Anbindung von Bildquellen unter Windows an die Bildverarbeitungsbibliothek PRAGMA

schriftliche Prüfungsarbeit
für die Bachelor-Prüfung des Studiengangs Angewandte Informatik
an der Ruhr-Universität Bochum

vorgelegt von

Zibner, Stephan Klaus Ulrich

Abgabe
17. Oktober 2007

erster Prüfer
Dr. Rolf P. Würtz
zweiter Prüfer
Dipl.-Inform. Guenter Westphal

Kurzfassung

Die Arbeit untersucht zur Anbindung einer Webcam unter dem Betriebssystem Windows an die Bildverarbeitungsbibliothek PRAGMA die Programmierschnittstellen WIA, Video for Windows und DirectShow auf Tauglichkeit im Sinne der Aufgabenstellung. Die Schnittstelle DirectShow wird im Bereich Bildquelleneinbindung und -konfiguration näher beschrieben und anschließend die Verwendung an einer praktischen Umsetzung erläutert. Abschließend wird auf den Funktionsumfang und den Lebenszyklus der programmierten Komponente eingegangen.

abstract

The following thesis inspects the programming interfaces WIA, Video for Windows and DirectShow for use as a connection between a webcam and the image processing library PRAGMA under Windows OS.
The integration and configuration of image sources in DirectShow is specified, followed by a commented practical implementation as example of utilization. A terminal statement of the scale of capacity and life cycle of the programmed component follows.

Stichworte

DirectShow, Webcam, Framegrabber, PRAGMA, Datenquelle

keywords

DirectShow, webcam, frame grabber, PRAGMA, data source

Inhaltsverzeichnis

1 Vorwort

Die vorliegende Arbeit entstand im Zuge der Bachelorprüfung im Studiengang "Angewandte Informatik" mit Studienschwerpunkt "Medien- und Kommunikationsinformatik". Die Arbeit ist praktisch ausgelegt, so dass neben dieser schriftlichen Ausarbeitung auch ein umfangreicher Quellcode Teil der Arbeitsleistung ist. Anhand der Arbeit soll gezeigt werden, dass eine komplexe Problemstellung mit dem vermittelten Wissen der Bachelorphase gelöst werden konnte.

2 Überblick

Diese Arbeit beschäftigt sich mit der Problematik, eine bildgebende Hardware unter festgesteckten Randbedingungen anzusprechen und diese dann zum Erhalt aktuellen Bildmaterials zu nutzen, welches zur weiteren Verwendung passend aufbereitet wird.

Die hier vorliegende Ausfertigung stellt dabei zugleich eine begründete Hinleitung zur gewählten Lösung, als auch eine Dokumentation zum entstandenen Quellcode dar.

3 Ziele

Übergeordnetes Ziel dieser Arbeit ist es, für die Bildverarbeitungsbibliothek PRAGMA eine Programmkomponente zu entwickeln und zu programmieren, die das Einbinden von Webcams als Bilddatenquellen unter PRAGMA auf Windowsplattformen ermöglicht.

Hieraus ergeben sich bereits einige Teilaspekte der Zielsetzung. So wird der zu entwickelnde Programmcode konform sein müssen zum Rest der Bibliothek, um Kompatibilität zu gewährleisten. Zudem soll der spätere Benutzer der Programmkomponente sich ausschließlich mit dessen Bedienung auskennen müssen, um mit der Webcam zu arbeiten, ohne dabei von der darunter liegenden spezifischen Programmierlösung und der eingesetzten Technologie näheres Wissen haben zu müssen. Dies ist insbesondere in Hinsicht auf die Plattformunabhängigkeit von PRAGMA von Bedeutung, da sich beispielsweise die Programmierlösung für Linux-Plattformen intern stark von der Lösung für Windows-Plattformen unterscheiden wird.

