche Dokumentation verfügbar. Es ist allerdings sinnvoll, die Entwicklung der `MediaControlAPI` für eventuelle zukünftige Netzwerkstreaming-Projekte weiterzuverfolgen.

2.2.6 FFMPEG-Projekt

Bei FFmpeg[ffm07] handelt es sich um ein Komplettpaket, um Video- und Audiodaten zu konvertieren, aufzunehmen und zu streamen. Es beinhaltet `libavcodec`, eine führende Audio/Video-Codec-Bibliothek. FFmpeg wurde zwar unter Linux entwickelt, lässt sich aber unter den meisten verbreiteten Betriebssystemen kompilieren. Das Projekt wurde aus verschiedenen Komponenten zusammengestellt. Das Kommandozeilen-Tool `ffmpeg` ermöglicht das Konvertieren von Videoformaten. Außerdem unterstützt es das Grabben und Codieren von einer TV-Karte in Echtzeit. Das `ffserver`-Modul ermöglicht die Live-Übertragung von Multimediadaten per HTTP-Stream. Eine RTSP-Implementierung befindet sich zur Zeit noch in der Entwicklung.
Bei dem `ffplay`-Media Player handelt es sich um ein simpeles Wiedergabeprogramm für Audio- und Videodateien, welches auf den FFmpeg-

Bibliotheken basiert. Die `libavcodec`-Bibliothek beinhaltet alle FFmpeg Audio/Video Encoder und Decoder. Die meisten Codecs wurden von Grund auf neu entwickelt, um die Geschwindigkeit und die Wiederverwendbarkeit zu optimieren. Um die gebräuchlichen Audio/Video-Formate zu parsen und zu generieren, wird die `libavformat`-Bibliothek verwendet. Das Erstellen des FFmpeg-Projekts im Visual Studio verläuft nach einigen wenigen Codeanpassungen erfolgreich. Beim Versuch einen Netzwerkstream anstelle einer Videodatei zu öffnen, stößt man an die Grenzen des Projektes. Da der Umweg, den Stream in kleinen Teilen in eine Datei zu schreiben und anschließend mit FFmpeg auszulesen, mit erheblichen Geschwindigkeitseinbußen verbunden ist, wird davon abgesehen FFmpeg im Rahmen des Framegrabbing zu verwenden. Es ist allerdings sehr gut dazu geeignet, zu Evaluierungszwecken vorhandene Testvideos einzulesen und frameweise zu bearbeiten.

2.2.7 Datenakquisition im Gesamtsystem

Libmpeg2 [lib05] ist eine freie Bibliothek, die das Decodieren von MPEG1 und MPEG2 Videostreams ermöglicht. Bei der Entwicklung wurde großen Wert darauf gelegt für das „main profile“ hunderptrozentige Konformität mit dem MPEG2-Standard zu erreichen. Außerdem wurde großer Aufwand in die Geschwindigkeit der Bibliothek investiert. Beim Decodieren eines üblichen DVD-Videostreams ohne die Bilder auf einem Display auszugeben, können auf einem Pentium III mit 666 MHz bis zu 110 Frames in der Sekunde erreicht werden. Dabei werden weniger als 20 Rechenzyklen für einen Ausgabepixel verwendet. Das heißt, dass in einem auf Libmpeg2 basierendem Media Player, die Display-Routine wahrscheinlich langsamer arbeiten wird als die Decodier-Routine. Obwohl plattformspezifische Optimierungen in Form von Assembler-Routinen genutzt werden, wird darauf geachtet immer eine generische C-Routine als Ersatz bereit zu halten. Dies führt dazu, dass Libmpeg2 unter verschiedensten Architekturen, wie x86, ppc, sparc, arm und sh4, verwendet werden kann. Das Netzwerk-Framegrabbing-Modul nutzt die Libmpeg2, in dem es der Funktion `mpeg2\_parse()` einen Buffer, der mit dem Stream aus dem Netzwerk gefüllt wird, bereitstellt und sie in einer Schleife aufruft. Anhand des zurückgegebenden Zustands lässt sich nun feststellen, ob ein Frame fertig decodiert wurde, oder ob noch weitere Daten in den Buffer gefüllt werden müssen.

```
if (you need a certain acceleration)
      mpeg2_accel(desired_acceleration);
handle=mpeg2_init();
info=mpeg2_info(handle);
```

```
while (data_available) {
        state=mpeg2_parse(handle);
        switch (state) {
        case STATE_BUFFER:
                read some data
                mpeg2_buffer(handle,data_start,data_end);
                break;
        case STATE_SLICE:
        case STATE_END:
                display the frame described in "info"
                mpeg2_convert (mpeg2dec, mpeg2convert_rgb32, NULL);
                fill data structure
                break;
        case STATE_INVALID:
                abort
                break;
        }
}
mpeg2_close(handle);
```

Die Funktion `mpeg2\_init()` initialisiert den Speicherbereich für die Decodierung. Dieser Speicherbereich wird am Ende durch den Aufruf von `mpeg2\_close(handle);` wieder zur Verfügung gestellt. Mit der Funktion `mpeg2\_accel` lässt sich der CPU-Typ für die Beschleunigung festlegen. Wird sie nicht aufgerufen, wird die beste verfügbare Beschleunigung automatisch erkannt. Die Funktion `mpeg2\_info()` erstellt eine Info-Struktur. In diese Struktur werden unter anderem die Sequenz und Einzelbilddaten geschrieben. Die Funktion `mpeg2\_parse()` liest die verfügbaren Daten und gibt den Zustand des Decoders zurück. Befindet sich der Decoder im Zustand `STATE\_BUFFER`, dann wurden alle im Buffer bereitgestellten Daten gelesen. Allerdings reichten sie nicht aus um einen kompletten Frame zu decodieren. In diesem Zustand ist es sinnvoll, mit Hilfe der Funktion `mpeg2\_buffer()` neue Daten aus dem Netzwerk in den Buffer zu füllen. Der Zustand `STATE\_SLICE` signalisiert, dass die letzte Zeile eines Frames erreicht wurde. Ist das Ende des Streams erreicht, befindet sich der Decoder im Zustand `STATE\_END`. In beiden Fällen macht es Sinn, das Display zu aktualisieren, bzw. die Datenstruktur zu füllen. Da der Decoder üblicherweise YUV-Farbwerte bereitstellt, müssen diese noch mit Hilfe der Funktion `mpeg2\_convert` in den RGB-Farbraum überführt werden. Der Zustand `STATE\_INVALID` zeigt an, dass der Buffer keine gültigen MPEG-Daten enthält und dass der Decoder beendet wird. Die vorgestellten Verfahren führen zu einer stabilen und effizienten

Lösung, die Bilddaten zur weiteren Verarbeitung bereitstellt.

3 Kalibrierung und Dewarping

In der Fotografie und in der Kinematorgrafie, benutzt man oft ein Weitwinkelobjektiv, dessen fokale Länge im Wesentlichen kürzer ist als die fokale Länge eines normalen Objektives für die Bildgröße, die durch die Kamera produziert ist. Im Allgemeinen ist das normale Objektiv für ein bestimmtes Format ungefähr der diagonale Abstand von den gegenüberliegenden Ecken der Bildblendenöffnung oder des Fotosensors. Extreme Weitwinkelobjektive, die nicht ein gradliniges Bild produzieren, werden Fisheye Objektive genannt. Die Haupteigenschaft eines Fisheye Objektivs ist, dass es eine Hauptverzerrung verursacht. Dieses kann auf zwei Arten gebrochen werden. Ein kreisförmiges Bild neigt im Allgemeinen dazu, an der breitesten Stelle (6mm) zu brechen und gibt an, wie nah das mögliche 360-Grad-Bild ist. Die extremste Ausprägung ist dabei eine Halbkugel, die angenommen werden kann. Dabei ist es egal, was das Bild darstellt. Von allen vorhandenen Objektiven sind Fisheyes die praktisch am seltensten benutzten Objektive. Anderseits können sie ein großer Spaß sein, um durch aufregende Effekte Leute zu faszinieren.
Konvergierende Vertikalen sind ein allgemeines fotografisches Problem und sie sind ausgeprägter, wenn ein Weitwinkelobjektiv verwendet wird. Durch die Verwendung einer Perspektivlinse kann dieses Problem behoben werden. Die beweglichen Elemente der Linse dienen dazu die konvergierenden Vertikalen zu beheben. Es heißt „photometrische (oder Sensor) Kalibrierung“.

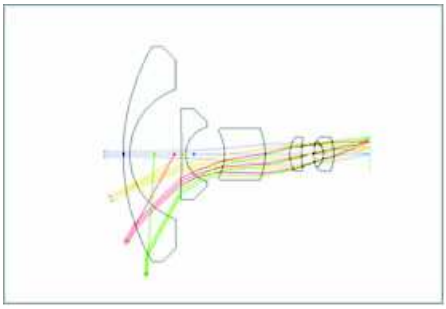


Abbildung 21: Strahlengang eines Superweitwinkelobjektives

Weitwinkelobjektive neigen zu Barrel Verzerrung und Telelinsen neigen zu PinCushion Distortion. Das PinCushion ist häufig seltener wahrnehmbar als Barrel.
Barrel Verzerrung veranlasst die Bilder kissenförmig auszusehen, wobei die Bildseiten sowohl konkav als auch konvex sein können. Viele digitale und

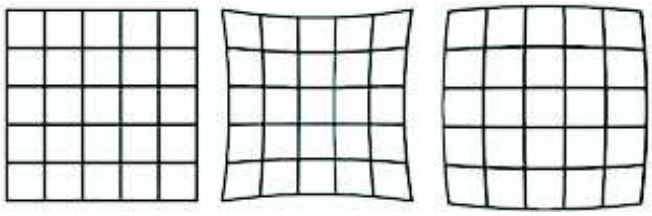


Abbildung 22: Original, PinCushion, Barrel

kompakte Filmkameras leiden unter dieser Art von Verzerrungen, besonders wenn sie ein Weitwinkelobjektiv hat. Die meisten digitalen Kameras haben nicht annehmbare Objektivverzerrungen. Das folgende Bild zeigt dieses unerwünschte Verbiegen, auch Barrel Verzerrung genannt. Ein Weitwinkelobjektiv verzerrt das Bild, indem es Geraden krümmt. Es ist nicht rechtwinklig, weil die Wand des Raumes, wie die Reflexion eines Verzerrungsspiegel im Spiegelkabinett verworfen wird. Alle Weitwinkelobjektive verursachen eine Barrel Verzerrung. Je breiter das Objektiv ist, desto mehr Effekte entstehen. Ein Fisheye Objektiv ist tatsächlich ein super-breites Weitwinkelobjektiv, dass sich in der Barrel Verzerrung unterscheidet.

Abbildung 23: Barell Verzerrung

Um diesen Effekten entgegen zu wirken könnte das aufgenommene Bild wiederum entzerrt werden. Für diese Entzerrung, nun folgend Dewarping genannt, muss zunächst die Verzerrung ermittelt werden. Dafür werden Aufnahmen von Schachbrettmustern oder ähnlichen Mustern gemacht, siehe Abbildung 24. Die Abstände und die Positionen der Ecken der Kästchen auf den Abbildungen sind konstant und bekannt. Diese können dann mit den Ecken der Testmuster auf den aufgenommenen Bildern verglichen werden. Ist die

Verzerrung einmal bekannt, kann jedes weitere aufgenommene Bild dewarped werden, so dass bei weiterer Verarbeitung der Bilder die Verzerrung der Linse keine Rolle mehr spielt.

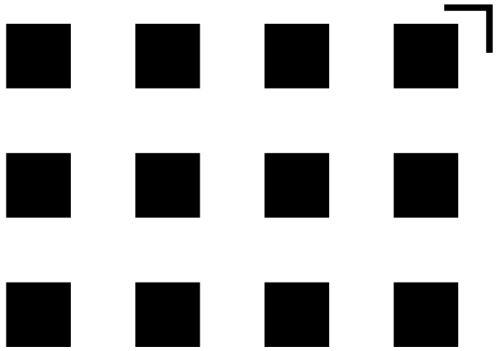


Abbildung 24: Testmuster

3.1 Dewarping

3.1.1 Eckpunkte des Testmusters finden

Als erster Arbeitsschritt beim Dewarping müssen zunächst einmal die Eckpunkte der Testmusterkästchen in den aufgenommenen Testbildern gefunden werden. Dazu gibt es mehrere Möglichkeiten, die von der PG 498 untersucht und benutzt worden sind. Dabei ist meist eine Methode eine Verbesserung der vorherigen.

OpenCV stellt die Methode `cvFindChessboardCorners` bereit, mit der angeblich automatisch die Eckpunkte eines Testmusters gefunden werden können. Diese Methode hat jedoch keinerlei verwertbare Resultate geliefert und wurde daher nicht weiter verwendet.

Bei der ersten funktionierenden einfachen Methode muss der Benutzer des Dewarping-Tools die Kalibrierungspunkte mit der Maus selber anklicken. Diese Methode ist sehr aufwändig und ungenau.

Als Verbesserung der vorherigen Methode kann mit OpenCV die Genauigkeit erhöht werden. Es werden zunächst wieder alle Messpunkte angeklickt und zwischen gespeichert. Nun wird mit der Methode `cvFindCornerSubPix` in der Umgebung jedes Punktes nach einer besseren Ecke gesucht. Diese Methode zeigt sehr gute Resultate, es werden alle Messpunkte richtig gefunden. Der Aufwand ist jedoch immer noch erheblich.

Um eine einfachere Handhabbarkeit zu erhalten, wurde diese Methode weiterentwickelt. Der Benutzer muss nun lediglich die vier Eckpunkte anklicken, welche ein Viereck aufspannen, in dem das Testmuster liegt. Da die Anzahl der Messpunkte zuvor bekannt ist, können erwartete Positionen für die Testpunkte berechnet werden. Sobald dies geschehen ist, kann wieder mit der OpenCV-Methode nach den wirklichen Positionen gesucht werden. Da ein rechteckiges Viereck zu erwarten ist, kann sogar mit drei Mausklicks das gewünschte Ergebnis erzielt werden. Damit ist der Aufwand um einen sehr großen Faktor verringert worden. Die Genauigkeit ist fast so gut wie bei der vorherigen Methode. Sie ist nur etwas schlechter, da es vorkommen kann, dass die vorberechneten Punkte zu weit von den tatsächlichen entfernt sind und diese dann nicht gefunden werden. Dennoch ist es einfacher das Verfahren noch einmal anzuwenden, als alle Testpunkte von Hand anzuklicken.

3.1.2 Originalpunkte eingeben

Zur Bestimmung der Verzerrung müssen nicht nur die verzerrten Punkte gefunden werden, sondern natürlich auch die Position der Punkte auf dem unverzerrten Testmuster.

Die aufwändigste und naivste Methode besteht daraus, alle Punkte hardcoded einzugeben. Dies ist natürlich nicht anpassungsfähig und liefert so schnell schlechte Ergebnisse.

Als Verbesserung der naiven Methode kann ein Rechteck durch vier Mausklicks angegeben werden. Da die Anzahl der Punkte und deren Abstände bekannt sind, kann aus den angeklickten Punkten ein Raster berechnet werden. Dies entspricht den originalen Punkten.

Zur noch besseren Vergleichbarkeit zwischen den Original- und Testpunkten wird das Originalbild auf die Größe und die Position des Testbildes affin transformiert. Nach dieser Transformation werden mit der gleichen Methode wie für die Testpunkte die Punkte gefunden.

3.2 Berechnung der Punkte im Gesamtsystem

Um die Kamerakalibrierung berechnen zu können benötigt man die Kalibrierungspunkte des in Abschnitt 6.5 erstellten Kalibrierungsmusters und dessen auf die Leinwand projiziertes Abbild. Dazu wählt der Benutzer erst das originale und dann das aufgenommene Muster aus. In beiden Bildern wird dann gemäß unten aufgeführtem Ablauf nach korrespondierenden Punkten gesucht.

Die Bestimmung der Kalibrierungspunkte basiert auf einer Kantendetektion mit Hilfe des Sobel Operators. Die Erklärung zur Kantendetektion mit Hilfe des Sobel Operators ist in Kapitel 4.1.5 zu finden. Zuerst wird das ursprüngliche Bild I_{normal} kopiert und invertiert. Das invertierte Bild sei im Folgenden mit I_{invert} bezeichnet. Im nächsten Schritt wird auf beiden Bildern der Sobel Operator angewandt, einmal in horizontaler $S_{horizontal}$ und einmal in vertikaler $S_{vertikal}$ Richtung. Man erhält somit vier neue Bilder.

1. $I_{snh} := I_{normal} * * S_{horizontal}$
2. $I_{snv} := I_{normal} * * S_{vertikal}$
3. $I_{sih} := I_{invert} * * S_{horizontal}$
4. $I_{siv} := I_{invert} * * S_{vertikal}$

Aus diesen berechnet man nun ein Gesamtbild I_{gesamt}.

$$I_{gesamt} := I_{snh} + I_{snv} + I_{sih} + I_{siv} \tag{4}$$

In I_{gesamt} wird nun die Menge M an Positionen gesucht, für die gilt:

$$\begin{aligned} M := \{(i,j) | & I_{gesamt}(i+1,j) > 100 \wedge I_{gesamt}(i-1,j) > 100 \\ & \wedge I_{gesamt}(i,j+1) > 100 \wedge I_{gesamt}(i,j-1) > 100\}. \end{aligned} \tag{5}$$

Der Wert 100 kommt daher, da es bei dem aufgenommenen Bild durch Kompression und aufgrund von wechselnden Lichtverhältnissen zu Artefakten kommt, sodass in den Kantenbilder nicht mehr nur die Werte 0 (keine Kante) und 255 (Kante) vorkommen.

Da die Suche nach den Positionen der Menge M zeilenweise, von oben nach unten, abläuft, kann man sich die Anzahl nh der in einer Zeile und die Anzahl nv der in einer Spalte vorkommenden Positionen merken. Diese Anzahlen sind für die Suche nach den korrespondierenden Punkten im aufgenommenen Kalibrierungsbild wichtig. Zum aufgenommenen Kalibrierungsbild I'_{normal} wird analog zu I_{normal} die Menge M' bestimmt. Als Ergebnis erhält man zwei Mengen M und M' sowie die Anzahlen nh und nv. Um nun die korrespondierenden Punkte zu erhalten wird die Menge M' erst nach den Y-Koordinaten sortiert. Danach werden immer nh große Abschnitte nach den X-Koordinaten sortiert. Nun enthalten beide Mengen, die korrespondierenden Punkte in der gleichen Reihenfolge und man kann die Kameraparameter bestimmen.

3.2.1 Radiale Basisfunktion

Die Radiale Basisfunktion (RBF) ist ein wichtiger Bestandteil des Dewarpings [RNM95]. Daher wird auf sie nun kurz näher eingegangen.
Eine Radiale Basis Funktion ist eine Realwertfunktion, deren Wert nur vom Abstand zum Ursprung abhängt:

$$p(\mathbf{x}) = \phi(\|\mathbf{x}\|), \mathbf{x} \in \mathbb{R}^2. \tag{6}$$

oder wechselweise von dem Abstand eines anderen Punktes $\mathbf{c} \in \mathbb{R}^2$, genannt Mitte.

$$p_c(\mathbf{x} - \mathbf{c}) = \phi(\|\mathbf{x} - \mathbf{c}\|) \tag{7}$$

Die Norm $\|\|$ ist normalerweise der euklidische Abstand. Radiale Basis Funktionen werden gewöhnlich verwendet, um näherungsweise Funktionen der folgenden Form aufzubauen:

$$y(\mathbf{x}) = \sum_{i=1}^{N} \lambda_i \cdot \phi(\|\mathbf{x} - \mathbf{c_i}\|). \tag{8}$$

Die folgende Tabelle gibt eine Übersicht über verschiedene Radiale Basis Funktionen ϕ(mit $r = \|\mathbf{x} - \mathbf{z}\|, \mathbf{x}, \mathbf{z} \in \mathbb{R}^2$):

Piecewise smooth		
MN	monomial	$\|r\|^{2m+1}$
TPS	thin plate spline	$\|r\|^{2m+} ln \|r\|$
Infinitely smooth		
MQ	multiquadric	$\sqrt{1 + (\epsilon r)^2}$
IQ	inverse quadratic	$\frac{1}{1+(\epsilon r)^2}$
IMQ	inverse MQ	$\frac{1}{\sqrt{1+(\epsilon r)^2}}$
GA	Gaussian	$e^{-(\tau r)^2}$

Die unbekannten Koeffizienten λ_i aus Geichung 8 können mit Hilfe des folgenden linearen Gleichungssystems bestimmt werden.

$$\begin{pmatrix} \phi(\|\mathbf{x}_1 - \mathbf{x}_1\|) & \dots & \phi(\|\mathbf{x}_1 - \mathbf{x}_n\|) \\ \vdots & \ddots & \vdots \\ \phi(\|\mathbf{x}_n - \mathbf{x}_1\|) & \dots & \phi(\|\mathbf{x}_n - \mathbf{x}_n\|) \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \tag{9}$$

Mit Hilfe dieser Koeffizient kann man nun die richtigen Positionen für die Pixel des verzerrten Bildes ausrechnen.

Zur Durchführung des Dewarping werden die Originalpunkte $\mathbf{x}_{i,org}$ des Testmusters und die verzerrten Punkte $\mathbf{x}_{i,dwp}$ im aufgenommenen Kamerabild mit Hilfe der zuvor beschriebenen Methoden bestimmt. Danach muss der Abstand $r_i = \|\mathbf{x}_{i,org} - \mathbf{x}_{i,dwp}\|$ ausgerechnet werden.

Gesucht ist nun eine Funktion $\mathbf{y} : \mathbb{R}^2 \to \mathbb{R}^2$ mit $\mathbf{y}(\mathbf{x}_{i,dwp}) = \mathbf{x}_{i,org}$. Setzt man nun den Abstand r_i in die radiale Basisfunktion ein, so erhält man folgendes Gleichungssystem:

$$\mathbf{y}(\mathbf{x}_{i,dwp}) = \sum_{i=1}^{N} \lambda_i \cdot \phi(r_i). \tag{10}$$

Damit können die Koeffizienten analog zu Gleichung 9 ausgerechnet werden. Bisher wurden die Punkte $\mathbf{x}_{i,org}$ und $\mathbf{x}_{i,dwp}$ benutzt, um die Koeffizienten λ_i zu berechnen . Nun wird noch mal die radiale Funktion benutzt, um das Dewarping für jeden Bildpunkt des aufgenommenen Bildes vorzunehmen. Für alle $N \times M$ Punkte $\mathbf{x}$ des Bildes ergibt sich die neue Position $\mathbf{y}$ durch

$$\mathbf{y} = \sum_{i=1}^{N} \lambda_i \cdot \phi(\|\mathbf{x} - \mathbf{x}_{i,org}\|) \tag{11}$$

mit den zuvor bestimmten λ_i. Durch Verschieben der Pixel in dem verzerrten Bild um den Abstand zwischen der korrekten Position und Position im verzerrten Bild entsteht das Dewarping-Bild.

3.2.2 Interpolation des korrigierten Bildes

Nach dem Dewarping kann das verbesserte Bild Störungen aufweisen. Diese können mittels Interpolation beseitigt werden. Daher wird nun kurz allgemein auf Interpolation eingegangen.

Interpolation ist der Prozess des Definierens einer Funktion, die spezifizierte Werte an spezifizierten Punkten annimmt. In mathematischen Unterbereichen der numerischen Analysis ist Interpolation eine Methode des Konstruierens von neuen Datenpunkten aus einen getrennten Satz bekannter Datenpunkte. In Wissenschaft und Technik hat man häufig eine Anzahl von Datenpunkten, die durch Experimente gemessen werden. Man versucht eine Funktion zu konstruieren, die diese Datenpunkte am besten beschreibt. Dies

wird Kurvenannäherung (curve fitting) genannt. Interpolation ist ein Spezialfall der Kurvenannäherung, in dem die Funktion die Datenpunkte genau durchlaufen muss. Ein anderes Problem, das mit der Interpolation zusammenhängt, ist die Approximation einer komplizierten Funktion durch eine einfache Funktion. Wir nehmen an, dass wir die Funktion kennen, aber sie zu kompliziert ist, um effizient ausgewertet werden zu können. Dann können wir einige bekannte Datenpunkte von der komplizierten Funktion auswählen und eine Nachschlagtabelle (Look-Up-Table) erstellen, um jeden Datenpunkt zu interpolieren und dadurch eine einfachere Funktion zu konstruieren. Wenn wir die einfache Funktion verwenden, um neue Datenpunkte zu errechnen, erhalten wir normalerweise nicht das gleiche Resultat wie mit der ursprünglichen Funktion. Abhängig von dem Problemgebiet und der Interpolationsmethode überwiegt der Gewinn der Einfachheit gegenüber der Störung.

Lineare Interpolation

Eine einfache Methode ist die lineare Interpolation. Im Allgemeinen arbeitet die lineare Interpolation mit zwei Datenpunkten (x_0, y_0) und (x_1, y_1).
Es sollen jetzt Punkte auf der Linie zwischen den beiden Datenpunkten mit

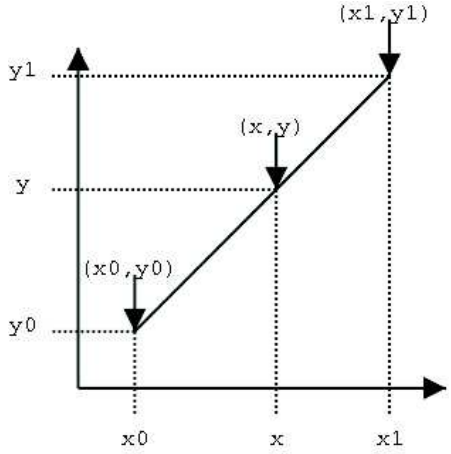


Abbildung 25: Lineare Interpolation

einem gegebenen x im Intervall $[x_0, x_1]$ ausgewählt werden.
Um dieses x zu finden wird der Strahlensatz benutzt. Dadurch wird folgende Formel aufgestellt:

$$\frac{y - y_0}{y_1 - y_0} = \frac{x - x_0}{x_1 - x_0}. \tag{12}$$

Der Interpolationskoeffizient α ist auf beiden Seiten der Gleiche.

$$\alpha = \frac{x - x_0}{x_1 - x_0} \tag{13}$$

$$\alpha = \frac{y - y_0}{y_1 - y_0}. \tag{14}$$

Der Wert für y lässt sich nun durch folgende Linearkombination beschreiben:

$$y = (1 - \alpha) \cdot y_0 + \alpha \cdot y_1. \tag{15}$$

Der Interpolationswert berechnet sich nun folgendermaßen:

$$y = y_0 + \frac{(x - x_0) \cdot (y_1 - y_0)}{(x_1 - x_0)}. \tag{16}$$

In Abbildung 25 wird dies graphisch dargestellt.

Bilineare Interpolation

In der Mathematik ist die bilineare Interpolation eine Erweiterung der linearen Interpolation für Funktionen mit zwei Variablen. Die Idee ist, lineare Interpolation zuerst in eine Richtung und dann in die andere Richtung durchzuführen.
Gegeben seien vier Punkte, die aneinander angrenzen $q_{11}(x_1, y_1)$, $q_{12}(x_1, y_2)$, $q_{21}(x_2, y_1)$, $q_{22}(x_2, y_2)$, gesucht ist eine Approximation für die Werte $f(x, y)$, mit $x_1 \leq x \leq x_2$, $y_1 \leq y \leq y_2$. Zuerst wird eine lineare Interpolation in x-Richtung ausführt. Es ergeben sich zwei Hilfswerte $f(\mathbf{r_1})$ und $f(\mathbf{r_2})$ der Hilfspunkte $\mathbf{r_1} = (x, y_1)$, $\mathbf{r_2} = (x, y_2)$:

$$f(\mathbf{r_1}) \approx \frac{x_2 - x}{x_2 - x_1} f(q_{21}) + \frac{x - x_1}{x_2 - x_1} f(q_{11}) \tag{17}$$

$$f(\mathbf{r_2}) \approx \frac{x_2 - x}{x_2 - x_1} f(q_{12}) + \frac{x - x_1}{x_2 - x_1} f(q_{11}). \tag{18}$$

Mit Hilfe der so errechneten Werte erfolgt nun eine lineare Interpolation in y-Richtung:

$$f(x, y) \approx \frac{y_2 - y}{y_2 - y_1} f(\mathbf{r_1}) + \frac{y - y_1}{y_2 - y_1} f(\mathbf{r_2}). \tag{19}$$

Nun haben wir das gewünschte Ergebnis für $f(x, y)$:

$$\begin{aligned} f(x,y) \approx & \frac{(x_2 - x)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} f(q_{11}) \\ & + \frac{(x - x_1)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} f(q_{21}) \\ & + \frac{(x_2 - x)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} f(q_{12}) \\ & + \frac{(x - x_1)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} f(q_{22}). \end{aligned} \tag{20}$$

Interpolation eines digitalen Bildes

Die Interpolation ist eine Methode, zur Erhöhung der Pixelanzahl in einem digitalisierten Bild. Einige digitale Kameras verwenden Interpolation, um ein größeres Bild als das aufgenommene zu produzieren oder digitalen Zoom zu realisieren. Die Software der Bildbearbeitung unterstützt eine oder mehrere Methoden der Interpolation. Wie glatt Bilder vergrößert werden - ohne ausgefranste Ränder, sogenannte Jaggies -, hängt von der Anwendung des richtigen Algorithmus ab. Die Methoden, die oft benutzt werden, sind:

- Nächste Nachbarinterpolation (Nearest Neighbor Interpolation)
- Bilinear Interpolation

Nachbarinterpolation ist die einfachste Methode. Der Algorithmus wählt einfach den Wert des nächsten Punktes aus und betrachtet die Werte der anderen Punkte nicht. Der Algorithmus ist sehr einfach durchzuführen, und wird normalerweise zusammen mit dem Mipmapping in Realzeit-3D Anwendungen verwendet. Der Algorithmus überträgt Farbwerte für eine Texturoberfläche.

Ein Beispiel:
Abbildung 26 zeigt das Originalbild in einer Auflösung von 106 × 40 Pixel. Das Aussehen nach 450-facher Vergrößerung mit Hilfe der Nachbarinterpo-

Abbildung 26: Originalbild 106 × 40

lation zeigt Abbildung 27:

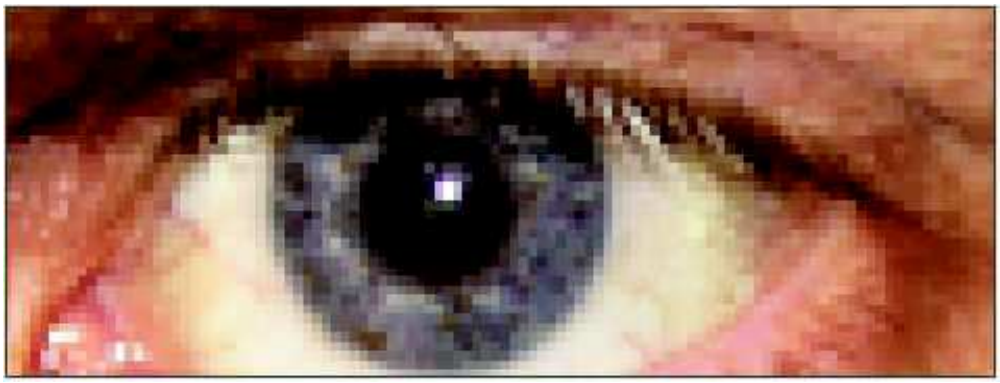


Abbildung 27: 450 fache Vergrößerung

Die meisten Bildbetrachtungs- und Renderingprogramme benutzen diese Art der Interpolation, um ein digitalisiertes Bild für genauere Prüfung zu vergrößern, weil sie nicht die Farbinformationen des Bildes ändern und kein Anti-Aliasing unterstützen. Aus dem gleichen Grund wird es nicht verwendet, um Fotos zu vergrößern, weil sie die Anzahl von Jaggies erhöhen. Die bilineare Interpolation berechnet den Wert eines neuen Pixels, durch den gewichteten Mittelwert der 4 Pixel (in einem 2×2 Block) in der nächsten Nachbarschaft. Die Mittelwertbildung hat einen Anti-Aliasingeffekt und produziert folglich verhältnismäßig glatte Ränder mit kaum sichtbaren Jaggies.

Abbildung 28: 450 fache Vergrößerung bilineare Interpoliert

Nach dem Dewarping erschienen Streifen in dem verbesserten Bild. Diese müssen noch entfernt oder zumindest vermindert werden. Zuerst muss man bestimmen, welche Pixel zu interpolieren sind. Wir haben schon jedes Bild, welches Dewarping braucht, initialisiert. Wenn der Pixel noch den Initialwert

nach dem Dewarping hat, so muss dieser interpoliert werden. Hier wird die bilineare Interpolation genutzt:
Es werden die vier nächsten Nachbarn untersucht. Dabei wird der Durchschnitt für jede Farbe ausgerechnet. Der Durchschnittswert ist der Wert, der für die Interpolation gebraucht wird.

3.3 Einsatz von Dewarping im Gesamtsystem

Das entwickelte Dewarpingsystem wird an zwei Stellen eingesetzt.
Im Serverteil der Anwendung müssen die aufgenommen oder empfangenen Bilder der Kameras dewarped werden, falls diese starke Verzerrungen beinhalten. Dewarping ist immer an eine spezielle Kamera und an deren Einstellungen angepasst. Es gibt also keine universellen Dewarpingeinstellungen. Daher muss für jede Kamera, die benutzt werden soll, ein Dewarping und Kalibrierungsprofil angelegt werden. Dies geschieht in den Konfigurationseinstellungen des Thresholdtools, welches zusätzlich zum Hauptprogramm von der PG 498 entwickelt worden ist.

4 Bildverbesserung

4.1 Digitale Filteroperatoren und Methoden zur Bildverbesserung

Wie bereits bei der Kalibrierung und dem Dewarping (vgl. Kapitel 3) der Fall, sollen die Methoden der Bildverbesserung weitere Probleme beheben, die durch die Aufnahme von Kameras entstehen. In erster Linie handelt es sich hierbei um Signalrauschen. Im Unterkapitel 4.1 werden dazu verschiedene Filteroperationen vorgestellt, um dieses zu beheben. Ferner geht die Bildverbesserung allerdings schon einen Schritt weiter und bereitet die Frames des Videostreams für die Segmentierung vor. Dabei ist es nun nicht mehr entscheidend, ein Frame objektiv zu verbessern, sondern bereits subjektiv in Hinblick auf eine Aufgabe zu verändern. Diese Funktion übernimmt der Sobel-Filter (siehe 4.1.5). Um einen Filter effizient durchzuführen, bedarf es verschiedener Hilfsfunktionen. Außerdem müssen diverse Analysen durchgeführt werden, um einen möglichst optimalen Ablauf zu garantieren. Diese werden im zweiten Teil dieses Kapitels genauer durchleuchtet (ab 4.2).

Reduktion des Signalrauschens

Bei der Suche nach einem kleinen roten Laserpunkt auf einer Leinwand ist es enorm wichtig, dass keine ungewollten Rauschflecken die Suche behindern. Diese Flecken, die durch technische Gegebenheiten der Kameras entstehen, können durch Weichzeichnungsfilter ausgeglichen werden. Dabei gilt es aber zu berücksichtigen, dass möglichst wenige Originalsignale verloren gehen. Für unsere Analysen haben wir daher aus Test-Frames drei verschiedene Bilder erstellt. Zunächst das Originalbild. Davon wurde ein absichtlich verrauschtes Bild erstellt, welches daraufhin wieder gefiltert werden konnte. Ein Rauschfilter wurde von uns bevorzugt, wenn sein gefiltertes Bild dem Originalbild näher kam, als das von anderen Filtern. Um dies vergleichen zu können und den möglichst optimalen zu finden, haben wir zwei unterschiedliche Testmethoden umgesetzt.

Signal-to-Noise-Ratio (SNR)

Die ersten Tests unserer Filter wurden mit der Signal-to-Noise-Ratio (SNR) bewertet. Dieser Wert errechnet sich aus der nachfolgenden Formel:

$$SNR = \frac{Nutzsignalleistung}{Rauschleistung} \tag{21}$$

In der Umsetzung muss man sich nun überlegen, wie man am effektivsten die Nutzsignalleistung und die Rauschleistung errechnet, um gut vergleichbare Ergebnisse zu erhalten. Die Werte entsprachen bei uns daher:

$$\begin{aligned} Nutzsignalleistung &= \sum_{Pixel} (Farbwert^2_{gefiltertesBild}) \\ Rauschleistung &= \sum_{Pixel} ((Farbwert_{gefiltertesBild} - Farbwert_{Originalbild})^2) \end{aligned} \tag{22}$$

Die Quadrierung verstärkt dabei das Ergebnis und gleicht zudem in der Rauschleistung potentielle Vorzeichenfehler aus. Der Bruch wird kleiner, sobald die Zahl im Nenner größer wird. Daher ist das Ergebnis der SNR besser, wenn ihr Wert größer als bei anderen ist.

Root-Mean-Square (RMS)

Die Root-Mean-Square (RMS) verfolgt einen leicht anderen Ansatz als die SNR. Es war uns wichtig auch diesen zweiten alternativen Vergleichswert bei der Bewertung der Filter heranzuziehen:

$$RMS = \sqrt{\frac{1}{Pixelanzahl} \cdot Rauschleistung} \tag{23}$$

Da hier die Rauschleistung nicht mehr in einem Nenner erscheint, ist das Ergebnis der RMS im Gegensatz zu dem der SNR besser, je geringer sein Wert ist.

4.1.1 Blurfilter

Als einfachsten Glättungsfilter haben wir einen simplen Blurfilter (Boxfilter) getestet, welcher zur Klasse der linearen Filter gehört [GW01]. Das Originalbild aus dem Hörsaal, zu sehen in Abbildung 29, haben wir mit weißem Rauschen gestört (siehe Abbildung 30) und dieses dann als Testbild für unsere Filter verwendet.

Mittels OpenCV haben wir hier das Bild jeweils mit einer quadratischen $N_F \times N_F$ Maske mit $N_F \in \{3, 5\}$ filtern lassen, wobei sich der betroffene Pixel im Zentrum aus den jeweiligen Werten seiner Umgebung wie folgt errechnete:

$$\mathbf{p} = \frac{1}{N_F^2} \sum_{i=0}^{(N_F-1)^2} \mathbf{p}_i \tag{24}$$

Abbildung 29: Originalbild aus dem Hörsaal

Abbildung 30: Originalbild und Bild mit hinzugefügtem weißen Rauschen

$$\begin{pmatrix} 255 & 133 & 0 \\ 144 & \mathbf{199} & 12 \\ 0 & 14 & 89 \end{pmatrix} \Rightarrow \begin{pmatrix} 255 & 133 & 0 \\ 144 & \mathbf{94} & 12 \\ 0 & 14 & 89 \end{pmatrix} \tag{25}$$

Die Ergebnisse nach dem Filtern sind wie erwartet ein wenig unscharf. Die SNR/RMS-Werte betragen für den 3×3 Blurfilter *33.098187 / 21.144004* und für den 5×5 Filter *39.298923 / 19.246620*.

Abbildung 31: Verrauschtes Eingabebild gefiltert mit 3×3 Blurfilter

Abbildung 32: Verrauschtes Eingabebild gefiltert mit 5×5 Blurfilter

4.1.2 Gaussfilter

Der Gaussfilter ist ein weiterer linearer Glättungsfilter [GW01]. Hierbei gehen die Werte der Nachbarpunkte des betroffenen Pixels anhand einer zweidimensionalen Gausglocke gewichtet in die Berechnung des neuen Wertes ein. Die Standardabweichung σ bestimmt hierbei die Stärke der Glättung. Die Gaussverteilung für einen Filter der Größe $N_F \times M_F$ mit Standardabweichung σ^2 wird bestimmt durch:

$$G(n,m) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{n^2+m^2}{2\sigma^2}\right) \quad \text{mit } n \in \{0,\dots N_F-1\}, m \in \{0,\dots M_F-1\} \tag{26}$$

Abbildung 33: Verrauschtes Eingabebild gefiltert mit 3×3 Gaussfilter

Unser Testbild haben wir ebenfalls mittels der OpenCV-Methode `void cvSmooth()` mit einem 3×3 und einem 5×5 Gaussfilter gefiltert. Die Ergebnisbilder werden in Abbildung 33 und 34 dargestellt. Hierbei ergeben sich als SNR/RMS-Werte für den 3×3 Filter *30.450860 / 22.107016* sowie *38.630167 / 19.505833* bei dem 5×5 Gaussfilter.

Abbildung 34: Verrauschtes Eingabebild gefiltert mit 5×5 Gaussfilter

4.1.3 Medianfilter

Der Medianfilter [GW01] ist ein nichtlinearer Filter aus der Gruppe der *Rangordnungsfilter*. Bei Rangordnungsfiltern werden die Grauwerte aus der Umgebung eines zu bearbeitetenden Pixels, welches sich im Zentrum dieser Umgebung befindet, und dessen eigener Grauwert in einer Liste der Größe nach sortiert. Beim Medianfilter ist der neue Wert des Pixels dann der Wert aus der Mitte dieser sortierten Liste und dieser ersetzt somit den alten Pixelwert. Allgemein bedeutet dies, dass bei einer Filtergröße von $N_F \times M_F$ der $\frac{N_F \cdot M_F}{2}$-te Wert aus der sortierten Liste der Größe $N_F \cdot M_F$ gewählt wird. Beispiel:

$$\begin{pmatrix} 255 & 133 & 0 \\ 144 & \mathbf{199} & 12 \\ 0 & 14 & 89 \end{pmatrix} \Rightarrow \begin{pmatrix} 255 & 133 & 0 \\ 144 & \mathbf{89} & 12 \\ 0 & 14 & 89 \end{pmatrix} \tag{27}$$

da (0,0,12,14,**89**,133,144,199,255).

Unter Zuhilfenahme der OpenCV-Methode `void cvSmooth()` wurde der Medianfilter jeweils mit einer 3×3 und einer 5×5 Umgebung getestet.Die SNR/RMS-Werte betrugen bei dem 3×3 Medianfilter für das Bild in Abbildung 35 *241.226305 / 7.271947* und für den 5×5 Filter aus Abbildung 36 *179.917638 / 8.392621*. Wie man gut sehen kann, werden beim Medianfilter alle Strukturen entfernt, bei denen die Ausdehnung in der Filterumgebung

weniger als $\frac{N_F \cdot M_F}{2}$ Pixel beträgt. Hierdurch werden Punkte, Linien und Ecken entfernt, Kanten von hell auf dunkel bleiben jedoch erhalten.

Abbildung 35: Verrauschtes Eingabebild gefiltert mit 3×3 Medianfilter

Abbildung 36: Verrauschtes Eingabebild gefiltert mit 5×5 Medianfilter

4.1.4 Bilateraler Filter

Bilaterales Filtern [TM98] gehört ebenfalls zu den nichtlinearen Filtermethoden und bietet gegenüber den bereits vorgestellten Verfahren den Vorteil, dass Rauschen (kleine Frequenzen) unterdrückt wird, aber scharfe Übergänge an Ecken und Kanten (hohe Frequenzen) erhalten bleiben. Der neue Pixelwert errechnet sich bei diesem Verfahren durch eine gewichtete Mittelwertbildung der Nachbarpixel wie auch schon bei den anderen Filtern. Das Gewicht besteht hier allerdings aus zwei Komponenten, der geometrischen Distanz und der photometrischen Ähnlichkeit. Nahegelegene Pixel sowie Pixel mit ähnlichem Bildwert fließen somit stärker in die Berechnung des neuen Wertes mit ein. Das Ergebniss des Filters kann man in Abbildung 37 sehen. Als SNR/RMS-Werte ergeben sich hier *33.110507 / 21.139700*.

Abbildung 37: Verrauschtes Eingabebild gefiltert mit biliteralem Filter

4.1.5 Sobelfilter

Zur Detektion von Kanten im Bild haben wir versucht einen Sobelfilter zu implementieren. Durch Anwendung des Sobel-Operators [GW01] erhält man ein Gradientenbild, in welchem ausschließlich die Übergänge von hell auf dunkel und umgekehrt als Kanten im Bild gezeichnet werden.

1	0	-1
2	0	-2
1	0	-1

-1	-2	-1
0	0	0
1	2	1

Abbildung 38: Sobel-Operator für vertikale und horizontale Kanten

Um eine effiziente Filterung mit dem Sobelfilter zu ermöglichen haben wir diese im Ortsfrequenzbereich durchgeführt, wodurch die Rechnung auf eine Matrixmultiplikation von dem Filter mit dem Bild reduziert wird. Hierzu haben wir das Eingabebild aus der Pic-Datenstruktur in ein Grauwertbild umgewandelt und dieses in die fftw-complex Datenstruktur geladen. Anschließend wurde das Bild mittels der fftw-Bibliothek fouriertransformiert und zentriert. Auch der Filter wurde transformiert und auf die Größe des zu filternden Bildes vergrößert indem er horizontal und vertikal aneinander gespiegelt wird.

...	...	...	1	0	-1	...	...	...
...	...	...	2	0	-2	...	...	...
...	...	...	1	0	-1	...	...	...
-1	0	1	**1**	**0**	**-1**	-1	0	1
-2	0	2	**2**	**0**	**-2**	-2	0	2
-1	0	1	**1**	**0**	**-1**	-1	0	1
...	...	...	1	0	-1	...	...	...
...	...	...	2	0	-2	...	...	...
...	...	...	1	0	-1	...	...	...

Abbildung 39: Methode zur Vergrößerung des Sobel-Operators

Der vergrößerte Filter wird dann ebenfalls per fftw transformiert. Die Filterung im Ortsfrequenzbereich wird dann durch die Multiplikation des Filters (A) mit dem Bild (B) durchgeführt. Das gefilterte Bild (C) ergibt sich dann durch den unten angegebenen Rechenweg, wobei X_{real} die Real- und X_{imag} die jeweiligen Imaginärteile darstellen:

$$\begin{aligned} C = A \cdot B &= (A_{real} + i \cdot A_{imag}) \cdot (B_{real} + i \cdot B_{imag}) \\ &= (A_{real} \cdot B_{real}) + (A_{real} \cdot i \cdot B_{imag}) + (i \cdot A_{imag} \cdot B_{real}) + (i^2 \cdot A_{imag} \cdot B_{imag}) \\ &= ((A_{real} \cdot B_{real}) - (A_{imag} \cdot B_{imag})) + ((i \cdot A_{real} \cdot B_{imag}) + (i \cdot A_{imag} \cdot B_{real})) \\ &= C_{real} + i \cdot C_{imag} \end{aligned} \tag{28}$$

Die neuen Werte wurden nach der Multiplikation in die jeweiligen Variablen C_{real} und C_{imag} der fftw-Datenstruktur gespeichert. Dann haben wir

das gefilterte Bild zurücktransformiert und die Farbwerte skaliert. Durch die Filterung erhält man das Bild aus Abbildung 40, indem ausschliesslich die im Ausgangsbild vorhandenen horizontalen und vertikalen Kanten zu sehen sind.

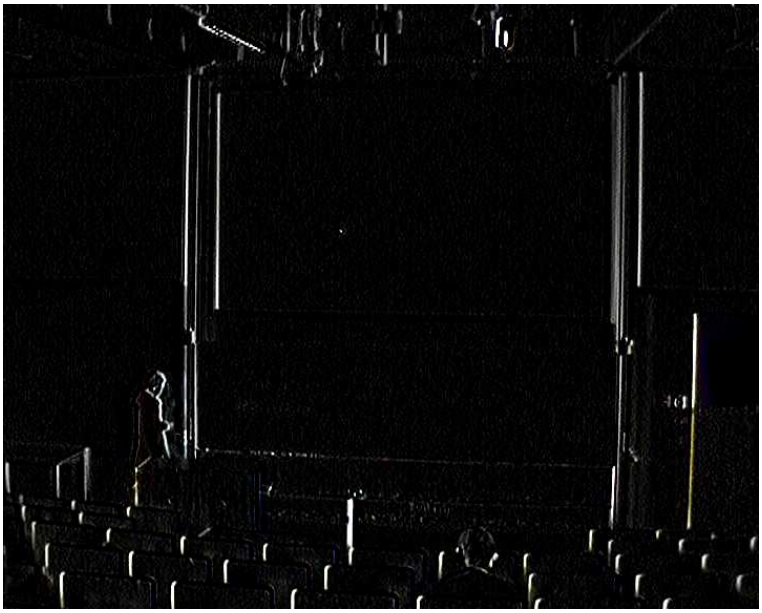


Abbildung 40: Originalbild mit Sobeloperator gefiltert

4.2 Diskrete Fourier-Transformation (DFT)

Für eine effiziente Implementierung der in Kapitel 4.1 erwähnten Filter braucht es optimierte Methoden der Bildverarbeitung. Damit eine möglichst echtzeitnahe Umsetzung erfolgen kann, haben wir uns für den Einsatz der *diskreten Fourier-Transformation (DFT)* entschieden. Diese überträgt die Bilddaten vom Ortsraum in den Frequenzraum, welcher einen klaren Geschwindigkeitsvorteil bei der Verrechnung der Filterkerne bietet. Dabei ist die DFT an das Fourier-Theorem angelegt, welches besagt, dass sich jede beliebige periodische Funktion als Summe von $cos(x)$ und $sin(x)$ Funktionen unterschiedlicher Frequenzen darstellen lässt. Zwei Komponenten $cos(x)$ und $sin(x)$ können ihrerseits wiederum eine komplexe Zahl z in Polarkoordinaten darstellen:

$$z = exp(i \cdot \phi) = cos(\phi) + i \cdot sin(\phi), \quad \text{mit } i = \sqrt{-1} \tag{29}$$

Bei einem Bild der Größe $M \times N$ arbeitet die DFT somit nach folgender

Formel:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(m,n) \cdot exp(-i \cdot 2 \cdot \pi \cdot (\frac{u \cdot m}{M} + \frac{v \cdot n}{N})), \qquad (30)$$

$$\text{mit } u \in \{0,1,\ldots,M-1\}, v \in \{0,1,\ldots,N-1\}$$

Da hierbei u und v von der gleichen Mächtigkeit wie m und n (den eigentlichen Bildkoordinaten im Ortraum) sind, kann man das transformierte Bild wieder als ein Array der gleichen Größe $M \times N$ des Originalbildes sehen. Es gilt aber zu beachten, dass eine Koordinate (u,v) der Frequenz (u,v) entspricht.
Zur Implementierung der DFT haben wir das Tool *FFTW* (siehe C.5) verwendet.

4.3 Powerspektren

Da die Foueriertransformierte $F(u,v)$ eines Signals $f(m,n)$ eine komplexwertige Funktion ist, nutzt man zur Visualisierung das so genannte *Powerspektrum* $|F(u,v)|$. Bei der Vorgehensweise musste in unseren transformierten Bildern zunächst für jedes Frequenzpaar (u,v) die Amplitude berechnet werden. Diese errechneten wir aus dem Punkt $F(u,v)$ mit dem Realteil $F(u,v)_{real}$ und dem Imaginärteil $F(u,v)_{imag}$ nach folgender Formel:

$$|F(u,v)| = \sqrt{F(u,v)_{real}^2 + F(u,v)_{imag}^2} \qquad (31)$$

Als nächstes mussten wir diese Werte zentrieren. Eingetragen in die `fftw_complex`-Datenstruktur, konnte man sich die Amplitudenwerte immer noch auf einem Array vorstellen, welches die gleiche Höhe und Breite unserer Eingangsbilder hat. Zum Zentrieren musste jeder Quadrant des Arrays an die gegenüberliegende Position versetzt werden. Somit sammeln sich die interessanten Informationen nicht mehr am Rand, sondern konzentriert in der Mitte an. Wir haben diese Verschiebung zunächst mit einem zusätzlichen Array realisiert. Das Originalarray wurde einmal komplett durchlaufen und mit Hilfe von mehreren Fallunterscheidungen der jeweilige Wert an die gleiche Stelle im gegenüberliegenden Quadranten des Zielarrays geschrieben. Später haben wir uns aus Effizienzgründen für eine Formel entschieden, die Fallunterscheidungen überflüssig machte:

$$F'(u,v)_{real} = F(u,v)_{real} \cdot (-1)^{u+v} \qquad (32)$$

$$F'(u,v)_{imag} = F(u,v)_{imag} \cdot (-1)^{u+v} \tag{33}$$

wobei u und v die Koordinaten von $F(u,v)$ im Originalarray sind. [FJ06]
Für die optische Ausgabe mussten letztlich die Amplitudenwerte intensiviert werden, da sonst die Details mit bloßem Auge nur schwer zu erkennen sind bzw. da Werte außerhalb der Skala von 0 bis 255 keine gültigen Informationen für Bildausgaben darstellen. Dazu wird jeder Punkt $|F'(u,v)|$ mit der nachfolgenden Formel in einen Punkt $|F''(u,v)|$ mit $0 \leq |F''(u,v)| \leq 255$ umgerechnet.

$$|F''(u,v)| = \frac{(|F'(u,v)| - Min) \cdot 255}{Max - Min} \tag{34}$$

mit:
Min : kleinste Amplitude in $|F'(u,v)|$
Max : größte Amplitude in $|F'(u,v)|$

4.4 Ringprojektion

Mittels Ringprojektion haben wir die jeweiligen Frequenzanteile des Bildes errechnet, um diese anschließend in die PCA (siehe Abschnitt 4.5) einfließen zu lassen [TT03]. Hierzu wurden vom Mittelpunkt des zentrierten Amplitudenbildes Kreise mit ansteigendem Radius gebildet. Werte die auf einem solchen Kreis liegen und somit um den selben Radius vom Mittelpunkt entfernt sind gehören zu ein und der selben Frequenz [Tea98]. Für jeden Ring, also jede Frequenz wurden die Werte aufsummiert und im Anschluss deren Mittelwert gebildet. Die maximale Frequenz ist hier nach dem Abstasttheorem auf die Hälfte der Bildbreite bzw. Bildhöhe begrenzt, abhängig davon welcher der beiden Werte der kleinere ist. Eine schematische Darstellung ist in Abbildung 41 zu sehen.

Die Ringprojektion (mittels der Klasse *BVring*) liefert uns zu einem eingegebenen zentrierten Amplitudenbild ein Array in dem die Frequenzen mit dem jeweiligen Anteil am Bild gespeichert werden. Dieses Array wird im Anschluss als Datensatz für die PCA verwendet.

4.5 Principal Component Analysis (PCA)

Um im späteren Programmablauf den Laserpunkt auf der Projektionsfläche besser und schneller finden zu können, haben wir versucht mittels einer *Principal Component Analysis (PCA)* entsprechende Merkmale für den Punkt zu finden. Die PCA auch bekannt als Hauptachsentransformation oder Karhunen-Loève-Transformation minimiert die Anzahl Merkmale auf einen

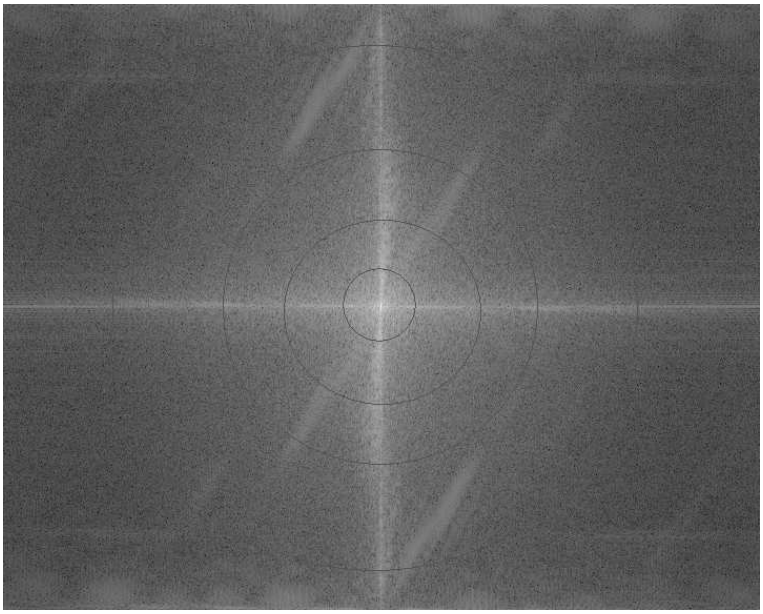


Abbildung 41: Ringprojektion auf zentriertem, logarithmiertem Amplitudenbild

Unterraum bei dem alle benötigten Informationen noch verfügbar sind. Als Eingabe der Analyse haben wir 100 unterschiedliche Bildausschnitte genommen, auf denen der Laserpunkt mit der Projektion im Hintergrund zu sehen ist, wie in Abbildung 42 dargestellt.

Mit Hilfe der Ringprojektion haben wir für jedes Bild ein Array berechnen lassen in dem die vorkommenden Frequenzen und deren Anteil am Bild gespeichert wurden. Diese Daten werden dann in einer Matrix gespeichert. Wir haben also 100 Bilder mit 25 unterschiedlichen Frequenzen, begrenzt durch die Größe der einzelnen Aufnahmen vom Laserpunkt, welche hier die Merkmale darstellen. Daraus ergibt sich ein 25-dimensionaler Raum welchen es mittels der PCA zu reduzieren gilt. Wir arbeiten also auf einer 100×25 Eingabematrix und versuchen darin Merkmale zu finden, welche typisch für den Laserpunkt sind. Als Grundlage für die PCA ist es notwendig zu wissen, dass die Varianz der Daten als Maß für deren Informationsgehalt gesehen werden kann. Die Varianz ist in der Stochastik definiert als dir Abweichung einer Zufallsvariable $X = (x_0, \ldots, x_{n-1})$ von ihrem Erwartungswert $\bar{X} = E(X)$ folgendermaßen definiert [Sie]:

$$var(X) = \frac{\sum_{i=0}^{n}(x_i - \bar{X})^2}{n-1} \tag{35}$$

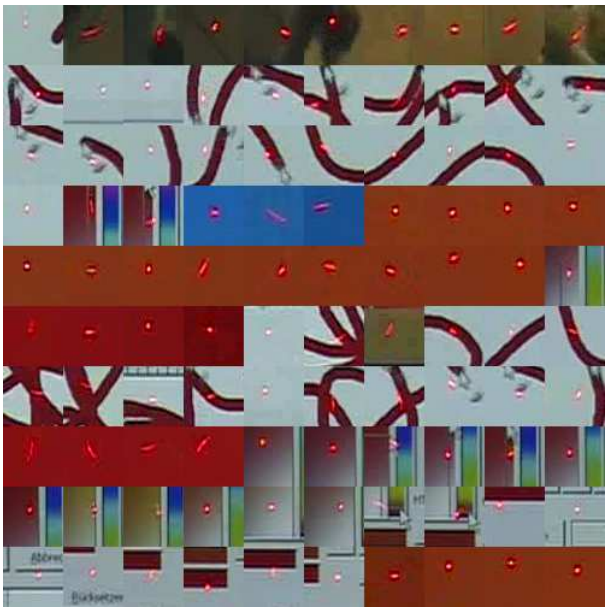


Abbildung 42: Bildausschnitte mit Laserpunkt zur PCA

In dem hier beschriebenen Kontext ist die Zufallsvarible X ein Eingabevektor $\mathbf{x}$ und der Erwartungswert der gemittelte Vektor der Eingabevektoren $\bar{\mathbf{x}}$. Somit lässt sich die Varianz für die hier verwendete PCA beschreiben als:

$$var(\mathbf{x}) = \frac{(\mathbf{x} - \bar{\mathbf{x}})^2}{n-1} \tag{36}$$

Hierdurch erhalten wir also die Varianz im eindimensionalen Raum. Um nun zwischen unseren verschiedenen Eingabevektoren die gesuchten Gemeinsamkeiten herauszufinden, müssen wir die Varianz der Vektoren unter Beachtung der Vektoren untereinander berechnen. Dies geschieht mittels der sogenannten Kovarianz. Die allgemeine Definition der Kovarianz zweier Zufallsvariablen ist folgendermaßen definiert. Sein $X = (x_0, \ldots, x_{n-1})$ und $Y = (y_0, \ldots, y_{n-1})$ zwei Zufallsvariablen, $\bar{X} = E(X)$ der Erwartungswert der Zufallvariable X und $\bar{Y} = E(Y)$ der Erwartungswert der Zufallsvariable Y, so ergibt sich die Kovarianz $cov(X, Y)$ als:

$$cov(X, Y) = \frac{\sum_{i=0}^{n} (x_i - \bar{X}) \cdot (y_i - \bar{Y})}{n-1} \tag{37}$$

Wie oben beschrieben bilden in unserem Kontext die Eingabevektoren die Zufallsvariablen. Da der Erwartungswert zweier unterschiedlicher Eingabevektoren gleich dem gemittelten Eingabevektor ist, folgt $\bar{X} = \bar{Y} = \bar{\mathbf{x}}$ und die

Kovarianz zweier Eingabevektoren vereinfacht sich zu:

$$cov(\mathbf{x}, \mathbf{y}) = \frac{(\mathbf{x} - \bar{\mathbf{x}}) \bullet (\mathbf{y} - \bar{\mathbf{x}})}{n - 1} \tag{38}$$

Aufgrund der Kommutativität der Multiplikation sieht man schnell, dass $cov(\mathbf{x}, \mathbf{y}) = cov(\mathbf{y}, \mathbf{x})$ ist, da sich die beiden Terme $(\mathbf{x} - \bar{\mathbf{x}}) \cdot (\mathbf{y} - \bar{\mathbf{x}})$ einfach vertauschen lassen. Wir brauchen also jeweils nur $cov(\mathbf{x}, \mathbf{y})$ berechnen. Während wir bei der Kovarianz weiterhin nur zwei Dimensionen haben, so haben wir bei unserer Eingabematrix deutlich mehr Vektoren. Um zum Beispiel in einen dreidimensionalen Raum rechnen zu können, müssen die verschiedenen Kovarianzen berechnet werden, wodurch sich dann die Kovarianzmatrix $\mathbf{C}$ ergibt [Smi02]. Gegeben seien hier nun die drei Vektoren $\mathbf{x}, \mathbf{y}, \mathbf{z}$. Die zugehörige Kovarianzmatrix baut sich dazu folgendermaßen auf:

$$\mathbf{C} = \begin{pmatrix} cov(\mathbf{x}, \mathbf{x}) & cov(\mathbf{x}, \mathbf{y}) & cov(\mathbf{x}, \mathbf{z}) \\ cov(\mathbf{y}, \mathbf{x}) & cov(\mathbf{y}, \mathbf{y}) & cov(\mathbf{y}, \mathbf{z}) \\ cov(\mathbf{z}, \mathbf{x}) & cov(\mathbf{z}, \mathbf{y}) & cov(\mathbf{z}, \mathbf{z}) \end{pmatrix} \tag{39}$$

Im allgemeinen Fall lautet die Formel für n Dimensionen dann:

$$\mathbf{C} = \{c_{i,j} | c_{i,j} = cov(Dim_i, Dim_j)\} \tag{40}$$

Da wie bereits erwähnt $cov(\mathbf{x}, \mathbf{y}) = cov(\mathbf{y}, \mathbf{x})$ gilt, ist die Kovarianzmatrix entlang ihrer Hauptdiagonalen symmetrisch. Wir haben die Kovarianzmatrix für unsere Eingabevektoren mittels der Newmat-Funktionen (siehe Seite 174) berechnen lassen. Zu der Kovarianzmatrix wurden dann ebenfalls per Newmat die Eigenvektoren und die zugehörigen Eigenwerte berechnet. Die Eigenvektoren geben die jeweiligen Richtungen an, in die sich die Werte innerhalb des n-dimensionalen Koordinatensystem ausdehnen. Um den Merkmalsraum zu veringern, reduziert man die Menge der zur Kovarianzmatrix gehörenden Eigenvektoren auf diejenigen, welche den größten Einfluss auf den Wertebereich haben. Wenn zum Beispiel nur eine geringe Anzahl von Werten sich in die Richtung eines bestimmten Eigenvektors ausdehnen, so kann dieser Eigenvektor entfernt werden ohne größere Verluste von Informationen die in den Daten enthalten sind. Entscheidend für die PCA ist es die Länge der Einzelvektoren auf eins zu reduzieren, was aber durch Newmat automatisch berechnet wird. Zu jedem Eigenvektor gibt es einen zugehörigen Eigenwert, welcher den Streckungsfaktor des Eigenvektors bei der Multiplikation mit der Matrix und damit auch die Gewichtung desselben angibt. Nach der Berechnung der Eigenvektoren und Eigenwerte wird entschieden, welche Eigenvektoren für diese relevant sind. Hierzu ordnet man die Eigenvektoren absteigend nach den jeweils zugehörigen Eigenwerten. Nun kann

man die Vektoren mit den höchsten Eigenwerten auswählen und die restlichen wegfallen lassen, wodurch sich die Dimension des Datenraums auf die Anzahl der verwendeten Eigenvektoren reduziert. Hierbei gehen natürlich wie bereits erwähnt Informationen verloren, was aber bei geringen Eigenwerten nicht viel Schaden anrichtet. Die gewählten Eigenvektoren bilden anschließend einen so genannten Merkmalsvektor. Multipliziert man diesen mit den gegeben Daten vom Anfang, so reduziert man die gegebenen Informationen auf den benötigten Teil, also somit die Dimension der gegebenen Daten. Dies ermöglicht es, in unseren gegebenen Bildern gezielter nach den Merkmalen des Laserpunktes zu suchen, da die entscheidenden Werte sich innerhalb des Merkmalraums befinden müssen.

5 Segmentierung des Laserpointers

5.1 Einleitung

Am Beginn der Arbeit der Segmentierungsgruppe stand das Finden geeigneter Verfahren, um den Laserpunkt innerhalb eines Bildes zu lokalisieren und seine Position zu bestimmen. Auf Grund längerfristiger Reparaturarbeiten innerhalb des Hörsaals, in welchem die fertige Anwendung laufen soll, wurden zunächst mit einem Camcorder verschiedene Aufnahmen eines Laserpunktes auf Leinwänden innerhalb unterschiedlicher Testszenarien generiert. Diese wurden anschließend mittels Kino [Abschnitt C.10] in den Rechner gespielt, in kleine Teilstücke zerlegt und mittels ffmpeg 2.2.6 in ein für die Arbeit mit OpenCV günstiges Format konvertiert. Hierfür fiel die Wahl auf MPEG1 mit einer Bitrate von 4Mbit/s. Dieses unkonventionelle Format stellte einen zufriedenstellenden Kompromiss zwischen Datenaufkommen, Qualität und von OpenCV unterstützten Formaten dar.
In ersten Versuchen wurden einige von OpenCV angebotene kantenorientierte Verfahren wie Sobel- und Laplaceoperator oder CannyEdge auf das Bild angewendet. Auf Grund der geringen Größe des Laserpunktes von nur wenigen Pixeln innerhalb der Bilder war er in den resultierenden Bildern kaum zu erkennen. Diese Erkenntnis brachte die Arbeitsgruppe dazu, im Folgenden mit bereichsorientierten Verfahren zu arbeiten.

5.2 Grundlagen

In diesem Abschnitt werden die Grundlagen für die innerhalb des Projektes verwendeten Verfahren zur Segmentierung des Laserpointers beschrieben.

5.2.1 Bereichsorientierte Verfahren

Kerngedanke bei der Segmentierung von digitalen Bildern ist es, das Bild in mehrere disjunkte Bereiche zu unterteilen, in denen jeweils ein bestimmtes Einheitskriterium E gilt. Wenn richtig segmentiert wurde, muss weiterhin gelten, dass für die Vereinigung zweier Bereiche kein gemeinsames Einheitlichkeitskriterium existiert. Grundsätzlich gibt es für das Auffinden solcher Teilgebiete zwei verschiedene Ansätze.

Beim kantenorientierten Ansatz wird versucht, die Kanten zwischen den Bereichen zu finden und geschlossene Gebiete zu lokalisieren. Dies bietet sich besonders bei markanten Konturlinien mit z.B. auffälligem Krümmungsverhalten an. Auf Grund der geringen Größe des Laserpunktes und der wenig

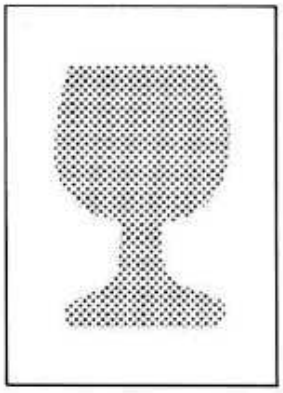


Abbildung 43: Bereichsorientierter Ansatz

Abbildung 44: Kantenorientierter Ansatz

markanten Kontur kam dieser Ansatz nicht in Frage.
Als vielversprechender erwies sich für den Laserpunkt ein bereichsorientierter Ansatz. Bei solchartigen Verfahren werden Masken erstellt, die für jedes Pixel angeben, in welchem Bereich es liegt. Für das Auffinden des Laserpointers ist das Ziel, das Eingabebild in zwei Bereiche zu unterteilen. Zum ersten Bereich zählen dabei alle Pixel, die den Laserpointer darstellen, im Idealfall ein Pixel, und zum zweiten Bereich zählen alle Pixel, die nicht dem gesuchten Punkt entsprechen. Um diese Unterteilung zu erreichen, wird jedes Pixel auf verschiedene Merkmale überprüft. Das Ergebnis der Auswertung eines Merkmals entspricht dabei jeweils einem Binärbild. Eine logische Und-Verknüpfung aller Binärbilder ergibt schließlich das nach Wunsch segmen-

tierte Ergebnisbild. Um Rechenzeit zu sparen wird innerhalb des Projektes nicht zwangsweise jeder Test auf jedem Pixel ausgeführt. Es wird jedes Pixel zunächst auf ein Merkmal geprüft und nur im Erfolgsfall werden auch die weiteren Tests durchgeführt.

5.2.2 Schwellwertverfahren

Einfache Schwellwertverfahren bilden die Grundlage des verwendeten Segmentierungsansatzes. Ausgehend von einem diskreten Bildsignal $f(m,n), 0 \leq m \leq M-1, 0 \leq n \leq N-1$ lässt sich ein einfacher Schwellwertoperator mathematisch folgendermaßen beschreiben.

$$g(m,n) = \begin{cases} I_1 & \text{für } 0 \leq f(m,n) \leq s \\ I_2 & \text{für } s \leq f(m,n) \leq f_{max} \end{cases} \tag{41}$$

Dabei stellen I_1 und I_2 zwei beliebige, aber unterschiedliche Werte dar und s eine zu definierende Intensitätsschranke. Als anschauliches Beispiel hierfür lässt sich die Segmentierung auf einem Graustufenbild anhand der Intensitätswerte angeben, wobei sich ein dunkles Objekt vor einem hellen Hintergrund befindet. Dieses gilt es vom Hintergrund hervorzuheben. $f(m,n)$ repräsentiert hier den jeweiligen Grauwert des betrachteten Bildpunktes. Anhand einer gesetzten Intensitätsschranke s lässt sich bestimmen, in welchen Bereich das Pixel fällt und welcher Wert, I_1 oder I_2, zugeordnet wird. In Abbildung 45 sieht man ein dem Beispiel entsprechendes Eingabebild, bei welchem der Hintergrund entfernt werden soll.

Abbildung 46 zeigt das Ergebnis einer gelungenen Segmentierung mittels Schwellwertoperator. Dabei wurden die $f(m,n) \leq s$ unverändert gelassen und die $f(m,n) > s$ auf f_{max} gesetzt, wodurch eine klare Trennung zwischen Objekt und Hintergrund gegeben ist. Anzumerken ist hierbei, dass die Wahl der Intensitätsschranke, des sogenannten Schwellwertes, von zentraler Bedeutung ist, was durch Abbildung 47 deutlich gemacht werden soll. Hier wurde das selbe Verfahren verwendet, allerdings wurde der Schwellwert zu niedrig angesetzt, so dass noch Reste des Hintergrundes vorhanden bleiben. Analog zu diesem Beispiel lassen sich noch zusätzliche Intensitätsgrenzen einfügen, um in mehrere Bereiche zu unterteilen.

Im Anwendungsfall der PG werden die Pixel, deren Wert über bzw. je nach Merkmal unter dem Schwellwert liegen, nicht weiter betrachtet, da sie nach diesem Test nicht mehr für den Laserpunkt in Frage kommen. So gesehen entsprechen Schwellwertoperatoren Hoch- bzw. Tiefpassfiltern, da sie, je nachdem ob die Werte oberhalb oder unterhalb des Schwellwertes von Interesse

Abbildung 45: Bild zur Segmentierung mittels Schwellwertoperator

Abbildung 46: günstig gewählter Schwellwert

sind, nur höhere oder tiefere Werte durchlassen. Führt man dieses Analogon weiter aus, kommt man leicht auf die Idee, zwei oder mehr Schwellwerte zu kombinieren und erhält so die Entsprechung eines Bandpasses bzw. einer Bandsperre für den Wertebereich. Es kommen nur solche Punkte in Frage, die innerhalb oder, je nach Definition, außerhalb eines bestimmen Wertebe-

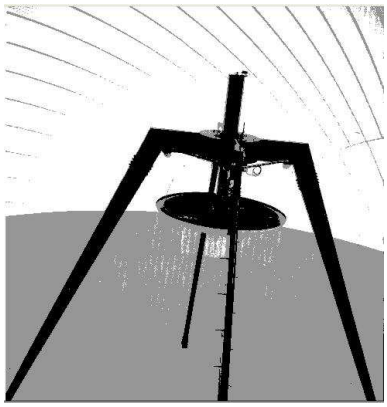


Abbildung 47: zu niedrig angesetzter Schwellwert

reiches liegen.
Die Wahl günstiger Schwellwerte erfolgte zunächst empirisch mit rein objektiver Bewertung der Qualität. Um die grob angenäherten Werte für die jeweiligen Umgebungsbedingungen zu optimieren hat die Projektgruppe ein Tool zur halbautomatischen Ermittlung der Schwellwerte entwickelt, welches in Kapitel 6 näher beschrieben wird.

5.2.3 Farbräume

Während bei einem einfachen Graustufenbild in der Regel nur ein Kanal für das Bild angegeben ist, welcher die Intensitätsinformationen enthält, gibt es für farbige Bilder verschiedene Möglichkeiten, diese im Rechner zu repräsentieren. Im Folgenden sind drei unterschiedliche Modelle zur Darstellung farbiger Bilder beschrieben, die sich jeweils ineinander konvertieren lassen. Jeder Farbraum bietet unterschiedliche Eigenschaften, weshalb es bei bestimmten Verfahren nützlich sein kann, den Farbraum im Vorfeld zu wechseln.

Der RGB-Farbraum Der RGB-Farbraum beschreibt ein additives Farbmodell ausgehend von den drei Grundfarben Rot(R), Grün(G) und Blau(B). Dabei wird versucht das Verhalten des menschlichen Auges nachzubilden, denn das Auge verfügt über getrennte Farbrezeptoren für die genannten drei Farben. Beim Wahrnehmen einer Farbe wird jeder Farbkanal einzeln

angeregt und durch eine anschließende Überlagerung entsteht der eigentliche Farbeindruck.
Technisch lässt sich dieses Modell sehr gut nachbilden. Pro Farbkanal steht eine zu definierende Bandbreite zur Verfügung (im konkreten Anwendungsfall kommen acht Bit pro Kanal zum Einsatz), um die Intensität dieses Kanals zu charakterisieren. Werden alle Kanäle auf 0 gesetzt, ist das Resultat schwarz, werden hingegen alle Kanäle auf den Maximalwert gesetzt, ergibt sich weiß, wie man in Abbildung 48 sehen kann.

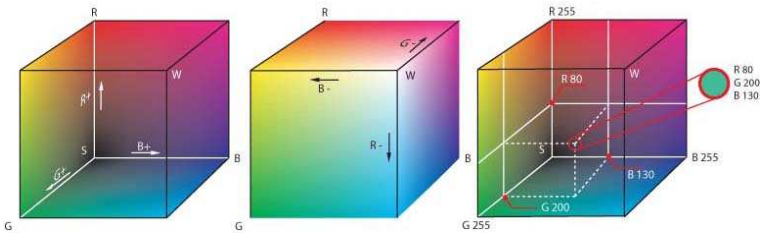


Abbildung 48: Der RGB-Farbraum

Das RGB-Modell ist Basis für die in Abschnitt 10.1.3 eingeführte Datenstruktur und stellt somit das Ausgangsformat für die weitere Bildbearbeitung dar.

Der HLS-Farbraum Der HLS-Farbraum mit seinen drei Komponenten H (engl. Abk. Hue), L (engl. Abk. Lightness) und S (engl. Abk. Saturation) hat gegenüber dem gebräuchlicheren RGB-Farbraum den Vorteil, dass er eher den Vorgehen in der Kunst entspricht und sich somit intuitiv durch Verändern der einzelnen Kanäle gezielt Farbeindrücke erzielen lassen. Die drei Komponenten H, L, und S sollen hier nun näher beschrieben werden.

- H entspricht dem Farbton.
- L entspricht der Helligkeit der Farbe.
- S entspricht der Sättigung.

Abbildung 49 zeigt anschaulich den Aufbau des HLS-Farbraumes.

Die Umrechnung eines 8-bit Bildes vom RGB-Farbraum in den HLS-Farbraum erfolgt, wie im nachfolgenden Pseudocode beschrieben:

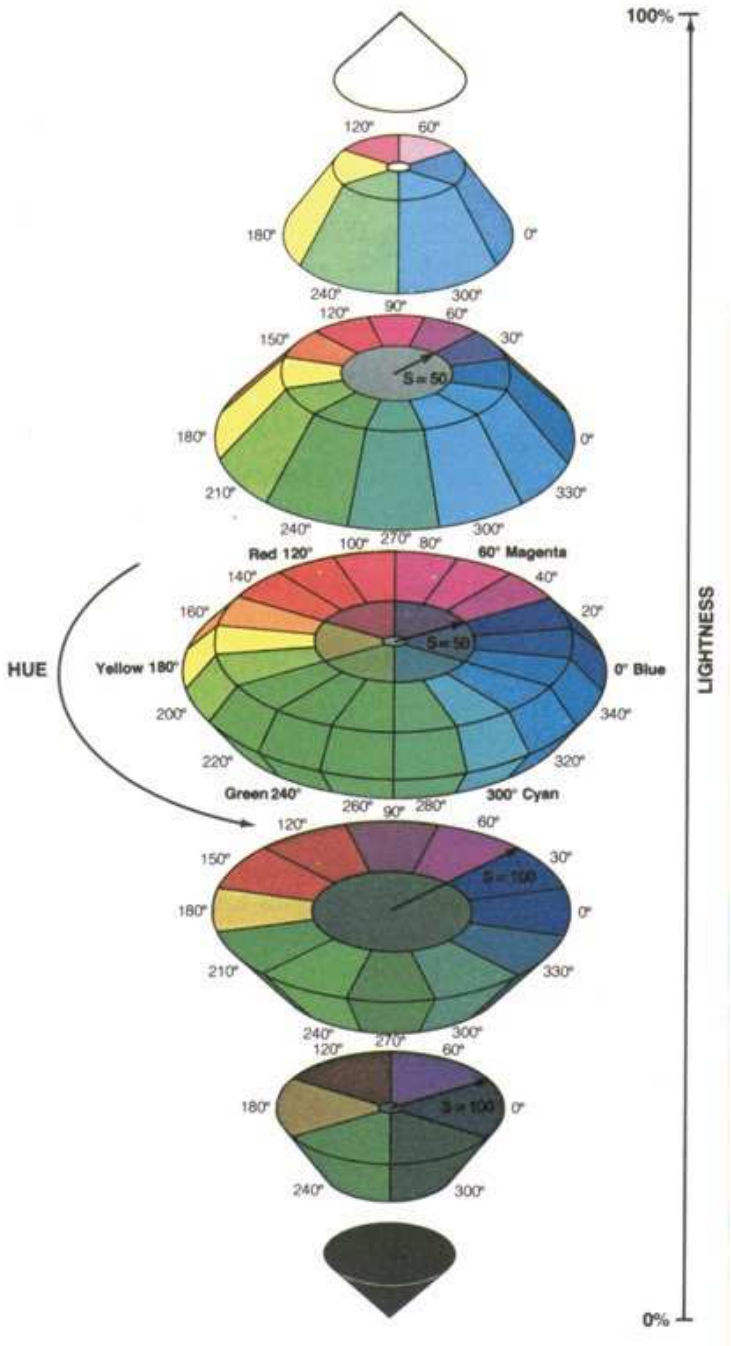


Abbildung 49: Der HLS-Farbraum [Sch99]

```
BEGIN
  V_{max} := max(R,G,B);
  V_{min} := min(R,G,B);

  L := (V_{max}+V_{min})/2;

  IF (L < 0.5) THEN S: = (V_{max}-V_{min})/(V_{max}+V_{min});
  ELSE S := (V_{max}-V_{min})/(2-(V_{max}+V_{min}));

  IF (V_{max} = R) THEN H := (G-B)*60/S;
  IF (V_{max} = G) THEN H := 180+(B-R)*60/S;
  IF (V_{max} = B) THEN H := 240+(R-G)*60/S;

  IF (H < 0) THEN H := H+360;

  H := H/2;
  L := L*255;
  S := S*255;
END;
```

Des Weiteren muss gesagt werden, dass die RGB-Werte vor der Umrechnung normiert werden müssen. Die letzten drei Zeilen sind speziell für die Umrechnung eines 8-bit Bildes hinzugefügt worden. Handelt es sich um 16-bit oder 32-bit Bilder, so müssen diese Zeilen durch die folgenden ersetzt werden.

Bei 16-bit Bildern:

```
H := H;
L := L*65535;
S := S*65535;
```

Bei 32-bit Bildern:

```
H := H;
L := L;
S := S;
```

Der HLS-Farbraum ist für die Projektgruppe wichtig, da er sich sehr gut dafür eignet Farbhistogramme zu berechnen, was im RGB-Farbraum nicht der Fall ist. Diese Histogramme sollen im Abschnitt 5.2.6 näher erläutert werden, da sie die zentrale Rolle beim Tracking der Person spielen und als Merkmal dienen.

Das YUV-Farbmodell Bisher wurde die Segmentierung auf dem RGB-Farbraum beschrieben. Das Problem, das sich dabei ergibt ist, dass dieser Farbraum sehr lichtempfindlich ist. Dies bedeutet, dass je nach Tageszeit und Lichteinfluss der Laserpunkt besser oder schlechter zu erkennen und somit zu segmentieren ist. Es stellt sich nun die Frage nach einem Farbraum oder Farbmodell, sodass diese Lichtempfindlichkeit verhindert oder reduziert wird.
Eine mögliche Lösung bietet das YUV-Farbmodell, das sich aus zwei Komponenten zusammensetzt. Zum einen der Luma und zum anderen der Chrominanz. Die Luma gibt die Lichtstärke pro Fläche an und ist korrespondierend zum Grauwert. Die Chrominanz ist die Komponente, die für den Farbanteil zuständig ist. Sie setzt sich wiederum aus zwei Komponenten zusammen. U beschreibt den blauen Farbanteil und V den roten Farbanteil.
Die Umrechnung zwischen RGB und YUV ist durch folgende Gleichungen bestimmt:

$$Y := 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{42}$$

$$U := (B - Y) \cdot 0.493 \tag{43}$$

$$V := (R - Y) \cdot 0.877 \tag{44}$$

Der RGB-Farbraum hat folgende Nachteile:

- Der RGB-Farbraum kann nicht alle Farben wiedergeben.
- Die Farbwahrnehmung hängt von der absoluten Helligkeit ab.
- Die Umgebung beeinflusst die Farbwahrnehmung.

Das YUV-Farbmodell bietet folgende Vorteile gegenüber dem RGB-Farbraum:

- Die Darstellung entspicht mehr dem visuellen menschlichen Empfinden.
- U und V können effizienter durch weniger Bits dargestellt werden.
- Der V-Kanal ist beleuchtungsunempfindlicher.

5.2.4 Farbsegmentierung

Der bisher beschriebene einfache Schwellwertoperator arbeitet auf einem eindimensionalem Merkmal eines ebenfalls eindimensionalen Wertebereichs, was z.B. eine einfache Segmentierung von Graustufenbildern gestattet. Versucht man dieses Verfahren direkt auf farbige Bilder anzuwenden, stößt man auf Probleme. In diesem Abschnitt werden daher mögliche Varianten der Anwendung eines Schwellwertoperators zur Farbsegmentierung auf RGB-Bildern dargestellt.
Ziel der Farbsegmentierung ist es, ein gegebenes Bild in verschiedene Bereiche mit verschiedenen Farbeigenschaften zu unterteilen. Wählt man naiv, z.B. um rote Bildanteile zu segmentieren, nur einen Schwellwert für die R-Werte und lässt B- und G-Werte außer Acht, liefert der Operator unter Umständen nicht das gewünschte Ergebnis, da für den resultierenden Farbeindruck alle drei Farbanteile eine Rolle spielen. Es ist also nötig, alle Farbkanäle mit einzubeziehen, was durch unterschiedliche Weisen modelliert werden kann.
Eine Möglichkeit dem Farbmodell gerecht zu werden, ist die Differenzenbildung zwischen jeweils zwei der Farbkanäle. Ausgehend vom RGB-Farbraum lassen sich folgende Differenzen bilden:

1. R-B
2. R-G
3. B-R
4. B-G
5. G-R
6. G-B

Anhand der in 5.2.2 beschriebenen Schwellwertverfahren kann das Bild mit Hilfe eines Schwellwertes t' in verschieden farbige Bereiche unterteilt werden.

1. Rot wenn 1 und $2 \geq t' \geq 0$
2. Blau wenn 3 und $4 \geq t' \geq 0$
3. Grün wenn 5 und $6 \geq t' \geq 0$

Eine andere Möglichkeit Bilder auf Grund von Farbinformationen bereichsorientiert zu segmentieren besteht in der Auswertung der Anzahl von prozentualen Anteilen der Farbwerte in einem Fenster. Bei diesem Verfahren wird

für jedes Pixel $\mathbf{p}$ bestimmt ob die gesuchte Farbe $k \in \{R, G, B\}$ überwiegt. Hierbei wird aber nicht die Farbdifferenz sondern es wird ein Quotient q angegeben. Dabei bezeichnet $i_{\mathbf{p}}$ mit z.B. $i = R$ den Wert des Roten Kanals an Pixel $\mathbf{p}$.

$$COL_k(\mathbf{p}) = \begin{cases} 1, & \text{falls } i_{\mathbf{p}} \leq q \cdot k_{\mathbf{p}} \wedge j_{\mathbf{p}} \leq q \cdot k_{\mathbf{p}} \wedge k_{\mathbf{p}} > t \\ 0, & \text{sonst} \end{cases} \tag{45}$$
$$\text{mit } i, j \in \{R, G, B\},\ i \neq j \neq k$$

Danach wird mit Hilfe eines Fensters g der Größe $N_F \times M_F$ die Anzahl der Pixel gezählt die einen überwiegenden Anteil an der Farbe k haben.

$$COL_k^F(n, m) = \sum_{a=0}^{N_F-1} \sum_{b=0}^{M_F-1} COL_k(\mathbf{p} = (a, b)) \tag{46}$$
$$\text{mit } a \in \{0, \ldots, N_f - 1\}, b \in \{0, \ldots, M_F - 1\}$$

Ist diese Summe wieder größer als ein zu bestimmender Schwellwert t', wird das Fenster als überwiegend aus der Farbe k bestehend ausgegeben. Das oben beschriebene Verfahren kann auch auf folgende Weise vereinfacht werden. Man berechnet den Vektor der Mittelwerte $\mathbf{m}$ der Farbwerte über dem Fenster. Von diesem Mittelvektor berechnet man nun, ob Bedingung 45 erfüllt ist.

Bisher wurde die Segmentierung auf dem RGB-Farbraum beschrieben. Man kann auch eine Segmentierung in den in Abschnitt 5.2.3 beschriebenen Farbräumen bzw. Farbmodellen vornehmen.
Die Farbsegmentierung auf Basis der YUV-Farbmodells besteht nun wieder in einem einfachen Schwellwertverfahren. Hierbei wird ein vorher zu bestimmender Schwellwert t auf einen der Kanäle angewandt. Dabei ist zu beachten, dass der V-Kanal sensitiv für rot und der U-Kanal sensitiv für blau ist. Wendet man einen Schwellwert auf den Y-Kanal an, lässt sich an Hand von Helligkeitsunterschieden segmentieren.
Auch im HLS-Farbraum kann man Schwellwerte auf die einzelnen Kanäle anwenden, wovon aber innerhalb des Projektes abgesehen wurde. Zur Segmentierung innerhalb dieses Farbraumes erwiesen sich Farbhistogramme als probates Mittel, auf welches in Abschnitt 5.2.6 eingegangen wird.

5.2.5 Integral-Bilder

Bei der Berechnung von Merkmalen innerhalb eines Bildes, die auf rechteckigen Summen basieren, bietet sich eine Repräsentation des Bildes als Integral-Bild an. Diese Repräsentation enthält an einer Position $Int(x, y)$ die Summe

des Quadrates, das (x, y) als untere rechte Ecke hat.

$$Int(x, y) := \sum_{x' \leq x, y' \leq y} I(x', y') \tag{47}$$

Diese Gleichung kann effizient nach folgendem Vorgehen in einem Durchlauf des Bildes berechnet werden.

$$s(x, y) = s(x, y - 1) + I(x, y) \tag{48}$$

$$Int(x, y) = Int(x - 1, y) + s(x, y) \tag{49}$$

Dabei bezeichnet $s(x, y)$ die kumulative Spaltensumme.
Mit Hilfe des Integral-Bildes lässt sich nun jede rechteckige Summe mit vier Arrayzugriffen berechnen. Die Differenz zweier Rechtecke kommt mit acht Arrayzugriffen aus.
Als Beispiel zur Bestimmung einer Rechtecksumme soll hier Abbildung 50 dienen. Der Wert an Position eins besteht aus der Summe über dem Rechteck A, der Wert an Position zwei aus der Summe über den Rechtecken A und B, der Wert an Position drei aus den Summen über A und C und an Position vier aus den Summen über A,B,C und D. Möchte man nun den Wert des Rechteckes D mit Hilfe des Integral-Bildes bestimmen so berechnet man $4 + 1 - (2 + 3)$, da die Summe über A im Wert von Position zwei und drei enthalten ist, muss der Wert von Position eins wieder aufaddiert werden. Die hier beschriebene Variante des Integral-Bildes findet seine Anwendung im Abschnitt 5.3.2 und in einer abgewandelten Form, als integrales Farbhistogramm im Abschnitt 5.2.6.

5.2.6 Farbhistogramme

Histogramme sind eine grafische Darstellung der Häufigkeitsverteilung von Bildpunkten. Bei den hier verwendeten Histogrammen handelt es sich um Farbhistogramme. Die Häufigkeitsverteilung wird in Form eines Vektors als Merkmal genutzt. Die einzelnen Stellen innerhalb des Mermalvektors werden Bins genannt und geben die Anzahl der Bildpunkte an, die innerhalb eines bestimmten Intervalls liegen.

Aufbau Der RGB-Farbraum reagiert sehr viel stärker auf Helligkeitsschwankungen als der HLS-Farbraum, deswegen wird vor der eigentlichen Berechnung des Farbhistogramms [Klu06] eine Konvertierung des RGB-Bildes in ein HLS-Bild, wie in Abschnitt 5.2.3 beschrieben, durchgeführt.

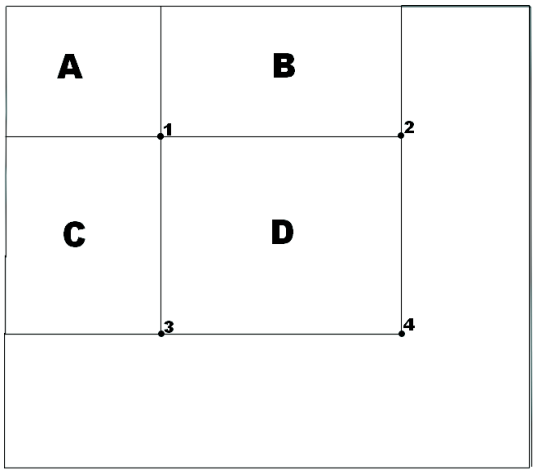


Abbildung 50: Berechnung einer Rechtecksumme im Integral-Bild, $D = 4 + 1 - (2 + 3)$

Der HLS-Farbraum wird in 32 Histogramm-Bins unterteilt, wobei 24 für den Farb- und Sättigungsanteil, also den Komponenten H und S entsprechen, und die übrigen 8 für den Helligkeitsanteil L reserviert sind.