Ein weiterer Aspekt der Zielsetzung stellt die Universalität der zu entwickelnden Komponente dar. So muss nicht nur die weiter oben erwähnte Plattformunabhängigkeit, also die strikte Trennung plattform-spezifischer Lösungen garantiert sein, sondern auch eine möglichst breit gefächerte Unterstützung diverser Hardware und Entwicklungsplattformen gewährleistet werden.

Zuletzt lässt sich aus den typischen Verwendungsgebieten einer Bildverarbeitungsbibliothek eine weitere Ausprägung der Zielsetzung ableiten. So sollte gerade bei Verwendung für Echtzeit-Erkennungsaufgaben, wie z.B. der Gesichtserkennung, eine Bildanforderung möglichst in Echtzeit unterstützt werden bei größtmöglicher Anzahl Bilder pro Zeiteinheit.

4 Stand der Technik

Webcams gewinnen gerade zur Zeit des Booms des Web 2.0 und der damit einhergehenden vermehrten Benutzung von Videoaufnahmen und -streams (z.B. Videoblogs, Videochat etc.) immer mehr an Bedeutung im alltäglichen Umgang mit dem PC. Damit liegt der Gedanke nahe, diese einfach zu erwerbenden und zu bedienenden Bilddatenquellen auch für die Bildverarbeitung zu benutzen. Dabei liegt gerade im Charakter der Webcam als Echtzeitbildquelle die Grundlage zur Verwendung im Sektor der Objekterkennung, speziell bei Echtzeitverarbeitung, wie beispielsweise der Gesichtserkennung im Zuge von Sicherheitskontrollen.

Die unter Windows vorhandenen Möglichkeiten zur Multimedia-Programmierung sind sehr umfangreich und mächtig und befinden sich teilweise, zusammen mit der Einführung des neuen Betriebssystems Windows Vista, stark im Umbau. Einige Lösungsansätze und die dazugehörigen Technologien werden nachfolgend genauer beschrieben.

5 Lösungswege

Nachfolgend werden einige Lösungsansätze vorgestellt und auf die Verwendbarkeit zur Aufgabenlösung analysiert. Dabei wird der schlussendlich eingeschlagene Lösungsweg detailliert begründet und beschrieben.

5.1 hardwarespezifisch

Der erste Lösungsweg ist im Vergleich zu allen anderen Ansätzen der radikalste. Ohne auf jedwede Programmierhilfestellung zurückzugreifen, kann selbstverständlich jedes angeschlossene Gerät direkt über die Hardwareschnittstelle (z.B. USB) oder über mitgelieferte Treiber angesprochen werden. Da ein Ziel dieser Arbeit die größtmögliche Loslösung von der eigentlichen Hardware ist, stellt diese Herangehensweise keine zufrieden stellende Lösung dar und fällt auch wegen unzureichender Dokumentation seitens der Hersteller, kaum zugänglicher Programmierbarkeit über die Hardwareschnittstelle (die Ansprache von USB in Usermode-Programmen wird nicht nativ unterstützt, so dass weitere Komponenten zugeladen werden müssen, z.B. LibUSB[1]) und mangelnder Kompatibilität zu verschiedenen Gerätetypen aus dem Fokus des Interesses.

Eine echte Alternative zur direkten Ansprache der Hardware stellt der Hersteller Logitech programmierinteressierten Käufern von hauseigenen Produkten zur Verfügung. So lassen sich sämtliche Webcams der Serie "Quickcam" über ein bereitgestelltes Software Development Kit ansprechen[2]. Trotz dieses ersten Standardisierungsversuches bleibt natürlich bei der Verwendung dieses SDKs die Hardwareauswahl stark eingeschränkt. Somit ist auch dieser Lösungsansatz im Sinne der Aufgabenstellung nicht brauchbar.