Die Sättigung sollte einen Wert von 16 aufweisen, da ein darunter liegender Wert auf Rauschen schließen lässt. Experimente mit anderen Werten brachten keine bedeutenden Verbesserungen mit sich. Die Zuordnung eines Bildpunktes $\mathbf{p}$ zu einem Histogramm-Bin erfolgt nun folgendermaßen:

$$h(i, rect) = \sum_{\mathbf{p} \in rect} \delta(B(\mathbf{p}) - i) \text{ mit } i \in \{1, \ldots, 32\} \tag{50}$$

Dabei ist

- rect das Teilrechteck auf dem das Farbhistogramm berechnet wird.
- i das i-te Bin innerhalb des Merkmalsvektors.
- $\mathbf{p}$ ein Bildpunkt innerhalb des Teilrechtecks rect.
- $$\delta(x) = \begin{cases} 1 & \text{falls } x = 0 \\ 0 & \text{sonst} \end{cases} \tag{51}$$

- $h(i, rect)$ gibt die Anzahl der Bildpunkte an, die in rect liegen und dem i-ten Bin zugeordnet wurden.

Die Funktion $B(\mathbf{p})$ gibt durch eine Fallunterscheidung an, auf welche Look-Up-Tabelle zugegriffen werden soll. Diese Look-Up-Tabellen haben wir bisher noch nicht näher beschrieben. Der Zugriff erfolgt in konstanter Rechenzeit und die Funktion ist somit effizient berechenbar.

$$B(\mathbf{p}) = \begin{cases} B_L(L_\mathbf{p}) & \text{für } 0 \leq S_\mathbf{p} < 16 \\ B_{HS}(H_\mathbf{p}, S_\mathbf{p}) & \text{für } 16 \leq S_\mathbf{p} \leq 255 \end{cases} \tag{52}$$

Das Histogramm mit Zentrum $\mathbf{p} = (x, y)$ und Breite w bzw. Höhe h ergibt sich nun durch die folgende Funktion:

$$Hi(p_x, p_y, w, h) = \frac{1}{w \cdot h} \cdot (h(1, rect), \ldots, h(32, rect))^T \tag{53}$$

Der Bruch $\frac{1}{w \cdot h}$ dient der Normierung des Farbhistogramms und erlaubt einen Vergleich von Histogrammen unterschiedlicher Ausgangsgröße.

Die Look-Up-Tabelle Die von der Projektgruppe verwendete Look-Up-Tabelle ist uniform aufgebaut.Dies bedeutet, dass wir die einzelnen Bereiche gleich groß gewählt haben. Zunächst muss ein eindimensionales Array $B_L(x)$ der Größe 256 erzeugt werden, welches als Look-Up-Tabelle der Helligkeit L dient. Der Helligkeitsanteil wird durch 8 Bins beschrieben, was bedeutet, dass die Look-Up-Tabelle in 8 gleichgroße Teile aufgeteilt werden muss. Diesen Teilen werden aufsteigend Nummern zugeordnet. Die zweidimensionale Look-Up-Tabelle $B_{HS}(x, y)$ dient dem Zugriff des Farb- und Sättigungsanteils. Diese LUT muss wiederum in gleichgroße Teile aufgeteilt werden, allerdings nicht in 8 sondern 24, da für Farb- und Sättigungsanteil 24 Bins zur Verfügung stehen. Die Nummerierung beginnt dabei mit der Nummer, bei der die erste LUT beendet wurde. Am Ende ist die LUT fertig.

Abstandsfunktion Die Abstandsfunktion soll dazu dienen zwei Farbhistogramme, hier A und B miteinander zu vergleichen. Der ausgegebene Wert ist dabei ein Übereinstimmungsmaß. Dabei ist zu beachten, dass ein kleiner Wert auf eine große Übereinstimmung schließen lässt.

$$\Delta_1(A, C) = \sqrt{1 - \sum_{i=1}^{32} \sqrt{A_i \cdot C_i}} \tag{54}$$

$$\Delta_2(A, C) = \sqrt{\sum_{i=1}^{32} (A_i - C_i)^2} \tag{55}$$

Falls beide Farbhistogramme identisch sind, so ist der Wert der Abstandsfunktion aus Gleichung 55 gleich Null, da die Subtraktion an jedem Bin eine Null liefert ist die Summe Null dadurch ebenfalls die Wurzel.

Bei Gleichung 54 muss man ein wenig näher hinschauen. Sind die Farbhistogramme identische, so gilt:

$$\begin{aligned} \Delta_1(A, A) =& \sqrt{1 - \sum_{i=1}^{32} \sqrt{A_i \cdot A_i}} \\ =& \sqrt{1 - \sum_{i=1}^{32} A_i} \end{aligned} \tag{56}$$

Da die A_i normiert sind, gilt für die Summe, dass sie eins ist, somit ist der Wert null.

Nun benötigt man noch den Begriff des Referenzfarbhistogramms, dabei handelt es sich um einen Teilbereich des Bildes, der verfolgt werden soll, im Fall der Projektgruppe um ein Bild der Zielperson. Um nun den Bereich des Bildes zu finden, der den geringsten Abstand zum Referenzhistogramm hat, müssen wir für jeden Bildpunkt ein Histogramm der Größe des Referenzhistogramms berechnen und damit dann die Abstandsfunktion auswerten. Bei einer Bildauflösung von 720×576 Pixeln und einer Referenzfarbhistogrammweite von 50 Pixeln und -höhe von 100 Pixeln sind das $720 \times 576 \times 50 \times 100 = 2073600000$ Berechnungen. Diese große Zahl an Berechnungen macht ein Real-Time Tracking unmöglich. Der große Aufwand ergibt sich vor allem durch unnötige Berechnung der Farbhistogramme, bei denen Zwischenergebnisse für darauf folgende Auswertungen benutzt werden könnten. Abhilfe schaffen die so genannten integralen Farbhistogramme.

Berechnung des integralen Farbhistogramms Ein integrales Farbhistogramm stellt die Erweiterung des in Abschnitt 5.2.5 beschriebenen Integralbildes dar. Hier wird nicht nur mit einem zweidimensionalen Integral gearbeitet, sondern das berechnete integrale Farbhistogramm ist dreidimensional. Die dritte Dimension besteht aus den Farbhistogrammen des HLS Farbraums. Im Folgenden wird das integrale Farbhistogramm mit I bezeichnet. Der Aufbau erfolgt nun Zeile für Zeile, so kann sukzessive für das gesamte Bild ein Farbhistogramm aufgebaut werden. Das Farbhistogramm an der Stelle $\mathbf{p} := (x, y)$ ergibt sich dann wie folgt:

$$\begin{aligned} I(p_x, p_y) =& I(p_x, p_y - 1) + I(p_x - 1, p_y) - I(p_x - 1, p_y - 1) + \\ & \left[(\delta(B(\mathbf{p}) - 1) ... \delta(B(\mathbf{p}) - 32))^T\right] \end{aligned} \tag{57}$$

Die Auswertung für den Bildpunkt $\mathbf{p}$ setzt sich einmal zusammen aus dem Wert an $I(p_x, p_y - 1)$ und $I(p_x - 1, p_y)$, dabei wurde allerdings $I(p_x - 1, p_y - 1)$ zweimal addiert, da bereits sowohl in die Berechnung von $I(p_x, p_y - 1)$ als auch $I(p_x - 1, p_y)$ eingegangen ist, muss dieser Wert einmal abgezogen werden. Der letzte Term gibt das Histogramm für das Pixel $\mathbf{p}$ an. Die Berechnung eines Histogramms Hi erfolgt nun, wie schon in Abschnitt 5.2.5 beschrieben durch einfache Addition und Subtraktion der zugrunde liegenden rechteckigen Summen.

$$Hi = I_4 + I_1 - I_2 - I_3 \tag{58}$$

Hierbei seien die Bezeichnungen analog zur Abbildung 50 des Abschnittes 5.2.5. Das I drückt aus, dass es sich nicht um einen zweidimensionalen Bereich, sondern um ein Histogramm an den entsprechenden Stellen handelt.

ROI Der Begriff ROI (engl. Abk. Regions of Importance) beschreibt den Bereich des Bildes, der überhaupt einer Untersuchung bedarf. Er wird im Verlauf dieser Ausarbeitung bei dem Partikelfilter benötigt, um weitere Rechenzeit zu sparen. Es handelt sich dabei um nichts anderes als den Bereich des Bildes, in dem zunächst einmal vermutet wird, dass Veränderungen stattgefunden haben und sich das gesuchte Referenzhistogramm befindet. Worauf diese Vermutungen beruhen und wie sie sich berechnen lassen, soll hier nicht thematisiert werden. Die ROI erlaubt es allerdings, die Berechnungen auf einen Teilbereich des Bildes zu reduzieren und somit die Anzahl der zu berechnenden Farbhistogramme zu senken und dadurch Rechenzeit zu sparen.

5.2.7 Differenzenbild

Ein Differenzenbild C ist, wie sich aus dem Namen bereits schließen lässt, das Ergebnis der Subtraktion zweier (aufeinanderfolgender) Bilder A und B. Die Interpretation des Differenzbildes ist dabei die folgende: Das Differenzenbild zeigt Veränderungen von Bild A zu Bild B und kann so Aufschluss über mögliche Bewegungen zwischen diesen beiden Bildern geben. Die Ergebnisse sind dabei umso besser, je weniger sich die beiden Bilder im Hintergrund unterscheiden und die Kamera bewegt wird. Die Verwendung für die PG ergibt sich somit fast automatisch. Die Verfolgung der Person innerhalb des Hörsaals erfolgt durch zwei statisch angebrachte Kameras, dabei verändert sich der Hintergrund kaum, leichte Veränderung durch Lichteinflüsse außer acht gelassen. Daher macht eine Nutzung von Differenzenbildern Sinn.
Die Berechnung der Differenz für einen Bildpunkt $\mathbf{p} := (p_x, p_y)$ im Bild C soll nun beispielhaft für ein RGB-Bild gezeigt werden. Dabei sind die Bildpunkte (p_{xa}, p_{ya}) und (p_{xb}, p_{yb}) die korrespondierenden Bildpunkte zu (p_x, p_y) aus Bild A und B. R_{xy}, G_{xy},B_{xy} sind dabei die RGB-Werte für den Bildpunkt (p_x, p_y) im Bild C, R_{xya}, G_{xya},B_{xya} sind dabei die RGB-Werte für den Bildpunkt (p_{xa}, p_{ya}) im Bild A und R_{xyb}, G_{xyb},B_{xyb} sind dabei die RGB-Werte für den Bildpunkt (p_{xb}, p_{yb}) im Bild B.

$$R_{xy} = |R_{xya} - R_{xyb}| \tag{59}$$

$$G_{xy} = |G_{xya} - G_{xyb}| \tag{60}$$

$$B_{xy} = |B_{xya} - B_{xyb}| \tag{61}$$

Um die Darstellung und das weitere Arbeiten mit den Differenzenbildern zu vereinfachen, werden diese in Binärbilder umgewandelt. Dabei zeigen weiße Bildpunkte keine Veränderung an und schwarze Veränderung.

Die Umrechnung eines RGB-Bildpunktes $\mathbf{p}$ mit RGB-Werten R_{xy}, G_{xy}, B_{xy} in einen Grauwertbildpunkt (p_{xg}, p_{yg}) und Grauwert Gr_{xy} erfolgt mit folgender Formel:

$$Gr_{xy} = 0.299 \cdot R_{xy} + 0.587 \cdot G_{xy} + 0.114 \cdot B_{xy} \tag{62}$$

Dieses Grauwertbild Gr kann nun mit einem Schwellwert t in ein Binärwertbild Bi umgewandelt werden, dass nur noch aus schwarzen und weißen Bildpunkten besteht. Dabei werden alle Bildpunkte der Grauwerte unterhalb dieses Schwellwertes liegen auf schwarz (0) gesetzt und alle Bildpunkte die

darüber liegen auf weiß (1) gesetzt.
Formal:

$$Bi_{xy} = \begin{cases} 1 \text{ falls } Gr_{xy} \geq t \\ 0 \text{ sonst} \end{cases} \tag{63}$$

Dieses resultierende Binärbild muss nur noch invertiert werden, um Bildpunkte **p** weiß darzustellen, an denen die Differenz null war, und Bildpunkte an denen Veränderung stattgefunden hat, schwarz darzustellen. Die folgende Bildersequenz stellt das ganze noch einmal graphisch dar.

Abbildung 51: Das Hintergrundbild des Hörsaals

5.3 Anwendung für den Laserpointer

In diesem Abschnitt wird auf die konkrete Anwendung der zuvor beschriebenen Verfahren innerhalb des Projektes eingegangen. Es werden zwei Verfahren beschrieben, welche alternativ benutzt werden können. Aufgrund der Erfahrungen die mit diesen einfachen Verfahren gesammelt wurde, entstand dann die letztendliche Lösung, die in Abschnitt 6.10 beschrieben wird.

Abbildung 52: Abbildung 51 mit einer Person

Abbildung 53: Das Differenzbild zwischen Abbildung 51 und Abbildung 52

5.3.1 Farbdifferenzen und V-Kanal

Erste Überlegungen zu einem bereichsorientierten Segmentierungsverfahren für den Laserpointer befassten sich mit der Suche nach einem geeigneten

Abbildung 54: Abbildung 53 mit Schwellwert

Abbildung 55: Abbildung 54 mit invertierten Schwellwert

Verfahren zur Farbsegmentierung, da der Laserpunkt durch sein charakte-

ristisches Rot ein sehr vielversprechendes Merkmal bot. Ein erster Ansatz, die roten Stellen im Bild zu ermitteln, basiert auf den in Abschnitt 5.2.4 eingeführten Farbquotienten. Jedes Einzelbild wird zeilenweise gefenstert durchlaufen, wobei jeweils folgender Test ausgewertet wird:
Ablauf:

1. wähle einen Anteil q am roten Farbwert, der nicht überschritten werden soll
2. wähle einen Schwellwert t für den roten Farbkanal
3. für alle Pixel innerhalb des Fensters überprüfe
   - hat das momentane Pixel einen Blauanteil der größer als $q \cdot R$
   - hat das momentane Pixel einen Grünanteil der größer als $q \cdot R$
   - hat das momentane Pixel einen Rotanteil der kleiner als t ist
   - dann dekrementiere einen Zähler
4. ist der überwiegende Anteil in dem Fenster nicht rot, so gib 0 aus sonst 1

Aus Performanzgründen entschied sich die Projektgruppe jedoch dazu, zu Gunsten des Farbdifferenzansatzes dieses Verfahren auszutauschen, da Farbdifferenzen sich ungefenstert einfacher berechnen lassen.
Nach Bildung der Farbdifferenzen bleiben noch mehrere Punkte übrig, da die Schwellwerte nicht zu hoch angesetzt werden dürfen. Um aus den verbleibenden Punkten den Laserpointer zu lokalisieren, wird der Wert des V-Kanals der jeweiligen Punkte überprüft und das höchstwertige Pixel des aktuellen Bildes als Position gewählt. Um Störungen durch statische Bildelemente zu unterdrücken, erfolgen die einzelnen Tests auf den Differenzen der aktuellen Kanäle mit denen des vorangegangenen Bildes. Alle drei Arbeitsschritte lassen sich dabei bei einem Bilddurchlauf realisieren, was gleichzeitig zur Folge hat, dass die Berechnung des Wertes für den V-Kanal nur für solche Bildpunkte erfolgen muss, deren Farbdifferenzen im Differenzenbild über einem Schwellwert liegen.
Der komplette Ablauf für ein Einzelbild:

```
int cur_r,cur_g,cur_b;
int last_r,last_g,last_b;
int rb_diff,rg_diff;
int y,ergebnis,max;
```

```
max = 0;
Pic cur_front = m_Frame->frameset[1];
Pic last_front = last_frame->frameset[1];
T* p_cur_front = cur_front[0][0];
T* p_last_front = last_front[0][0];
FOR int i = 0 TO i < frame_height DO
BEGIN
  FOR int j = 0 TO j < frame_width DO
  BEGIN
    cur_b = (int)*(p_cur_front++);
    cur_g = (int)*(p_cur_front++);
    cur_r = (int)*(p_cur_front++);
    last_b = (int)*(p_last_front++);
    last_g = (int)*(p_last_front++);
    last_r = (int)*(p_last_front++);
    cur_r -= last_r;
    cur_g -= last_g;
    cur_b -= last_b;

    IF(cur_r < 0) THEN DO cur_r = 0;
    IF(cur_g < 0) THEN DO cur_g = 0;
    IF(cur_b < 0) THEN DO cur_b = 0;
    rb_diff = cur_r - cur_b;
    IF rb_diff >= 25 THEN DO
    BEGIN
      rg_diff = cur_r - cur_g;
      IF (rg_diff >= 25) THEN DO
      BEGIN
        y=299*cur_r + 587*cur_g + 114*cur_b;
        ergebnis = ((cur_r*1000 - y));
        IF ergebnis > max THEN DO
        BEGIN
          max = ergebnis;
          lp_pos_x = j;
          lp_pos_y = i;
        END;
      END;
    END;
  END;
END;

IF max==0 THEN DO
  lp_pos_x=lp_pos_y=0;
```

Das Verfahren erwies sich als robust und effizient, so dass zunächst überlegt wurde, jeweils den kompletten Frame zu durchsuchen, aber es erwies sich als performanter, den Suchraum einzuschränken. Zu diesem Zweck kommt ein Partikelfilter zum Einsatz, welcher in Abschnitt 7.6.2 näher beschrieben wird.

5.3.2 Integralbilder mit Differenzen

Bei näherer Betrachtung der aufgenommenen Kamerabilder fiel auf, dass der Laserpointer durch die Parameter der Kamera und den begrenzten Dynamikumfang der Daten im Rechner häufig überstrahlt war, so dass das Zentrum nicht grellrot wie im realen Bild sondern weiß wirkte. Der in Abschnitt 5.3.1 beschriebene Ansatz funktionierte bei korrekt gewählten Schwellwerten trotzdem sehr gut, aber durch diese Erkenntnis wurde die Projektgruppe motiviert, einen alternativen Ansatz zu testen.
Dieser Ansatz basiert auf Differenzenbildern, um statische Bildinhalte auszuschließen. Dazu wird zunächst aus den einzelnen Farbkanälen des aktuellen und vorangegangenen Bildes der jeweilige Grauwert des betrachteten Pixels subtrahiert. Dieser wird invertiert und anhand eines Schwellwertes binarisiert. Die 1-wertigen Bildpunkte entsprechen denen mit hohen Intensitätswerten im Differenzenbild und kommen daher für den Laserpointer in Frage. Das resultierende Binärbild wird nicht direkt abgespeichert, sondern in ein Integralbild entsprechend Abschnitt 5.2.5 überführt. Der Vorteil des Integralbildes liegt darin, dass sich Rechtecksummen sehr effizient berechnen lassen, was im Fall der konkreten Anwendung einer Beschleunigung der Berechnung eines Fensters um das betrachtete Pixel entspricht. Bei diesem Verfahren wird in einem 5×5 Fenster die Anzahl der 1-wertigen Punkte durch einfache Aufsummierung gezählt und anschließend die Position mit der höchstwertigen Summe ausgegeben. Folgender Code soll die Funktionalität verdeutlichen:

```
IF(!integral_image) DO
  allocate_integral_image(frame_width,frame_height);

IF(!p_previous_akk_row_pos) DO
BEGIN
  p_previous_akk_row_pos = new unsigned int[frame_width];
  p_parp_temp = p_previous_akk_row_pos;
END;
```

```
p_previous_akk_row_pos = p_parp_temp;
unsigned int* p_previous_integral_pos = *integral_image;

int end_x = frame_width;
int end_y = frame_height;
int window_size = 5;
int double_window_size = 2*window_size;
int threshold = 200;
int min_distance = frame_width + 1;
int current_distance = frame_width + 1;
FOR int i = start_y TO end_y DO
BEGIN
  p_previous_akk_row_pos = p_parp_temp;
  p_integral_current_pos = integral_image[i];
  FOR int j = start_x TO end_x DO
  BEGIN
    current_color_b = (int)*(p_current_image++);
    current_color_g = (int)*(p_current_image++);
    current_color_r = (int)*(p_current_image++);
    last_color_b = (int)*(p_last_image++);
    last_color_g = (int)*(p_last_image++);
    last_color_r = (int)*(p_last_image++);
    current_color_gray = 0.299*current_color_r
          + 0.587*current_color_g + 0.114*current_color_b;
    last_color_gray = 0.299*last_color_r + 0.587*last_color_g
          + 0.114*last_color_b;
    abs_diff = abs(current_color_gray - last_color_gray);
    abs_diff = 255 - abs_diff;
    abs_diff > threshold ? abs_diff = 0 : abs_diff= 1;
    *p_integral_current_pos = 0;
    j==0 ? p_previous_integral_pos = p_integral_current_pos
          : p_previous_integral_pos = p_integral_current_pos -1;
    i==0 ? *(p_previous_akk_row_pos) = 0 : 0;
    *(p_previous_akk_row_pos)= *(p_previous_akk_row_pos)+abs_diff;
    *(p_integral_current_pos++) = *(p_previous_integral_pos++)
          + *(p_previous_akk_row_pos++);
    IF((i > 246) && (i < 500) && (j > 155) && (j < 512)) DO
    BEGIN
      window_heigth_pos = i - double_window_size;
      window_width_pos = j - double_window_size;
        window_sum = integral_image[i][j]
              + integral_image[window_heigth_pos][window_width_pos]
              - (integral_image[window_heigth_pos][j]
```

```
            +  integral_image[i][window_width_pos]);
        (j-window_size + 1) > start_x ? window_mid_pos_x =
            (j-window_size + 1): window_mid_pos_x = start_x ;
        (i-window_size + 1) > start_y ? window_mid_pos_y =
            (i-window_size + 1): window_mid_pos_y = start_y ;
      current_distance =(int)sqrt((double)(abs(lp_pos_x
            -window_mid_pos_x)+abs(lp_pos_x-window_mid_pos_x)));
      IF((max_value<window_sum)&&(current_distance<min_distance))DO
      BEGIN
        min_distance = current_distance;
        max_value = window_sum;
        (j-window_size + 1) > start_x ? max_value_pos_x
            = (j-window_size + 1): max_value_pos_x = start_x ;
        (i-window_size + 1) > start_y ? max_value_pos_y
            = (i-window_size + 1): max_value_pos_y = start_y ;
      END;
    END;
  END;
END;
lp_pos_x = max_value_pos_x;
lp_pos_y = max_value_pos_y;
```

5.4 Verworfene Erweiterung

5.4.1 Intensitätsverlauf

Auf der Suche nach weiteren Merkmalen, die den Laserpointer klassifizieren sollten, wurden mehrere unterschiedliche Aufnahmen des Punktes unter starker Vergrößerung betrachtet. Wie man in Abbildung 56 erkennen kann, ist es auffällig, dass das Zentrum des Laserpointers von sehr starker Intensität war, teilweise sogar für das menschliche Auge weiß erschien. Um dieses Zentrum herum nahm die Intensität gleichmäßig ab. Diese Erkenntnis war Veranlassung dazu, diesen Bestand als Merkkmal zu definieren.

Um den Test effizient zu gestalten, wurde auf eine radiale Betrachtung der Region um die jeweiligen Pixel verzichtet und stattdessen kreuzförmig in vier Richtungen verfahren. Ausgehend vom Zentrum wurde nach oben, unten, links und rechts geprüft, ob die Intensität zum jeweiligen Nachfolger in dieser Richtung abnimmt. Da der Test in dieser einfachen Form sehr empfindlich auf Helligkeitsschwankungen im Bild reagierte, wurden zwei Schwellwerte eingeführt, um die Empfindlichkeit des Merkmals zu steuern. Der Intensitätsabfall zum Nachfolger musste einerseits größer sein als eine untere

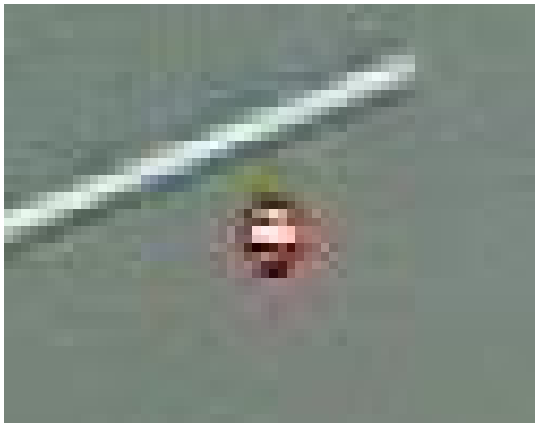


Abbildung 56: Detailaufnahme des Laserpointers

Schranke, um eine Abgrenzung gleichmäßiger Flächen zu erreichen, und andererseits kleiner als eine obere Schranke, damit harte Grenzübergänge nicht fälschlicherweise hinzugenommen wurden. Weiterhin musste eine Anpassung bezüglich der betrachteten Ausdehnung vorgenommen werden, um der Pixelgröße des Laserpointers zu entsprechen. Ein letzter Parameter beeinflusst, bei wie vielen der umgebenden Pixel diese Bedingungen erfüllt sein müssen, damit das betrachtete Pixel akzeptiert wird.

```
short weite = 3;
short diffmax=9;
short diffmin=1;
short counter = 4 * weite;
short num_passed = 4;
double grayin, grayout;
int h;
FOR h = 0 TO weite && (counter > 4) DO
BEGIN
  IN JEDE DER VIER RICHTUNGEN DO
  grayin = INTENSITÄT IN h SCHRITTEN IN RICHTUNG
  grayout = INTENSITÄT IN h+1 SCHRITTEN IN RICHTUNG
  IF((grayin-grayout>diffmax)||(grayin-grayout<diffmin))DO
  BEGIN
    --counter;
  END;
END;
```

Da der Laserpointer ein Punkt höchster Intensität ist, ist es sehr unwahr-

scheinlich, dass sich in seiner Nähe Punkte befinden, deren Intensität noch höher ist. Im Idealfall würde der Laserponiter ein globales Maximum darstellen. Um lokale Maxima von geringer Gesamtintensität auszuschließen, wird zusätzlich zu den bisher genannten Tests noch einmal in jede Richtung bei einem weiter entfernten Punkt geprüft, ob dieser heller ist und nur im negativen Fall der aktuelle Punkt behalten.

```
short schalter = 0;
IN JEDE DER VIER RICHTUNGEN do
  grayin = INTENSITÄT IN h SCHRITTEN IN RICHTUNG
  grayout = INTENSITÄT IN h+20 SCHRITTEN IN RICHTUNG
  if((grayout-grayin) >= 0)
  schalter=1;

IF ((counter > num_passed)&&(schalter==0)) THEN DO
BEGIN
  PUNKT AKZEPTIERT
END;
```

Nach ersten vielversprechenden Anwendungen auf einer Serie von Testvideos entschied sich die PG schließlich doch gegen die Verwendung dieses Verfahrens. Zum einen besteht die Rückprojektionsfläche in der Testumgebung aus einer Kunststoffscheibe, in der der Laserpointer ausfächert und so empfänglicher für diesen Test ist. Dies ist aber in der später angestrebten Anwendung nicht der Fall, was sich in Testaufnahmen aus der realen Umgebung belegten. Zum anderen war der Test im Vergleich zu den anderen genutzten Verfahren sehr rechenaufwändig und brachte keinen signifikaten Informationsgewinn.

5.4.2 Farbhistogramme

Um eine verbesserte Erkennung des Laserpointers zu erreichen, wurde auch versucht, die in Abschnitt 5.2.6 beschriebenen Farbhistogramme auf den Laserpointer anzuwenden. In der Anwendung erwies es sich als schwierig, dynamisch ein geeignetes Referenzhistogramm für den Laserpointer zu erstellen. Deshalb wurden zwei statische Referenzhistogramme erzeugt, wovon das günstigere gewählt wurde. Dieser Kompromiss war einer der Gründe, weshalb dieses Verfahren später fallengelassen wurde. Weiterhin zog der Laserpointer bei schnellen Bewegungen Spuren auf den Einzelbildern, welche für die Segmentierung durch Farbhistogramme ungünstig sind. Ein weiteres Problem lag in dem im Vergleich zu den in Abschnitt 5.3 genannten Verfahren erheblich höherem Rechenaufwand für die Berechnung der Farbhistogramme. Eine

komplette Berechnung der Einzelbilder wäre zu aufwändig gewesen, weshalb in diesem Zusammenhang erste Versuche mit einem Partikelfilter unternommen wurden, worauf in Abschnitt 7.6.1 kurz eingegangen wird.

5.5 Segmentierung der Person

Anders als bei der Segmentierung des Laserpointers wurde für die Segmentierung der Person innerhalb des Hörsaals von vornherein ein Trackingansatz angestrebt. Aus diesem Grund wird an dieser Stelle kein kompletter Ablauf wie in Abschnitt 5.3 angegeben, sondern es wird lediglich die grundlegende Idee für die Auswertung der Merkmale gegeben. Das erfolgreiche Tracking einer Person erfordert eine möglichst genaue Charakterisierung der Zielperson in Form von Merkmalen bzw. Merkmalsvektoren. Diese Merkmale können bspw. Größe, Haarfarbe, Kleidung usw. sein. Allerdings muss bei der Auswahl der Merkmale stets beachtet werden, dass die Anzahl so gering wie möglich, aber trotzdem charakteristisch ist. Merkmale, die für uns Menschen charakteristisch sind, müssen für den Computer keinensfalls charakterisch sein oder andersherum. Man muss sich also viele Gedanken um dieses Gebiet machen, um gute Ergebnisse zu erzielen.
Als Merkmal für die Person entschied sich die Projektgruppe für die in Abschnitt 5.2.6 eingeführten Farbhistogramme. Diese eignen sich sehr gut, um die charakteristischen Eigenheiten der Person zu beschreiben und der hohe Aufwand für die Berechnung wird dadurch kompensiert, dass durch den Einsatz eines Partikelfilters der Suchraum eingeschränkt wird, worauf in Abschnitt 7.5 näher eingegangen wird. Ein wesentlicher Punkt, der gegen das Verwenden von Farbhistogrammen für den Laserpointer sprach war das schwierige Erzeugen geeigneter Referenzhistogramme. Unter einschränkenden Rahmenbedingungen ist es der Projektgruppe gelungen, dieses Problem für die Anwendung auf die Person zu beheben, was in Abschnitt 7.5 erklärt wird und wodurch sich Farbhistogramme nun als sehr geeignetes Mittel herausgestellt haben.

6 Schwellwert-Konfiguration

6.1 Motivation

In den vorangegangenen Kapiteln wurden Verfahren zur Segmentierung des Bildes mit dem Laserpunkt besprochen. Diese Verfahren basierten auf Schwellwerten, die empirisch bestimmt wurden. Da wir zum Testen unseres Systems verschiedene Kameras sowie Leinwände zur Verfügung hatten, (Rückprojektionsleinwand, Hörsaalleinwand) bestand das Problem darin, einen passenden Schwellwert zu bestimmen, sodass das Verfahren auf allen Leinwänden funktioniert. Weiterhin erschweren unterschiedliche Lichtverhältnisse die exakte Bestimmung eines Schwellwertes. Aufgrund dieser Tatsache haben wir ein Tool entwickelt, dass unabhängig vom Gesamtsystem, halbautomatisch passende Schwellwerte für eine Kamera-Leinwand-Kombination bestimmt. Aufgrund der Unabhängigkeit vom Gesamtsystem wurde weiterhin die Berechnung der Kameraparameter in diesem Tool realisiert. Als erstes soll auf die Grundlagen eingegangen werden, mit denen die spätere Funktionalität realisiert wird. Das Tool hat folgende Bestandteile, die im Anschluss an den Grundlagenteil erläutert werden.

- Erstellen eines Kalibrierungsbildes
- Erstellen eines „Pfadbildes“
- Setzen der Projektionsflächengrenzen
- Halbautomatische Bestimmung von Schwellwerten der verwendeten Merkmale
- Berechnung der Gewichtung der verwendeten Merkmale

6.2 Varimax

6.2.1 Grundlagen

Varimax ist eine Methode in der Faktoranalyse. Nachdem man eine Menge von Faktoren einer Datenanalyse extrahiert hat, möchte man gerne diese Faktorenmenge analysieren. Diese Faktorenmenge muss nicht immer die volle Datenmenge umfassen. Nimmt man als Ausgangsbasis die PCA aus Kapitel 4.5 so kann man sich z.B. auf die Faktoren beschränken, die zu einer hohen kumulativen Varianz führen. Um die Analyse der Faktoren zu vereinfachen versucht man diese durch Rotation aussagekräftiger zu gestalten. Es gibt

zwei Arten von Rotationen, eine in der die entstehenden Achsen orthogonal zueinander sind und eine in der das nicht der Fall ist. Da man meist nicht alle Faktoren bei der Rotation betrachtet kann dies dazu führen, dass die Varianz der rotierten Faktoren aus dieser Untermenge gegenüber der Menge, die alle Faktoren umfasst, abnimmt. Dabei ist aber die Gesamtvarianz unverändert, es ändert sich nur die Partition. Im Weiteren soll unter dem Begriff Ladung, in Anlehnung an Kapitel 4.5, der Anteil eines Merkmals X sein, die auf den Faktor Y lädt. Bei der PCA wäre also so ein Faktor ein Eigenvektor der Kovarianzmatrix.

6.2.2 Orthogonale Rotation

Die orthogonale Rotation wird mit Hilfe einer Rotationsmatrix $\mathbf{R}$ beschrieben. In dieser Matrix steht jede Zeile für einen ursprünglichen Faktor und jede Spalte für einen rotierten Faktor. Es steht also an Position (i, j) der Kosinus des Winkels zwischen der alten und der neuen Achse. Eine Rotationsmatrix eines zweidimensionalen Koordinatensystems hätte demnach folgendes Aussehen:

$$\mathbf{R} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{64}$$

Abbildung 57 stellt diese orthogonale Rotation zwischen zwei Faktoren als Rotation eines Koordinatensystems dar.

6.2.3 Verfahren

Das Varimax-Verfahren wurde 1958 von Henry F. Kaiser entwickelt und ist eines der weit verbreitetsten Verfahren unter den orthogonalen Rotationen. Es liefert eine Rotation der Faktoren, sodass ein Faktor wenige hohe Ladungen und viele kleine Ladungen hat. Das bedeutet, dass die Ladungsmatrix $\mathbf{L}$ der Faktoren so transformiert wird, dass die Unkorreliertheit der Faktoren erhalten bleibt, siehe Abbildung 57. Ein zweites Merkmal der Rotation ist, dass der Zusammenhang zwischen Merkmal und Faktor erhalten bleibt. Die nun folgende Herleitung des Varimax Verfahrens ist weitestgehend aus [Zei00] entnommen. Das Varimax-Verfahren basiert auf der folgenden zu ma-

ximierenden Gleichung:

$$\begin{aligned} s^2 &= \sum_{k=1}^{q} s_k^2 \\ &= \sum_{k=1}^{q} \left(\frac{1}{p} \sum_{j=1}^{p} \left(l_{jk}^2 - \overline{l_{jk}^2} \right)^2 \right) \\ &= \frac{1}{p} \sum_{k=1}^{q} \sum_{j=1}^{p} l_{jk}^4 - \frac{1}{p^2} \sum_{k=1}^{q} \left(\sum_{j=1}^{p} l_{jk}^2 \right)^2 \end{aligned} \tag{65}$$

Dabei bezeichnet l_{jk}^2 die quadratische Ladung des j-ten Merkmals auf den k-ten Faktor, $\overline{l_{jk}^2}$ der Mittelwert der quadratischen Ladungen, q die Anzahl der Faktoren und p die Anzahl der Ladungen.
Bei dem Varimax-Verfahren wird dann folgende Erweiterung der obigen Gleichung verwendet, die auf Henry F. Kaiser zurückgeht und als Varimax-Kriterium bezeichnet wird.

$$\begin{aligned} V &= p^2 \sum_{k=1}^{q} \left(\frac{1}{p} \sum_{j=1}^{p} \left(z_{jk}^2 - \overline{z_{jk}^2} \right)^2 \right) \\ &= p \sum_{k=1}^{q} \sum_{j=1}^{p} z_{jk}^4 - \sum_{k=1}^{q} \left(\sum_{j=1}^{p} z_{jk}^2 \right)^2 \end{aligned} \tag{66}$$

mit

$$z_{jk} = \frac{l_{jk}}{k_j}, \text{mit } j \in \{1, \ldots, p\}, k \in \{1, \ldots, q\} \tag{67}$$

und

$$k_j^2 = \sum_{k=1}^{q} l_{jk}^2 \tag{68}$$

Die z_{jk}, aus den Gleichungen 66 und 67, stellen eine Normierung der Ladungen l_{jk} dar. Als $\mathbf{Z}$ soll im Folgenden die normierte Ladungsmatrix $\mathbf{L}$ bezeichnet werden.

Die Berechnung einer Rotation, die das Kriterium aus Gleichung 65 maximiert, ist ein iteratives Verfahren, welches nach jedem Schritt das Kriterium 66 auswertet. Als Abbruchkriterium des Verfahrens benutzt man folgende Gleichung:

$$\frac{V_n - V_{n-1}}{V_n} > 0.00001, \tag{69}$$

$p_{k,k'}$	$q_{k,k'}$	β
positiv	positiv	$0°$
positiv	negativ	$180°$
negativ	positiv	$0°$
negativ	negativ	$-180°$

Tabelle 2: Berechnungshilfe für den Rotationswinkel θ

dabei bezeichnet V_n den Wert des Kriteriums 66 nachdem n-mal alle Faktoren rotiert wurden. Als nächstes soll nun die Berechnung der Rotation von q Faktoren in einem Iterationsschritt beschrieben werden.
Zuerst werden für zwei benachbarte Faktoren folgende Hilfsgrößen berechnet. Seien $\mathbf{f}_k$ und $\mathbf{f}_{k'}$ zwei benachbarte Faktoren in der normierten Ladungsmatrix $\mathbf{Z}$, berechne:

$$a_{k,k'} := \sum_{j=1}^{p} \left(z_{jk}^2 - z_{jk'}^2\right) \tag{70}$$

$$b_{k,k'} := \sum_{j=1}^{p} \left(2 \cdot z_{jk}^2 \cdot z_{jk'}^2\right) \tag{71}$$

$$c_{k,k'} := \sum_{j=1}^{p} \left(z_{jk}^2 - z_{jk'}^2\right)^2 - \sum_{j=1}^{p} \left(2 \cdot z_{jk}^2 \cdot z_{jk'}^2\right)^2 \tag{72}$$

$$d_{k,k'} := 2 \sum_{j=1}^{p} \left(z_{jk}^2 - z_{jk'}^2\right) \cdot \left(2 \cdot z_{jk}^2 \cdot z_{jk'}^2\right) \tag{73}$$

$$p_{k,k'} := p \cdot d_{k,k'} + 2 \cdot a_{k,k'} \cdot b_{k,k'} \tag{74}$$

$$q_{k,k'} := p \cdot c_{k,k'} - a_{k,k'}^2 + b_{k,k'}^2 \tag{75}$$

Der Rotationswinkel der beiden betrachteten Faktoren $\mathbf{f}_k$ und $\mathbf{f}_{k'}$ ergibt sich dann durch

$$\theta_{k,k'} = \frac{1}{4}\left(\beta + \arctan\left(\frac{p_{k,k'}}{q_{k,k'}}\right)\right) \tag{76}$$

Der Summand β lässt sich aus Tabelle 2 übernehmen. Mit Hilfe des mit Gleichung 76 bestimmten Rotationswinkels lässt sich nun die Rotationsmatrix $\mathbf{R}$ aus Gleichung 64 aufstellen.

$$\mathbf{R} := \begin{pmatrix} \cos(\theta_{k,k'}) & -\sin(\theta_{k,k'}) \\ \sin(\theta_{k,k'}) & \cos(\theta_{k,k'}) \end{pmatrix} \tag{77}$$

Die rotierte Ladungsmatrix $\mathbf{Z}'$, für den nächsten Iterationsschritt, ergibt sich nun durch Multiplikation von $\mathbf{Z}$ mit $\mathbf{R}$.

$$\mathbf{Z}' = \mathbf{R} \cdot \mathbf{Z} \tag{78}$$

Sind alle Paare von Eigenvektoren rotiert so wird das Kriterium aus Gleichung 66 ausgewertet. Ist der Anteil der Differenz von neuem zu altem Wert des Kriteriums 66 echt größer als ein vorher definierter Schwellwert, hier 0.00001, so wird die Rotation beendet und als Ergebnis erhält man die rotierten Faktoren.

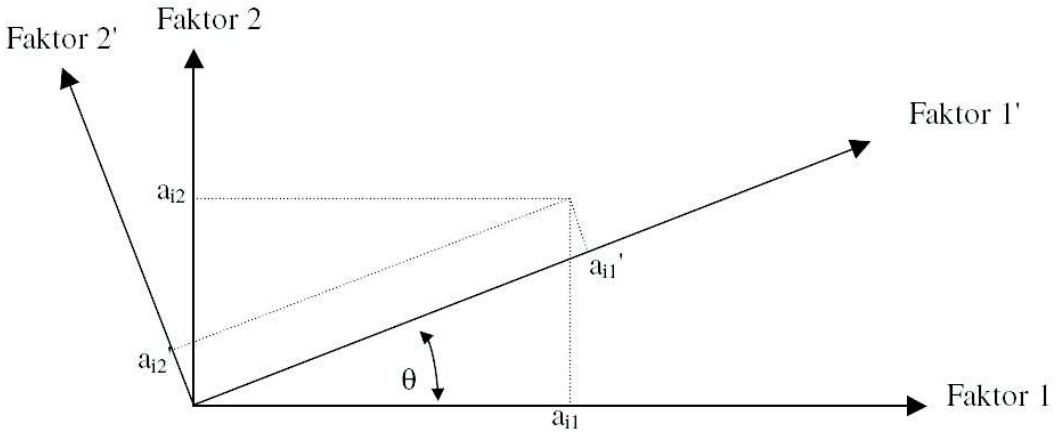


Abbildung 57: Orthogonale Rotation zweier Faktoren um den Winkel θ

6.3 Fuzzy Logik

6.3.1 Grundlagen

Der Begriff „fuzzy“ bedeutet soviel wie „unscharf, verschwommen“. Seit dem Jahre 1965 wurde durch L.A. Zadeh der Begriff der „Fuzzy Methoden“ in der angewandten Mathematik verwendet. Hiermit ließen sich Modelle, die auf unscharfen Daten beruhen, beschreiben. Parameter die empirisch ermittelt werden unterliegen einer natürlichen Unschärfe oder lassen sich nur vage bestimmen. Weiterhin kann die Bestimmung von präzisen Daten sehr komplex sein, oder es kann durch Modellfehler dazu kommen, dass man gleiche Werte nicht genau zu einer Klasse von vordefinierten Werten zuordnen kann. Als Beispiel sei hier die gesundheitsschädliche Strahlendosis gegeben. Bei

dieser Situation lässt sich keine konkrete Schranke angeben, ab wann eine Gesundheitsschädigung eintritt. Um so eine konkrete Schranke zu modellieren, müsste man sämtliche Auswirkungen auf die Gesundheit quantifizieren und in Relation setzen. Dieses wäre aber aufgrund der Komplexität des Sachverhalts kaum möglich.
Die unscharfe Modellierung von Mengen ist aber kein Konstrukt, was nur in der Mathematik vorkommt, jeder Mensch benutzt im Alltag unscharfe Mengen und trifft unscharfe Schlussfolgerungen. Um diesen Sachverhalt zu verdeutlichen betrachte man folgendes Beispiel:
Es seien zwei vage Aussagen gegeben:

- die meisten Männer sind groß
- die meisten Männer sind muskulös

Die Behauptung „Kai ist wahrscheinlich groß und muskulös“ würde niemand bezweifeln. Der obige Sachverhalt ließe sich auch mit stochastischen Methoden behandeln. Geht man von folgender Stichprobe aus:

- 90% der Männer sind größer als 170 cm
- 90% der Männer sind muskulös,

so würde man zu folgender Schlussfolgerung kommen. P(„muskulös“ $\wedge$ „$\geq$170“) = 81%. Kai wäre also mit einer Wahrscheinlichkeit von 81% groß und muskulös. An diesem Beispiel sieht man, dass die Merkmale Größe und Muskeln statistisch unabhängig sein müssen, damit man sie einfach miteinander multiplizieren kann. Man muss also zusätzliche statistische Annahmen machen, um mit den genau definierten Merkmalen umgehen zu können.
Ein weiterer Bereich, der mit unscharfen Begriffen beschrieben wird, ist die Steuerung von Systemen. Wir betrachten zuerst eine scharfe Steuerung eines Roboterfahrzeuges:

- Bewege dich 488 Meter geradeaus bis zur Kreuzung,
- drehe dich 82° im Uhrzeigersinn,
- bewege dich 5% der bis hierhin zurückgelegten Strecke
- bis zu dem Gebäude, das einen RGB-Wert von (0,255,0) hat.

Diese recht exakte Steuerung ließe sich mit dem nötigen Aufwand in einem Roboterfahrzeug integrieren. Ein Mensch, den man als unscharfen Roboter bezeichnen kann, würde mit folgenden unscharfen Aussagen zurechtkommen:

- Gehe ca. einen halben Kilometer bis zu einer Kreuzung,
- dann rechts,
- nach einem kurzen Stück
- erreicht man ein grünes Haus.

Es hat sich herausgestellt, dass sich die unscharfe Steuerung als wesentlich robuster erwiesen hat. So wird eine unscharfe Steuerung z.B. bei der U-Bahn in Sendai (Japan) erfolgreich verwendet.

Die obigen zwei Beispiele haben zwei Gebiete gezeigt in denen man „Fuzzy-Methoden“ einsetzt:

1. Beschreibung von unscharfer Mengenzugehörigkeit
2. Die Steuerung mit unscharfen Begriffen

Da die Fuzzy-Logik ein großes Gebiet der angewandten Mathematik ist, werden in den nächsten Abschnitten nur die Grundlagen von unscharfen Mengen und deren Verknüpfung beschrieben, da diese auch später in unserem Programm eingesetzt werden.

6.3.2 Fuzzy Mengen

In der klassischen Mengenlehre ist eine Menge A beschrieben durch eine Anzahl von Elementen einer Grundmenge X. Jedes Element der Grundmenge X gehört entweder zu der Menge A oder nicht. Dieses lässt sich durch eine Zugehörigkeitsfunktion beschreiben.

$$m_A(x) = \begin{cases} 1, & \text{wenn } x \text{ zu } A \text{ gehört} \\ 0, & \text{wenn } x \text{ nicht zu } A \text{ gehört} \end{cases} \tag{79}$$

Hingegen ist in der „Fuzzy-Theorie“ die Zugehörigkeitsfunktion graduell. Die Funktion kann also beliebige Werte annehmen. Zur besseren Übersicht kann man diese Werte auf das Intervall $[0 \ldots 1]$ normieren.
Die Definition einer Fuzzy-Menge B besteht aus dem Element der Grundmenge und seiner Zugehörigkeitsfunktion.

$$B = \{(x, m_B(x)) | x \in X\} \tag{80}$$

Diese Definition ist nicht von rein theoretischer Natur, sondern hat ganz konkrete Anwendungsgebiete. In einem sicherheitskritischen Bereich kann

ein Messwert von 7 eine Schwelle zwischen Sicherheit und Gefahr bedeuten. Ein Messwert von 6.999 kann aber damit nicht als völlig sicher und ein Messwert von 7.0001 als völlig unsicher bezeichnet werden. Eine graduelle Zugehörigkeit kann in diesem Beispiel dafür benutzt werden um einen unsicheren Bereich festzulegen. Abbildung 58 zeigt eine Möglichkeit eine graduelle Zugehörigkeit zu modellieren. Ein unsicherer Bereich könnte dort Anfangen, wo die Funktion einen vorgegebenen Wert unterschreitet, z.B. gilt ein Messwert mit Zugehörigkeit ≤ 0.8 als unsicher.

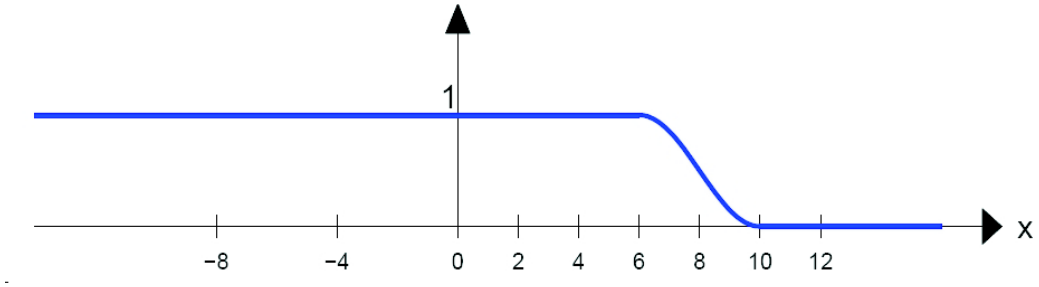


Abbildung 58: Zugehörigkeit von Messwerten zum sicheren Bereich [Obe03]

6.3.3 Verknüpfungen auf Fuzzy Mengen

Die wichtigsten Verknüpfungen der klassischen Mengenlehre sind die Konjunktion und die Disjunktion zweier Mengen. Der Begriff der T-Norm wird als Verallgemeinerung der logischen Konjunktion angesehen [Sch06].
Definition T-Norm:
τ heißt T-Norm wenn die folgenden Bedingungen erfüllt sind:

1. $\forall x \in [0 \ldots 1]$ gilt: $\tau(x, 0) = 0 \wedge \tau(x, 1) = x$
2. τ ist monoton, d.h. $\forall x, x', y, y' \in [0 \ldots 1]$ gilt: Wenn $x \leq x' \wedge y \leq y'$ dann folgt daraus $\tau(x, y) \leq \tau(x', y')$
3. τ ist kommutativ, d.h. $\forall x, y \in [0 \ldots 1]$ gilt: $\tau(x, y) = \tau(y, x)$
4. τ ist assoziativ, d.h. $\forall x, y, z \in [0 \ldots 1]$ gilt: $\tau(x, \tau(y, z)) = \tau(\tau(x, y), z)$

Der unscharfe Durchschnitt $A \cap B$ zweier unscharfer Mengen A und B ist eine T-Norm und lässt sich mit Hilfe der Zugehörigkeitsfunktionen beschreiben.

$$m_{A \cap B}(x) := \min(m_A(x), m_B(x)) \tag{81}$$

Weiterhin wird der Begriff S-Norm als Verallgemeinerung der logischen Disjunktion angesehen.
Definition S-Norm:
σ heißt S-Norm wenn die folgenden Bedingungen erfüllt sind:

1. $\forall x \in [0 \dots 1]$ gilt: $\sigma(x, 0) = x \wedge \sigma(x, 1) = 1$
2. σ ist monoton, d.h. $\forall x, x', y, y' \in [0 \dots 1]$ gilt: Wenn $x \leq x' \wedge y \leq y'$ dann folgt daraus $\sigma(x, y) \leq \sigma(x', y')$
3. σ ist kommutativ, d.h. $\forall x, y \in [0 \dots 1]$ gilt: $\sigma(x, y) = \sigma(y, x)$
4. σ ist assoziativ, d.h. $\forall x, y, z \in [0 \dots 1]$ gilt: $\sigma(x, \sigma(y, z)) = \sigma(\sigma(x, y), z)$

Die unscharfe Vereinigung $A \cup B$ zweier unscharfer Mengen A und B ist eine S-Norm und lässt sich mit Hilfe der Zugehörigkeitsfunktionen beschreiben.

$$m_{A \cup B}(x) := \max(m_A(x), m_B(x)) \tag{82}$$

Im ersten Fall ergibt sich der Graph des Durchschnittes, indem man jeweils den kleineren der beiden Werte $m_A(x)$, bzw. $m_B(x)$, auswählt, im Falle der Vereinigung wählt man den größeren der beiden Werte. Die Abbildungen 59 und 60 stellen dieses bildlich anhand der Zugehörigkeitsfunktionen dar. Man kann leicht nachvollziehen, dass die Definitionen 81, 82 eine Erweiterung des klassischen Mengendurchschnitts und der Mengenvereinigung sind.

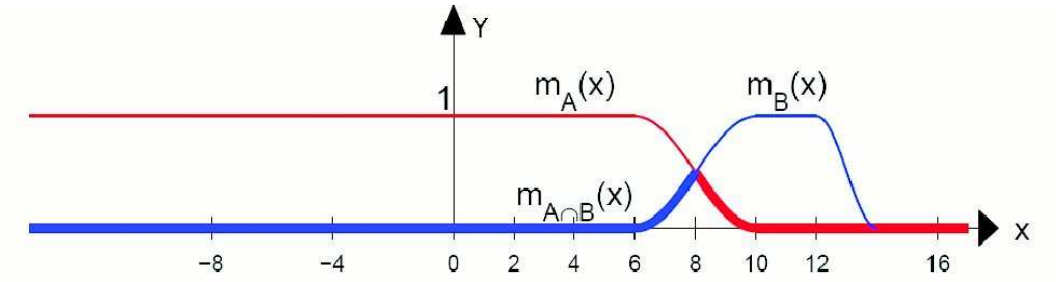


Abbildung 59: Schnitt zweier Fuzzymengen A und B [Obe03]

6.3.4 Unscharfe Zahlen und Fuzzy Repräsentationen

Die Darstellung einer unscharfen Menge kann nur dann praktikabel sein, wenn die Referenzmenge X endlich und nicht sehr groß ist. Da dieses nur selten der Fall ist muss man versuchen die Fuzzy-Menge gut zu approximieren.

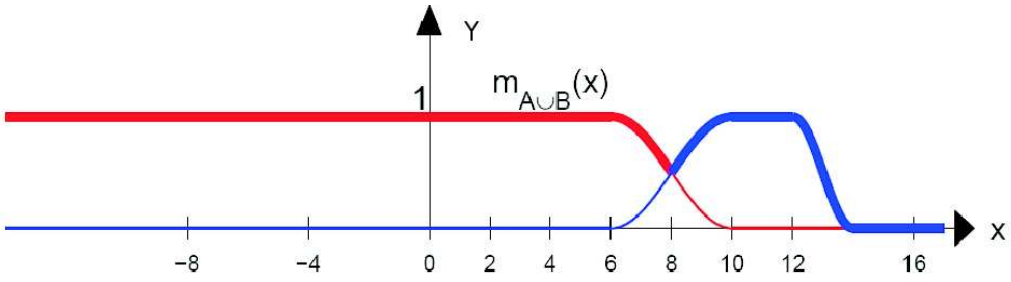


Abbildung 60: Vereinigung zweier Fuzzymengen A und B [Obe03]

Dazu wählt man als Zugehörigkeitsfunktion eine einfach zu beschreibende Funktion. Eine Auswahl der meist benutzen Funktionen soll nun vorgestellt werden.
Die einfachste Funktion ist eine monoton steigende bzw. fallende Funktion.

$$m_{a,b}(x) = \begin{cases} 0, & \text{falls } x \leq a \\ \frac{x-a}{b-a}, & \text{falls } a < x \leq b \\ 1, & \text{falls } x > b \end{cases} \tag{83}$$

Ein Beispiel zeigt Abbildung 61 a).
Weiterhin können exponentielle Funktionen als Beschreibung genutzt werden.

$$m_{a,b}(x) = \begin{cases} 0, & \text{falls } x \leq b \\ 1 - e^{-a(x-b)}, & \text{falls } x \geq b \end{cases} \tag{84}$$

Ein Beispiel zeigt Abbildung 61 b).
Als weiteres können Kombinationen von Funktionskonstrukten genutzt werden. Eine Dreiecksfunktion setzt sich z.B. aus einer monoton steigenden und einer monoton fallenden Funktion zusammen.

$$m_{a,b,m}(x) = \begin{cases} 0, & \text{falls } x \leq m - a \\ \frac{x-(m-a)}{a}, & \text{falls } m - a < x \leq m \\ 1 - \frac{x-m}{b}, & \text{falls } m < x \leq m + b \\ 0, & \text{falls } x > m + b \end{cases} \tag{85}$$

Ein Beispiel zeigt Abbildung 61 c).
Eine weitere, zusammengesetzte Funktion, ist die Trapezfunktion, die sich wieder aus einer monoton steigenden und einer monoton fallenden Funktion

zusammensetzt aber dazwischen, ein konstantes Plateau hat.

$$m_{a,b,m_1,m_2}(x) = \begin{cases} 0, & \text{falls } x \leq m_1 - a \\ \frac{x-(m-a)}{a}, & \text{falls } m_1 - a < x \leq m_1 \\ 1, & \text{falls } m_1 < x \leq m_2 \\ 1 - \frac{x-n}{b}, & \text{falls } m_2 < x \leq m_2 + b \\ 0, & \text{falls } x > m_2 + b \end{cases} \tag{86}$$

Ein Beispiel zeigt Abbildung 61 d).
Als Spezialfall einer Fuzzy Menge kann man die unscharfen Zahlen sehen. Nach Definition besteht die Zugehörigkeitsfunktion $m_A(x)$ einer unscharfen Zahl z aus einem monoton steigenden Bereich, einem eindeutigem zentralem Wert w mit $m_A(w) = 1$ und einem monoton fallenden Bereich. Aufgrund dieser Definition kann man unscharfe Zahlen mit Hilfe der oben aufgeführten Funktionen 83-86 beschreiben.

6.3.5 Verknüpfung unscharfer Zahlen

Um die Addition beziehungsweise Subtraktion von unscharfen Zahlen definieren zu können, betrachtet man das Erweiterungsprinzip von L.A. Zadeh, welches die Auswertung von Funktionen auf unscharfen Zahlen ermöglicht.
Es sei $f(x, y)$ eine Funktion auf zwei Variablen und a, b zwei unscharfe Zahlen, so ist der Funktionswert $f(a, b)$ definiert durch die Zugehörigkeitsfunktionen:

$$m_{f(a,b)}(z) := \sup_{z=f(x,y)} min(m_a(x), m_b(y)) \tag{87}$$

Anhand dieser Definition lässt sich nun die Addition beziehungsweise Subtraktion als Spezialfall von Gleichung 87 beschreiben durch:

$$m_{a+b}(z) := \sup_{z=a+b} min(m_a(x), m_b(y)) \tag{88}$$

$$m_{a-b}(z) := \sup_{z=a-b} min(m_a(x), m_b(y)) \tag{89}$$

6.4 Bresenham´s Line Algorithm

Der Bresenham´s Line Algorithm dient in der Computergrafik dem Zeichnen von Geraden auf einem Rasterbildschirm. Der Algorithmus wurde von Jack Bresenham einem Programmierer bei IBM entwickelt. Der Algorithmus kommt nur mit der Addition von ganzen Zahlen aus und ist deswegen

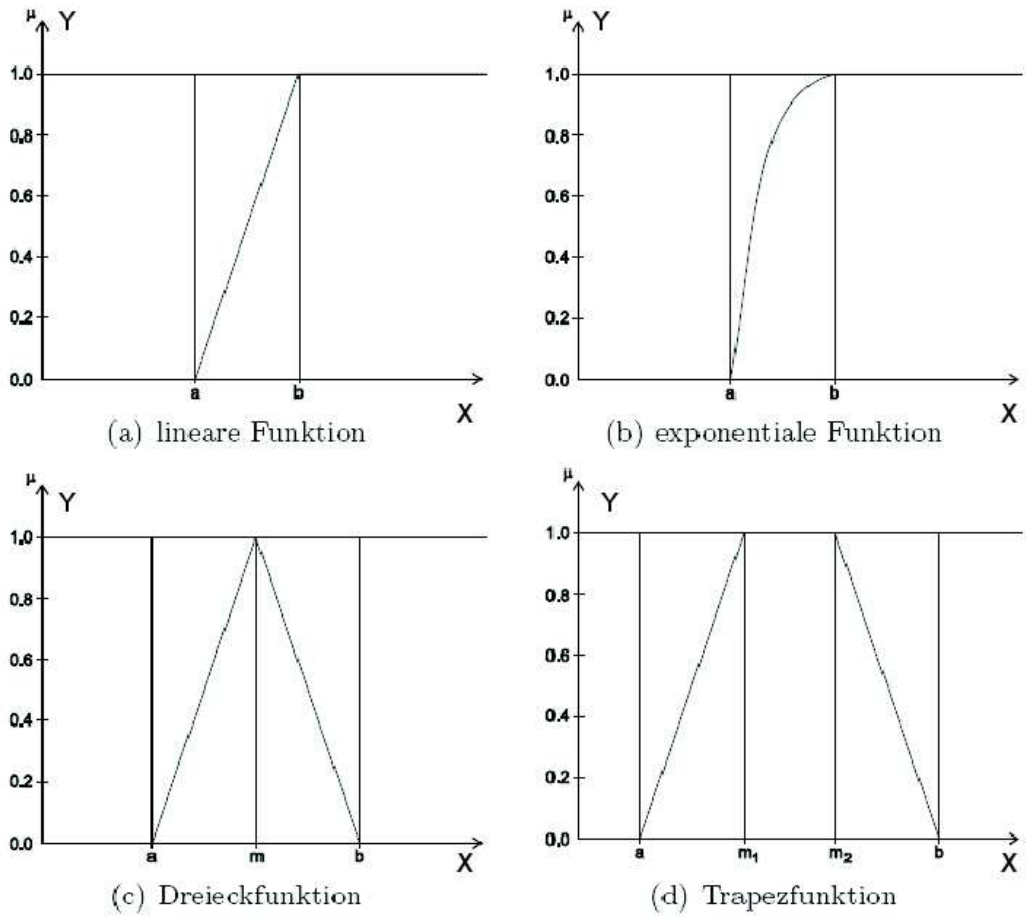


Abbildung 61: Verschiedene Zugehörigkeitsfunktionen [Sch06]

leicht und effizient implementierbar. Die, bei der Approximation der kontinuierlichen Werte, durch diskrete Werte, auftretenden Fehler, werden durch den Algorithmus minimiert. Mit einer Erweiterung des Grundalgorithmus, der nur für Geraden gedacht war, kann man Kreisabschnitte zeichnen. Auch bei dieser Erweiterung kommt man ganz ohne Multiplikationen oder Divisionen aus. Die Kreisgleichung kann so umgestellt werden, dass beim Zeichnen wieder nur Additionen verwendet werden. Als weitere Operation beim Zeichnen von Kreisen kommt das Shiften von Binärzahlen hinzu, die sich einfach und effizient durch Hardware realisieren lässt. Aufgrund der einfachen Implementierung und der Verwendung von Hardwarebausteinen werden der Bresenham-Algorithmus und dessen Erweiterungen heute in vielen Geräten, wie Plottern und Grafikkarten, eingesetzt.

6.4.1 Grundlagen

Die mathematische Grundlage des Bresenham´s Line Algorithm ist die Geradengleichung der folgenden Form:
Es seien zwei Punkte $\mathbf{p} := (p_x, p_y)$ und $\mathbf{q} := (q_x, q_y)$ gegeben. O.B.d.A. sei $\mathbf{q} > \mathbf{p}$, somit erhält man die Steigung $d_x := q_x - p_x$ in X-Richtung und $d_y := q_y - p_y$ in Y-Richtung. Weiterhin sei $d_x > d_y$.

$$y = p_y + (x - p_x) \cdot \frac{d_y}{d_x} \tag{90}$$

Diese Gleichung lässt sich durch Umformung auf die Form

$$0 = d_x - (y - p_y) - d_y \cdot (x - p_x) \tag{91}$$

bringen.
Wenn man anstatt der Null eine Fehlervariable f_d einführt, dann hat diese folgende Eigenschaften:
Sei $\mathbf{p}_r$ die momentane Position auf dem Rasterbildschirm und $\mathbf{p}_g$ die momentane Position die man anhand der Gleichung 90 bestimmen kann, dann gilt:

$$f_d \begin{cases} < 0 \text{ wenn } \mathbf{p}_r \text{ unterhalb von } \mathbf{p}_g \text{ liegt} \\ = 0 \text{ wenn } \mathbf{p}_r \text{ mit } \mathbf{p}_g \text{ übereinstimmt} \\ > 0 \text{ wenn } \mathbf{p}_r \text{ oberhalb von } \mathbf{p}_g \text{ liegt} \end{cases} \tag{92}$$

Weiterhin stellt man fest, dass f_d um d_y verringert wird, wenn man x um Eins erhöht und um d_x erhöht wird, wenn man y um Eins erhöht. Unter der Voraussetzung $d_x > d_y$ wird f_d wieder größer gleich Null wenn y um Eins erhöht wird. Die Idee des Algorithmus, der im nächsten Abschnitt beschrieben wird, ist nun die inkrementelle Bestimmung von f_d und die Entscheidung der Schrittrichtung, anhand der drei Fälle aus Gleichung 92.

6.4.2 Verfahren

Bei der Beschreibung des Algorithmus beschränkt man sich auf den ersten Oktanten, also Geraden mit einer Steigung im Intervall [0...1]. Für die anderen Fälle $\mathbf{q} < \mathbf{p}$ und $d_x < d_y$ kann man mit Fallunterscheidungen denselben Ansatz verfolgen. Der Algorithmus besteht nun aus folgenden Schritten. Zuerst wird die aktuelle Position in der größeren Richtung erhöht. Danach wird die Fehlervariable f_d um den Wert d_y verringert. Wird die Fehlervariable f_d echt kleiner Null, so wird die aktuelle Y-Position um Eins erhöht. Als Initialisierung für f_d nimmt man den Wert $\frac{d_x}{2}$, dies bedeutet, dass man eine Y-Steigung von Eins hat und damit auf der Hälfte der Strecke d_x einen

Schritt in Y-Richtung macht.
Der hier beschriebene Algorithmus wird in unserem Programm im Abschnitt 6.6 verwendet. Abbildung 62 zeigt eine Linie, die mit Hilfe des Algorithmus gezeichnet wurde. Die rote Linie beschreibt die originale Strecke zwischen den beiden Punkten, die blaue Linie ist die Rasterdarstellung, die der Algorithmus liefert. Hingegen wird in Abschnitt 6.8 eine leicht abgewandelte Form des Algorithmus verwendet.

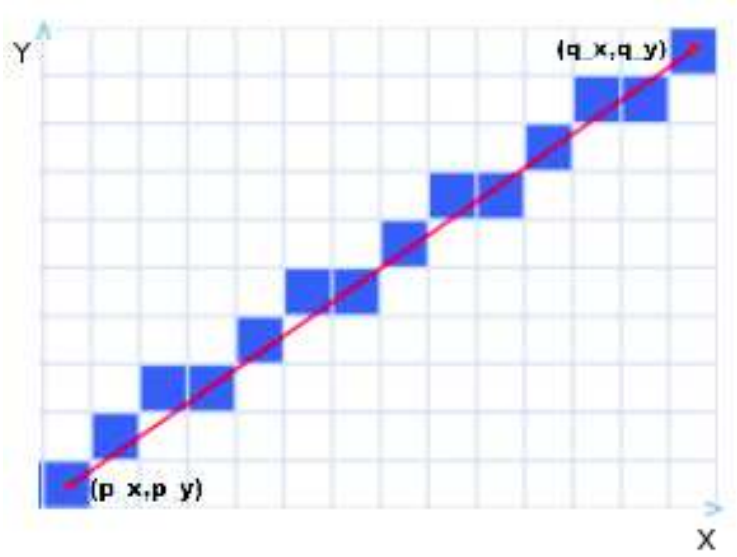


Abbildung 62: Rasterdarstellung einer Strecke zwischen zwei Punkten

6.5 Erstellen des Kalibrierungsbildes

Um eine Kamera wie in Kapitel 3 zu kalibrieren, benötigt man ein Kalibrierungsbild. Dieses Bild kann dann vor die Kameras gehalten werden um die Kameraparameter zu bestimmen. Das Tool bietet die Möglichkeit ein Schachbrettmuster zu erzeugen. Dieses Muster besteht dann aus einer vom Benutzer wählbaren Anzahl an schwarzen und weißen Flächen. Das erzeugte Bild kann dann entweder abgespeichert werden um es auszudrucken, oder es wird bildschirmfüllend angezeigt um es von der Projektionsfläche abfilmen zu können. Weiterhin werden diese Kalibrierungspunkte auf der Leinwand benötigt, um eine affine Transformation für Abschnitt 6.8.3 zu berechnen.

6.6 Erstellen eines „Pfadbildes“

Als Pfadbild ist hier ein Bild gemeint, welches nur einen Pfad enthält, der aus zwei Linien auf schwarzem Grund besteht (Abbildung 63). Wir haben uns für Blau als Farbe für die Linien entschieden, da sie gut auf hellem, sowie dunklem Untergrund zu erkennen ist. Dieser Pfad wird mit Hilfe von zehn vorgegebenen Punkten bestimmt und durchläuft das Bild von links nach rechts. Das Pfadbild hat die Auflösung des verwendeten Ausgabegerätes, sodass es über die komplette Projektionsfläche projiziert werden kann. Der Hintergrund des Pfades kann mit Hilfe des Transparent-Dialogs aus Abschnitt 10.1.2 durchsichtig gemacht werden, sodass man die normale Projektion im Hintergrund sieht. Dieses Bild wird mit Hilfe des „Bresenham's Line Algorithm“ erstellt, siehe Abschnitt 6.4.2. Um später die Position des Laserpunktes im aufgenommenen Bild bestimmen zu können, wird nicht die Linie, die direkt durch die vorgegebenen Punkte verläuft, gezeichnet, sondern basierend darauf ein Pfad, der aus zwei Linien besteht. Dazu werden die zehn vorgegebenen Punkte um zehn Pixel nach oben und nach unten verschoben. Dieses führt dazu, dass der Laserpunkt nicht direkt auf einer gezeichneten Linie liegen muss, was zu einer Verfälschung seiner Werte führen würde. Der Laserpunkt liegt vielmehr zwischen den beiden verschobenen Linien, in einem Pfad, auf dem Hintergrund. Dieses hat zwei Vorteile, erstens kommt es zu keiner Verfälschung der Werte die später für eine Merkmalsbestimmung benötigt werden und zweitens kann der originale, nicht verschobene Pfad, der durch die vorgegebenen Punkte führt, zur Berechnung der voraussichtlichen Position des Laserpunktes genutzt werden, siehe Abschnitt 6.8. Abbildung 63 zeigt so einen Pfad auf einer Powerpointpräsentation mit einem Laserpunkt zwischen den beiden Linien.

6.7 Setzen der Projektionsflächengrenzen

Zum Setzen der Projektionsflächengrenzen wird ein aufgenommenen Bild angezeigt und der Benutzer kann durch klicken auf die obere linke und untere rechte Ecke der Projektionsfläche im Bild einen Bereich festlegen, in dem die Projektion liegt. Diese Bestimmung hat den Vorteil, dass beim Segmentieren störende Bereiche im Bild, die nicht zu der zu untersuchenden Projektion gehören, nicht mehr betrachtet werden müssen und so zu falschen Ergebnissen führen. Weiterhin ergibt sich ein Laufzeitvorteil, wenn nicht das gesamte aufgenommene Bild verarbeitet werden muss.

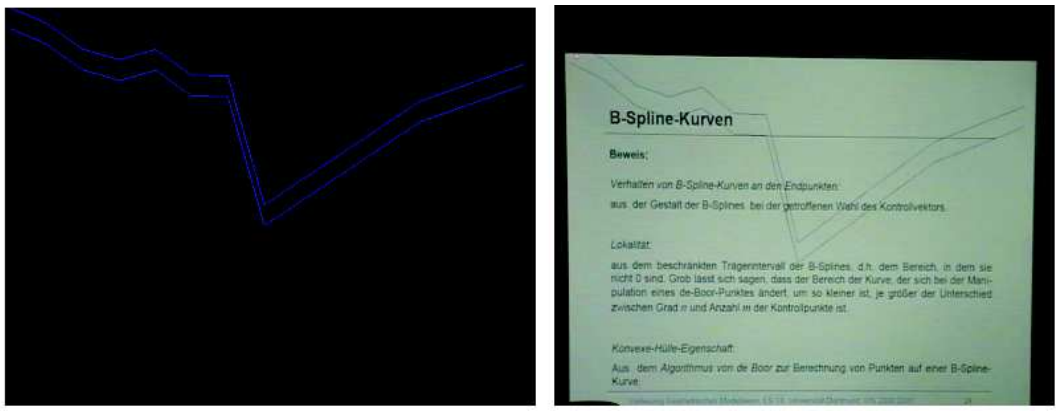


Abbildung 63: links das Pfadbild, rechts Pfadbild auf Präsentation

6.8 Bestimmung von Schwellwerten

Die von uns verwendeten Merkmale seien folgendermaßen bezeichnet:

1. $M_1 := R - B$
2. $M_2 := R - G$
3. $M_3 := V$

Diese drei Merkmale wurden schon in dem Abschnitt 5.3.1 zur Segmentierung eines Bildes anhand von Farbdifferenzen verwendet. Die Bestimmung der Schwellwerte für die einzelnen Merkmale wird von dem Tool halbautomatisch bewerkstelligt und besteht aus drei Schritten:

1. Aufnahme eines Testvideos.
2. Manuelle Bestimmung der Startwerte.
3. Automatische Anpassung der Startwerte an das Testvideo.

6.8.1 Aufnahme eines Testvideos

Mit Hilfe des in 6.6 beschriebenen Pfadbildes wird ein Testvideo aufgenommen, auf dem der Benutzer den Pfad mit dem Laserpunkt abfährt. Hierbei muss darauf geachtet werden, dass eine konstante Geschwindigkeit eingehalten wird und sich der Laserpunkt zwischen den beiden Linien aufhält.

Die Einschränkung auf eine konstante Geschwindigkeit ist nur für die Bestimmung der Schwellwerte wichtig, bei der späteren Anwendung muss diese Einschränkung nicht mehr eingehalten werden. Unter dem Pfadbild können nun typische Anwendungen, wie z.B. Präsentationen, ausgeführt werden, um eine bessere Anpassung der Werte zur späteren Anwendung zu bekommen. Das aufgenommene Video wird nun als Einzelbilder in einem vorgegebenen Verzeichnis gespeichert. Anstatt ein Testvideo aufzunehmen kann man auch die momentan aufgenommenen Bilder sofort als Einzelbilder speichern, ohne den Umweg über die Videoaufnahme zu gehen. Weiterhin kann man auch mehrere Sets von Einzelbildern mit verschiedenen Hintergrundanwendungen aufnehmen. Wie bei jedem lernbasiertem Verfahren so ist auch hier die Anzahl und die Ausgewogenheit der Einzelbildersets entscheidend für die Genauigkeit der zu bestimmenden Werte. Je größer und unterschiedlicher die Menge der Testsets ist, desto besser kann das Verfahren an die gegebenen Bedingungen angepasst werden.

6.8.2 Manuelle Bestimmung der Startwerte

Bei der manuellen Bestimmung der Startwerte werden dem Benutzer eine frei wählbare Anzahl von abgespeicherten Bildern eines Testsets gezeigt. Hierbei ist darauf zu achten, dass nicht alle Bilder aus einer nahezu gleichen Konfiguration stammen. Es sollte sich der Laserpunkt auf den betrachteten Bildern z.B. einmal auf hellem und einmal auf dunklem Hintergrund befinden. In jedem Bild kann der Benutzer mit der Maus den Laserpunkt markieren. Um die markierten Positionen wird ein Fenster, der Größe des Laserpunktes gelegt und die drei Merkmale innerhalb dieses Fensters werden berechnet. Aufgrund dieses Vorgehens führen wir nun eine erweiterte Bezeichnung der Merkmale M_i ein. Mit $M_i^q(\mathbf{p})$ wird das Merkmal M_i an Position q innerhalb des Fensters, das als Mittelpunkt die Position $\mathbf{p} := (x, y)$ hat, bezeichnet. Die Positionen q starten in der linken oberen Ecke des Fensters mit dem Wert 0, durchlaufen das Fenster zeilenweise und enden in der rechten unteren Ecke mit dem Wert $(N_F - 1) \cdot (M_F - 1)$. Die Werte n und m stellen die Breite und Höhe des Fensters dar. In unserer Anwendung gehen wir von einer Fenstergröße von 3×3 Pixeln aus. Weiterhin sei $M_i(\mathbf{p})$ der Wert des Merkmals M_i an der Stelle $\mathbf{p}$. Bei der Berechnung der $M_i^q(\mathbf{p})$ werden auch die jeweiligen oberen und unteren Grenzen $Upper_i$ und $Lower_i$ der Merkmale M_i vermerkt. Weiterhin werden in einer Häufigkeitsverteilung $H_i(M_i)$ das Auftreten des Merkmalwertes M_i innerhalb des Fensters festgehalten. Um später eine Gewichtung der Merkmale berechnen zu können, werden die $M_i^q(\mathbf{p})$ Werte in eine Menge P aufgenommen. Hierbei soll darauf hingewiesen werden, dass für die Menge P die Merk-

male $M_i^q(\mathbf{p})$ alle als eigenständig betrachtet werden. Somit erweitert sich die Menge der untersuchten Merkmale von vormals drei nun auf $27 = 3{\cdot}(3{\cdot}3)$.

6.8.3 Automatische Anpassung

Bei der automatischen Anpassung der Startwerte werden folgende Schritte ausgeführt:

- Bestimmung der voraussichtlichen Position $\mathbf{w}_t$
- Bestimmung der tatsächlichen Position $\mathbf{p}'$
- Vermerken der Werte $M_i^q(\mathbf{p}')$
- Anpassen der Verteilungen H_i

Um die voraussichtliche Position zu bestimmen, wird eine abgewandelte Form des in Abschnitt 6.4.2 vorgestellten Bresenham's Line Algorithm verwendet. Der Algorithmus wird diesmal nicht dazu genutzt einen ganzen Pfad zu zeichnen, sondern es wird die Position $\mathbf{w}_t := (x, y)$ eines Pfades zum momentanen Zeitpunkt t bestimmt. Die Position $\mathbf{w}_t$ wird auf folgende Weise berechnet. Zuerst wird eine Schrittweite s festgelegt, danach werden s Schritte des Bresenham's Line Algorithmus von der zuletzt bestimmten Position $\mathbf{w}_{t-1}$ aus ausgeführt und als Ergebnis $\mathbf{w}_t$ ausgegeben. Dieses Vorgehen hat den Vorteil, dass es zu einer konstanten Geschwindigkeit führt. Würde man nicht mit Hilfe der Schrittweite s die Position bestimmen, sondern sich z.B. nach dem X- oder Y-Wert von $\mathbf{w}_t$ richten, hätte man Änderungen in der Geschwindigkeit. Diese Änderungen treten auf, da der Pfad quer über das Bild führt und somit einmal die X-Distanz zweier Punkte größer als die Y-Distanz ist und umgekehrt. Die Position $\mathbf{w}_t$ wird nun mit Hilfe der geometrischen Transformation $\mathbf{R}$, die bei der Kalibrierung errechnet wurde, in das Koordinatensystem des momentan zu bearbeiteten Bildes transformiert:

$$\mathbf{w}'_t := \mathbf{R} \cdot \mathbf{w}_t \tag{93}$$

Als nächstes wird zeilenweise Pixel für Pixel das Bild abgelaufen und für jede Pixelposition das Merkmal $M_i(\mathbf{p})$ bestimmt. Gilt für den Wert $M_i(\mathbf{p})$, dass $Lower_i \leq M_i(\mathbf{p}) \leq Upper_i$ ist, so wird der euklidische Abstand e_p zwischen $\mathbf{p}$ und $\mathbf{w}'_t$ berechnet. Die Werte $M_i(\mathbf{p})$, die innerhalb der Grenzen liegen, werden in einer Hilfsstruktur gespeichert und anhand ihrer e_p Werte aufsteigend sortiert. Die Position $\mathbf{p}'$ mit dem geringsten Abstand $e'_p = \min_{\mathbf{p}}\{e_p\}$

wird als tatsächliche Position des Laserpunktes im momentanen Bild angenommen. Es werden die zugehörigen $M_i^q(\mathbf{p}')$ Werte im umgebenden Fenster mit Hilfe der vorher berechneten und abgespeicherten Werte bestimmt und in der Verteilung H_i erhöht. Weiterhin werden diese positiven Werte in die Menge P aufgenommen. Für alle anderen Positionen wird anhand einer Abstandsgewichtung $w(\mathbf{p})$ die Verteilung H_i an den Stellen $M_i^q(\mathbf{p})$ verringert. Als Abstandsgewichtung wurde folgende Funktion verwendet:

$$w(\mathbf{p}) := exp\left(1 - \frac{\alpha}{e_p}\right) \tag{94}$$

In Gleichung 94 wird das α so gewählt, dass weit entfernte Fenster ein hohes Gewicht erhalten. Da die Entfernung von der Bildgröße abhängt muss das α passend gewählt werden. In Testläufen haben wir den Wert $\alpha = 50$, für die Bildgröße 720×576, ermittelt.
Damit keine negativen Werte in H_i entstehen wird der Wert an Position $H_i(M_i^q(\mathbf{p}))$ nur anteilig nach folgender Gleichung verringert:

$$\begin{aligned} H_i(M_i^q(\mathbf{p})) =& H_i(M_i^q(\mathbf{p})) - w(\mathbf{p}) \cdot (0.01 \cdot H_i(M_i^q(\mathbf{p}))), \\ & \text{mit } i \in \{1,2,3\},\ q \in \{0,\ldots,(N_F-1)(M_F-1)\} \end{aligned} \tag{95}$$

Als weiterer Ansatz könnte man für alle $i \in \{1,2,3\}$ eine Häufigkeitsverteilung H_i^p für die Werte im Fenster zur positiven Position $\mathbf{p}'$ und H_i^n für die Werte aller anderen negativen Fenster berechnen, diese dann normalisieren und dann gegeneinander verrechnen.

Nachdem alle Testbilder bearbeitet wurden, werden die Verteilungen H_i normalisiert. Damit ist die automatische Bestimmung der Schwellwerte abgeschlossen und man kann die Werte, die echt größer Null sind, aus den Verteilungen H_i ablesen und als Schwellwertbereiche verwenden.

6.9 Berechnung der Gewichtung

Als nächstes soll eine Gewichtung W_i^q der Merkmale M_i^q berechnet werden. Die Lösung dieses Problems kann auf verschiedenste Weisen geschehen, wie z.B. mit linearer Optimierung oder dem Gradientenabstiegsverfahren. Wir haben uns aber dazu entschlossen, die schon für die FFTW-Merkmalsbestimmung verwendete PCA zu nutzen.

Zuerst werden für die M_i^q aus der Menge P die normalisierten Werte $\widehat{M_i^q} := H_i(M_i^q)$ bestimmt. Diese Werte bilden nun die Grundlage für die in Abschnitt 4.5 definierte PCA. Zuerst werden zu den $\widehat{M_i^q}$ für jedes i und

q der Mittelwert berechnet. Mit diesen Mittelwerten wird dann eine Kovarianzmatrix aufgestellt. Als Ergebnis der PCA erhält man $3 \cdot (3 \cdot 3)$, $3 \cdot (3 \cdot 3)$-dimensionale Eigenvektoren, die so genannten Faktoren und die zugehörigen Eigenwerten der Kovarianzmatrix. Das eigentliche Ziel der PCA, die Datenreduktion kommt hier aber nicht zum Einsatz. Es wird also nicht anhand der kumulativen Varianz entschieden, welche Faktoren ausreichen um die Eingabemenge zu approximieren. Damit werden alle Eigenvektoren, analog zu 4.5, als Faktoren aufgefasst. Eine Position j, eines Eigenvektors $\mathbf{u}$ drückt somit die Ladung des Merkmals M_j auf den Faktor $\mathbf{k}$ aus. Die zu den Eigenvektoren gehörenden Eigenwerte sollen später als Gewichte der einzelnen Merkmale aufgefasst werden. Diese Gewichte müssen nun den Merkmalen zugeordnet werden. Da aus den Eigenvektoren nicht sofort ersichtlich ist, auf welches Merkmal sie eine hohe Ladung haben, muss die Position j' des Eigenvektors $\mathbf{u}$ gefunden werden, der die größte Varianz bezüglich aller anderen Eigenvektoren hat. Hierbei kommt das in Abschnitt 6.2 vorgestellte Varimax-Verfahren zum Einsatz. Bei diesem Verfahren werden die Eigenvektoren so rotiert, dass sie jeweils an wenigen Stellen eine hohe Varianz haben und an allen anderen Stellen eine geringe Varianz. Während der Ausführung der Varianzmaximierung muss darauf geachtet werden, dass die Eigenwerte passend zu ihren rotierten Eigenvektoren sortiert werden. Bei dem hier verwendeten Varimax-Verfahren bedeutet ein Wert nahe Eins an einer Position j' im rotierten Eigenvektor $\mathbf{u'}$, dass das Merkmal $M_{j'}$ eine hohe Ladung auf $\mathbf{u'}$ hat. Die Gewichtung W_i^q des Merkmals $M_{j'}^q$ besteht nun aus dem Eigenwert des Eigenvektors $\mathbf{u'}$.

6.10 Anwendung

Die Erkennung des Laserpunktes mit den so bestimmten Merkmalsverteilungen H_i und den Gewichtungen W_i^q läuft nun folgendermaßen ab. Als erstes wird um die zu untersuchende Position $\mathbf{p}$ ein Fenster in der Größe des Laserpunktes gelegt. Innerhalb des Fensters werden die Merkmale $M_i^q(\mathbf{p})$ berechnet und danach der Wert $\widehat{M}_i^q(\mathbf{p})$ bestimmt. Diese Werte werden nun als unscharfe Mengen, analog zu Abschnitt 6.3.4, aufgefasst. Dieses Vorgehen erlaubt es nun die in Kapitel 6.3 aufgeführte Gleichung 88 zur Bestimmung der Summe von unscharfen Zahlen anwenden zu können. Es ergibt sich folgender Wert für das momentane Fenster F_k:

$$p(F_k) := \min_i \left(\sum_{q=0}^{(N_F-1)\cdot(M_F-1)} W_i^q \cdot \widehat{M}_i^q \right) \tag{96}$$

Das Fenster F_g in dem der Laserpunkt enthalten ist, wird nun folgendermaßen bestimmt:

$$F_g := \sup_k(p(F_k)) \tag{97}$$

Da eine Suche im kompletten Bild durch die Fensterung zu zeitintensiv ist, haben wir uns dazu entschlossen die Merkmalsverteilungen H_i und die Gewichte W_i^q als Eingabe für einen Partikelfilter zu verwenden, siehe Kapitel 7.4.

Weiterhin können die Verteilungen H_i dazu benutzt werden das primitive Verfahren aus Abschnitt 5.3.1 adaptiv zu gestalten. Hierbei würde man nur die drei Verteilungen H_i mit Hilfe der Methoden aus dem vorherigen Abschnitt bestimmen, ohne eine Gewichtung W_i^q zu berechnen. Danach würde man, ohne eine Fensterung, für jedes Pixel $\mathbf{p}$ überprüfen, ob $H_i(M_i(\mathbf{p}))$ einen Wert echt größer Null hat und nach dem Pixel $\mathbf{p}'$ suchen, für das $H_i(M_i(\mathbf{p}'))$ maximal ist.

7 Tracking

7.1 Motivation

In dem hier verwendeten Kontext der Bildverarbeitung dient das Tracken der Vorhersage der Lage des Laserpointers in den einzelnen Frames der Kamerastreams, die einmal den Benutzer und einmal die Projektionswand abfilmen. Mit Hilfe dieser Vorhersage lässt sich der Suchraum unseres Objektes einschränken, sodass nicht mehr das gesamte Bild nach dem Laserpointer durchsucht werden muss. Weiterhin bietet der verkleinerte Suchraum den Vorteil einer vielleicht leichteren Suche des Objektes, da hier weniger störende Einflüsse vorhanden sein können. Dieses führt zu einer besseren Auswahl der zuvor beschriebenen Segmentierungsverfahren. Weiterhin ergibt sich durch die Suche in einem kleineren Suchraum mit anderen Segmentierungsalgorithmen auch eine Laufzeiteffizienz.

7.2 Definition

Unter dem Begriff Tracking versteht man alle Bearbeitungsschritte, die nötig sind, um ein sich bewegendes Objekt zu verfolgen. Das Ziel besteht darin, Informationen über den Verlauf der Bewegung des Objektes herauszufinden. Hierbei sollen Messfehler bzw. Messrauschen wenig Einfluss auf die Bewegung haben. Die Informationen müssen nicht nur immer aus der Position des Objektes bestehen, sondern können auch Geschwindigkeit, Beschleunigung oder andere vorhersagbare Werte, wie zum Beispiel Börsenkurse sein.

Das Tracken eines Objektes besteht aus zwei Schritten, der Vorhersage und der Korrektur. Im Vorhersageschritt wird mit Hilfe eines Bewegungsmodells der neue Zustand geschätzt. Im Korrekturschritt wird dann der vorhergesagte Zustand mit dem tatsächlichen Zustand verglichen. Der tatsächliche Zustand stammt von einer Messung und wird durch das Beobachtungsmodell modelliert.

7.3 Mathematische Grundlagen und Struktur

Das Tracking eines sich bewegenden Objektes lässt sich in folgender Weise mathematisch beschreiben.
Sei X_i die Zufallsvariable, die den Zustand des Objektes in Schritt i angibt, und Y_i der, durch eine Messung festgestellte Zustand, y_i gibt den tatsächlichen Zustand im Schritt i an.

1. Vorhersage des nächsten Zustandes anhand von den zuvor beobachteten Zuständen:
$$P(X_i|Y_0 = y_0, ..., Y_{i-1} = y_{i-1}) \tag{98}$$
2. Korrektur, mit deren Hilfe der vorhergesagte und der tatsächliche Zustand gewichtet miteinander verrechnet werden.
$$P(X_i|Y_0 = y_0, ..., Y_i = y_i) \tag{99}$$

Bei der Korrektur ist darauf zu achten, dass die Gewichtung zwischen gemessenem und vorhergesagtem Zustand richtig gewählt wird. Ist die Gewichtung des Messwertes zu stark, so kann es bei verrauschten Messwerten eher dazu kommen, dass ein falscher Zustand angenommen wird. Wird hingegen die Vorhersage zu stark gewichtet, so können große Veränderungen der Messwerte erst langsam angenommen werden, es kommt also zu einer Verzögerung der tatsächlichen Bewegung. Beide Szenarien können, in Hinsicht auf die uns gestellten Aufgabe, dazu führen, dass der Suchbereich zu klein oder zu groß gewählt wird und das Auffinden des Objektes so verzögert oder unnötig erschwert wird. Dieses würde dann auch wieder zu einer Laufzeitverschlechterung führen, die in Realzeitanwendungen unerwünscht ist.

Zur Vereinfachung wird im Folgenden angenommen, dass die bedingten Wahrscheinlichkeiten 98 und 99 sich wie in einem Hidden Markov Modell verhalten. Dies bedeutet, dass

1. Der vorhergesagte Zustand nur von dem vorherigen Zustand abhängig ist.
$$P(X_i|X_0, ..., X_{i-1}) = P(X_i|X_{i-1}) \tag{100}$$
2. Neue Messungen nur vom direkten vorhergesagten Zustand abhängen und stochastisch unabhängig sind.
$$P(Y_i, Y_j, ..., Y_k|X_i) = P(X_i|Y_i)P(Y_j, ..., Y_k|X_i) \tag{101}$$

Zur Bestimmung dieser Wahrscheinlichkeiten verwendet man nun das Bayes´sche Gesetz:
Sei $P(X_0)$ die Wahrscheinlichkeit des Startzustandes und Y_0 die erste Messung so ergibt sich folgende Wahrscheinlichkeit der Vorhersage.

$$P(X_0|Y_0 = y_0) = \frac{P(y_0|X_0)P(X_0)}{P(y_0)} \tag{102}$$

Für den allgemeinen Fall gilt:

$$P(X_i|y_0, ..., y_{i-1}) = \int P(X_i|X_{i-1})P(X_{i-1}|y_0, ..., y_{i-1})dX_{i-1} \tag{103}$$

Für die Korrektur gilt:

$$P(X_i|y_0, ..., y_i) = \frac{P(y_i|X_i)P(X_i|y_0, ..., y_{i-1})}{\int P(y_i|X_i)P(X_i|y_0, ..., y_{i-1})dX_{i-1}} \tag{104}$$

Zur genauen Herleitung dieser Formeln siehe [Sch05].
Die Lösung dieser Gleichung hängt vom verwendeten Beobachtungs- und Bewegungsmodell ab.

7.3.1 Monte-Carlo-Integration

Wenn man ein sich bewegendes Objekt in einer Folge von Bildern verfolgen möchte, muss man das Objekt erst in den einzelnen Bildern erkennen. Meist ist das Objekt durch eine bestimmte Form oder Farbgebung charakterisiert. Diese Merkmale können aber auch, z.B. durch verschiedene Beleuchtung der Szene, im Hintergrund auftauchen, sodass man mehrere wahrscheinliche Objektpositionen hat, ohne dass man genau weiß, welche die korrekte ist. In der Wahrscheinlichkeitsverteilung der Lage treten also mehrere Peaks auf, die weiterverfolgt werden müssen. Man benötigt für die Korrektur der Vorhersage den Erwartungswert und die Varianz der Wahrscheinlichkeitsverteilung zur Bestimmung der Kovarianzmatrizen. Dieses bedeutet, dass man hochdimensionale Wahrscheinlichkeitsdichten integrieren muss. Dieses ist sehr aufwändig und damit zu rechenintensiv für Realzeitanwendungen. Als Lösung diese Problems verwendet der Partikelfilter die Methode der Monte-Carlo-Integration zur Bestimmung der Integrale, die hier kurz beschrieben werden soll.

Die Monte-Carlo-Integration basiert auf dem Gesetz der großen Zahlen, was besagt, dass:

$$P\left[E(X) = \lim_{x\to\infty} \frac{1}{n}\sum_{i=1}^{n} x_i\right] = 1 \tag{105}$$

ist.
Daraus folgt für den Erwartungswert einer Zufallsvariable X, dass

$$E(X) \approx \frac{1}{n}\sum_{i=1}^{n} x_i \tag{106}$$

Wendet man dieses dann auf eine Zufallsvariable g mit Dichte p an, so ergibt sich, dass:

$$\int_{y\in\Omega} f(y)\ dy \approx \frac{1}{n}\sum_{i=1}^{n}\frac{f(x_i)}{p(x_i)} \tag{107}$$

mit $f(x) = g(x)p(x)$ ist.
In Worten bedeutet dies, dass man den Erwartungswert einer Zufallsvariable mit Hilfe der Summe von endlich vielen Samples x_i (hier Partikel genannt) mit Dichte p bestimmen kann.
Bei dem normalen Partikelfilter werden die Partikel gleichverteilt gewählt. Um aber die oben angesprochenen Peaks in der Wahrscheinlichkeitsverteilung besser herauszuarbeiten, gibt es eine Erweiterung des Partikelfilters, das sogenannte importance sampling, bei dem die Partikel mit Gewichten versehen werden. Partikel in den interessanten Regionen der Wahrscheinlichkeitsverteilung werden somit höher gewichtet. Abbildung 64 zeigt dieses beispielhaft an einer beliebigen Verteilungsfunktion.

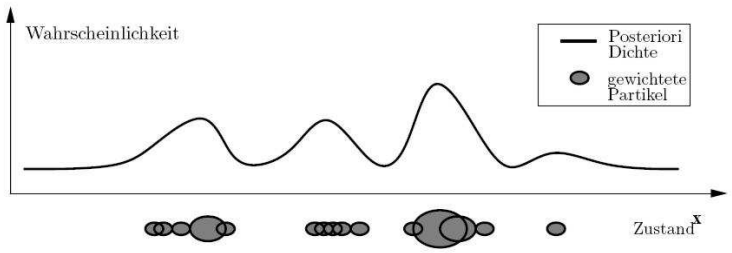


Abbildung 64: Verteilungsapproximation mit gewichteten Partikeln [Sch05]

7.3.2 Schematischer Ablauf

Anhand der unter Abschnitt 7.3.1 beschriebenen Grundlagen soll nun kurz auf den Ablauf einer Iteration des Partikelfilters eingegangen werden. Grundlage bildet hier die in Abschnitt 7.3 aufgestellte Formel Nr.(104) für den Korrekturschritt, die der Übersicht halber hier noch einmal aufgeführt wird.

$$P(X_i|y_0,...,y_i) = \frac{P(y_i|X_i)P(X_i|y_0,...,y_{i-1})}{\int P(y_i|X_i)P(X_i|y_0,...,y_{i-1})dX_{i-1}} \tag{108}$$

Der Term $P(X_i|y_0,...,y_{i-1})$ entspricht der Formel zur Vorhersage des nächsten Zustandes.

$$P(X_i|y_0,...,y_{i-1}) = \int P(X_i|X_{i-1})P(X_{i-1}|y_0,...,y_{i-1})dX_{i-1} \tag{109}$$

Hier gibt $P(X_i|X_{i-1})$ das Bewegungsmodell an, mit dem der alte Zustand mit Hilfe der aktuellen Wahrscheinlichkeitsverteilung (zweiter Faktor) in den neuen Zustand überführt wird.
Der Faktor $P(y_i|X_i)$ im obigen Integral der Formel Nr.(104) beschreibt das Beobachtungsmodell, mit dessen Hilfe die aktuelle Messung mit dem prädizierten Zustand verglichen wird. Bei dieser Wahrscheinlichkeitsverteilung kann es nun durch falsche Messungen zu mehreren Peaks kommen. Die Approximation dieser Verteilung wird nun mit Hilfe von Partikeln realisiert.
Auf diesen Vorüberlegungen basierend ergibt sich folgender schematischer Ablauf:

1. „Resampling“:Auswahl eines Partikels aus der Menge. Partikel mit hoher Wahrscheinlichkeit können mehrfach gewählt werden, Partikel mit kleiner Wahrscheinlichkeit gar nicht.

2. Für die ausgewählten Partikel wird mit Hilfe des Bewegungsmodells der neue Zustand bestimmt.

3. Bestimmung des neuen Gewichts mit Hilfe des Beobachtungsmodells.

Mit diesen neu gewichteten Partikeln wird im Korrekturschritt der neue Objektzustand geschätzt.
Abbildung 65 zeigt einen Iterationsschritt des Condensation Algorithmus von Isard und Blake. Der Algorithmus stellt eine Möglichkeit zur Realisierung eines Partikelfilters da und arbeitet mit dem oben beschriebenen importance sampling.

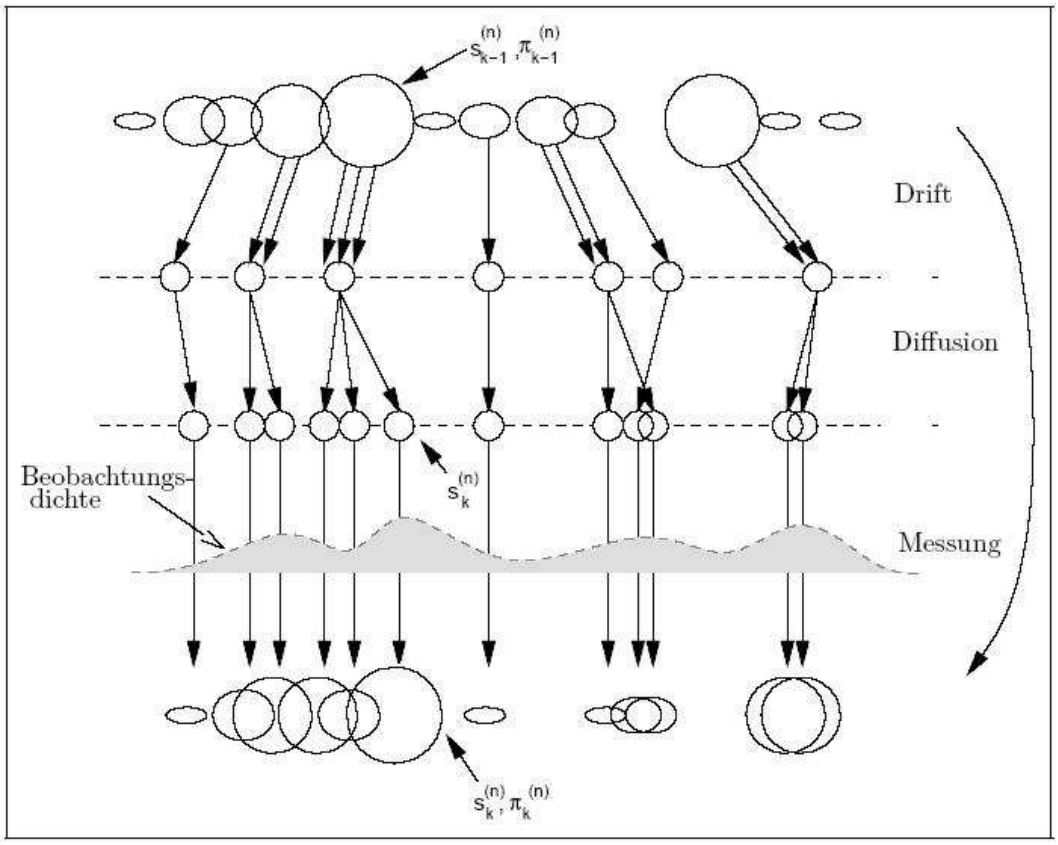


Abbildung 65: Ablauf des Condensation Algorithmus [Sch05]

7.4 Partikelfilter

7.4.1 Einführung

Gegenstand dieses Abschnittes soll der Gebrauch des Partikelfilters in der Projektgruppe sein. Eine kurze informelle Einleitung soll allerdings nicht vorenthalten werden.
Der Partikelfilter ist eine Methode das Tracking eines Objektes mit Hilfe von Merkmalen, die dieses Objekt mathematisch gut beschreiben, zu verfolgen. Das Merkmal ist dabei sehr wichtig und ist im Fall der Projektgruppe, dass in Abschnitt 5.2.6 ausführlich beschriebene Farbhistogramm. Die erzielten Ergebnisse sind allerdings von vielen weiteren Parametern abhängig, wie z.B. der Anzahl der Partikel. Je größer die Anzahl um so größer die Wahrscheinlichkeit das Objekt verfolgen zu können, umso größer allerdings auch der Effizienzverlust. Die Anzahl der Partikel

muss also möglichst gut gewählt werden. Die Partikel beschreiben dabei im Grunde nichts anderes als den Zustand eines Bildpunktes $\mathbf{p} := (x, y)$. Der Zustand kann dabei viele Parameter annehmen. Im Fall der PG sind das:

- die aktuelle Position $\mathbf{p}$
- die Geschwindigkeit des Partikels sowohl in X als auch in Y-Richtung
- den Wert der Abstandsfunktion zum Referenzhistogramm M
- die Gewichtung des Partikels w

Diese Parameter dienen dazu, dass Zielobjekt verfolgen zu können. Im folgenden Abschnitt sollen die einzelnen Methoden in Pseudocode aufgeführt und ihre Arbeitsweise erläutert werden.

7.4.2 Die Methoden des Partikelfilters

```
PartikelFilter(int pos_x,int pos_y,int particle_number)
BEGIN
  number_of_last_positions := 2;
  noise_x := 2.0;
  noise_y := 2.0;

  FOR int i := 0 TO particle_number DO
  BEGIN
    particles.push_back(
     Partikel
     (
     (int)NormalRandom((float)pos_x,noise_x),
     (int)NormalRandom((float)pos_y,noise_y),
     0.0,
     0.0
     )
     );
  END;

  position first_position;
  first_position.pos[0] := pos_x;
  first_position.pos[1] := pos_y;
```

```
  last_positions.push_back(first_position);
END;
```

Diese Methode erzeugt einen Partikelfilter. Die Übergabeparameter sind dabei, die beim ersten mal gefundene Position und die Anzahl der Partikel die um diese Position gestreut werden sollen. Hier kann davon ausgegangen werden, dass die Position zu 100 Prozent die wirkliche Position der Zielperson ist. Warum diese Annahme gilt, wird später erläutert. Innerhalb der Methode werden desweiteren Parameter für die Anzahl der zuletzt berechneten Positionen gespeichert, um aus den beiden Punkten einen Richtungsvektor für das Bewegungsmodell bestimmen zu können. Die Partikel werden in der Schleife um die Position, die übergeben wird, gestreut, und zwar mit dem Rauschwert $noise_x$ in X- und $noise_y$ in Y-Richtung.

```
set_particles(vector<Partikel> new_particles)
BEGIN
  particles.clear();
  FOR int i := 0 TO particle_count DO
  BEGIN
    particles.push_back(new_particles.at(i));
  END;
END;
```

Diese Methode bekommt als Übergabeparameter aus der aufgerufenen Klasse den Vektor aller Partikel mit den neu berechneten Parametern, wie z.B. der Abstandsfunktion. Diese werden dann einfach in einen neuen Vektor übernommen, der zur Weiterverarbeitung dient.

```
calculate_weights(void)
BEGIN
  float sigma := 0.13;
  float pi := 3.14159265;
  float temp_1 := 1.0/((sqrtf(2.0*pi)*sigma));
  float temp_2 := -1.0/(2.0*sigma*sigma);

  FOR int i := 0 TO particle_count DO
  BEGIN
    particles.at(i).weight := temp_1*
      exp(temp_2*(particles.at(i).match*particles.at(i).match));
```

```
  END;
END;
```

Diese Methode ist eine der Zentralsten des Partikelfilters. Sie berechnet aus dem Wert der Abstandsfunktion eine Gewichtung für das entsprechende Partikel. Wie in Abschnitt 5.2.6 angesprochen, bedeutet ein kleiner Wert der Abstandsfunktion eine große Übereinstimmung zum Referenzfarbhistogramm. Die Gewichtung des Partikels sollte hoch sein, je kleiner der Abstand ist. Die dort angegebene e-Funktion schafft genau dieses. Dieser Wert geht dann in den Zustand des Partikels als Gewichtung ein. Wichtig an dieser Stelle ist zu erwähnen, dass die Summe aller Partikelgewichtungen nicht zu eins aufaddiert werden kann.

```
resample()
BEGIN
  float sum_match := 0.0;
  sort(particles.begin(),particles.end(),matchsort);
  vector<Partikel> resampled_particles;

  FOR int i := 0 TO particle_count DO
  BEGIN
    sum_match += particles.at(i).weight;
    resampled_particles.push_back(particles.at(i));
  END;

  particles.erase(particles.begin(),particles.end());

  FOR int i := 0 TO particle_count DO
  BEGIN
    resampled_particles.at(i).cumweight := resampled_particles.at(i).weight/
     sum_match;
    if(resampled_particles.at(i).cumweight > 1/particle_count)
    BEGIN
      particles.push_back(resampled_particles.at(i));
    END;
  END;

  int number_of_missing_particles := particle_count-particles.size();

  position new_position;
```

```
  IF(!particles.empty()) THEN
  BEGIN
    new_position.pos[0] := particles.at(particles.size()-1).pos[0];
    new_position.pos[1] := particles.at(particles.size()-1).pos[1];
  END;
END;
```

Diese Methode dient der Auswahl bestimmter Partikel aus der Partikelmenge. Dabei werden die Partikel zunächst einmal normiert, indem die einzelnen Gewichtungen aufaddiert und jede Gewichtung eines Partikels durch diese Summe geteilt wird. Dadurch erhält man eine Gewichtung die sich zu eins summiert. Darauf folgend wird geprüft, ob die Gewichtungen der einzelnen Partikel nicht zu klein geworden sind. Fällt die Gewichtung eines Partikels unter diese Schranke wird es aus der Partikelmenge entfernt, ansonsten wird es beibehalten. Das Löschen eines Partikels führt nun allerdings nicht dazu, dass die Zahl der Partikel kleiner wird, sondern lediglich dazu, dass das Partikel um die zuletzt am höchsten gewichtete Position gestreut wird.

```
predict_particles()
BEGIN
  float x := calculate_max_distance(particles);
  FOR i := 0 TO particle_count-1 DO
  BEGIN
    particles.at(i).pos[0] += (int)particles.at(i).vel[0];
    particles.at(i).pos[1] += (int)particles.at(i).vel[1];
    particles.at(i).pos[0] := (int)NormalRandom(particles.at(i).pos[0],x/2);
    particles.at(i).pos[1] := (int)NormalRandom(particles.at(i).pos[1],x/2);

    IF (particles.at(i).pos[0] < 0)
    BEGIN
      particles.at(i).pos[0] := 0;
    END;

    IF (particles.at(i).pos[1] < 0)
    BEGIN
      particles.at(i).pos[1] := 0;
    END;
  END;
END;
```

Diese Methode berechnet wie weit die Partikel nun gestreut werden müssen, dabei wird jedes Partikel um die Hälfte des größten Abstands zum höchst gewichteten Partikels normal verteilt gestreut.

```
calculate_velocity()
BEGIN
  float velocity_x;
  float velocity_y;
  float distance_old_new;
  float norm_velocity_x;
  float norm_velocity_y;
  float max_distance;

  IF (last_positions.size() <= 1) THEN
  BEGIN
    velocity_x := 0.0;
    velocity_y := 0.0;
    distance_old_new := 0.0;
    norm_velocity_x := 0.0;
    norm_velocity_y := 0.0;
    max_distance := 0.0;
  END;
  ELSE
  BEGIN
    velocity_x := ((float)last_positions.at(1).pos[0]
                -(float)last_positions.at(0).pos[0]);
    velocity_y := ((float)last_positions.at(1).pos[1]
                -(float)last_positions.at(0).pos[1]);
    distance_old_new := sqrt((velocity_x)*(velocity_x)
                +(velocity_y)*(velocity_y));
    norm_velocity_x := velocity_x/distance_old_new;
    norm_velocity_y := velocity_y/distance_old_new;
    max_distance := calculate_max_distance(particles);
  END;

  FOR int i := 0 TO particle_count-1 DO
  BEGIN
    particles.at(i).vel[0] := norm_velocity_x*max_distance;
    particles.at(i).vel[1] := norm_velocity_y*max_distance;
  END;
```

```
END;
```

Diese Methode beschreibt die Berechnung des Bewegungsmodells mit Hilfe der letzten beiden ermittelten Positionen. Der Abstand der letzten Positionen wird berechnet, normiert und dann um die Länge des größten Abstandes vom höchst gewichteten Partikel zu allen anderen Partikeln in die errechnete Richtung gestreut.

```
vector<Partikel> get_particles()
BEGIN
  return particles;
END;
```

Diese Methode gibt den Vektor mit den Partikeln zurück.

7.5 Anwendung auf die Person

Dieser Abschnitt soll nun die Methoden aus den Abschnitten 5.2.6, 5.2.7 und 7.4 verbinden und zeigen wie sie zusammenarbeiten, um die Position der Person zu bestimmen. Dazu soll nun informell ein grober Ablauf skizziert werden.

7.5.1 Ablauf

Der Algorithmus basiert in erster Linie darauf, dass sich im ersten erfassten Bild nur die zu verfolgende Person befindet. Dies spielt eine wichtige Rolle, da das Differenzenbild sonst nicht zu einem eindeutigen Ergebnis kommen würde. Die Differenzenz zwischen dem aktuellen Bild und dem Hintergrundbild würde viele schwarze Bildbereiche aufweisen, die alle potenzielle Personen sein könnten. Die Einschränkung liefert uns dahingehend einen Vorteil, als dass sie diesen Fall nicht beachtet und den Algorithmus daher robuster und fehlerunabhängiger macht.
Das Differenzenbild mit seiner schwarzen Punktwolke liefert nun die Positionen, die einen Unterschied zum Hintergrundbild aufweisen. Dort wird Aktivität vermutet. Die Punktwolke hilft nun die Größe des Farbhistogramms zu berechnen. Punkte der Punktwolke werden aufsummiert und durch ihre Anzahl geteilt. Es wird also ein Mittelwertpunkt berechnet. Darauf folgend wird von allen Punkten, aus der Punktwolke, die Abstände zum

Mittelwertpunkt berechnet und aufsteigend sortiert. Im Anschluss werden nur 90 Prozent, der am nächsten liegenden Bildpunkte zum Mittelwertpunkt betrachtet. Ausreißer oder Rauschen innerhalb des Differenzenbildes kann so beseitigt werden. Die minimale X-Position, minimale Y-Position, maximale X-Position und die maximale Y-Position bilden dann den Rahmen des Referenzhistogramms. Die Differenz der minimalen und maximalen X-Position bzw. Y-Position liefert dann die Breite und Höhe des Farbhistogramms, welches dann als Referenzfarbhistogramm dient.
Die Position des Referenzfarbhistogramms wird nun an den Partikelfilter übergeben, der um diese Position normalverteilt eine bestimmte Anzahl von Partikeln streut. An diesen Positionen wird dann ein Farbhistogramm berechnet. Aus den gestreuten Positionen können wiederum die maximalen und minimalen Koordinaten bestimmt werden, um die ROI zu ermitteln und die Größe des zu berechnenden Integrals zu verkleinern. Nur innerhalb dieses Bereiches muss ein integrales Farbhistogramm berechnet werden. Dann wird für jedes Partikel das Farbhistogramm berechnet. Der Matchwert eines Partikels errechnet sich mit Hilfe der Abstandsfunktion durch Vergleich des Farbhistogramms mit dem Referenzfarbhistogramm. Sind für alle Partikel die Matchwerte berechnet, so werden diese nun in eine Gewichtung umgewandelt. Da wie schon erwähnt niedrige Matchwerte ein hohe Übereinstimmung aufweisen, wird dazu die in Kapitel 5.2.6 beschriebene e-Funktion verwendet. Aus diesen Gewichtungen, wird nun ein normiertes Gewicht berechnet, indem alle einzelnen Gewichtungen aufsummiert und durch ihr Gesamtgewicht geteilt werden.
Dieses normierte Gewicht gibt dann an, ob ein Partikel behalten wird oder ob es neu gestreut wird. Die zu kleinen Gewichte werden um die zuletzt ausgebene Postion gestreut, es geht also kein Partikel verloren. Die Menge der Partikel wird dann aufsteigend nach Gewichtung sortiert. Dieses führt dazu, dass das höchst gewichtete Partikel vorne ist. Dieses Partikel stellt die vermutete Position des zu verfolgenden Objektes dar. Die anderen Partikel werden nun mit Hilfe der Methoden des Bewegungsmodelles in die Richtung des höchst gewichteten Partikels gestreut. Dieser Ablauf wiederholt sich für jedes Bild.

7.5.2 Bildlicher Ablauf

Abbildung 66 zeigt wie sich aus dem Differenzenbild und der daraus resultierenden Punktwolke ein Referenzfarbhistogramm berechnen lässt. Dieser rote eingerahmte Bildausschnitt wird nun im Verlauf des Algorithmus verfolgt.

Abbildung 66: Die Abbildung zeigt den Ausschnitt des Bildes, der als Referenzhistogramm dienen wird

Abbildung 67 zeigt wie sich die Partikel um die höchst gewichtete Position streuen und dazu führen, dass Bewegungen der Personen erfasst werden können.

7.6 Anwendung auf den Laserpointer

Die im folgenden beschriebenen Verfahren basierten auf dem in Abschnitt 7.4 beschriebenen Partikelfilter, welcher hier nicht erneut komplett erläutert werden soll. Es werden stattdessen nur die charakteristischen Eigenschaften der jeweiligen Anwendung auf den Laserpointer beschrieben.

7.6.1 Ansatz mit Farbhistogrammen

Das im folgenden beschriebene Verfahren stellt einen frühen Ansatz zum Tracking des Laserpointers dar, welches entwickelt wurde, um den Rechenaufwand durch die Farbhistogramme zu kompensieren.
Zunächst wird auf dem gesamten Bild nach Abschnitt 5.4.2 nach dem Laserpointer gesucht. Anschließend werden um diese Position randomisiert gleichverteilt Partikel gestreut. Für die Vorhersage der Bewegung wird ein recht kompliziertes Modell gewählt. Es werden die letzten drei gefundenen sowie

Abbildung 67: Die Abbildung zeigt die gestreuten Partikel in Rechteckform, dabei ist das blaue das höchst gewichtete Rechteck

die aktuelle Position betrachtet. Für die Richtung der Bewegung wird über die Innenwinkel gemittelt und für die Weite über die Abstände der Positionen, wobei die chronologisch jüngeren Punkte stärker bei der Mittelung gewichtet werden, als die älteren. Für die eigentliche Bewegung kamen zwei Alternativen zum Einsatz. Bei der ersten Variante werden alle Partikel um den wie oben beschrieben berechneten Bewegungsvektor verschoben und bei der zweiten Variante wird nur die aktuelle Position verschoben und um diesen Punkt neu gestreut.
Dieses Verfahren wurde verworfen, da sich zum einen bei der Auswertung der Merkmale durch Farbhistogramme einige Probleme ergaben, was in Abschnitt 5.4.2 näher erläutert wird, und zum anderen das Bewegungsmodell ungünstig gestaltet war und der Punkt häufig verloren wurde.

7.6.2 Ansatz mit Farbsegmentierung

In diesem Abschnitt soll das aktuell benutzte Verfahren zum Tracking des Laserpointers beschrieben werden. Zunächst wird die initiale Position des Laserpointers anhand des in Abschnitt 6.10 Verfahrens lokalisiert. Um diese Position werden normal gleichverteilt Partikel gestreut. Um die Bewegungsrichtung zu bestimmen wurde folgendes Bewegungsmodell genutzt. Es wurde

jeweils der Bewegungsvektor zwischen der aktuellen Position und der vorherigen bestimmt und auf Länge eins normiert. Die für die Bewegung genutzte Länge des Vektors ergibt sich aus dem Abstand zwischen der aktuellen Position und dem am weitesten entfernten Partikel. Anschließend werden alle Partikel mit Hilfe des so gewonnenen Bewegungsvektors verschoben. Zur Bestimmung der Gewichtung der einzelnen Partikel wird Gleichung 97 für alle Partikel berechnet.

8 Treiber

Um die Positionierung des Cursors an die dem Laserpointer entsprechende Position auf dem Desktop umzusetzen, sowie das Abfangen und Einfügen von Rechtsklick-Ereignissen, wurde die Entscheidung getroffen, einen Maustreiber zu schreiben, der diese Aufgaben übernehmen sollte. Da die Treiberentwicklung kein einfaches Fachgebiet ist, wird im ersten von zwei Kapiteln auf die Grundlagen eingegangen, sowie ein kleiner Abriss über die Historie von Treibern gegeben, um den heutigen Stand einordnen zu können. Im zweiten Kapitel soll der Werdegang dieses Treibers vom Konzept, über den Entwicklungsprozess inklusive seiner Sackgassen, bis hin zum fertigen Treiber beschrieben werden und auf einem groben Niveau auch die interne Funktion des Treibers erläutert werden. Diese grobe Beschreibung ist nötig, da bei der Treiberentwicklung unzählige Details der Betriebssystemarchitektur beachtet werden müssen, welche von dem eigentlichen Konzept ablenken würden.

8.1 Grundlagen

Einen Treiber zu programmieren ist in den meisten Fällen eine große Herausforderung, da zu dieser Aufgabe ein tiefgehendes Verständnis der darunterliegenden Hardware, sowie der Software des Betriebssystems - in unserem Fall *Microsoft Windows XP* - gehört. Ein Fehler kann das System abstürzen lassen oder sogar die Hardware beschädigen. Zur Entwicklung des Maustreibers, mit dem die gefundenen Pointerkoordinaten an das System übergeben werden, gehört nun auch Basiswissen, um überhaupt erst einen solchen Treiber entwerfen zu können. Dieser Abschnitt soll einige Grundlagen vermitteln, um später in Abschnitt *Entwicklung des Treibers* den eigentlichen Entwicklungsprozess mit seinen Designentscheidungen nachvollziehen zu können. Begonnen wird mit einer einfachen Definition eines Treibers. Darauf folgen ein Einblick in den Aufbau eines Treibers und die Organisation von Treibern im Betriebssystem. Den Schluss bildet eine Einführung in den spezifischen Kontext von Maustreibern und Filtertreibern mit ihren Besonderheiten.

8.1.1 Eine kurze Definition des Begriffes “Treiber“

Ein Treiber ist kein reguläres Windows Programm, wie man es kennt. Ein Treiber ist vielmehr eine Bibliothek von standardisierten Routinen, die das Betriebssystem aufruft, um mit bestimmten Endgeräten zu kommunizieren. Ein wichtiges Merkmal ist, dass es keine main-Methode gibt und der Treiber somit auch nur arbeitet, wenn das Betriebssystem eine Anforderung schickt.

Beispiele für Situationen, in denen solche Anforderungen geschickt werden sind:

- ein Benutzer schließt ein PnP - Gerät an den Computer an oder entfernt es.
- eine Anwendung führt Schreib- und Leseoperationen auf einem Gerät aus (z.B. CD-ROM).
- Geräte werden aus Effizienzgründen vom Betriebssystem abgeschaltet, etwa bei Powermanagement auf Notebooks.

Treiber werden im sogenannten Kernelmode ausgeführt, was bedeutet, dass ihnen tiefere Eingriffe in das System gewährt sind, als etwa Usermode-Anwendungen, die lediglich auf die Win32API zugreifen, welche ihrerseits die nötigen Kernelmodefunktionen anstößt (vgl. Abbildung 68). Gerade weil ein Treiber so viel mehr Freiheiten hat, ist bei dessen Design höchste Vorsicht geboten. Das Betriebssystem geht davon aus, dass Kernelmode-Programme verantwortungsvoll mit den Systemressourcen umgehen und führt daher keine Überprüfung durch, ein kleiner Fehler in der Programmierung kann daher das System höchst instabil machen oder sogar abstürzen lassen.
Aufgaben eines Treibers sind etwa:

- Verwaltung der PnP Funktionalität
- PowerManagement
- Dispatch-Funktionen, um IRP's (Input-/Output-Request Packages, Anforderungen des Betriebssystems an den Treiber) an den nächsten Treiber im Device-Stack zu senden.
- Ladevorgang des Treibers

8.1.2 WDM-Treiber

DOS Die ersten PC's enthielten bis zu 640KB addressierbaren "echten" Speicher - in Form eines Chips. Erweiterungskarten wurden mittels Schaltern auf einen bestimmten I/O-Kanal eingestellt. Über die config.sys Datei ließen sich RealMode-Treiber laden. Die Anwendungen mussten über diese Treiber oder - weil das Interface des DOS-Betriebssystems unzureichend war - direkt mit diesen Erweiterungskarten kommunizieren. Dies bedeutete einen erheblichen Aufwand, die Treiber wurden in Assembler programmiert und griffen auf Schnittstellen des BIOS zurück.

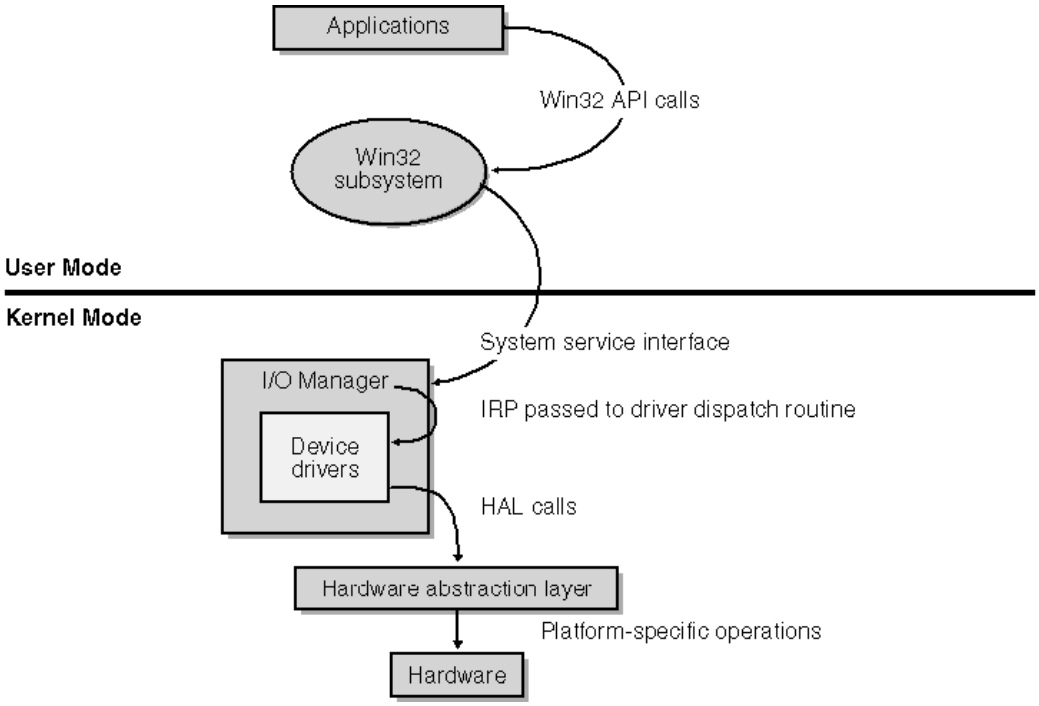


Abbildung 68: Unterteilung in User- und Kernelmode

DOS Weiterentwicklungen Auch nachdem Microsoft in späteren DOS-Versionen weitere Speicherverwaltungstechniken einführte (Protected Mode und Virtual Memory) blieben die Realmode-Treiber die gleichen.

Windows Windows war ursprünglich lediglich ein graphisches Frontend für DOS und nutzte ebenfalls Realmode-Treiber, aber über die Zeit entwickelten sich einige Windows Treiber für Standard-Geräte, wie Maus, Tastatur oder Display. Diese Treiber trugen die Endung .DRV und waren ebenso, wie die Realmode-Treiber in Assembler geschrieben. Windows und dessen Programme konnten diese Standard-Treiber nutzen, DOS-Programme mussten aber immer noch auf Realmode-Treiber zurückgreifen.

Windows 3.0 und Multitasking Windows 3.0 führte Multitasking ein, also die gleichzeitige Ausführung mehrerer Programme. Microsoft setzte dies durch Virtual Machines um, d.h. für jedes Programm wurde eine eigene virtuelle Maschine simuliert, mit eigenen Instanzen der Hardware des Systems. Dies führte aber zu einem Problem: Mehrere Programme konnten gegenläufige Anweisungen an ein und das selbe Gerät senden. Die Lösung waren sogenannte virtuelle Gerätetreiber, die zwischen den Virtual Machines und dem

jeweiligen Realmode-Treiber vermitteln sollten. Diese virtuellen Gerätetreiber hatten die Dateiendung VxD.386, wobei das x für den jeweiligen Gerätetyp stand. Es wurde aber außerdem die Unterteilung in User-Mode und Kernel-Mode Programme eingeführt (vgl. *Eine kurze Definition des Begriffes "Treiber"*). Kernelmodeprogramme werden als "vertrauenswürdig" angesehen, d.h. das System geht davon aus, dass sie robust und veranwortungsvoll mit der Harware umgehen; das Betriebssystem und Treiber gehören zu den Kernelmode-Programmen. Usermode-Programme dagegen werden als nicht "vertrauenswürdig" angesehen. Trotz dieser Unterscheidung wurde von Microsoft noch kein echter Speicherschutz eingeführt.

Die NT-Familie und jede Menge Treiber Microsoft entwickelte nun ein komplett neues Betriebssystem, das auf Sicherheit und Stabilität ausgelegt war, doch waren die Hardwareanforderungen nicht für einen durchschnittlichen PC ausgelegt und so wurde die ursprüngliche Betriebssystemfamilie mit Windows 3.1, Windows 3.11 und Windows 95 weitergeführt. Windows 95 führte PnP-Fähigkeiten ein und die VxD-Treiber für Windows 95 brauchten die I/O Maps nicht mehr zu berücksichtigen. Dennoch: für Windows 3.11 musste diese PnP-Fähigkeit zusätzlich in einem zweiten Treiber gefasst werden. Schließlich verlangte auch die NT-Betriebssystemfamilie einen eigenen Treiber, da mit der neuen Kernelarchitektur auch ein neues Treibermodell eingeführt wurde. Ein Hardwarehersteller musste also zu diesem Zeitpunkt bis zu drei verschiedene Treiber bereitstellen.

WDM Um diese Menge an unterschiedlichen Treibermodellen zu beseitigen, wurde ab Windows 98 das sogenannte WDM (Windows Driver Model) eingeführt, um diese zu ersetzen. Mit diesem Treibermodell war es möglich, für alle Betriebssysteme, ob NT- oder Windows 9x - Famlilie, nur noch einen einzigen Treiber bereitstellen zu müssen.

Dieses Treibermodell gilt nun auch noch für Windows XP, die Zielplattform für dieses Projekt. Obwohl für Windows Vista ein noch neueres Modell vorgestellt wurde, hat dieses viel Ähnlichkeit mit dem WDM, so dass ein Nebeneinander von komplett unterschiedlichen Treibermodellen, wie früher, nicht mehr auftritt.

WDM-Treiber sind nicht mehr in Assembler geschrieben, sondern werden als Sammlung von Subroutinen programmiert. Das Dateiformat eines Treibers ist exakt das gleiche, wie bei einer normalen Windowsanwendung, der einzige Unterschied, abgesehen von der Dateiendung „sys", ist das Fehlen der Main-Methode.

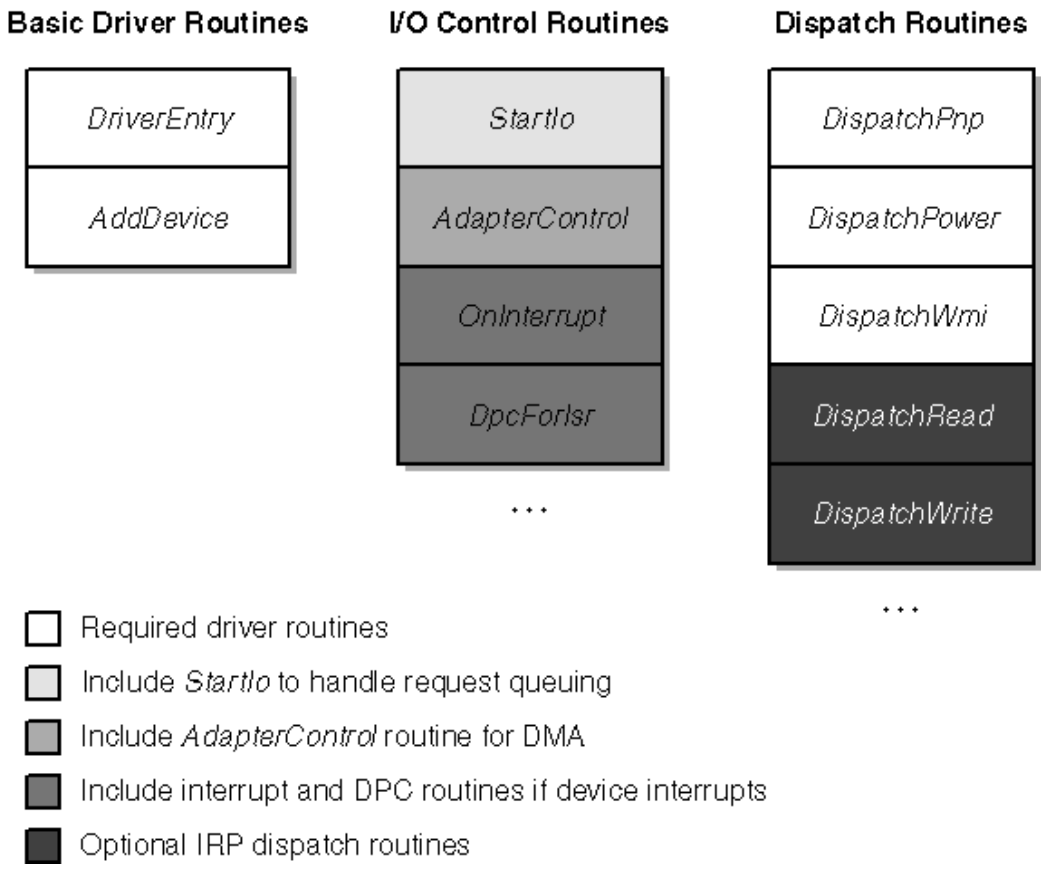


Abbildung 69: Typische Routinen eines Treibers (nicht vollständig)

Diese Subroutinen haben unter anderem die Aufgabe IRP's (=InputOutput Request Package) für das PowerManagement, und Plug&Play-Verhalten zu behandeln (Abbildung 69), aber auch Schreib- und Lesevorgänge auf das Device. Diese Routinen werden von Windowskernel bei Bedarf aufgerufen. Der Treiber stellt dem Betriebssystem also durch die standardisierten Subroutinen eine Schnittstelle zur Hardware zur Verfügung.

Driver Entry Diese Funktion nimmt globale Initialisierungen vor, wenn der Treiber das erste mal geladen wird. Ein Treiber kann zum Beispiel auch mehrfach geladen werden, wenn mehrere identische Geräte an einen Computer angeschlossen werden, etwa zwei Computermäuse desselben Typs.

AddDevice Während die DriverEntry-Funktion nur einmal für jeden Gerätetyp aufgerufen wird, wird die AddDevice-Funktion einmal für jedes angeschlossene Device aufgerufen.

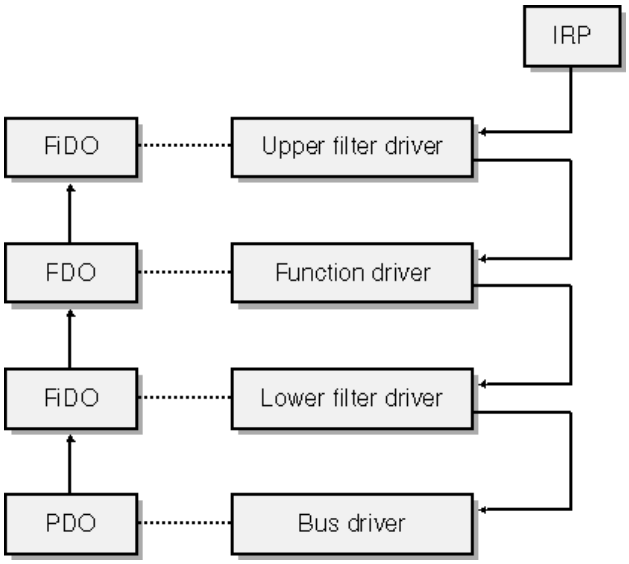


Abbildung 70: Der Device-Stack

weitere Subroutinen Es gibt noch weitere wichtige Subroutinen, die von den einzelnen Teilen des Betriebssystems aufgerufen werden, um z.B. PnP-Operationen oder Powermanagement durchzuführen. Auf diese soll hier aber nicht im Einzelnen eingegangen werden, da diese für die Entwicklung des Maustreibers nur von untergeordnetem Interesse sind. Auf die für den vorliegenden Treiber wichtigen Routinen wird später explizit eingegangen.

8.1.3 Driver Stacks

Es gibt für Geräte generell mehrere Treiber, die aufeinander aufbauend die Funktionalität des Gerätes umsetzen. Die Aufgaben der einzelnen Treiber reichen vom Bustreiber, der für die Anbindung der Hardware an den Rechner verantwortlich ist (z.B. PCI-Bus oder USB-Bus), bis zum Funktionstreiber, der für die eigentliche Funktionalität des Gerätes verantwortlich ist. Mindestens diese beiden Treiber hat jeder sogenannte Device Stack. Darüber hinaus existieren Filtertreiber, die zusätzliche Aufgaben übernehmen oder Klassentreiber für spezielle Geräteklassen, die immer wiederkehrende Aufgaben einer Geräteklasse sammeln (z.B. für alle Drucker, alle CD-ROM Laufwerke, usw.) (Abbildung 70). Alle IRP's, die einer dieser Treiber nicht bearbeitet, werden an den im Stack darunterliegenden Treiber weitergeleitet.

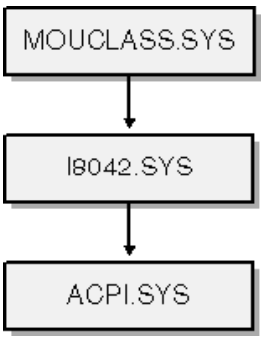


Abbildung 71: Device-Stack für PS/2-Mäuse

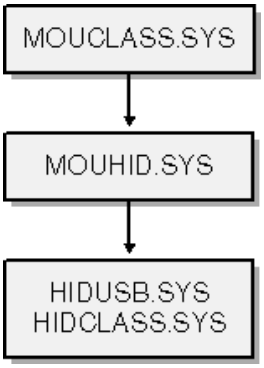


Abbildung 72: Device-Stack für USB-Mäuse (HID-Devices)

8.1.4 Maustreiber

Generell gibt es zwei mögliche Arten von Maustreibern: Treiber für PS/2 - und HID kompatible USB-Mäuse. Beiden gemein ist, dass sie sich in einen Device Stack einfügen, an dessen Spitze MOUCLASS.SYS steht. Dieser Treiber stellt dem Betriebssystem ein einheitliches Interface zu Verfügung. Daher kann man PS/2- und USB-Mäuse gleichwertig mit dem Betriebssystem verwenden. Ältere serielle Mäuse werden hier nicht besprochen, da ihre Architektur etwas komplizierter ist, aber es soll erwähnt werden, dass auch über deren Device-Stack der mouclass - Treiber liegt. Abbildung 72 veranschaulicht den Device Stack für eine USB-Maus.HIDUSB.SYS (ein HIDCLASS minidriver) fungiert als ein Bus Treiber, während MOUHID.SYS als ein Function Treiber agiert. MOUCLASS.SYS ist ein oberer Klassenfilter für die Maus-Geräteklasse. Abbildung 71 ist ein vergleichbares Diagramm für geeignete PS/2 Mäuse auf einem anderen System. I8042.SYS ist der Funktionstreiber für den PS/2 Maus Port. Auf diesem bestimmtem System, ACPI.SYS (der Treiber mit Hauptverantwortung für das Power Management) agiert

als bus driver. Auffällig ist, dass beide Device Stacks MOUCLASS.SYS gemein haben. MOUCLASS stellt dem System eine konsistente Schnittstelle für Computermäuse zur Verfügung. Daher kann man PS/2 und USB-Mäuse gleichwertig benutzen.

8.1.5 Filtertreiber

Ein Filtertreiber fügt sich über oder unter dem Funktionstreiber in den Device Stack ein und filtert die Nachrichten an den Treiber, d.h. alle Nachrichten werden vom Filter-Treiber empfangen, evtl. bearbeitet und an den im Stack darunter liegenden Treiber weitergeleitet. Es ist auch möglich, mehrere dieser Filter übereinanderzulegen (vgl. Abbildung 73). Anwendung finden solche Treiber als Filter für ganze Geräteklassen, bei denen bestimmte Aufgaben immer auf die gleiche Weise bearbeitet werden müssen. Aber auch Fehler in bestimmten Treibern können so ohne Kenntnis der Interna eines solchen fehlerhaften Treibers behoben werden, indem einfach die betreffenden Aufgaben vom Filter übernommen und die restlichen Subroutinen an den eigentlichen Treiber weitergeleitet werden.

8.2 Entwicklung des Treibers

8.2.1 Grundkonzeption

Um Mausaktionen abfangen, modifizieren und hinzufügen zu können, wurde beschlossen einen Filtertreiber für Computermäuse zu schreiben, der über dem entsprechenden Function-Treiber liegt. So wird es ermöglicht global die Tastendrücke und Bewegungen der Maus abzufangen, zu modifizieren, zu ersetzen oder eigene Aktionen einzufügen. Ursprüglich wurde lediglich ein PS/2-Treiber geplant und entwickelt, später jedoch dessen Funktionalität auf alle Maus-Geräte erweitert, d.h. alle Geräte, die den MOUCLASS-Filter als Klassenfilter besitzen. Für eine Auflistung der verschiedenen Treiber-Versionen, siehe Abbildung 74.

8.2.2 Eingriff in die Verarbeitung

Ein erster Treiber entstand durch Modifikation des "*moufilter*"-Beispiels aus dem Begleitmaterial zum DDK. Es ist mit diesem Treiber möglich in den Device-Stack einer PS/2 Maus einzugreifen und die Parameter zu verändern. Als Beispiel wurde hier die Y-Komponente der Bewegung negiert, so dass

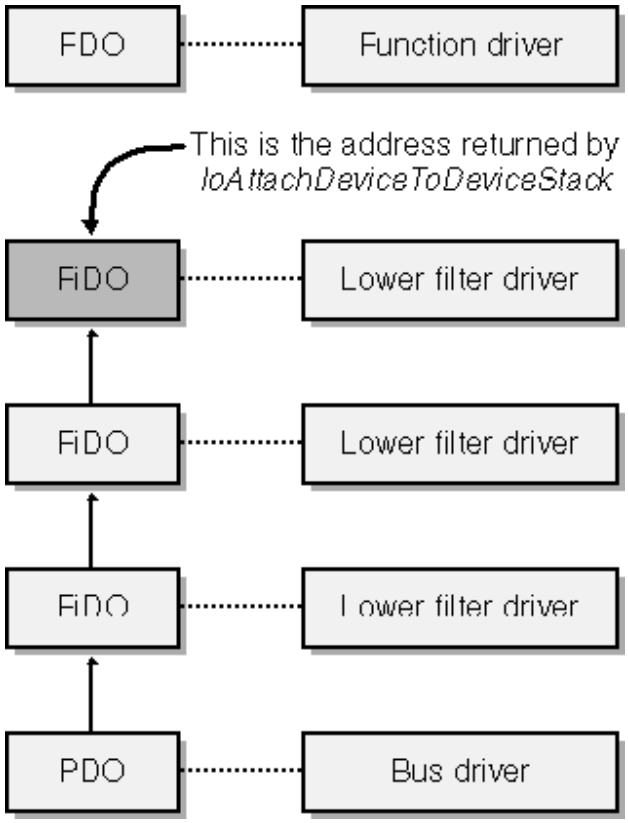


Abbildung 73: Stapel der Filtertreiber im Device-Stack

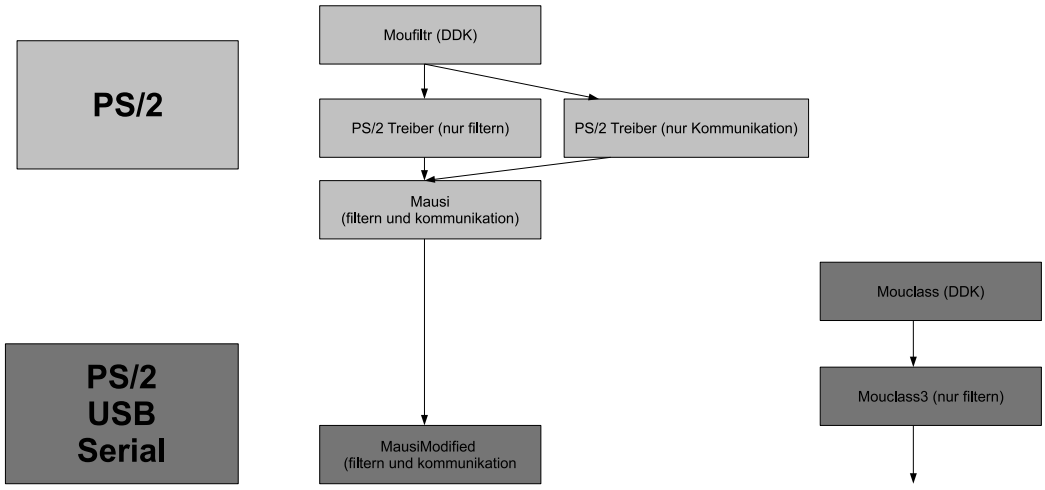


Abbildung 74: Treiberversionen

eine Aufwärtsbewegung der Maus den Cursor nach unten bewegt und umgekehrt. Anhand dieses Projektes war eine Einarbeitung in die Konzepte der Treiberprogrammierung einfach möglich.

8.2.3 Kommunikation mit dem Treiber

Ein weiterer Treiber wurde durch Modifikation eines minimalen Filtertreibers "*filtr*" aus dem Begleitmaterial zu [One03]. erstellt. Es ist diesem Treiber möglich, Nachrichten einer User-Mode-Anwendung zu empfangen, um z.B. später neue Cursorkoordinaten an den Treiber zu senden und die Manipulation der Mausdaten entsprechend zu gestalten. Diese Kommunikation läuft über sogenannte MS-DOS Devices. Ein solches DOS-Device ist mit einem Laufwerk oder einer COM-Schnittstelle vergleichbar und in der Tat handelt es sich dabei ebenfalls um DOS-Devices. Ein solches Device lässt sich wie eine normale Datei zum Schreiben bzw. Lesen öffnen. Der Treiber bekommt bei solchen Aktionen eine Benachrichtigung durch das Betriebssystem, indem die betreffende Subroutine aufgerufen wird. Ein Treiber registriert für diesen Vorgang zunächst einen solchen MS-DOS Device Name über den er zum Schreiben geöffnet werden kann, i.d.R. geschieht dies in der DriverEntry-Routine. Obwohl anhand dieses Treibers das Konzept der Kommunikation zwischen Usermode-Anwendungen und Treibern des Kernelmodes studiert werden konnte, stellte sich nach einigen Tests mit diesen beiden ersten Treibern heraus, dass es nicht möglich ist, ein Mausdevice zum Schreiben zu öffnen, um ihm Nachrichten zu senden. Dieses Unvermögen von Mausdevices, eine Kommunikationsverbindung mit dem Usermode herzustellen, ist darauf begründet, dass das Device schon vom Betriebssystem geöffnet ist, um Informationen über die Mausbewegungen und -aktionen zu empfangen. Es können nicht zwei Kommunikationsverbindungen gleichzeitig aufgebaut werden, d.h. dass so eine Usermodeanwendung niemals direkten Zugriff auf ein solches Device haben kann. Als Lösung wurde ein Umweg über ein zweites Device gewählt, dessen einziger Zweck es ist als Nicht-Maus-Device Nachrichten anzunehmen und innerhalb des Treibers das eigentliche Mausdevice zu steuern. Diese Technik wird auch als *Out-Of-Band-Communication* bezeichnet (vgl. Abbildung 75). So war es möglich einen Filtertreiber für PS/2-Mäuse zu schreiben, der auf Nachrichten aus dem Usermode bestimmte Mausaktionen blockiert, ändert oder ersetzt. Später wurde diese Technik auf den späteren allgemeinen Maustreiber übertragen. Der Maustreiber kann auf drei verschiedene Nachrichten aus dem Usermode reagieren:

1. Setzen der XY-Koordinaten: Der Treiber ändert die Zeigerposition des Mauszeigers auf dem Desktop.

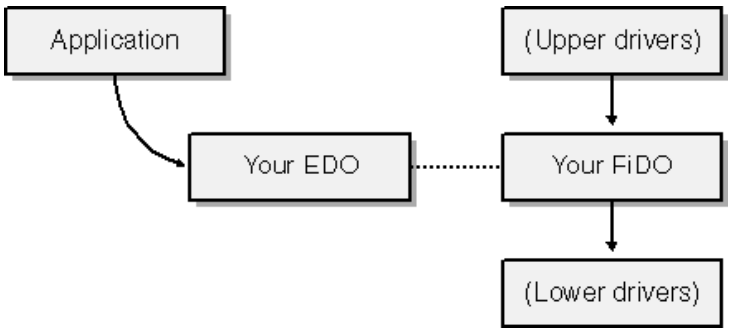


Abbildung 75: Out-Of-Band-Communication

2. Ausschalten der normalen PS/2 Maus: Da das System durch den Laserpointer gesteuert werden soll, kann hier die normale PS/2-Maus für die Dauer der Anwendung abgeschaltet werden.

3. Einschalten der normalen PS/2 Maus: Schaltet die normale Funktionalität der PS/2-Maus wieder ein.

Das Ein- bzw. Ausschalten der normalen angeschlossenen Maus hat hier den Zweck, das deren Mausbewegungen, die Bewegung des Cursors über den Laserpointer stören können.

8.2.4 PS/2 vs. USB

Der erste funktionierende Treiber, mit dem die Mausinformationen - Tastendrücke, Bewegungen - verändert werden und neue hinzugefügt werden konnte, war ein PS/2-Treiber. Dieser Treibertyp ist durch eine Unterstützung von Filtern durch gleich zwei Funktionen, die in die Verarbeitungs-Pipeline eingreifen für erste Schritte sehr geeignet. Dieser Treiber erlaubte die Manipulation der Zeigerkoordinaten und die Manipulation von Datenpaketen der Maus, sogenannten MOUSE_INPUT_PACKET's, die Angaben über die Bewegung der Maus, sowie der betätigten Tasten enthalten. Es stellte sich jedoch bald heraus, dass dieser Treiber nur geladen wird, wenn eine PS/2-Maus beim Bootvorgang angeschlossen ist. Der für den Einsatz bestimmte Laserpointer wird jedoch über den USB-Bus an den Rechner angeschlossen. Weil auch ähnliche Geräte standardmäßig nur über den USB-Bus angeschlossen werden können, wurde daher mit der Entwicklung eines USB-Treibers begonnen. Nachdem anfängliche Versuche den Klassenfilter "*mouclass*", dessen Source-Code ebenfalls im DDK mitgeliefert wird, zu manipulieren fehlschlugen, wurde nach einiger Recherche beschlossen, den Ursprünglichen PS/2-Treiber weiterzuentwickeln. So wurde der gesammte PS/2-spezifische Code entfernt und der

Treiber so zu einem Klassenfilter verallgemeinert, der für alle Mousedevices, ganz gleich über welchen BUS sie an den Rechner angeschlossen werden, gilt. Die PS/2-spezifischen Funktionen, die in die Verarbeitungs-Pipeline eingreifen, wurden nun durch eigene Routinen ersetzt. Das Ergebnis ist nun ein Filtertreiber für alle Mausgeräte, der es erlaubt MOUSE_INPUT_DATA - Packete in den Datenstrom einzufügen (setzen der gefunden Koordinaten), zu entfernen (Ausschalten der normalen Maus) oder zu manipulieren (Abfangen bzw. einfügen von Rechtsklick-Ereignissen für die spätere Verknüpfung mit dem Pie-Menu).

9 3D-Laserverlängerung

Ein Ziel der Projektgruppe ist es, den Laserstrahl in die 3D-Szene eines beliebigen Programmes, das 3D-Graphik bietet, zu verlängern. Um dies umzusetzen, wurde ursprünglich die Entwicklung eines eigenen Graphiktreibers geplant, später jedoch zugunsten einer einfacheren Lösung aufgegeben.
In diesem Abschnitt soll zunächst ein kleiner Überblick über die Aufgaben bei der Graphiktreiberprogrammierung gegeben werden, um die weitere Diskussion der Alternativen zu motivieren. Schließlich soll auf die gewählte Lösung mittels Proxy-DLL's eingegangen werden und deren Anwendung auf die OpenGL- sowie die Direct3D-Bibliothek besprochen werden.

9.1 Graphiktreiber

Ein Graphiktreiber muss im Gegensatz zu anderen Treibern (vgl. Abschnitt Treiber) eine Unmenge von verschiedenen API's anbieten. Diese API's beinhalten unter anderem Schnittstellen für die GDI (Graphics Device Interface), DirectX und OpenGL. Die Anzahl dieser bereitzustellenden Routinen ist so umfangreich, dass eine Umsetzung dieses Ansatzes im Rahmen dieser Projektgruppe nicht praktikabel war. Wie umfangreich diese API's sind soll nur das Beispiel der DDI-Routinen der GDI zeigen, die ein Graphiktreiber dem System zur Verfügung stellen **muss**. Diese umfassen mehr als 100 Routinen, die zur Verfügung gestellt und auf die Hardware abgestimmt werden müssen. Dazu kommen noch umfangreichere Bibliotheken, wie OpenGL und DirectX, die ein Graphiktreiber zwar nicht umsetzen muss, aber aus Performancegründen sollte. Für das Projekt der PG498 währen diese Bibliotheken zwingend erforderlich gewesen, da gerade in diese eingegriffen werden soll. Auch eine Lösung analog zu Filtertreibern, wie sie für die Maussteuerung umgesetzt wurde ist im Bereich der Graphiktreiber nicht umsetzbar, da hier kein Treiberstack vorhanden ist muss man, um eingreifen zu können, den originalen Graphiktreiber ersetzen.

9.2 Alternativen

Es wurde über mehrere Wochen intensiv nach Alternativen zur Nutzung eines Graphiktreibers gesucht. Die betrachteten Alternativen umfassen folgende Methoden:

- **DLL's von OpenGL bzw. DirectX manipulieren**
  Diese Methode würde es erfordern sich auf eine Version der Bibliotheken festzulegen und die Systemdateien (DLL's) zu ersetzen. Hier wäre

ein großer Nachteil gewesen, Inkonsistenzen mit Folge- oder Vorgängerversionen zu riskieren, darüber hinaus werden die Systemdateien durch einen Sicherheitsmechanismus des Windows-Betriebssystems geschützt, der nicht einfach auszuhebeln ist.

- **Funktionen des nVidia-Treibers nutzen**
  Der Graphiktreiber der Firma nVidia verfügt über eine API, mit der man begrenzt in die Funktion des Treibers eingreifen kann. Diese Funktionalität ist aber nicht umfangreich genug und zudem zu schlecht dokumentiert, um sie nutzen zu können.

- **Eine eigene Bibliothek schreiben, die von Programmierern benutzt werden kann**
  Diese Methode würde es zwar ermöglichen eine maximale Performance zu erreichen, jedoch wäre diese Möglichkeit auf Programme begrenzt, deren Programmierer diese Bibliothek in ihr Programm einbauen. Zudem wäre eine weitere Anpassung des jeweiligen Progammes nötig.

- **API-Hook**
  Es gibt begrenzte Möglichkeiten, API-Aufrufe eines Programmes oder einer DLL umzuleiten, diese gehen aber entweder mit niedriger Performance oder einem Eingriff in die Binärdateien einher, die zu unerwünschten Systemverhalten führen können. So kann z.B. in einer ausführbaren Datei die Tabelle, in der die dynamischen Link-Adressen der DLL-Funktioen stehen, manipuliert werden. Diese Lösung bietet jedoch viel Spielraum für Fehler, da falsche Adressen in dieser Tabelle zu nicht vorhersehbarem Verhalten führen können.

- **Proxy DLL**
  Mit dieser Methode kann eine Anwendung an eine andere DLL umgeleitet werden, die die gleichen Funktionen exportiert, wie die originale DLL. Der Unterschied zum API-HOOK liegt hier darin, dass nicht einzelne Funktionsaufrufe umgeleitet werden oder Dateien manipuliert werden, sondern der komplette Suchpfad nach einer DLL geändert wird. Dies hat den Vorteil, die nicht veränderten Funktionen über Forwarding an die Original-DLL weiterleiten zu können und nur die geänderten umsetzen zu müssen. Zudem können sehr viele, wenn auch nicht alle, Programme dadurch beeinflusst werden. Die Programmdateien werden im Übrigen nicht angetastet, was der Sicherheit zu gute kommt.

9.3 DLL-Umleitung (Proxy-DLL)

Die hier gewählte Methode leitet die Aufrufe an die Original-DLL um, über eine neue DLL, die die Aufrufe entsprechend der eigenen Anfordrungen verändert. Das Windows-Betriebssystem stellt dem Benutzer die beiden Möglichkeiten des „dot local“ - Files und des „manifest“ - Files zur Verfügung, um den Suchpfad einer Anwendung zu seinen DLL's zu manipulieren. Im Folgenden sollen beide Methoden kurz vorgestellt werden, wenn auch nur eine (Manifest-Files) hier genutzt wird.

9.3.1 dot local - Files

Es existiert eine denkbar einfache Methode, ein Programm anzuweisen, eine DLL zuerst im eigenen Verzeichnis, anstatt sofort im Windows Systemverzeichnis zu suchen. Man erstellt im Anwendungsverzeichnis eine Datei mit dem gleichen Namen, wie die Anwendung mit dem Suffix „.local“. Die Datei kann leer sein, da ihre bloße Anwesenheit die Anwendung schon anweist, den Suchpfad zu ändern.
Laut Microsoft Developer Network ist dieses Feature dazu gedacht, Anwendungen, die von einer bestimmten Bibliotheksversion abhängen und mit neueren Versionen nicht mehr zusammenarbeiten, die Möglichkeit zu geben, mit einer lokalen Kopie der alten Version weiterzuarbeiten.

9.3.2 Manifest - Files

Für den gleichen Zweck wurden in neueren Windows Versionen die Manifest-Files eingeführt, die diese Aufgabe dort übernehmen. Anders als die „dot local“ Files sind diese Dateien nicht leer, sondern bestehen aus XML-Code, welcher die genaue namentliche Angabe der DLL's, welche umgeleitet werden sollen enthält und welcher auch über die DLL-Umleitung hinausgehende Änderungen erlaubt, etwa Angaben zu Sicherheitseinstellungen, z.B. der Betriebssytemversion, auf der die Anwendung ausschließlich lauffähig sein soll.
Im Zusammenhang mit der PG498 wird dieser Mechanismus allerdings genutzt, um eine Proxy-DLL anzusprechen, die den für die Laserstrahldarstellung nötigen Code enthält und für die übrigen Routinen die Original-Bibliothek anspricht. (vgl. Abbildung 76)

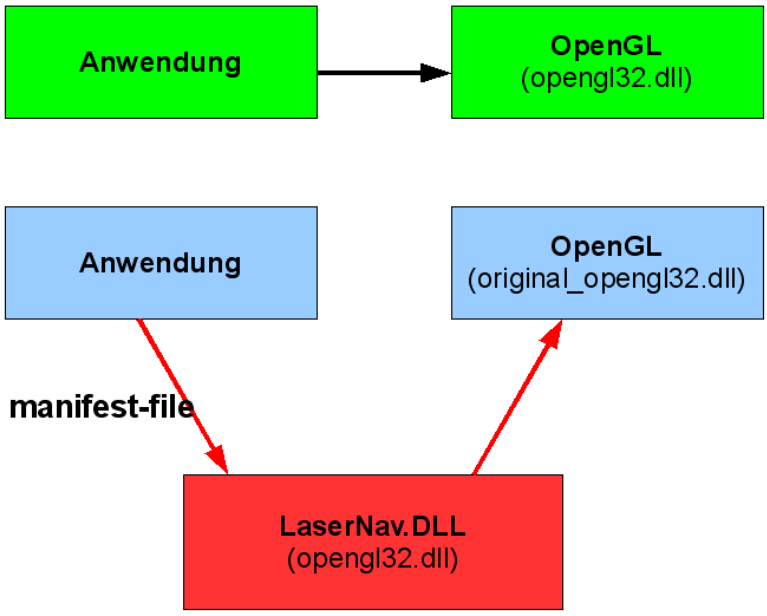


Abbildung 76: Proxy-DLL Prinzip

9.4 Eingriff in OpenGL und Berechnung des Laserstrahls

Die Proxy-DLL für OpenGL basiert auf der opengl32.dll, welche sich standardmäßig im „Windows/system32“ Verzeichnis befindet. Es wurde die glEnd - Funktion so manipuliert, das nach der regulären Beendigung des Zeichnens zusätzlich der Laserstrahl durch eine Linie im 3D-Raum verlängert wird.
Als Ausgangspunkt für die Berechnung der 3D-Koordinaten dieser Linie stehen die Koordinaten des Laserpunktes relativ zum Bildschirm, sowie der Richtungsvektor in die Tiefe der 3D-Szene zur Verfügung (aus der ServerApp-Anwendung). Berechnet werden, sollen die beiden Punkte im dreidimensionalen Raum der Anwendung P1 und P2 (vgl. Abbildung 78), zwischen denen der Laserstrahl eingezeichnet werden soll. Die Berechnung geht in folgenden Schritten vor sich:

1. Abfragen der Modelview- und Projection-Matrizen
   ```
   glGetFloatv(GL_PROJECTION_MATRIX, proj);
   glGetFloatv(GL_MODELVIEW_MATRIX, modl);
   ```

2. Kombination der Matrizen, um die Gesamtmatrix zu erhalten

$$clip = proj \cdot modl \tag{110}$$

3. Berechnen der Ebenengleichungen, die die Seitenkanten der Sichtpyramide beschreiben

```
m_Frustum[0][0] = clip[ 3] - clip[ 0];
m_Frustum[0][1] = clip[ 7] - clip[ 4];
m_Frustum[0][2] = clip[11] - clip[ 8];
m_Frustum[0][3] = clip[15] - clip[12];
...
m_Frustum[5][0] = clip[ 3] + clip[ 2];
m_Frustum[5][1] = clip[ 7] + clip[ 6];
m_Frustum[5][2] = clip[11] + clip[10];
m_Frustum[5][3] = clip[15] + clip[14];
```

   und anschließende Normalisierung.

4. Berechnen der Eckpunkte der Sichtpyramide aus den Ebenengleichungen

   Hier wird aus jeweils drei Ebenengleichungen der Schnittpunkt dieser Ebenen bestimmt. Für Matrix 111, in der die Parameter der drei Ebenengleichungen in den Zeilen stehen und die drei Koordinaten des Schnittpunktes x, y und z gilt Gleichung 112.

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix} \tag{111}$$

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{112}$$

   Formt man dies nach den Koordinaten um, so erhält man Formeln 113 bis 115, die benutzt werden, um aus den jeweils drei Ebenengleichungen, die Koordinaten des Schnittpunktes x, y, z zu bestimmen.

$$x = -\frac{b \cdot (g \cdot l - h \cdot k) + c \cdot (h \cdot j - f \cdot l) + d \cdot (f \cdot k - g \cdot j)}{a \cdot (f \cdot k - g \cdot j) + b \cdot (g \cdot i - e \cdot k) + c \cdot (e \cdot j - f \cdot i)} \tag{113}$$

$$y = \frac{a \cdot (g \cdot l - h \cdot k) + c \cdot (h \cdot i - e \cdot l) + d \cdot (e \cdot k - g \cdot i)}{a \cdot (f \cdot k - g \cdot j) + b \cdot (g \cdot i - e \cdot k) + c \cdot (e \cdot j - f \cdot i)} \tag{114}$$

$$z = -\frac{a \cdot (f \cdot l - h \cdot j) + b \cdot (h \cdot i - e \cdot l) + d \cdot (e \cdot j - f \cdot i)}{a \cdot (f \cdot k - g \cdot j) + b \cdot (g \cdot i - e \cdot k) + c \cdot (e \cdot j - f \cdot i)} \tag{115}$$

   Aus diesen Berechnungen findet man die Koordinaten der Punkte A bis H (vgl. Abbildung 77)

5. Berechnen von Hilfsvektoren aus den Eckpunkten (vgl. Abbildung 78)

```
Vb[0]=C[0]-A[0];
Vb[1]=C[1]-A[1];
Vb[2]=C[2]-A[2];

Vh[0]=B[0]-A[0];
Vh[1]=B[1]-A[1];
Vh[2]=B[2]-A[2];

Vt[0]=D[0]-A[0];
Vt[1]=D[1]-A[1];
Z=Vt[2]=D[2]-A[2];

Vp[0]=A[0];
Vp[1]=A[1];
Vp[2]=A[2];
```

6. Berechnen der Endpunkte P1 und P2 des Laserstrahls (vgl. Abbildung 78)

```
P1[0]=Vp[0]+x * Vb[0];
P1[1]=Vp[1]+y * Vh[1];
P1[2]=Vp[2];

P2[0]=P1[0]+Z*LaserVektor[0];
P2[1]=P1[1]+Z*LaserVektor[1];
P2[2]=P1[2]+Z*LaserVektor[2];
```

9.5 Viewportkorrektur

Die bisherige Berechnung ging von der Anzeige als Vollbild aus, bei Fenstern aber muss für den Viewport korrigiert werden. Zu diesem Zweck werden die Bezeichnungen aus Abbildung 79 eingeführt.

Der erste Schritt besteht darin, die Werte x und y aus dem Code für die Berechnung der Endpunkte durch Werte zu ersetzen, die nicht mehr relativ zum gesamten Bildschirm sind, sondern relativ zum Fenster (vgl. Abbildung 79).

Der zweite Schritt betrifft perspektivische Effekte. Wird ein Strahl gerade

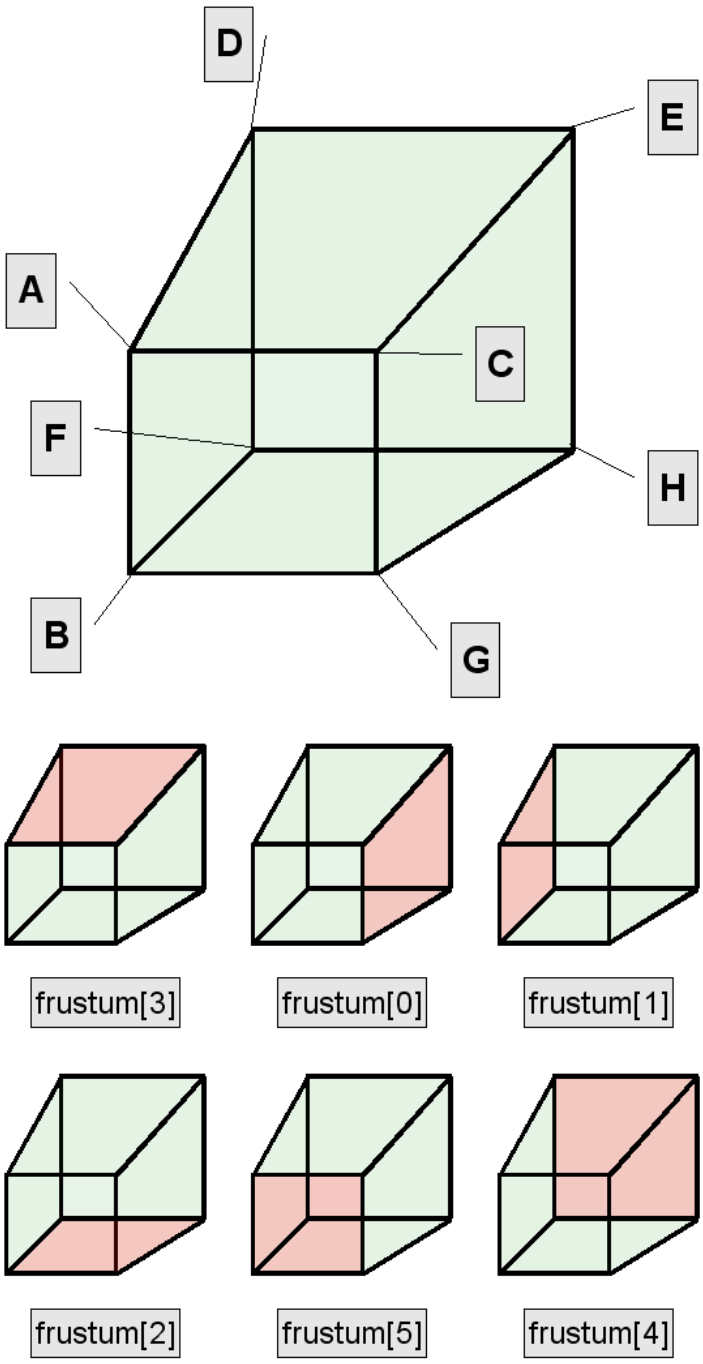


Abbildung 77: Bezeichnungen der Flächen und Punkte des Frustums

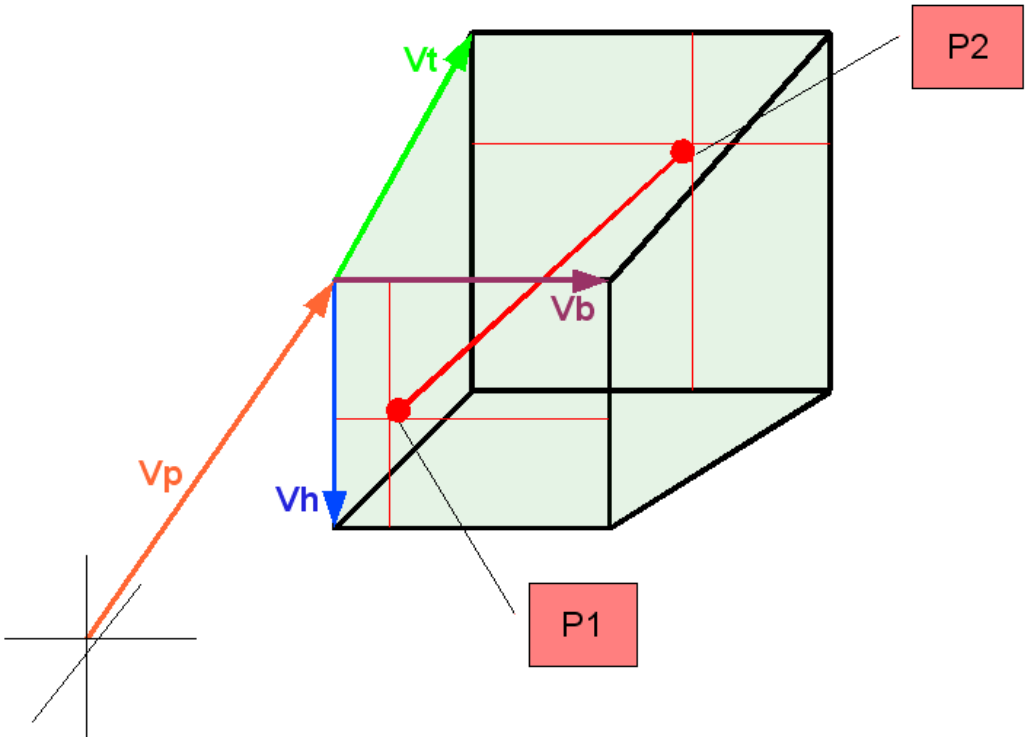


Abbildung 78: Endpunkte

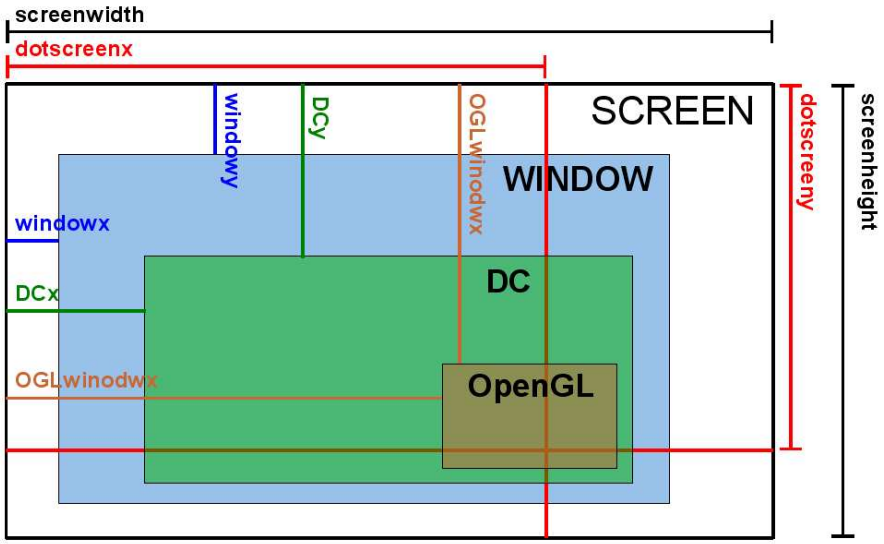


Abbildung 79: Bezeichnungen der Viewportkorrektur

in die Bildebene projeziert, so wird der Strahl dennoch als Linie erscheinen, da sich die Sichtpyramide in die Tiefe des Raumes erweitert. Da dieser Effekt unerwünscht ist, muss er korrigiert werden. Dazu werden die x- und y-Koordinaten von Punkt P2 in Relation zur Breite der vorderen Clip-Ebene gesetzt und dieses Verhältnis auf die hintere Clip-Ebene übertagen, um die entsprechenden x- und y-Koordinaten auf der hinteren Clip-Ebene zu erhalten.

```
Verhältnis-X = (P2.x-A.x) / (C.x-A.x);
Verhältnis-Y = (P2.y-A.y) / (B.y-A.y);

P2.x =D.x + Verhältnis-X * (E.x-D.x);
P2.y =D.y + Verhältnis-Y * (F.y-D.y);
```

9.6 Portierung auf Direct3D 9

Nicht alle Progamme die 3D-Grafik bieten benutzen OpenGL, mehr und mehr verbreitet sich auch die ursprünglich nur für Spiele entworfene Bibliothek DirectX mit ihrer Komponente Direct3D (D3D). Bei der Übertragung der API-Hook Lösung auf Direct3D muss jedoch der grundlegende Unterschied zwischen diesen beiden Bibliotheken beachtet werden: OpenGL ist eine Sammlung von Funktionen und die gesammte API basiert auf einem Zustandsautomaten. DirectX dagegen ist objektorientiert, was bedeutet dass hier mehrere Objekte der originalen Schnittstelle überschrieben werden müssen.
Diese Eigenart macht die Lösung entsprechend aufwändiger. Folgende DirectX-eigene Objekte wurden überladen:

- IDirect3D9
- IDirect3DDevice9

In letzterem findet sich die Methode *EndScene*, welche sich analog zu *glEnd* verhält.
Bei der Berechnung der beiden Endpunkte wurden ebenso, wie bei der Lösung für OpenGL die Darstellungsmatritzen abgefragt. Hier ist jedoch wieder ein Unterschied zwischen den beiden API's zu beachten: OpenGL benutzt eine Modelview Matrix und eine Projection-Matrix, Direct3D dagegen trennt die Modelview-Matrix weiter in Model-Matrix und View-Matrix. So dass hier weitere Abfragen und Matritzenmultiplikationen nötig sind.
Die restliche Berechnung bleibt jedoch gleich und Unterschiede finden sich

nur noch in den verwendeten API-spezifischen Funktionen und den verwendeten Datenstrukturen, etwa Strukturen zur Speicherung von Matritzen u.ä..

10 Gesamtsystem

10.1 Grundkonzept

Das Gesamtsystem ist schon aufgrund der Systemanforderungen auf zwei Programme aufgeteilt(Abbildung 87): die Client-Applikation (ClientApp), die auf dem Rechner läuft, dessen Dektop mit dem Laserpointer gesteuert werden soll und der Server-Applikation (ServerApp), die auf dem Hörsaalrechner läuft. Die Aufgabe des Clienten ist es, eine GUI für die Steuerung des Servers und dessen Engine bereitzustellen, sowie eine Benutzerverwaltung umzusetzen und schließlich die empfangenen Mauskoordinaten an den Maustreiber weiterzuleiten. Der Server enthält die eigentliche Engine zur Erkennung des Laserpointers und sendet die gefundenen Koordinaten an den Clienten. Zusätzlich gehört zum Gesamtsystem noch der Maustreiber, der auf dem Clientrechner installiert sein muss (vgl. Abschnitt *Treiber*), um die Mauskoordinaten setzen zu können, sowie die Pie-Menu Komponente, welche bei einen Rechtsklick Zugriff auf verschiedene Systemfunktionen bietet, z.B. den normalen Rechtsklick, den Aufruf einer Tastenkombination oder den Aufruf einer Anwendung. Durch die Hörsaalaustattung vorgegeben, sind die Hardwarekomponeten der Kameras und des Dallmeier-Systems, welches über eine Ethernetverbindung mit dem Server-Rechner verbunden ist.

10.1.1 Client

Primär ist der Client lediglich eine GUI für die Steuerung des Servers - umgesetzt durch die MFC-Bibliothek -, darüberhinaus bietet dieser Systemteil eine Verwaltung für Netzwerkeinstellungen und Benutzerdaten. Außerdem werden vom Server empfangene Daten verarbeitet: XY-Koordinaten werden an den Maustreiber weitergeleitet und empfangene Logfiles, die Auskunft über den Verlauf der Berechungen im Server geben, für die Einsicht und Bearbeitung angeboten.

10.1.2 Pie Menü

Zur einfacheren Interaktion des Laserpointers mit der GUI sollte das normale Kontextmenü, das normalerweise bei einem Rechtsklick erscheint durch ein eigenes anpassbares Menü ersetzt werden. Bei der Form dieses alternativen Menüs wurde die Form eine Ringes ausgewählt, der wie ein Kuchen in mehrere Stücke unterteilt ist, welches den anklickbaren Knöpfen entspricht. Daher wird diese Menüform auch Pie Menü genannt. Dieses Pie Menü sollte um den Laserpointer herum erscheinen, so dass jeder Knopf auf dem Menü vom

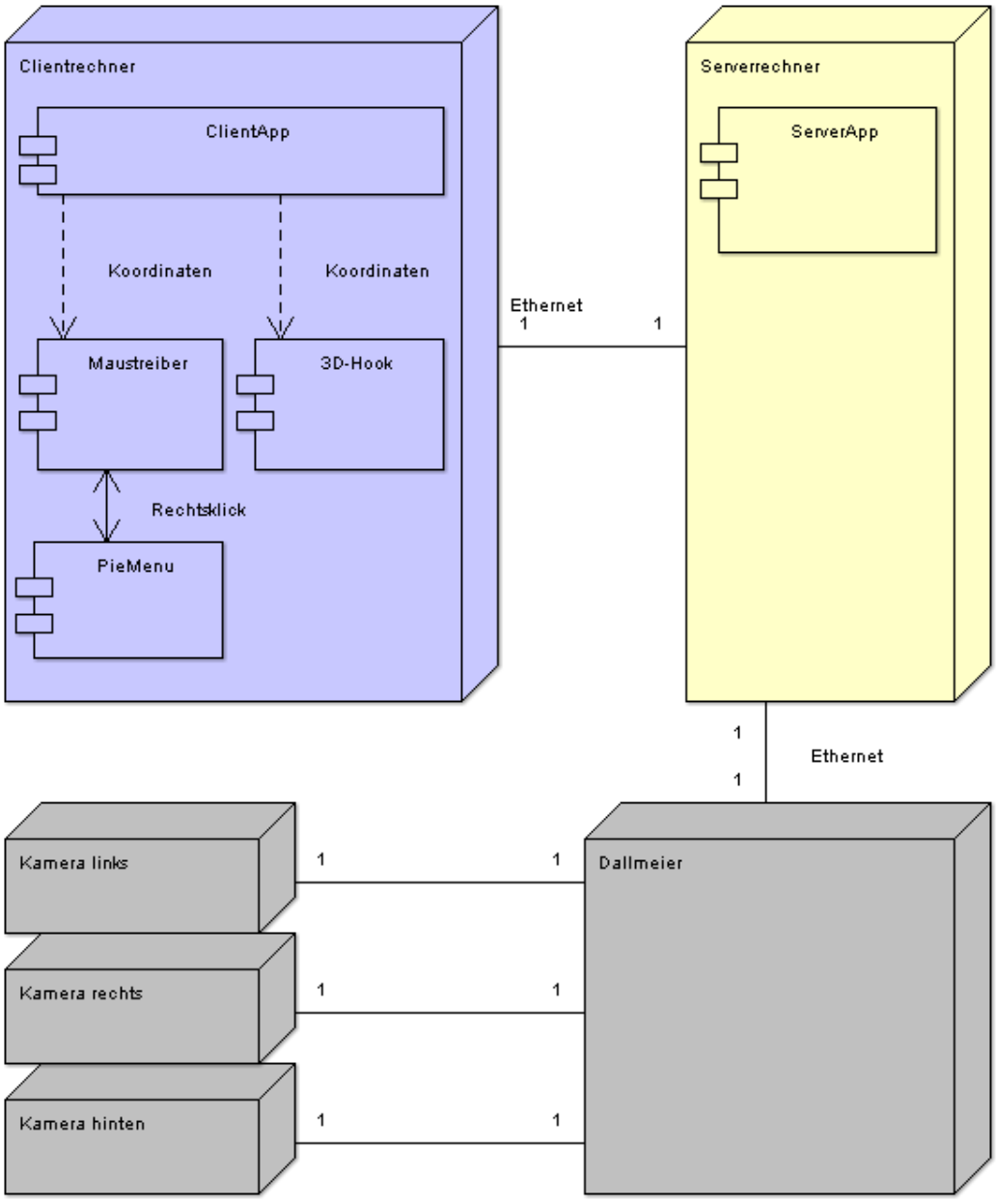


Abbildung 80: Das Gesamtsystem

Laserpointer gleich weit entfernt ist, was das Arbeiten erleichtern soll. Weiterhin sollen die Aktionen der Knöpfe frei konfigurierbar sein, damit nur die relevanten vorhanden sind und die unnötigen nicht stören.

Aufbau

Zur Realisierung des Pie Menüs wurde als Grundlage ein TransparentDialog gewählt. Dieser erbt von CDialog. Der TransparentDialog hat als eigene Farbe Transparenz ausgewählt und lädt zusätzlich ein Bitmap. Daher ist nur dieses Bitmap zu sehen, welches dann das eigentliche Pie Menü darstellt. Um zu wissen, ob gerade ein Rechtsklick ausgeführt worden ist, sendet das Pie Menü, während es im Hintergrund läuft, ständig Anfragen in Form von IOCTLs an den Maustreiber der PG 498. Dieser verhindert das Auftauchen des normalen Kontextmenüs, informiert das Pie Menü jedoch darüber, dass ein Rechtsklick stattgefunden hat, so dass es in den Vordergrund gebracht werden kann.

Die wichtigen Methoden hierbei sind:
void OnPaint(), welche das Bitmap darstellt, und
void OnSize(), welche eine Region erstellt, positionieren und deren Größe verändert

Lokalisierung der Mausposition

Damit das Pie Menü auf Mausklicks reagieren und die entsprechenden Befehle ausführen kann, muss es wissen, ob und an welcher Stelle auf das Pie Menü geklickt worden ist. Diese Funktionalität wird in OnMouseMove() realisiert. Hier wird ein Mittelpunkt und ein Radius angegeben. Wenn die Maus in den Kreisbereich hereinkommt, wird das PieMenu informiert, in welchem Kreis der Mauszeiger ist. Wenn nun die Maus gedrückt wird, dann bekommt OnLButtonDown() die Nachricht, und weiß, welche Funktion ausgeführt werden soll.

Konfiguration des Pie Menüs

Durch das Control Panel des Pie Menüs kann man die Funktion und das Icon jedes Buttons verändern. Das Control Panel ist ein Kind-Fenster des Pie Menüs. Durch ComboBoxen kann die Funktionalität jedes Knopfes eingestellt werden. Dazu werden Messages an das Eltern-Fenster geschickt, wenn

die Auswahl einer ComboBox geändert wird. Dazu wird die Methode OnCbnSelchangeComboboxex*() benutzt. Das Eltern-Fenster holt die Messages ein, und bestätigt.

Anforderungen

Für den Kern des gesamten Systems wurde eine Datenstruktur benötigt, welche die aufgenommenen Bilder in jedem Stadium ihrer Verarbeitung zur Verfügung stellt. Diese Struktur ist im Prinzip ein dreidimensionales Feld, definiert durch Breite, Höhe und der Anzahl zu haltender Werte pro Pixel. Es ist essentiell, dass die Struktur sowohl effizient nutzbar ist als auch ein geringes Maß an Zugriffskomfort bietet. Effizient heißt hierbei, dass die Struktur konsekutiv durchlaufen werden kann, wobei es möglich sein muss in konstanter Zeit von einem Datenelement zum nächsten zu gelangen. Außerdem muss ein akquiriertes Kamerabild schnell in die Struktur aufgenommen werden können. Zugriffskomfort heisst hierbei, dass man einzelne Datenelemente auch direkt lesen oder setzen kann.

Lösungsansätze

Zuerst wurde versucht das gesamte Feld mit new[] zu reservieren. Dies führte nicht zum gewünschten Ergebnis, da es nicht möglich war ein dreidimensionales Feld entsprechender Größe zu allozieren. Schließlich wurde versucht für jede Zeile einzeln Speicher zu reservieren, was aber den effizienten Durchlauf verhindert hätte.

Lösung

10.1.3 Die Datenstruktur

Letztendlich wurde die Struktur wie in Abbildung 81 aufgebaut, wobei der Übersichtlichkeit halber nicht alle Zeiger im zweiten Block dargestellt wurden. Dazu wurden drei zusammenhängende Speicherblöcke mit malloc reserviert. Im ersten wurden Zeiger auf die einzelnen Spalten gehalten. Im zweiten wurden Zeiger auf die einzelnen Pixel gespeichert. Im dritten wurden dann die eigentlichen Daten platziert. Dabei ist es wichtig, dass die Pixel wie dargestellt referenziert werden, um den zeilenweisen Zugriff mittels [x][y] zu ermöglichen. Durch einfaches Inkrementieren eines Zeigers auf den dritten Block können die Daten wie verlangt effizient durchlaufen werden. Dies ist um ein Vielfaches schneller als die Zeigerarithmetik zur Berechnung der Position aus den absoluten Indizes. Die grau schattierten Felder in Abbildung

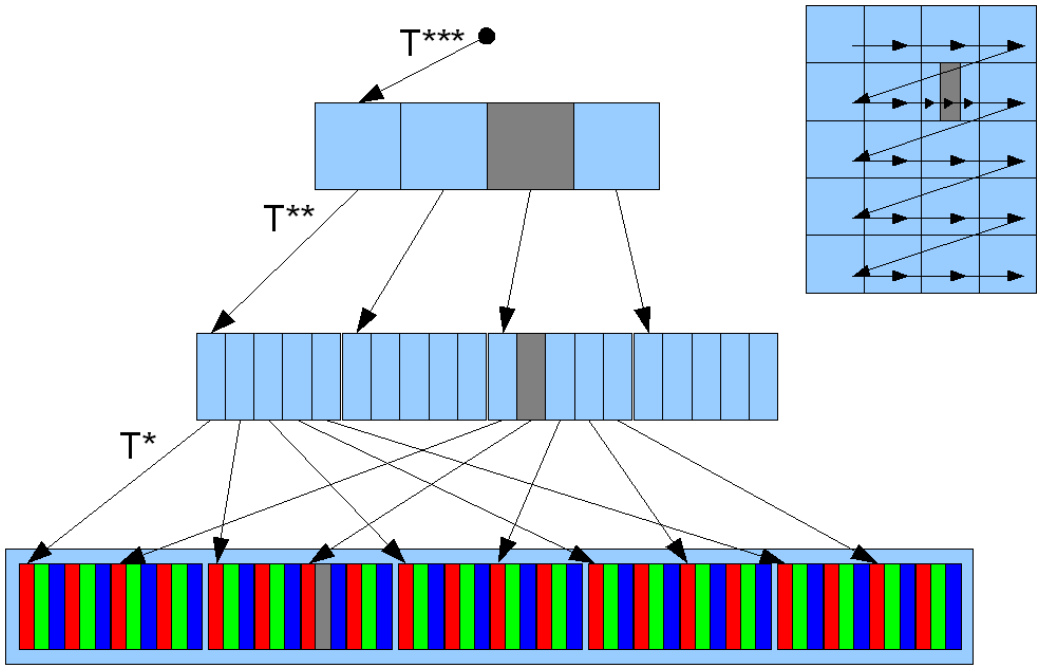


Abbildung 81: Die Datenstruktur

81 zeigen exemplarisch, wie man das Datenelement [3][2][2] mittels impliziter Zeigerarithmetik (zentral) und mittels Durchlauf (oben rechts) erreichen kann.

10.1.4 Server

Dieser Systemteil enthält die eigentliche Funktionalität. Zentrales Element ist hier der Controller, der alle Vorgänge steuert und kontrolliert (vgl. Abbildung 93). Jede Verarbeitungsstufe, vom Framegrabbing bis zum Tracking, soll als eigenes Subsystem in einem Thread laufen und der Controller als ein weiterer Thread freie Resourcen zuordnen. Die Daten werden in einer Pipeline nach jeder Verarbeitungsstufe an die nächste Stufe weitergeleitet. So werden im Framegrabbing die Bilddaten beschafft, im Dewarping diese Daten von eventuellen Verzerrungen befreit, im Preprocessing für die weitere Pipeline aufbereitet, in der Segmentierung die Koordinaten des Laserpointers erkannt und schließlich im Tracking die Position der Person im Raum bestimmt. Die Queues haben die in Abbildung 82 gezeigte Struktur. Sie halten eine Menge von Bildern für ein Subsystem in Warteposition. Für eine Queue ist jeweils eine maximale Menge von Bildern vorgegeben, damit bei Verzögerungen oder Fehlern die Queues nicht unbegrenzt wachsen.

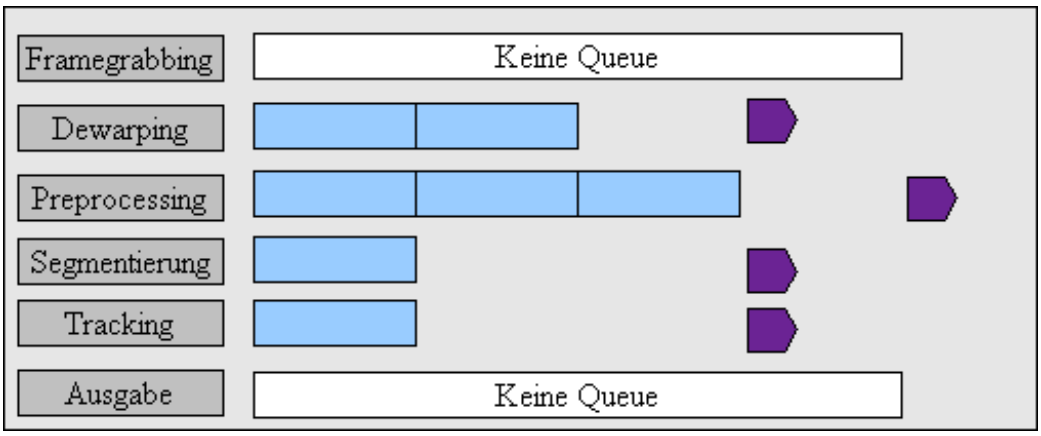


Abbildung 82: Queues des Servers

Subsystem Um dieses Threadkonzept komfortabel umsetzen zu können, wurde die externe Klasse CThread, die grundlegende Threadfunktionalität kapselt, um einige für dieses Projekt wichtige Funktionen erweitert und diese in der Klasse SubSystem zusammengefasst. Diese Klasse bietet dem Controller eine einheitliche Schnittstelle für die Verwaltung der jeweiligen Komponente, etwa die Möglichkeit die Bearbeitung eines Bildes zu starten oder diese zu unterbrechen, bzw. ganz zu stoppen. Auf der anderen Seite stellt diese Klasse aber auch den Untergruppen eine einfache Schnittstelle zur Verfügung ihre Subssysteme zu programmieren. Es muss lediglich das Subsystem abgeleitet und die Methode SingleStep umgesetzt werden. Die gesamte Verwaltung der Threadfunktionalität übernimmt die Subsystemklasse.

Controller Der Controller steuert die gesamte Bearbeitung eines Bildes über eine Reihe von Subsystemobjekten:

1. Framegrabbing: Diese Komponente lädt die Kamerabilder in das System und stellt sie in einem für die spätere Verarbeitung geeignetem Format zur Verfügung.

2. Dewarping: Hier werden Fehler der Optik, etwa Fischaugenverzerrungen oder ähnliches, korrigiert.

3. Preprocessing: In diesem Subsystem wird die Bildqualität für die weitere Verarbeitung verbessert.

4. Segmentierung: Bei dieser Komponente handelt es sich um die eigentliche Erkennung des Laserpunktes auf der Leinwand.

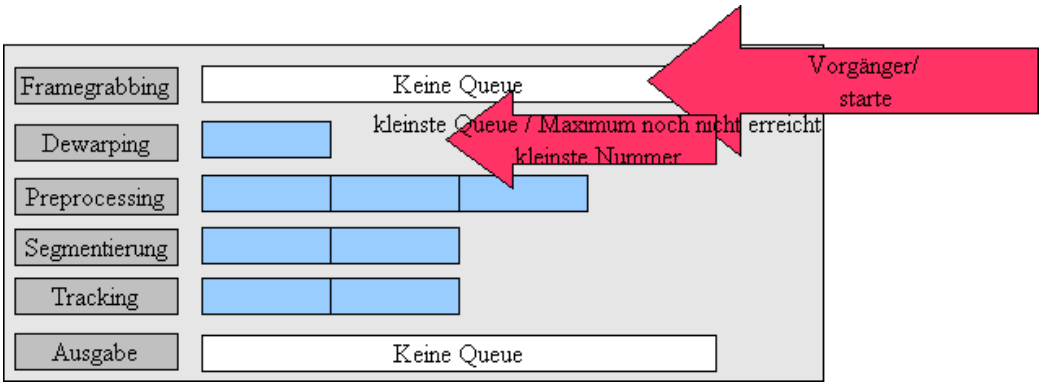


Abbildung 83: Einfaches Beispiel für die zweite Regel zur Bearbeitung der Daten

5. Tracking: Diese Komponente dient der Erkennung der Position der Person mit dem Laserpointer im Hörsaal, um einen Richtungsvektor zu gewinnen, mit dem der Laserstrahl in die 3D-Szene verlängert werden kann.

Der Controller übergibt jedem Subsystem ein Bild für die Bearbeitung und empfängt es danach wieder. Die Bilder werden in einem Objekt der Klasse DataContainer gehalten und verwaltet. Ist ein Subsystem mit der Bearbeitung fertig, so wird das Bild in die nächste Queue eingestellt. Das Scheduling der Subsysteme, erfolgt gemäß folgender Regeln, die innerhalb des Controller-Threads immer wieder abgefragt werden:

1. Lösche alte Bilder

2. Starte Bearbeitung des Threadvorgängers des Threads mit der kleinsten Queue, der sein Maximum nicht schon erreicht hat. (Bei Gleichheit zählt die kleinste Nummer!) Damit wird bei leeren Queues es direkt mit dem Framegrabbing begonnen. Als Beispiel betrachte man Abbildung 83: Die Dewarpingqueue ist momentan die kleinste Queue und somit wird das Vorgänger-Subsystem das Framegrabbing-Subsystem gestartet. Abbildung 84 zeigt eine etwas schwierigere Situation: hier haben zwei Queues die gleiche und gleichzeitig auch die kleinste Größe (Segmentierung und Tracking). Hier wird mit dem im Stack zuerst stehenden Subsystem fortgefahren, also mit dem Segmentierungs-Subsystem.

3. Starte die Bearbeitung des Threads mit der größten Queue, dessen Nachfolger sein Maximum noch nicht erreicht hat. (Bei Gleichheit

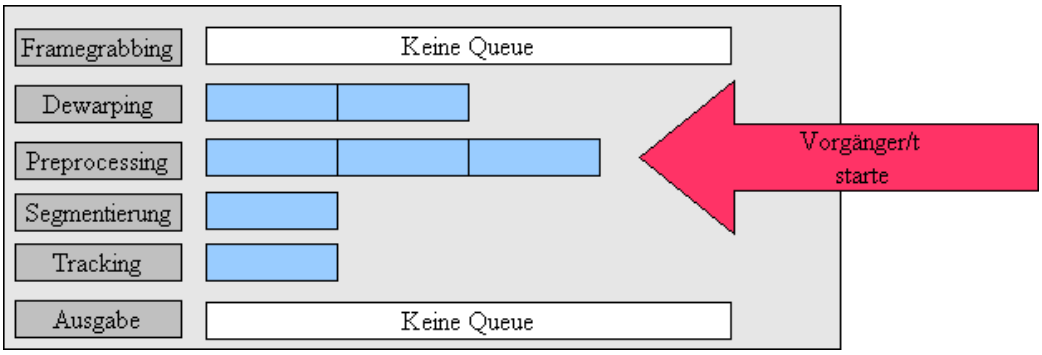


Abbildung 84: Schwierigeres Beispiel zur zweiten Regel zur Bearbeitung der Daten

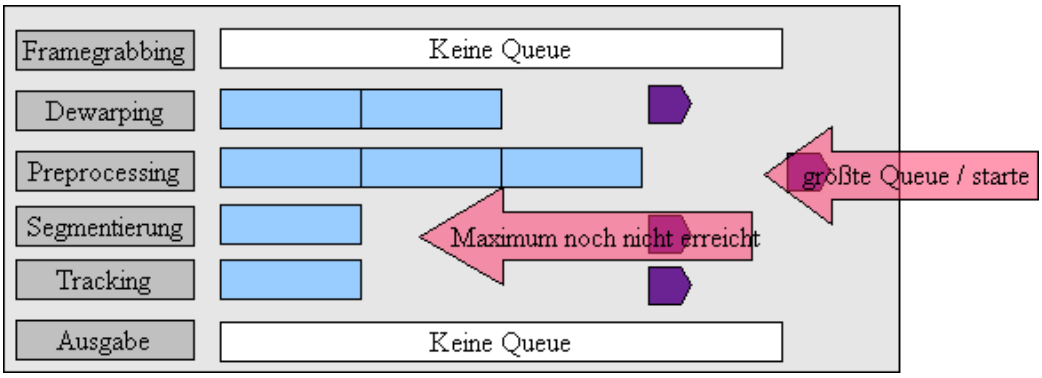


Abbildung 85: Beispiel für die dritte Regel zur Bearbeitung der Daten

zählt die größte Nummer!) Die Queue-Size der (Ausgabe ist immer "Maximum noch nicht erreicht". Man betrachte als Beispiel für diese Regel, die Situation in Abbildung 85: Die Preprocessing-Queue ist die momentan größte Queue und ihre Nachfolger-Queue (Segmentierung) hat ihr Maximum noch nicht erreicht, somit wird das Preprocessing-Subsystem gestartet.

10.2 Entwicklung

Am Anfang stand die Grundkonzeption mittels eines UML-Modellierungstools (ARGO-UML vgl. Abschnitt *Tools*). Nachdem diese erste Phase abgeschlossen war, wurde mit der Programmierung der einzelnen Klassen begonnen. Die Subsystem- und Controllerklassen, sowie die Klassen für die Datenverwaltung wurden zuerst und parallel in ständiger Absprache aufgebaut, da diese Komponenten für die späteren Schritte grundlegend

sind. Im nächsten Schritt konnten mit Hilfe dieser Komponenten und der in ihnen definierten Schnittstellen die einzelnen Untergruppen ihre Subsysteme unabhängig voneinander schreiben. Parallel zur Entwicklung der Subsysteme wurde nun mit der Programmierung des Servers begonnen, die vorrangig darin bestand den Controller mit seiner Regelverwaltung umzusetzen. Beim Test der Funktionalität konnte nicht auf die Subsysteme zurückgegriffen werden, da diese sich noch in der Entwicklung befanden. Als Ersatz wurden Dummy-Klassen geschaffen, die die Schnittstellen und das Verhalten der fertigen Komponenten simulierten. Nachdem dieses Gerüst stand und die Bearbeitung der Subsysteme noch nicht abgeschlossen war, wurde die GUI für den Client entwickelt. Diese Komponenten wurde mittels der *Microsoft Foundation Classes* (MFC) entworfen und umgesetzt. Mit den fertigen Framegrabbing- und Segmentierungs-Subsystemen wurden erste Testläufe möglich. Die noch unfertigen Subsysteme wurden durch Dummy-Komponenten ersetzt, die lediglich als Platzhalter zu verstehen sind. Zu diesem Zeitpunkt trat das Projekt erwartungsgemäß in eine Debuggingphase ein, da nun erstmals mit echten Daten - nicht mehr mit Platzhaltern - gearbeitet wurde und sich einige Speicherverwaltungsfehler zeigten. Zu diesem Zeitpunkt wurde die Logfile-Komponete immer wieder den Anforderungen angepasst, um die Bearbeitungsschritte des Servers genau analysieren zu können. Als letzter Schritt wurden Server und Client durch die NetworkThread-Klasse und jeweils eine TCP- und UDP-Verbindung miteinander verbunden.

11 Validierung

Im Zuge der Validierung wird überprüft, ob die Entwicklungsergebnisse den im Pflichtenheft (siehe Anhang D) formulierten Anforderungen entsprechen. Um festzustellen, ob ein Produkt eine bestimmte festgelegte Aufgabe tatsächlich löst, ist eine umfangreiche Testphase notwendig.
Oft werden das Debuggen und das Testen von Software fälschlicherweise als ein und dieselbe Tätigkeit angesehen. Es handelt sich allerdings tatsächlich um zwei verschiedene Tätigkeiten. Unter dem Testen von Software versteht man die stichprobenartige Ausführung eines Testobjektes, um die Qualität des Softwaresystems zu erhöhen und Fehler zu finden. Dabei sollen Fehlerwirkungen, also das Nichterfüllen bestimmter Anforderungen gezielt und systematisch aufgedeckt werden. Beim Debuggen handelt es sich hingegen um die Lokalisierung und die Behebung von Fehlerzuständen seitens des Software-Entwicklers.

Der Testprozess lässt sich in fünf Phasen einteilen:

1. Testplanung: Eine Teststrategie wird festgelegt und die benötigten Ressourcen (prüfender Projektteilnehmer, Werkzeuge, Geräte) werden eingeplant.

2. Testspezifikation: Eine Bestimmung von Testfällen aufgrund aller Dokumente, aus denen sich Anforderungen ableiten lassen, wird durchgeführt. Testdaten beinhalten nicht nur Eingabedaten, sondern auch Vorbedingungen, Nachbedingungen, Randbedingungen und das vorher festgelegte Soll-Verhalten.

3. Testdurchführung: Die Tests werden getreu der Testspezifikation durch den Prüfer an dem Testobjekt ausgeführt.

4. Testprotokollierung: Das Testprotokoll enthält Informationen über die Objekte, den Zeitpunkt, die Intensität, die Teilnehmer und die Ergebnisse der Prüfung.

5. Auswertung und Bewertung des Testers: Unterschiede zwischen Soll- und Ist-Verhalten können eine Fehlerwirkung aufzeigen. Allerdings gilt es zu bedenken, dass sowohl das Soll-Ergebnis als auch die Testumgebung und/oder die Spezifikation Fehler enthalten haben können.

Viele Testfälle lassen sich zu Testszenarios aneinanderreihen, in dem man übliche Abfolgen von Benutzereingaben nacheinander testet. Das Ergebnis

eines Testfalles wird somit als Ausgangssituation für einen nachfolgenden Testfall genutzt. Somit lassen sich viele Testfälle effektiv in einem Testlauf ausführen.

11.1 Reviews

Der Begriff Review steht für ein statisches Prüfverfahren für Dokumente, welches nicht maschinell durchgeführt wird. Alle Dokumente, die im Laufe des Entwicklungsprozesses erstellt werden, können Objekt eines Reviews werden, also z.B. Anforderungsbeschreibungen, Programmspezifikationen, das Benutzerhandbuch oder das Pflichtenheft. Die Überprüfung der Semantik dieser Dokumente ist in der Regel nur durch ein oder mehrere Personen möglich.
Ein Review besteht aus fünf Phasen:

1. Planung: Die Dokumente für das Review werden ausgewählt. Der Aufwand, der Termin und der Ort des Reviews wird bestimmt.

2. Einführung: Den am Review beteiligten Personen werden alle notwendigen Informationen, wie das zu untersuchende Dokument, das Pflichtenheft und andere Richtlinien zur Verfügung gestellt. Prüfkriterien in Form von Checklisten sind zudem hilfreich, um Abweichungen und Fehler zu erkennen.

3. Vorbereitung: Alle Teilnehmer bereiten sich individuell auf das Review vor. Die Gutachter notieren ihre Fragen, Kommentare und Mängel.

4. Reviewsitzung: Die Sitzung wird von einem Moderator geleitet und hat einen festen Zeitrahmen, der nicht überschritten werden soll. Nötigenfalls wird sie zu einem späteren Termin fortgesetzt. Es wird darauf geachtet Kritik nur am Dokument und nicht am Autor zu äußern. Dieser sollte nicht den Eindruck bekommen, sich verteidigen zu müssen. Allerdings sollte er darauf vorbereitet sein, die Ausführungen in seinem Dokument zu rechtfertigen. Stilfragen zum Dokument werden nicht erörtert, weiterhin sollen keine Problemlösungen entwickelt werden. Im Protokoll sollen sowohl gute Passagen als auch kritische, Haupt- und Nebenfehler vermerkt werden. Am Ende wird entschieden, ob das Objekt akzeptiert wird, oder ob es noch einmal zu einem späteren Zeitpunkt einem Review unterzogen werden soll. Das Protokoll wird von allen Anwesenden unterzeichnet.

5. Nachbereitung: Der Autor behebt die Mängel entsprechend des Reviewprotokolls. Falls das Objekt nicht akzeptiert wurde, wird eine neue Reviewsitzung angesetzt.

Sollte während des Reviews ein Mangel an Personal, Fachwissen oder Vorbereitung auftreten, ist die Sitzung abzubrechen und die Mängel zu beheben. Andernfalls ist man mittels des Reviews nicht in der Lage, die Qualität des zu prüfenden Objektes zu verbessern.

11.2 Statische Analyse

Mit Hilfe der statischen Analyse lassen sich fehlerhafte Stellen in Dokumenten erkennen, ohne das Prüfobjekt auszuführen. Zudem lassen sich anhand der statischen Analyse Messgrößen für die Qualität des Dokumentes ableiten. Da diese Überprüfung maschinell durchgeführt wird, muss das Prüfobjekt eine formale Struktur besitzen. Dabei handelt es sich oft um die formellen Beschreibungen der technischen Anforderungen, der Software-Architektur und vor allem des Programmcodes.
Die Werkzeuge ermöglichen es folgende Fehlerzustände zu erkennen:

- Verletzung der Syntax: Alle Compiler führen eine statische Analyse des Programmcodes durch, um zu überprüfen, ob die Syntax der Programmiersprache verletzt wurde.

- Konventions- und Standardabweichungen: Der Verstoß gegen Standards und Konventionen kann anhand von sprachspezifischen Analysatoren ermittelt werden. Die Konventionen sollten im Vorfeld des Projektes festgelegt werden.

- Datenflussanomalien: Die Datenflussanalyse verfolgt die Verwendung von Daten im Verlauf des Programmcodes und deckt Datenflussanomalien auf. Zu diesen Anomalien gehören unter anderem die referenzierende Verwendung einer Variable ohne vorherige Initialisierung oder die Nichtverwendung eines Variablenwertes. Diese Anomalien müssen nicht zwangsläufig zu Fehlerzuständen führen, sie sind allerdings ein Anzeichen für eine „unsaubere“ Implementierung.

- Kontrollflussanomalien: Kontrollflussgraphen stellen unverzweigte Sequenzen von Aweisungen als Knoten dar. Innerhalb dieser Sequenzen verändert sich der Programmablauf nicht. Aus „IF-THEN-ELSE“-Konstrukten o.ä. lassen sich Verzweigungen zwischen den Knoten erstellen. Kontrollflussanomalien stellen Unstimmigkeiten im Ablauf des Programms dar, wie z.B. nicht erreichbare Anweisungen und Sprünge in oder aus Schleifen heraus. Auch diese Anomalien müssen nicht zwingend zu Fehlerzuständen führen, sie zeugen aber von schlecht strukturierter Programmierung.

Die statische Analyse ist nicht in der Lage, alle Fehler zu erkennen. Es existieren Fehler, die erst zur Laufzeit auftreten können, z.B. wenn eine Division durchgeführt wird und dem Divisor zur Laufzeit eine Null zugewiesen wird. Allerdings stellt die statische Analyse eine effiziente Möglichkeit dar, im Vorfeld eines Reviews Unstimmigkeiten in Dokumenten aufzudecken.

11.3 Dynamisches Testen

Das dynamische Testen erfolgt zur Laufzeit des Prüfobjektes auf einem Testsystem. Es gilt nachzuweisen, dass das Programm die spezifizierten Anforderungen erfüllt. Aufgrund der großen Anzahl der Testfälle ist es notwendig systematisch möglichst viele Anforderungen zu überprüfen, um möglichst viele Fehlerwirkungen nachweisen zu können. Das Testobjekt muss in ablauffähiger Form vorliegen. Sollten Programmteile noch nicht fertig implementiert sein, so sind sie durch Platzhalter zu ersetzen, die ihr Eingabe/Ausgabe-Verhalten simulieren. Zudem ist es sinnvoll einen Testtreiber zu entwickeln, um das Testobjekt möglichst zeiteffizient mit Eingaben zu versorgen.

Blackbox-Verfahren

Bei dem Blackbox-Verfahren sind keine Informationen über die inneren Abläufe des Prüfobjektes notwendig. Die Testfälle konzentrieren sich auf die Überprüfung der funktionalen Spezifikation. Das Verhalten des Testobjektes wird von außen betrachtet. Der innere Zustand des Objektes wird nicht berücksichtigt. Die einzige Steuerung erfolgt durch die Eingaben. Das Blackbox-Verfahren eignet sich gut zur effizienten Funktionsüberprüfung großer Softwareprojekte. Befinden sich allerdings Fehler in der Spezifikation oder in den Anforderungen und wurden diese im Testobjekt umgesetzt, sind sie nicht mehr auffindbar, da sich das System ja wie erwartet verhält.

Whitebox-Verfahren

Das Whitebox-Verfahren analysiert die Abläufe im Innern des Testobjektes. In Ausnahmefällen ist es sogar möglich, in den Ablauf einzugreifen. Die Testfälle werden aus der Programmstruktur abgeleitet. Es handelt sich um ein strukturelles Testverfahren, da die Komponentenhierarchie, der Kontrollfluss und der Datenfluss des Testobjektes berücksichtigt werden. Das Whitebox-Verfahren ermöglicht eine gründliche Analyse der Vorgänge innerhalb eines Testobjekts. Es ist sogar in der Lage Fehler, die aus der Spezifikation übernommen wurden, aufzudecken. Allerdings eignet es sich nur für

die unteren Testebenen, da das Bestreben, ein komplettes System mit dem Whiteboxverfahren zu testen, einen zu großen Aufwand bedeuten würde.

11.4 Ergebnisse der Validierung

Ausführliche Tests sind für Softwareprojekte dieser Größe unvermeidlich, um die Qualitätsmerkmale Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Flexibilität und Übertragbarkeit zu gewährleisten. Je früher mit Testphasen im Entwicklungsprozess begonnen wird, desto schneller können Fehlerzustände erkannt und behoben werden, bevor sie in späteren Stadien der Softwareentwicklung, einen Mehraufwand für Korrekturen, Mängel an Funktionen oder sogar den Misserfolg des ganzen Projektes verursachen. Allerdings sind vollständige Tests in der Praxis nicht durchführbar. Durch die Vielzahl der kombinatorischen Möglichkeiten von Eingaben und Randbedingungen ergibt sich eine nahezu unbegrenzte Anzahl von Testfällen, die zu prüfen wären. Zur Zeit existiert kein umfangreiches, vollkommen fehlerfreies Softwaresystem und die Chancen, dass in naher Zukunft Werkzeuge für absolut lückenlose Tests entwickelt werden, sind gering. Auch das 3D-LaserNAV-Projekt ist da keine Ausnahme. Trotz der Tatsache, dass nicht alle Fehlerzustände erkannt werden können, waren alle Tests während des Entwicklungsprozesses sinnvoll. Sie haben die Projektteilnehmer vor unabsehbarem Mehraufwand im späteren Verlauf der Entwicklung bewahrt und ein hohes Maß an Softwarequalität für den Benutzer gewährleistet.

12 Fazit

12.1 Die Projektarbeit

Die Projektarbeit in den vergangenen zwei Semestern war eine gute Möglichkeit anhand praktischer Arbeiten, die während des Studiums erlangten theoretischen Kenntnisse erfolgreich umzusetzen. Trotz der häufig zeitintensiven Projektarbeit kam der Spaß an der Aufgabe nie zu kurz, sondern trug vielmehr intensiv zur Motivation bei. Dies lag zum einen an der sehr interessanten Aufgabenstellung und dem Entwurf einer Anwendung für die es eine reale Anforderung gab. Zum anderen an dem positiven Arbeitsklima innerhalb der Projektgruppe, das dazu beitrug, dass die Motivation meist gut war. Herausfordernd war die sehr hohe Einarbeitungszeit aufgrund der Komplexität des Themas. In vielen Bereichen mussten daher diverse Verfahren implementiert und evaluiert werden, bis sich ein Lösungsansatz als der geeignetste herausstellte. Da sich die Projektgruppe dazu entschlossen hatte mit dem Prototypenmodell zu arbeiten, wurden im ersten Semester für die einzelnen Module die entsprechenden Prototypen entworfen und implementiert. Für jedes Modul wurde eine Arbeitsgruppe gebildet, die sich in ihr jeweilig zugeordnetes Themengebiet einarbeitete.
Im zweiten Semester konzentrierte sich die Projektgruppe darauf, die Schnittstellenspezifikation sowie die graphische Benutzerschnittstelle zu entwerfen und die einzelnen Prototypen im finalen Gesamtsystem zusammenzuführen. Die Weiterentwicklung der Module verlief parallel zu diesem Entwicklungsschritt, bis zuletzt die Funktionalität des Programms getestet wurde. Abschließend bleibt festzuhalten, dass die Zielvorgaben aus dem Pflichtenheft realisiert wurden.

12.2 Danksagung

Unser Dank gilt dem Lehrstuhl VII „Graphische Systeme“ des Fachbereich Informatik an der Universität Dortmund für die Bereitstellung der Räumlichkeiten und der technischen Ausstattung. Im Besonderen danken wir Prof. Dr. Heinrich Müller und Dipl. Inform. Frank Weichert für die hervorragende Unterstützung in den vergangenen zwei Semestern.

A Handbuch

A.1 Vorwort

Der Hörsaal innerhalb des neuen Informatikgebäudes an der OH-14 ermöglicht mittels einer modernen INFITEC-basierten stereographischen Projektion, die Integration der PG-Umsetzung in eine professionelle Multimedia-Umgebung. Abbildung 86 zeigt das Multimediaequipment schematisch und vermittelt eine mögliche Herangehensweise zur Realisierung der Aufgabenstellung.

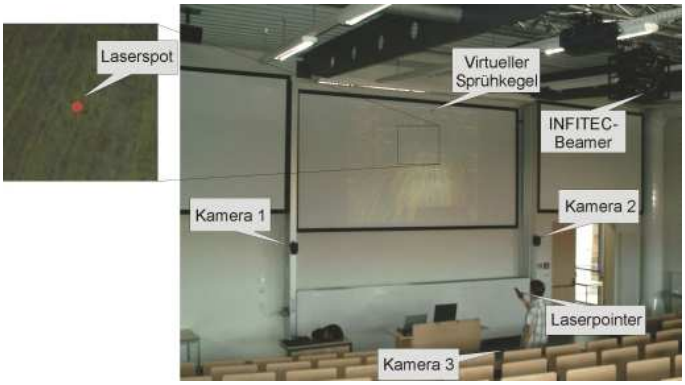


Abbildung 86: Darstellung des Informatik-Hörsaals mit seiner stereographischen Projektionswand und den zugehörigen Multimediakomponenten

Mittels der beiden Domkameras eins und zwei kann der Laserpointer getrackt und seine Position im (3D-) Raum bestimmt werden. Ergänzend erlaubt Kamera drei die Positionsdetektion des Laserpunktes auf der Projektionsfläche. In Verbindung mit den Informationen zu den räumlichen Modalitäten kann der reale Laserstrahl L1 als virtueller Suchstrahl L2 in die visualisierte stereographische Szene „verlängert“ werden - zur Interaktion im virtuellen 3D-Raum.

A.1.1 Was ist 3D-LaserNAV

Aktuelle Realisierungen der oben genannten Problematik decken jeweils nur einen Teilbereich ab oder sind auf Grund ihres Designs vergleichsweise nicht oder nur eingeschränkt unter realen Anwendungsszenarien einsetzbar. Selbst unter Verwendung expliziter 3D-Eingabegeräte (z.B. 3D-Maus) ist vielfach keine intuitive oder kooperative Nutzung durch mehrere Personen gegeben.

Das von der Projektgruppe zu entwickelnde System „3D-LaserNAV“ soll alle Teilaspekte integriert bearbeiten und durch einen modularen Aufbau gewährleisten, dass sich zukünftige Anwendungen problemlos integrieren lassen. Daher ist auch die resultierende prototypische Umsetzung nicht auf die einleitend aufgezeigte Anwendung spezifiziert, sondern erlaubt eine generalisierte Nutzung und Erweiterung der Erkenntnisse der Projektgruppe. So findet sich das prinzipielle Anwendungsszenario in vielfältigen Bereichen wieder, die eine virtuelle oder augmentierte 3D-Visualisierung benötigen. Sei es im Kontext einer komplexen Datenvisualisierung, 3D-Operationsplanung, bei der Visualisierung von Fräs- und Zerspanungsprozessen, zur dreidimensionalen Repräsentierung mathematischer Simulationen oder zur Einbettung innerhalb einer erweiterten multimedialen Lernumgebung.

A.1.2 Programmablauf

Das Programm besteht aus einer Server- und einer Clientanwendung. Die Serveranwendung wird auf dem Server gestartet und sammelt die Daten, die über den Dallmeier von den Kameras ankommen (siehe Abbildung 87).

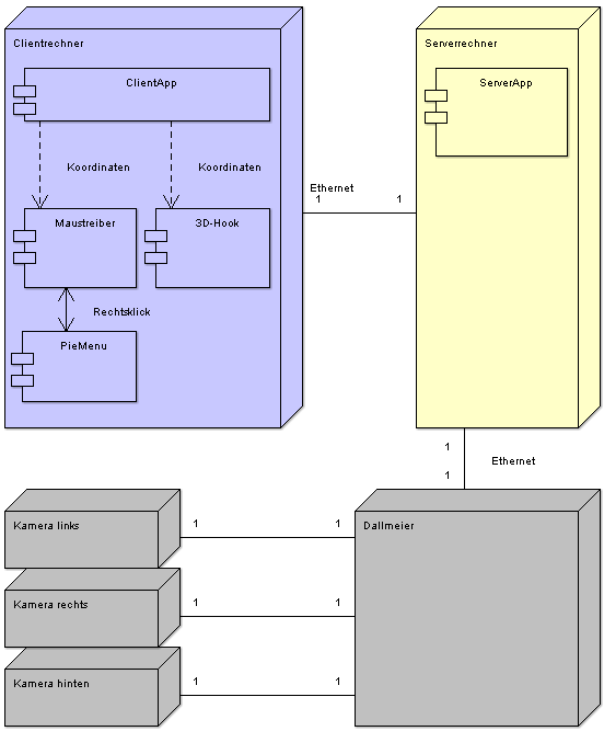


Abbildung 87: Das Gesamtsystem

Anschließend werden die Daten verarbeitet, um die Koordinaten des Laserpunktes im Bild zu erhalten. Auf dem Präsentationsrechner wird eine Clientanwendung gestartet, welche mit der Serveranwendung verbunden wird und die Berechnung auf dem Server anstößt. Die Koordinaten, die auf dem Server errechnet worden sind, werden an den Clientrechner geleitet, wo sie mittels installiertem Maustreiber die Position des Cursors auf den Client festlegen. Die Daten zur Verbindung zwischen Server und Client werden im Client angelegt und können dort auch verwaltet werden. Um zwischen der Anwendung im 2D- und im 3D-Modus zu wechseln wird ebenfalls die Clientanwendung verwendet. In der Clientanwendung können verschiedene Benutzerprofile angelegt und gespeichert werden, die unter anderem auch die Farbe des verlängerten Laserstrahls festlegen.

A.2 Installation

Es muss jeweils auf dem Client und auf dem Server die zugehörige Anwendung vorhanden sein. Die beiden nötigen Programme „Client.exe“ und „Server.exe“ werden dort abgelegt und gestartet. Auf dem Präsentationsrechner, also dem Client, ist es nötig den modifizierten Maustreiber zu installieren, der die Koordinaten aus der 3D-LaserNAV-Anwendung auf den Mauscursor anwendet. Die Installation des Treibers geschieht über eine geeignete Installationsroutine.

A.2.1 Hardwarevoraussetzungen

Folgende Hardware wird für den Betrieb von 3D-LaserNAV vorausgesetzt:

- Graphikkarte mit zwei DVI Ausgängen
- Prozessor mit 3 GHz und 1024 MB Ram
- Drei im Dreiecksraum positionierte Kameras
- Laserpointer mit zwei Knöpfen und einem USB/PS2 Anschluss für die Funkübertragung

A.2.2 Softwarevoraussetzungen

Folgende Software wird für den Betrieb von 3D-LaserNAV vorausgesetzt:

- WindowsXP (mit Service Pack 2 und zusätzlich installiertem Windowstastaturpaket)

- Treiber für OpenGL und DirectX
- Treiber für die Graphikkarte und Maus (gestellt von der PG 498)
- more3DStereo-Software für die Stereoprojektion
- Präsentationtsanwendung (z.B. Microsoft PowerPoint)

A.3 Bedienung des Programms

Zu Beginn muss die Serveranwendung auf dem Server im Vorbereitungsraum gestartet werden. Diese läuft während des gesamten Betriebs und wartet zunächst auf eine mögliche Verbindung mit der Clientanwendung. Die Clientanwendung (siehe Abbildung 88) wird auf dem Präsentationsrechner gestartet und initialisiert eine Verbindung zum Server. Nach starten der Anwendung befindet man sich im „LogView“ Fenster, welches in einem weiteren Abschnitt genauer erklärt wird.

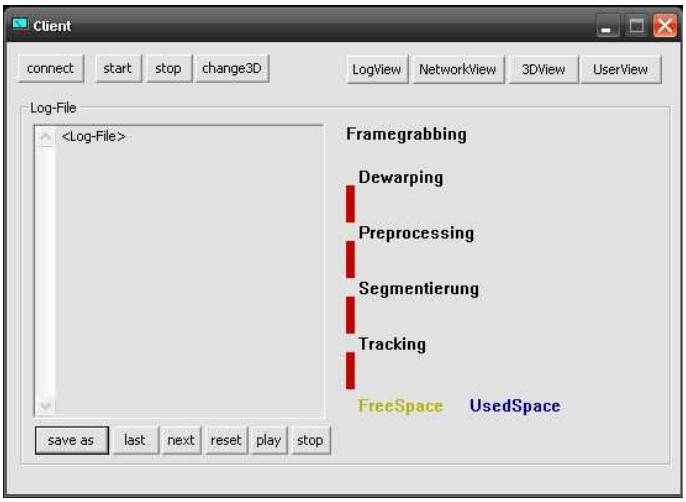


Abbildung 88: Die Clientanwendung nach dem Start

Beim ersten Start der Anwendung müssen die nötigen Netzwerkeinstellungen konfiguriert werden. Dies geschieht im Fenster `Network View`, welches von der Startseite aus direkt über den zugehörigen Button erreichbar ist (siehe Abbildung 89). Dort werden die IP und die UDP- und TCP-Ports zum Server sowie die UDP- und TCP-Ports der Clientanwendung eingetragen. Die eingetragenen Werte werden durch drücken der Buttons `Set ServerIP`, `SetServerPorts` und `SetClientPorts` übernommen. Man kann einen Servernamen eingeben und die Netzwerkkonfigurationen mit dem Button „Save

Server“ speichern. Vorhandene Konfigurationen werden in der `Serverlist` im unteren Bereich des NetworkView angezeigt. Diese lassen sich auswählen und können dann per `Load Server` die aktuelle Konfiguration ersetzen. Mit `"New Server` werden die Eingabefelder automatisch geleert und ein neuer Server kann angelegt werden. Nicht mehr benötigte Konfigurationen können gelöscht werden, indem man sie in der ServerList auswählt und dann auf den Button `Delete Server` drückt. Nach der ersten Konfiguration bleiben die Werte bei jedem Programmneustart gespeichert und müssen nicht neu eingegeben werden, solange die IP oder Ports unverändert bleiben.

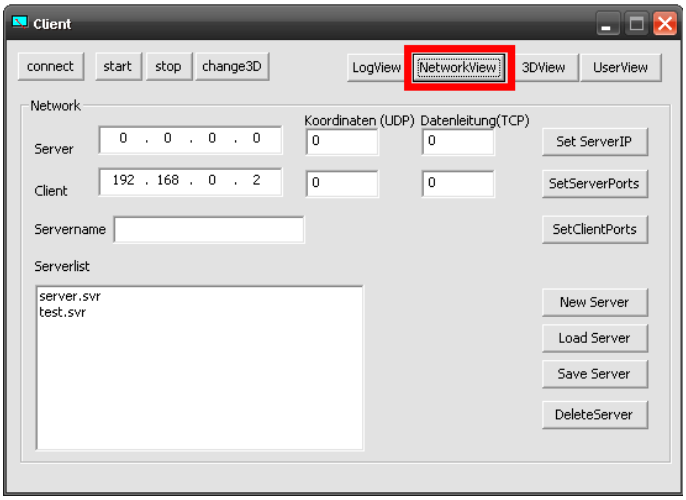


Abbildung 89: NetworkView in der Clientanwendung

Über die drei Buttons `connect`, `start` und `stop` in der oberen linken Ecke lässt sich die Verbindung und die Berechnung der Mauskoordinaten aktivieren. Nach der Konfiguration der Netzwerkeinstellungen und dem Start der Serveranwendung auf dem Aufnahmerechner kann man mit dem Button `connect` die Verbindung zum Server aufbauen. Nach erfolgreicher Verbindung zur Serverapplikation startet man die Berechnung der Koordination des Laserpointers mit dem Button `start` und kann diese auch wieder mit `stop` beenden. Um vom 2D-Modus in den 3D-Modus zu wechseln betätigt man den Button `change3D`, wodurch die Serveranwendung im 3D-Modus zusätzlich die Position des Laserpointers findet, um den in die 3D-Ebene verlängerten Laserstrahl darstellen zu können. Während der Client mit dem Server verbunden ist und mit diesem Daten austauscht, wird ein Logfile geschrieben, welches Informationen über den Ablauf der Anwendung bereitstellt. Das Logfile lässt sich nach einem „stop“ der Berechnung im `LogView` genauer analysieren (siehe Abbildung 90).

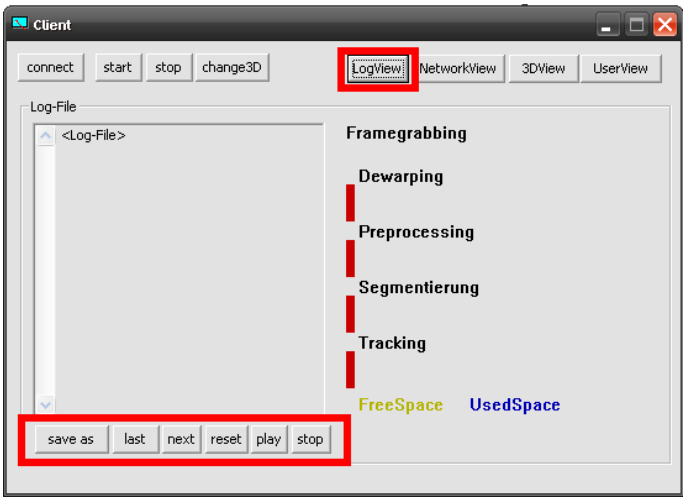


Abbildung 90: LogView in der Clientanwendung

Die Daten aus dem Logfile werden in der Box auf der linken Seite im Fenster angezeigt und es lässt sich über den Button `save as` abspeichern. Um die komplexen Daten aus dem Logfile graphisch anschaulich darzustellen, kann man die einzelnen Zeilen des Logfiles über die Buttons `last` und `next` durchgehen, wobei die Informationen und der Zustand der Anwendung auf der rechten Seite für den jeweiligen Teilbereich dargestellt wird. Um diese Darstellung für das ganze Logfile automatisch durchlaufen zu lassen, kann man diese mit dem Button `play` starten und mit `stop` wieder beenden. Mittels `reset` kann man das aktuelle Logfile wieder zurücksetzen.

Das Fenster `3DView` (siehe Abbildung 91) ermöglicht es, verschiedene Styles aus der Drop-Down-Box für den 3D-Laserstrahl auszuwählen.

Zu jedem Benutzer können mehrere Profile angelegt werden, welche unter anderem auch die Farbe des verlängerten Laserstrahls enthalten. Um ein Profil zu erstellen oder zu bearbeiten, wechselt man in das Fenster `UserView` (siehe Abbildung 92). Hier lässt sich der Name für das neue Profil eingeben und es können die zum verwendeten Laserpointer passenden Attribute eingestellt werden. Eine automatische Erkennung des verwendeten Laserpointers kann für unterstützte Modelle über den Button `detect` erfolgen. Das erstellte Profil kann anschließend mit dem Button `SaveUser` gespeichert und mit dem Button `LoadUser` wieder geladen werden. Mit `DeleteUser` kann ein angelegtes Profil wieder aus der Liste gelöscht werden. Der Button `NewUser` setzt

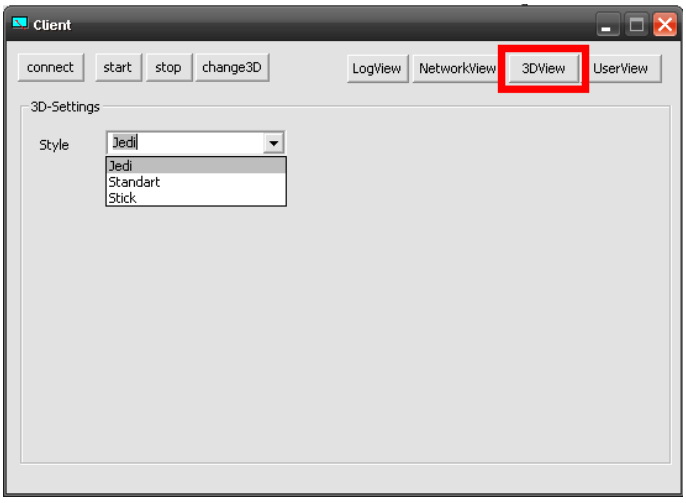


Abbildung 91: 3DView in der Clientanwendung

die Eingabefelder wieder zurück.

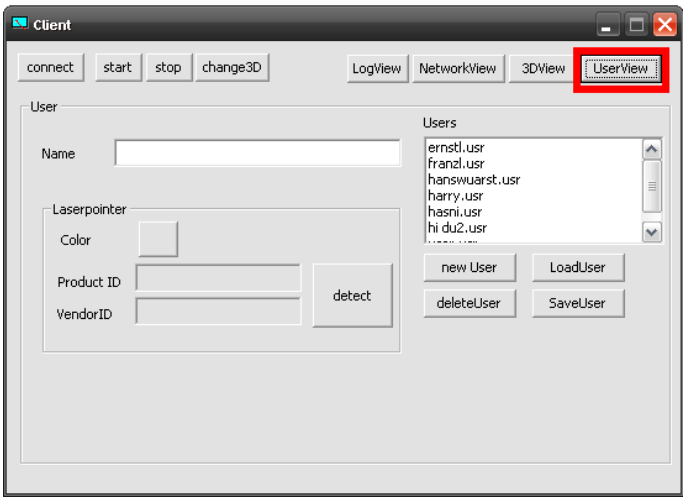


Abbildung 92: UserView in der Clientanwendung

B Klassendokumentation

B.1 Subsystem

Für die einzelnen Subsysteme, die alle nebenläufig arbeiten und von der Controller-Klasse gesteuert werden sollen, wurde eine Basisklasse entworfen. Diese Subsystem-Klasse kapselt das Verhalten eines Threads, welches sie von der CThread-Klasse erbt. Die Schnittstelle zum Controller, der zentralen Kontroll-Klasse der Server-Applikation, wird über die Methoden *start*, *stop* und *continue* bereitgestellt. Die jeweiligen Subsysteme (Framegrabbing, Dewarping, usw.) können für den eigentlichen spezifischen Code die Methode *SingleStep* überladen, die den nebenläufigen Codeteil enthält.

B.2 Controller

Die Controller-Klasse ist die zentrale Kontrollinstanz der Server-Applikation. Hier wird den jeweiligen Subsystemen ihre Aufgabe zugeteilt und bestimmt, wann sie mit ihrer Arbeit beginnen sollen. Außerdem werden die Daten der Pipeline hier weitervermittelt. Die Controller-Klasse verwendet die drei Regeln für das Scheduling der Subsysteme, welche im Abschnitt „Das Gesamtsystem“ ausführlich beschrieben sind. Diese Klasse stellt außerdem die Netzwerkverbindungen zum Stream der Kamerabilder, sowie der Client-Applikation her.

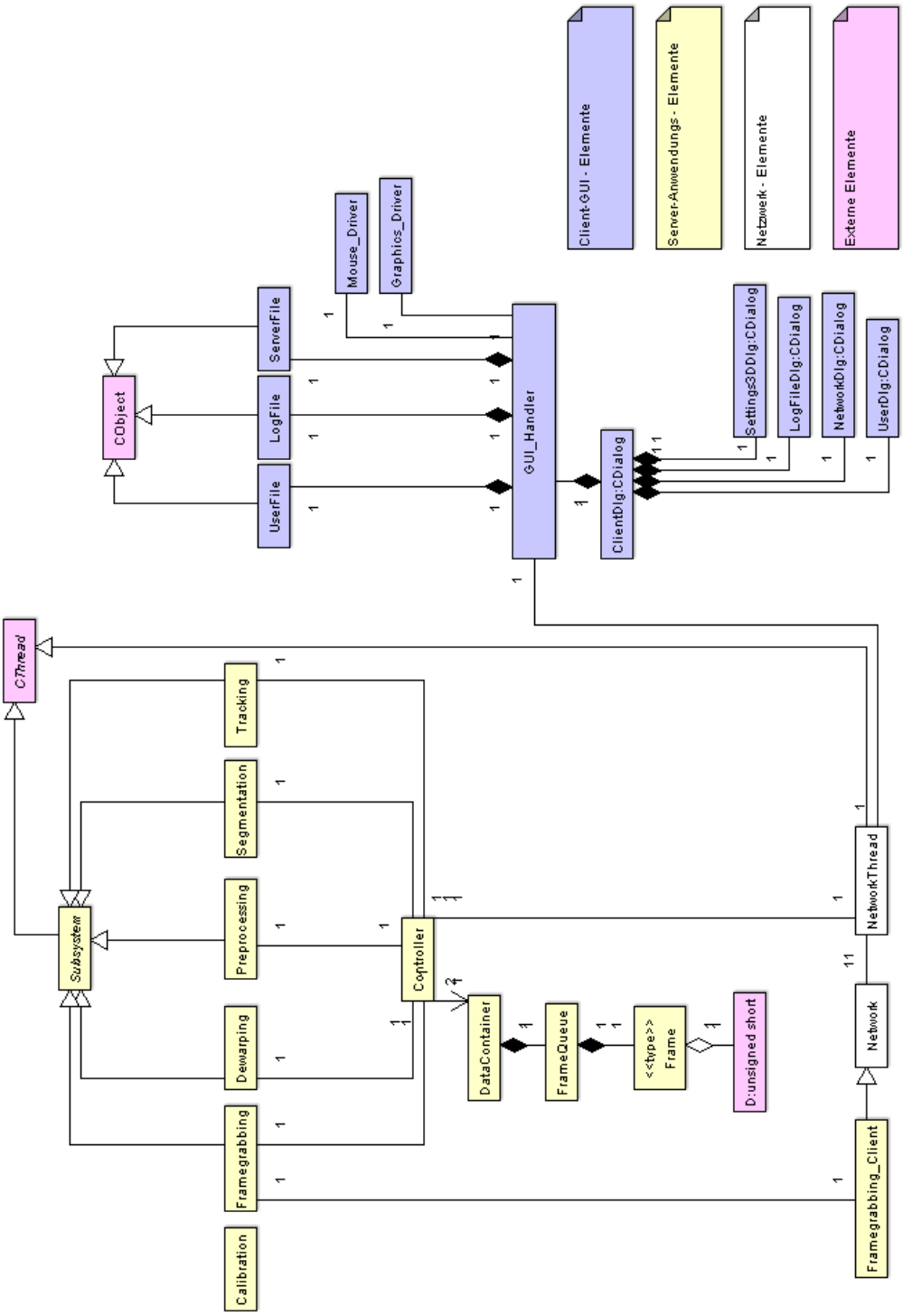


Abbildung 93: Klassendiagramm

C Werkzeuge

C.1 VLC media player

Der VLC media player wird seit 1999 vom VideoLAN-Team entwickelt. Dieses besteht aus Studenten der französischen Ingenieurschule École Centrale Paris in Châtenay-Malabry bei Paris und Entwicklern aus über 20 Ländern, unter anderem den USA, den Niederlanden, Norwegen und Deutschland. Das Programm steht unter der GPL und kann somit kostenlos verbreitet, aber auch von jedermann verbessert werden. Aus urheberrechtlichen Gründen ist die Wiedergabe von DRM-geschützten Formaten nicht möglich. Die aktuelle Version ist 0.8.6c und wurde am 17. Juni 2007 verbreitet und von der Projektgruppe 498 eingesetzt.

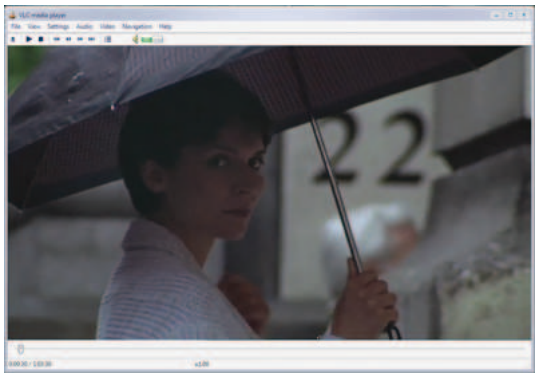


Abbildung 94: Wiedergabe eines Videos auf einem VLC media client

C.2 OpenGL

Um für die Verlängerung des Laserstrahls in die 3D-Szene eines beliebigen Programms eingreifen zu können, war es selbstverständlich unerlässlich eben diese 3D-Bibliotheken, wie das jeweilige Programm zu nutzen. Die OpenGL-Bibliothek ist eine von zwei Standardbibliotheken und stellt dem Nutzer eine Schnittstelle zur Verfügung, mit der es möglich ist, sowohl 2D- aber insbesondere auch 3D-Grafik umzusetzen. Es handelt sich um eine Programmierschnittstelle, die heute auf einer breiten Palette von Betriebssystemen verfügbar ist. So gut wie alle Programme, die 3D-Graphik bieten und auf Windows-Betriebssystemen laufen, nutzen entweder OpenGL oder das Konkurrenzprodukt von Microsoft DirectX, somit war die Wahl dieser beiden Bibliotheken obligatorisch.

C.3 Visual Studio 2005

Als Entwicklungsumgebung Microsoft VisualStudio 2005 gewählt, welches sich neben der allgemeinen Softwareentwicklung auch als Standard im Bereich der Treiberentwicklung herausgestellt hat. Dies gilt nicht zuletzt, weil Microsoft hierfür das DDK (s.u.) bereitstellt. Mit dieser Entwicklungsumgebung können Treiber abseits der Kommandozeile kompiliert werden. Bei der Entwicklung stehen hier umfangreiche Hilfen zur Verfügung, z.B. Intellisense, Codevervollständigung usw. .

C.4 LaTeX

Mit LaTeX2_ϵ wurden sämtliche Dokumente erstellt, die während der Projektphase zu erstellen waren. Zum einen wurden sowohl die wöchentlichen Sitzungsprotokolle als auch die umfangreicheren Berichte, wie Pflichtenheft und Seminarband mit dem Textsatzsystem verarbeitet. LaTeX bietet umfangreiche Pakete zur Erstellung wissenschaftlicher Dokumente. Besonders hervorzuheben ist hierbei beispielsweise die Formelgebung sowie die Möglichkeit einzelne Dokumente in separaten Dateien abzuspeichern, um sie letztendlich in einem großen Dokument wie diesem Endbericht zusammenzufügen.

C.5 Fastest Fourier Transform in the West (FFTW)

Die umfangreiche *FFTW* bietet effiziente Methoden, um Bilder fouriertransformieren zu können. Sie verarbeitet die eingehenden Datensätze intern in der eigenen Datenstruktur `fftw_complex[x][y]`, welche sich für die Darstellung komplexer Zahlen als äußerst hilfreich erwies. Es handelt sich dabei um ein Array mit jeweils zwei `double`-Zahlen pro Index `[x]`. Im zweiten Index `[y]` kann dementsprechend nur eine null oder eins stehen, wobei null bedeutet, dass man gerade den Realteil einer komplexen Zahl betreachtet und bei eins den Imaginärteil. Für unser Programm war es somit entscheidend, unsere eigene Datenstrukter in ein `fftw_complex` zu übertragen, was auf Grund der guten Zugriffsmöglichkeiten auf beide Datenstrukturen durch einen einfachen und schnellen Listendurchlauf erledigt werden konnte.
Um die Transformation durchzuführen, brauchten lediglich die von der *FFTW* zu Verfügung gestellten Funktionen aufgerufen werden. Dabei mussten wir uns entscheiden, ob wir die Transformation in der Variante `MEASURE`, `ESTIMATE` oder `PATIENT` verwenden [FJ06]. In Hinblick auf den Tradeoff zwischen Geschwindigkeit und Verlässlichkeit haben wir uns für `ESTIMATE` entschieden.

C.6 Newmat

Die Matrizenbibliothek Newmat11 beta [Dav06] bietet Datenstrukturen und Operationen zur effizienten Durchführung von Matrizenoperationen. Einige der im Projekt genutzten Standardklassen erben von Datenstrukturen der Bibliothek. Die genutzten Operationen sind Multiplikationen, Additionen und Invertierungen, sowie zur Berechnung der Kovarianzmatrix, der Einvektoren und der Eigenwerte. Die Operationen sind effizient und gut leserlich im Code.

C.7 gnuplot

Bei Transformationen mittels der *FFTW* oder vergleichbaren Methoden wurde während des Entwicklungsprozesses ein Hilfsmittel benötigt, welches möglichst schnell und unkompliziert Datensätze eines Arrays visuell anzeigen konnte. Wir haben hierfür *gnuplot* verwendet. Wir mussten zunächst eine einfache Methode entwickeln, die ein beliebiges zweidimensionales Array in eine Datei ausgibt. Diese Datei konnte problemlos in *gnuplot* eingelesen werden. Die Werte wurden dabei dann bspw. als Koordinaten interpretiert. Je nach Befehlsaufruf, konnte man die Punkte auch mit Linien versehen anzeigen lassen, so dass Funktionsgraphen in Form von Kurven oder Parabeln erkennbar wurden.
Gnuplot selbst und die Routinen zur Dateispeicherung sind nicht mehr ins Endergebnis unserer Projektgruppe eingeflossen. Sie dienten lediglich zur Entwicklung.

C.8 Concurrent Versions System (CVS)

Für die Verwaltung von Programmcode bei jedem größeren Projekt, bei dem auch mehrere Personen gleichzeitig an diesem Code arbeiten, ist die Verwendung eines Versionierungssystems unerlässlich. Diese Systeme verwalten den gleichzeitigen Zugriff auf dieselbe Datei und fügen die Änderungen wenn möglich automatisch zusammen oder bieten die Option dies manuell zu tun. Darüberhinaus werden die einzelnen Dateiversionen mit Versionsnummern versehen, die bei Bedarf auch wieder in einen früheren Zustand überführt werden können. Es existieren zwei Standards in diesem Bereich *Subversion* und *CVS* (Concurrent Versions System). Subversion ist als Ersatz für das ältere CVS entworfen worden und behebt einige Schwächen des CVS-Systems. Dennoch stand auf dem Server des Lehrstuhls lediglich CVS zur Verfügung, welches später allerdings immer wieder Probleme verursachte. Diese Probleme bestanden insbesondere darin, dass sich bestimmte Dateien des Programmcodes nicht mehr auschecken ließen.

C.9 OpenCV

OpenCV ist eine offene Programmbibliothek, welche Algorithmen für digitale Bildverarbeitung unter C bzw. C++ zur Verfügung stellt. Es ist eine große Anzahl von teilweise stark parameterisierbaren Filtern sowie Datenstrukturen zur internen Verarbeitung von Bilddaten vorhanden. Die Bibliothek wurde von Intel veröffentlicht und ist aktuell in der Version 1.0 verfügbar. Auf Grund der Vielfalt der zusammengefassten Algorithmen und der einfachen Implementierungsmöglichkeiten wurde OpenCV in der frühen Entwicklungsphase eingesetzt, um schnell unterschiedliche Ansätze in der Segmentierung oder der Bildverbesserung testen zu können. Im endgültigen Produkt wird aus Performancegründen auf die Verwendung von OpenCV verzichtet.

C.10 Kino

Kino ist ein nichtlinearer DV Editor unter Linux und wurde verwendet, um Testaufnahmen eines DV-Camcorders in den Rechner einzuspielen und interessante Abschnitte aus ihnen herauszuschneiden. Zur Verwendung kamen die Versionen 0.9.4 und 1.0.0.

C.11 VMware

Bei der Entwicklung von Treibern, die im Gegensatz zu „normalen“ Programmen und DLL's im Kernelmode laufen, ist es selbst bei sorgfältigem Design des Treibers nicht auszuschließen, dass mit falschen Speicheraddressen gearbeitet wird. Dies ist auf eine andere Speicherverwaltung im Kernelmode zurückzuführen und führt unter Umständen zu einen kompletten Systemabsturz mit Anzeige der „Blue Screen“-Fehlermeldung, welche genau für Meldungen im Kernelmode nicht mehr behebbarer Fehler geschaffen wurde. Um das Entwicklngs- und Testsystem nicht zu gefährden, wurden die ersten Testläufe in einer virtuellen Testumgebng durchgeführt, die einen realen Computer simuliert. Es gibt eine Vielzahl solcher VM-Software, die Wahl fiel hier auf VMware der Firma VMware Inc. .

C.12 ArgoUML

Für den Systementwurf wurde Argo-UML als UML-Modellierungstool gewählt. Dieses stellt ein frei verfügbares Java basiertes Tool dar, das den Anforderungen gerecht wurde, Stabilität bot und eine einfache Einarbeitung erlaubte.

C.13 DDKWizard

Zur Erleichterung der Arbeit wurde ein sogenannter Wizard eingesezt, um ein Visual Studio-Projekt anzulegen. Es handelt sich um eine Erweiterung der Visual Studio Umgebung. Dieser Wizard führt alle Projekeinstellungen, die für die Treiberentwicklung nötig sind automatisch durch, legt die Projektdateien an und nimmt dem Programmierer so diese fehleranfällige Arbeit ab.

C.14 DebugView

Um Debugausgaben aus dem Kernelmode darstellen zu können, wurde auf DebugView zurückgegriffen, ein Freeware-Programm, vergleichbar mit der Konsolenausgabe von Visual Studio. Allerdings ermöglicht es dieses Programm Debugnachrichten aus dem Kernelmode abzufangen.

C.15 WinDDK

Damit man mittels Visual Studio 2005 Treiber entwicklen kann, benötigt man zusätzlich das Driver Development Kit, kurz "DDK", welches alle erforderlichen Bibliotheken, Tools und Kompiler mitliefert. Im Gegensatz zu Visual Studio 2005 ist das DDK frei verfügbar. Man kann mit dem enthaltenen Kompiler auch ohne das Visual Studio Treiber kompilieren, jedoch bietet dieses umfangreichere Hilfsmittel, als lediglich die Kommandozeile des DDK-Kompilers. Zum DDK gehören außerdem eine umfangreiche Dokumentation und Beispielprojekte zur Treiberprogrammierung.

C.16 Matrox Imaging Library (MIL)

Die Matrox Imaging Library, entwickelt von Matrox Electronic Systems Ltd. (Abbildung 95), ist ein Software Entwicklungs Toolkit für die industrielle und medizinische Bildverarbeitung und Bildanalyse. Sie stellt eine komplette und leicht zu verwendende Programmierbibliothek für die Bilderfassung, Bildverarbeitung, Bildanalyse, Bildanzeige und Bildarchivierung dar. Neben Multithreading wird auch Multiprocessing unterstützt.

Beispielhafter Programmcode für die Aufnahme und kontinuierlichen Darstellung einer Videosequenz:

```
#include <stdio.h>
#include <stdlib.h>
```

Abbildung 95: Logo von Matrox Electronic Systems Ltd.

```
#include <conio.h>
#include <mil.h>
#define IMAGE_WIDTH 640
#define IMAGE_HEIGHT 480
#define NBGRAB 8 /* Number of image buffers in the sequence. */
void main(void)
{
MIL_ID MilApplication, /* Application identifier. */
MilSystem, /* System identifier. */
MilDigitizer, /* Digitizer identifier. */
MilDisplay, /* Display identifier. */
MilImage[NBGRAB], /* Number of image buffers in the sequence. */
MilImageDisp; /* Display image buffer identifier. */
long n;
/* Allocate defaults. */
MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, &MilDisplay,
&MilDigitizer, M_NULL);
/* Allocate sequence storage image buffers. */
for (n=0; n<NBGRAB; n++)
{
MbufAlloc2d(MilSystem, IMAGE_WIDTH, IMAGE_HEIGHT, 8L+M_UNSIGNED,
M_IMAGE+M_GRAB, &MilImage[n]);
```

```
}
/* Allocate a display image buffer. */
MbufAlloc2d(MilSystem, IMAGE_WIDTH, IMAGE_HEIGHT, 8L+M_UNSIGNED,
M_IMAGE+M_GRAB+M_PROC+M_DISP, &MilImageDisp);
/* Display activation. */
MbufClear(MilImageDisp, 0x0);
MdispSelect(MilDisplay, MilImageDisp);
/* Print a message. */
printf("Press enter to record the sequence.\n");
getchar();
/* Grab the sequence. */
for (n=0; n<NBGRAB; n++)
{
MdigGrab(MilDigitizer, MilImage[n]);
}
/* Play the sequence until a key is pressed. */
while( !kbhit() )
{
/* Play the sequence once. */
for (n=0; n<NBGRAB; n++)
{
/* Copy one image to the screen. */
MbufCopy(MilImage[n],MilImageDisp);
}
}
/* Free image buffers. */
MbufFree(MilImageDisp);
for (n=0; n<NBGRAB; n++)
{
MbufFree(MilImage[n]);
}
/* Free defaults. */
MappFreeDefault(MilApplication, MilSystem, MilDisplay,
MilDigitizer, M_NULL);
```

C.17 Digitizer Configuration Format (DCF)

Matrox Imaging bietet detaillierte Kameraschnittstellen-Beschreibungen für alle Kameras an, die schon an die Matrox Hardware angepasst wurden. Diese Datenblätter beschreiben die Kamera, erklären welche Betriebsarten auf

welche Weise unterstützt werden und dokumentieren die benötigte Kabelverbindung. Zusätzlich beinhalten sie auch die DCF-Dateien (Digitizer Configuration Format), die für die Programmierung der Hardware benötigt werden. Der Inhalt einer DCF ist auszugsweise angegeben:

```
/Matrox Electronic Systems Ltd.
/Copyright 1999.
/
/PAL compatible DCF template for rapid channel switching
/in field mode.
/
/Requires MIL/MIL-Lite 6.01 with the MIL 6.01 driver for
/Matrox Corona/Meteor-II upgrade as well as
/Matrox Corona/Meteor-II with rev. B of the video decoder.
/
/Note: this DCF can only be opened and edited in
/Matrox Intellicam version 2.06 (included with MIL 6.1).
/
/
[CAMERA_NAME]
PAL compatible
[CONFIG_FILE]
50CF
CORONA
Mon Jan 10 14:10:16 2000
[INFO_FILE_REV]
0002.0036.0000
CORONA
[GENERAL_PARAMETERS]
GEN_MATCH_HW                                    0x1
GEN_SAVED_W_ERR                                 0x0
CT_LS                                           0x0
CT_FS                                           0x1
CT_CONV_INVERTED                                0x0
CT_TAPS                                         0x0
...
```

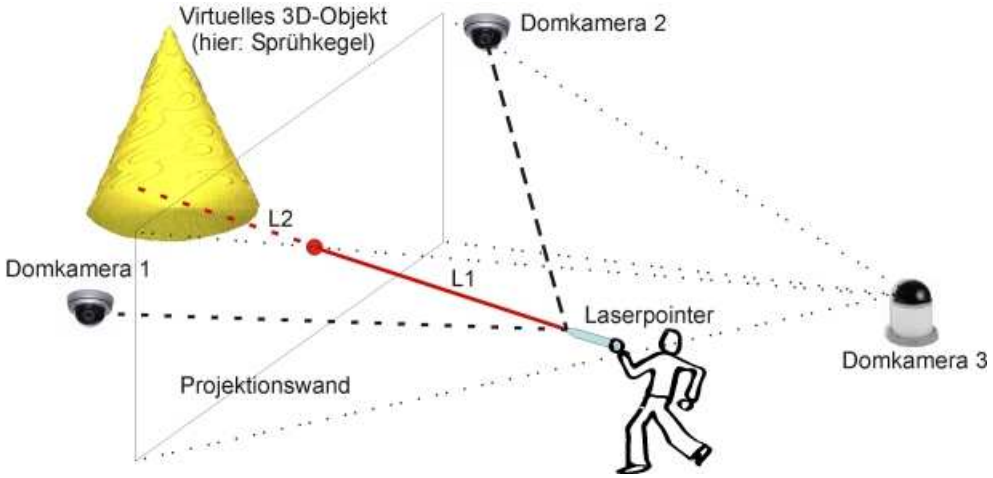
D Pflichtenheft

D.1 Zielbestimmungen

Als Ausgangspunkt ergeben sich folgende Forschungsprojekte:

- *Zerfallsprozesse* von Flüssigkeitslamellen bei Hohlkegeldüsen mit den Anwendungsbereichen Motoren und Herstellungsprozesse.
- *Operationsplanung* bei Lippen-Kiefer-Gaumenspalten mit dem Anwendungsbereich die Analyse der muskulären Kraftentwicklung als Basis virtuell optimierter rekonstruktiver Chirurgie.

Diese Projekte werden mittels einer Stereoprojektion visualisiert und von einer größeren Gruppe von Personen betrachtet. Dazu wird eine zu der Projektionsart zugehörige 3D-Brille verwendet. Es hat sich herausgestellt, dass bei einem größeren Publikum die Interaktionsmöglichkeiten mit der Projektion eher begrenzt sind.
Aus diesem Beweggrund wurde die Projektgruppe 498 des Fachbereichs Informatik Lehrstuhl VII der Universität Dortmund gebildet und mit der Aufgabe betreut, eine multimodale Mensch-Maschine-Interaktion für immersive, dreidimensionale virtuelle Realitäten zu entwickeln.



D.2 Musskriterien

Die im Folgenden aufgeführten Leistungen sind für das Produkt und seinen erfolgreichen Einsatz unabdingbar und müssen am Ende des Projekts realisiert worden sein.

Akquisition

Die Videodaten der Kameras durchlaufen, bevor sie der Bildverbesserung zugeführt werden, mehrere Stationen. Die Domkameras senden ein analoges FBAS-Signal an die Dallmeier Video-Streaming-Server. Dort wird das Bild digitalisiert und anschließend im MPEG-2-Format codiert. Der Codec bietet die nötige Effizienz, um die Daten innerhalb der Server zu archivieren und sie über das Netzwerk zu streamen. Im Empfänger müssen sie anschließend decodiert und synchronisiert werden, um eine korrekte Berechnung der Raumkoordinaten zu gewährleisten. Aus den Videostreams wird eine Datenstruktur erstellt und der weiteren Bildverarbeitung zur Verfügung gestellt. Die Struktur wird die Möglichkeit bieten, Einzelbilder einer Group of Pictures mit einer entsprechenden Timestamp zu übergeben. Die Effizienz dieser Vorgänge bestimmt zu einem erheblichen Anteil die Reaktionsgeschwindigkeit des Systems zur Interaktion in der dreidimensionalen stereographischen Realität.

D.2.1 Datenaufbereitung

Kalibrierung

Unter Verwendung von eingebrachten Markierungsgittern erfolgt eine Kalibrierung der Kameras [Zha00].

Dewarping

Basierend auf dem Ergebnis der Kalibrierung erfolgt das Dewarping (Entzerrung) der Streamingdaten [RM92].

Bildverbesserung

Trotz hochwertiger Aufnahmemodalitäten kann es zu geometrischen Verzerrungen und Diskontinuitäten (z.B. Rauschen, Reflektionen) innerhalb einzelner Sequenzen kommen. Diese Artefakte werden durch Methoden der digitalen Bildverbesserung behoben [SHB07]. Zusätzlich werden Beleuchtungsveränderungen innerhalb der Videostreams kompensiert.

Diagramm - Prozessmodell Kalibrierung

D.2.2 Segmentierung und Tracking

Segmentierung

Um die Segmentierung des Laserpunktes auch in zeitkritischen Situationen effizient innerhalb des Videostreams von Domkamera drei (Hörsaalende) zu ermöglichen, wird der Laserpunkt automatisch mit geeigneten Algorithmen der digitalen Bildverarbeitung vom Hintergrund getrennt, wobei auch Inhomogenitäten bzgl. Farbe und Kontrast berücksichtigt werden. Neben geometrischen und Farb-assoziierten Merkmalen [GW01], werden auch topologische Informationen über den vorhergehenden Bewegungspfad des Laserpunktes berücksichtigt.

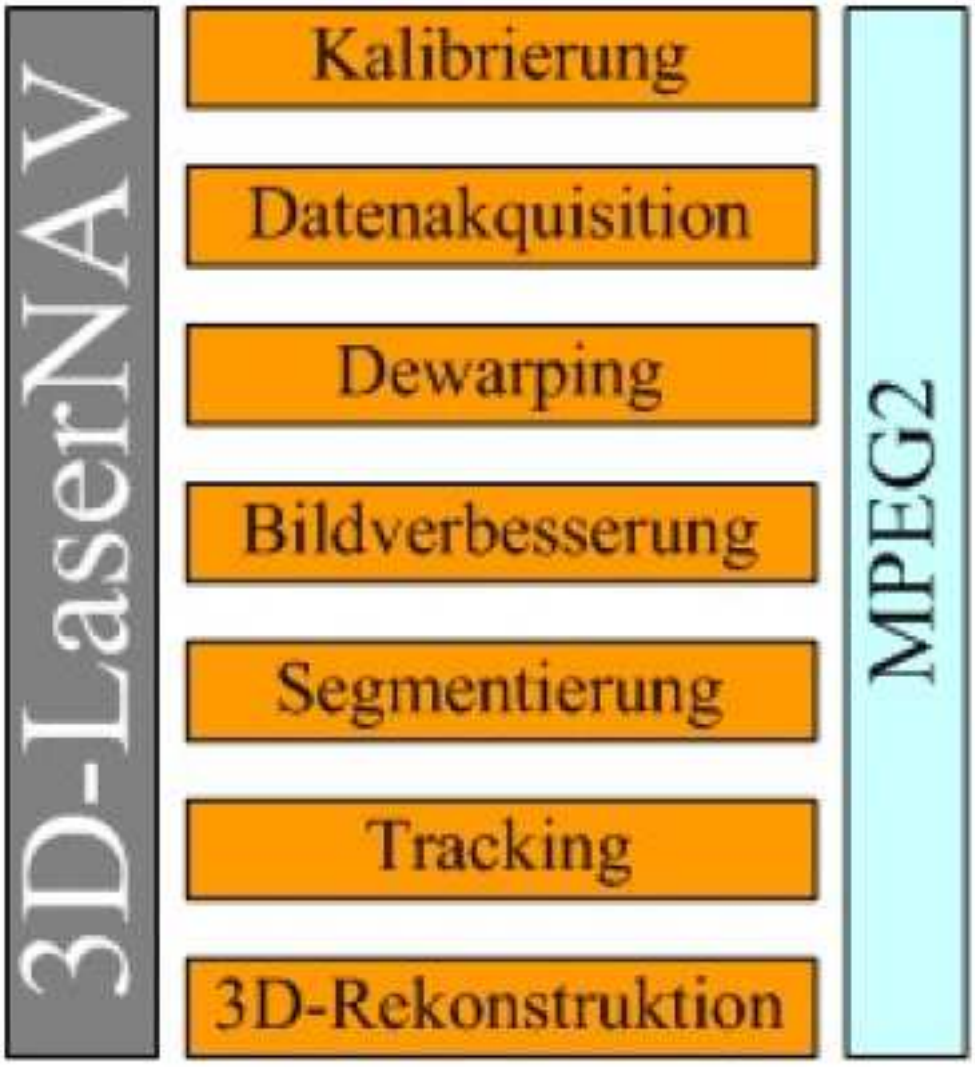
Tracking

Zur topologischen Positionsdetektion des Laserpointers muss dieser in den korrelierenden Aufnahmen der Kamera eins und zwei (Hörsaalfront) detektiert werden, um sowohl das Tracking des Laserpointers robust zu halten, als auch Probleme durch Überdeckungen zu kompensieren. Modellbasierte Trackingverfahren, wie z.B. probabilistische Ansätze (Kalman-Filter oder Conditional Density Propagation) finden hier Verwendung [IB96].

D.2.3 3D-Rekonstruktion

Für den im vorhergehenden Schritt segmentierten Laserpointer ist nun die tatsächliche räumliche Lage bestimmt. Dieses ist möglich, da zu jedem Zeitpunkt zwei Aufnahmen unter einem vorgegebenen Winkel vorliegen und die korrelierenden Aufnahmen in einem geometrischen Zusammenhang (epipolare Geometrie) stehen [Zha00]. Aus den dreidimensionalen Informationen kann die Lokalisation im Raum bestimmt werden und in Kombination mit der Position des Laserpunktes auf der Projektionswand, die Orientierung des Laserstrahls.

Diagramm - Modulsicht

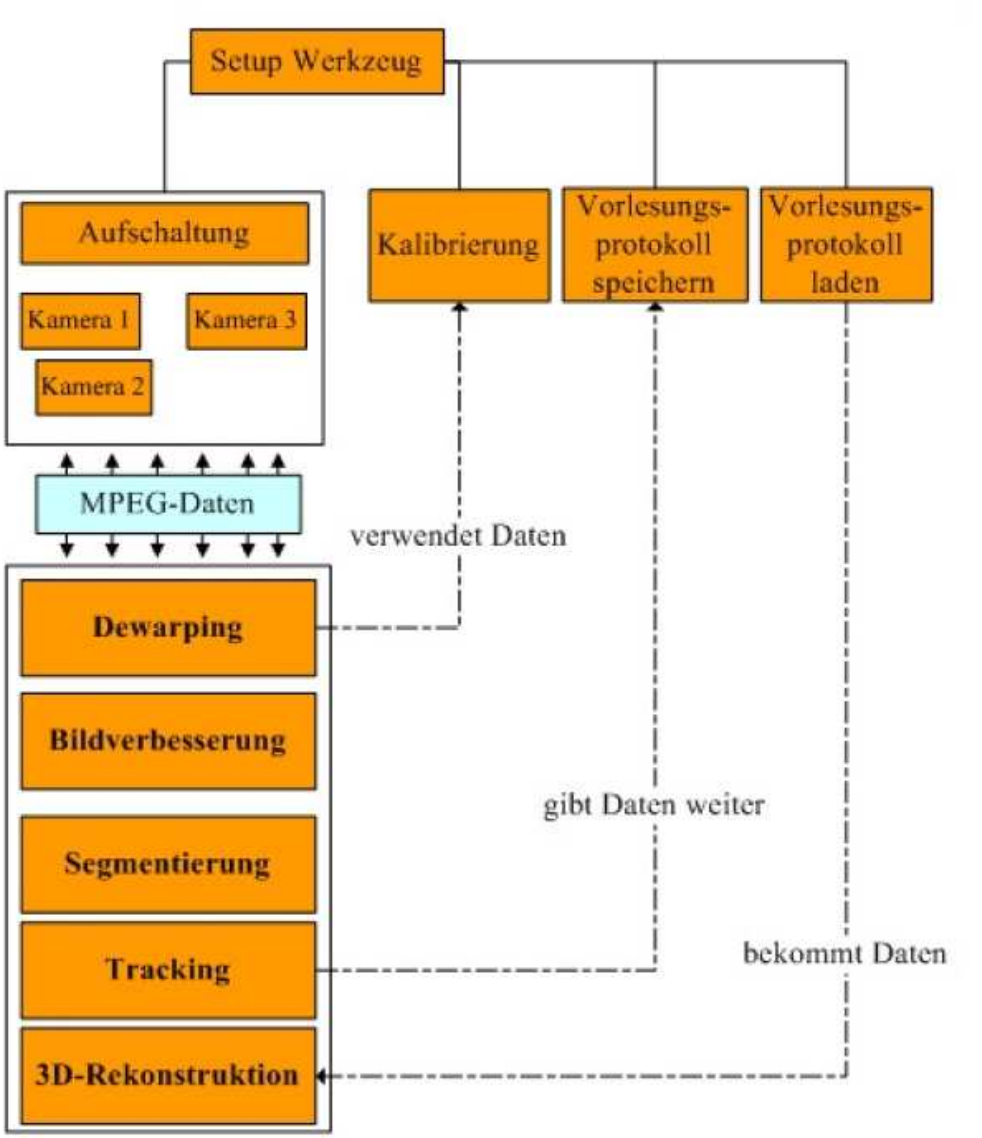


3D-Interaktionsschnittstelle

D.2.4 Maustreiber

Es wird ein eigener Maustreiber für den Laserpointer entwickelt. Mit diesem ist es dann möglich, elementare Mausinteraktionen (z.B. Mausklicks) [Kir98] auch mit einem Laserpointer durchzuführen.

Diagramm - Übergreifendes Modulschema



D.3 Wunschkriterien

Kriterien, die allgemein gehalten, nicht funktional dem Produkt zugeordnet werden können, sind an dieser Stelle aufgelistet. Diese Kriterien sind nicht für den Abschluss des Projekts zu erfüllen, sondern können von der PG 498 zur Zufriedenheit des Kunden noch ergänzt werden.

- Visualisierung des Laserstrahls in der Projektion
- Multi-User Fähigkeit
- Gestenerkennung
- Nutzung einfacher Laserpointer ohne Buttons und ohne Qualitätsminderung
- Verdeckungsunempfindlichkeit
- Variable/extreme Lichtverhältnisse
- Erweitertes Graphisches User Interface
- Vorlesungsmitschnitt/-aufzeichnung
- Interaktion eines Benutzers mit 2 Pointern gleichzeitig
- Anlegen von Profilen für Anwendungen

D.4 Abgrenzungskriterien

Abgrenzungskriterien sind Kriterien, die bewusst von der PG 498 nicht realisiert werden.

- Unbegrenzter Einsatzbereich
- Feste Reaktionszeiten
- Schrifterkennung mittels Laserpointer
- Mehrfacheinsatz von gleichfarbigen Laserpointern
- Maximale Benutzeranzahl von mehr als 3 abwechselnden Benutzern
- Andere Software-, Hardware- oder Videoformate

D.5 Produkt-Einsatz

Anwendungsbereich

Das durch die Projektgruppe 498 realisierte Produkt, wird ausschließlich für Präsentationen in praktischen Anwendungsfeldern (z.B. Motoren, medizinische Sprühgeräte, Lackspritzanlagen) eingesetzt.

Zielgruppe

Benutzer und Zielgruppen dieses Systems sind Ärtze, Mechaniker, etc. mit durchschnittlichen PC-Kenntnissen.

Betriebsbedingung

Als Betriebsbedingung wird ein wissenschaftlicher Arbeitsplatz erwartet. Dieser muss über die im Pflichtenheft in Kapitel 6 beschriebene Konfiguration und Anschluss an die zu verwendende Technik besitzen.

D.6 Produkt-Umgebung

Software

Folgende Software muss auf dem Anwenderarbeitsplatz vorhanden sein, um die größtmöglichste Funktionalität des Produkts sicherzustellen:

- Betriebssystem - WindowsXP (Service Pack 2 und zusätzlich installiertem Windowstastaturpaket)
- Treiber für OpenGL und DirectX
- Treiber für die Graphikkarte und Maus (gestellt von der PG 498)
- Software - more3DStereo für die Stereoprojektion und weitere zu verwendende Programme

Hardware

Folgende Hardware muss auf dem Anwenderarbeitsplatz vorhanden sein, um die größtmöglichste Funktionalität des Produkts sicherzustellen:

- Graphikkarte mit zwei DVI Ausgängen
- Prozessor mit drei GHz und 1024 MB Ram
- Drei im Dreiecksraum positionierte Kameras
- Laserpointer mit zwei Knöpfen und einem USB Anschluss für die Funkübertragung

Schnittstellen

- Zugriff auf die Bilddaten der drei vorhandenen Kameras
- UBS-Schnittstelle am Präsentationsrechner

D.7 Produktfunktionen

Im Folgenden sind die Funktionsbausteine der 3D-LaserNAV-Applikation aufgelistet. Funktionen sind mit ***/F Funktionsnummer/*** eindeutig bezeichnet. Ein nachgestelltes *W* kennzeichnet ein Wunschkriterium.

/F10 W/ Profil anlegen
Ein neues Profil wird für eine im Speicher schon ausgeführte Anwendung erstellt. Dabei wird ein Profilname als Eingabe erwartet. In diesem Profil befinden sich nach Abschluss Angaben über den verwendeten Anzeigemodus (2D oder 3D).

/F20 W/ Profil ändern
Ein bereits angelegtes Profil wird angezeigt. Erwartet wird eine Änderung der bestehenden Information, ansonsten wird keine Korrektur erfolgen.

/F30 W/ Profil löschen
Ein bereits angelegtes Profil wird entfernt.

/F50/ Kamera kalibrieren
Der Benutzer kann an dieser Stelle wählen, wieviele Kameras er für die Interaktion verwenden möchte. Die Kalibrierung wird anhand des Diagramm aus Punkt 2.3 durchgeführt.

/F60/ Kamera aufschalten
An dieser Stelle ist es möglich, vorhandene Kameras zu aktivieren. Die 3D-LaserNAV Software wertet die eingehenden Bilddaten aus. Möglichkeiten sind hier nur Rück-Kamera oder Rück-Kamera und beide Front-Kameras.

/F70/ Kamera abschalten
Bereits angeschaltete Kameras werden deaktiviert. Diese Deaktivierung erfolgt nur auf der Softwareebene, das heißt, die gelieferten Bilddaten werden nicht für die Interaktion herangezogen. Die Kamera an sich ist noch aktiv.

/F80/ Einstellungen abrufen
Es wird eine Übersicht über die aktuelle Konfiguration angeben. Die Kameradaten, die im Programm an sich nicht einsehbar sind, werden hier in einem Frame je Kamera angezeigt. Zusätzlich gibt es noch Informationen über die aktuell eingesetzten Treiber, Kalibrierungsdaten und Systemparametern.

/F90/ Protokoll einsehen
Mit dieser Funktion wird das Protokoll angezeigt.

/F100/ Protokoll exportieren
Das Protokoll kann in einem Textformat an auswählbarer Stelle abgespeichert werden.

/F110/ Laserpointer aufschalten
Der Laserpointer übernimmt nun die Funktion der Maus. Beide Zeigeinstrumente agieren parallel. Als Bedingung für diesen Schritt gilt, dass die Kameras aufgeschaltet sind.

/F120/ Laserpointer abschalten
Laserpointer und Maus agieren nicht mehr simultan. Die Maus ist nun wieder alleiniges Zeigeinstrument. Die Funktionen des Laserpointers mittels seiner Tasten werden ebenso deaktiviert.

/F130 W/ Laserpointer 3D-Darstellung aufschalten
Es wird zunächst eine Form der Darstellung ausgewählt (z. Bsp. eine Linie). Diese Form wird dann in die aktive Projektion, vom virtuellen Augenpunkt aus, eingeblendet. Bedingung ist hier, dass alle drei benötigten Kameras aufgeschaltet sind.

/F140 W/ Laserpointer 3D-Darstellung abschalten
Das Einzeichnen in die 3D-Darstellung wird deaktiviert.

/F150 W/ Sitzung aufnehmen
Eine Sitzung wird aufgenommen. Als Eingabe wird ein Sitzungsname für die Sitzungsdatei erwartet. Interaktionsdaten und Bilddaten werden kontinuierlich in die Datei geschrieben.

/F170 W/ Sitzung abspielen
Eine vorher abgespeicherte Sitzung wird abgespielt. Interaktionsmöglichkeiten des Laserpointers werden für die Dauer deaktiviert.

D.8 Produktdaten

Profildaten W

Die Profildaten werden lokal auf der Festplatte hinterlegt.

Sitzungsdaten W

Neben den Profildaten werden auch die Sitzungsdaten lokal hinterlegt.

D.9 Produktleistungen

Reaktionszeit

Die von der PG 498 gestellte Anwendung wird die Kriterien einer Echtzeitanwendung einhalten.

Benutzeroberfläche

Da man davon ausgehen muss, dass sich die Gruppe der Benutzer nicht nur aus Experten mit Computererfahrung zusammensetzt, wird eine grafische, menüorientierte und vor allem eine intuitiv bedienbare Benutzeroberfläche eingesetzt. Die Entwurfsphasen des Software-Engineering werden beachtet.

Die Einbettung eines Menüs in die dreidimensionale Abbildung wird an dieser Stelle als Wunschkriterium angesehen.

Qualitäts-Zielbestimmung

Die dargestellte Tabelle definiert die Schwerpunkte der Entwicklungsarbeit. Die Entwicklergruppe wird weitere Qualitätsrichtlinien aufstellen, welche die Qualität des Endprodukts sichern und die Einhaltung der oben festgelegten Kriterien überwachen. Dazu wird vom Projektmanagement ein angemessenes Qualitätsmanagement-Handbuch gepflegt, welches Dokumentation, Testfälle und die Modellierungsgrundsätze festlegt. Autarke Modultests werden die korrekte Funktionalität der Komponenten sichern.

Produktqualität	Sehr gut	Gut	Normal
Funktionalität (Funktionsumfang)			**x**
Effizienz (Zeitverhalten)	**x**		
Übertragbarkeit (Kompatibilität)			**x**
Zuverlässigkeit (Fehlertoleranz)			**x**
Benutzbarkeit (Erlernbarkeit)	**x**		

D.10 Globale Testszenarien

Es werden Variationen der Umgebung als Testfälle behandelt, damit ein unproblematischer Einsatz der Software garantiert werden kann. Die Testreihe umfasst folgende Elemente:

- Varianten der Raumbeleuchtung
- Laserpointer und dessen Portierbarkeit. Einsatz verschiedener Modelle
- Vorhandene Personen in der Einsatzumgebung und deren Einfluss auf die Funkübertragung der verwendeten Laserpointer
- Störquellen im Bezug auf Laserpointer und Kamera (konkurrierende Laserpointer, Teilverdeckung einer Kamera, etc.)

D.11 Entwicklungsumgebung

Software

- CVS, Visual Studio 2005,
- CAD Software, OpenCV
- Powerpoint, Farcry, Maya
- LaTeX und dafür benötigte Umgebung (z.Bsp. TeXnicCenter oder MikTeX)
- more3DSTEREO, Nvidia Treiberpaket
- WDF/WDM Treiberpaket
- DirectX, OpenGL
- FTP Client
- ICQ, Email, Browser (Firefox), Bugzilla
- UML Tool (z.Bsp. Together)

Hardware

- 1024 MB RAM
- ein Desktop PC, entsprechend der Konfiguration aus dem Medienwagen (bei Programmsplit dann zwei PCs)
- drei Farbkameras inklusive Anschluss, Kabel, etc.
- ein Laserpointer mit zwei Knöpfen und USB Anschluss

Orgware

- Internetanschluss
- Handbuch für das Creston Touchpanel im Hörsaal
- Handbuch für die Dallmeier und PVIEW Elemente
- Anschlussdiagramm für den Hörsaal

Entwicklungsschnittstellen

- Datenzugriff auf das Dallmeierprotokoll
- Daten- und Anwendungszugriff auf den Maustreiber
- Datenexport auf Präsentationsrechner

Literatur

[Abd03] H. Abdi. Factor Rotations in Factor Analyses. Technischer Report, The University of Texas at Dallas, 2003.

[Bre65] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, pages 25–30, 1965.

[Dav06] R. B. Davies. Newmat11 beta, http://www.robertnz.net/nm_intro.htm, 2006.

[dir04] DirectX 9.0 Software Development Kit, http://www.microsoft.com/downloads/, 2004.

[Fea90] J. D. Foley and et al. *Computer graphics: principles and practice (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[ffm07] FFmpeg, http://ffmpeg.mplayerhq.hu/, 2007.

[FJ06] M. Frigo and S. G. Johnson. Fftw - for version 3.1, http://www.fftw.org/, 2006.

[GW01] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[Han97] C. Hand. A Survey of 3D Interaction Techniques. *Computer Graphics Forum*, pages 269–281, 1997.

[IB96] M. Isard and A. Blake. Contour Tracking by Stochastic Propagation of Conditional Density. In *ECCV (1)*, pages 343–356, 1996.

[Kea01] J. Kniss and et al. Interactive Texture-Based Volume Rendering for Large Data Sets. *IEEE Computer Graphics and Applications*, pages 52–61, 2001.

[Kir98] C. Kirstein. Interaction with a Projection Screen Using a CameraTracked Laser Pointer, 1998.

[Klu06] M. Kluger. Partikelfilterbasierter, virtueller Kameraassistent zur Verfolgung einer Person in einer Gruppe von Menschen, 2006.

[lib05] libmpeg2 - a free MPEG-2 video stream decoder, http://libmpeg2.sourceforge.net/, 2005.

[Obe03] M. Oberguggenberger. Fuzzy-Logik- Was ist das?, 2003.

[One03] W. Oney. *Programming the Windows Driver Model, 2. Edition.* Microsoft Press, 2003.

[Par03] T. Park. A Note on Terse Coding of Kaiser's Varimax Rotation Using Complex Number Representation. Technischer Report, 2003.

[RM92] D. Ruprecht and H. Müller. Image Warping with Scattered Data Interpolation Methods. Technical Report 443, 1992.

[RNM95] Detlef Ruprecht, Ralf Nagel, and Heinrich Müller. Spatial free-form deformation with scattered data interpolation methods. *Computers and Graphics*, pages 63–71, 1995.

[Sac01] O. Sachse. Demoapplikation Serviceroboter fürs Kinderzimmer - Integration von Bildverarbeitung, Handlungsplanung und Navigation. Diplomarbeit, Fachhochschule Brandenburg, Fachbereich Informatik und Medien, 2001.

[Sch99] G. Schnabel. Die Farbmodelle HSV und HLS - Widersprüche in Theorie und Praxis, 1999.

[Sch05] G. Schwarzenberg. Objektverfolgung mit Partikel-Filtern. Diplomarbeit, Fakultät für Informatik,Universität Karlsruhe (TH), 2005.

[Sch06] T. Schramm. Segmentierung und Volumenbestimmung von Tumoren in MRT Mammographien. Diplomarbeit, Universität Dortmund,Fachbereich Informatik, Lehrstuhl VII, 2006.

[SHB07] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision.* Thomson-Engineering, 2007.

[Sie] J. Siegemund. Hauptkomponentenanalyse. In *Proseminar „Robuste Signalidentifikation“.*

[Smi02] L. I. Smith. A tutorial on Principal Components Analysis, 2002.

[Str05] T. Strutz. *Bilddatenkompression : Grundlagen, Codierung, Wavelets, JPEG, MPEG, H.264.* Vieweg, Wiesbaden, 3., aktualisierte und erweiterte auflage edition, 05 2005.

[Tea98] Y. Y. Tang and et al. Ring-Projection-Wavelet-Fractal Signatures: A Novel Approach to Feature Extraction. *IEEE Transactions on circuits and systems II: Analog and digital signal processing*, 45(8), 1998.

[TM98] C. Tomasi and R. Manduchi. Bilateral Filtering for Gray and Color Images. In *ICCV*, pages 839–846, 1998,.

[TT03] D. M. Tsai and Y. H. Tsai. Defect detection in textured surfaces using color ring-projection correlation. *Mach. Vision Appl.*, pages 194–200, 2003.

[vlc07] VideoLAN - Free Software and Open Source video streaming solution for every OS!, http://www.videolan.org/, 2007.

[Zei00] J. Zeidler. Untersuchung des steganographischen Algorithmus F4. Diplomarbeit, Technische Universität Dresden Fakultät Informatik Institut für Theoretische Informatik, 2000.

[Zha00] Z. Zhang. A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1330–1334, 2000.