[1] http://libusb-win32.sourceforge.net/

[2] http://www.golem.de/0009/9753.html

5.2 Windows Image Acquisition

Die Windows Image Acquisition (kurz WIA) ist Bestandteil des Microsoft Windows SDK und damit frei erhältlich. WIA dient als standardisierte Schnittstelle zu geläufigen Formen bildgebender Hardware, wie Scannern, Digitalkameras und digitalen Videokameras. Diese Schnittstelle wurde mit dem Betriebssystem Windows Me eingeführt[3] und stellt aufgrund der Standardisierung der Schnittstelle eine inoffizielle Weiterentwicklung der stark treiber-abhängigen Schnittstelle TWAIN dar[4].

WIA bietet neben der Verwendung von Einzelbild-Quellen auch die Möglichkeit, Streaming-Quellen, wie zum Beispiel Webcams, anzusprechen und von diesen Bilder zu erhalten, so dass die Verwendung dieser Schnittstelle für den Rahmen dieser Arbeit geprüft werden muss.

Positiv herauszuheben ist die Standardisierung der Schnittstelle. Ein Großteil aktueller Hardware wird vom WIA Device Manager erkannt und kann demzufolge als Bildquelle dienen. Somit wäre die Kompatibilität für viele Hardwaretypen gesichert.

Ein ganz klarer Nachteil sind die mangelnden Konfigurationsmöglichkeiten der Schnittstelle im Umgang mit Videoquellen, zu denen auch Webcams zählen. Da das Hauptaugenmerk im Umgang mit Videoquellen unter WIA ganz klar auf die Darstellung des Videostreams in einem Vorschaufenster liegt, sind die Einstellmöglichkeiten auf die Auflösung eben jener Bilddarstellung beschränkt[5].

Die eigentliche Bildfunktion (*TakePicture*[6]) hat weitere Eigenheiten, die eine Verwendung im Sinne der Zielsetzung dieser Arbeit untersagen. So erhält man weder einen direkten Zugriff auf den Videostream, noch kann eine programmierinterne Repräsentation eines Bildes (z.B. als Pixelmap) angefordert werden, da die angebotene Methode das angeforderte Bild direkt als (JPEG-)Datei mit dem übergebenen Dateinamen in einem angegebenen Dateiordner ablegt.
Für die Weiterverarbeitung unter PRAGMA müsste also dieses Bild erst wieder aus dieser Datei eingelesen werden, was im Zusammenspiel mit dem zeitintensiven Speichervorgang eine nicht akzeptable Verzögerung zwischen Anforderung und Erhalt der Bilddaten darstellt.

Ein letzter Kritikpunkt ist die Beschränkung der Benutzbarkeit der aktuellen WIA-Schnittstellenversion 1.0 auf die Betriebssysteme Windows Me und Windows XP[7] [8]. Eine neuere Version der Schnittstelle (WIA 2.0) wird zusammen mit dem Betriebssystem Windows Vista erarbeitet, wird aber hier nicht weiter betrachtet aufgrund der Inkompatibilität zu älteren Betriebssystemen und der Unvollständigkeit der Dokumentation und der Schnittstelle zum Zeitpunkt dieser Arbeit.

[3] http://en.wikipedia.org/w/index.php?title=Windows_Image_Acquisition&oldid=149638866
[4] ://en.wikipedia.org/w/index.php?title=Windows_Image_Acquisition&oldid=149638866
[5] http://msdn2.microsoft.com/en-us/library/ms629896.aspx
[6] http://msdn2.microsoft.com/en-us/library/ms629905.aspx
[7] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wia/wia/overviews/startpage.asp
[8] http://msdn2.microsoft.com/en-us/library/ms629896.aspx

5.3 Video for Windows

Video for Windows[9] (kurz VfW) ist ein Multimedia Framework und der Vorgänger von DirectShow. Im Verlauf der Jahre 1996/97 ging VfW zuerst in die Technologie ActiveMovie auf, welche kurze Zeit darauf Bestandteil der Technologiesammlung DirectX 5 bzw. der (damaligen) Teilkomponente DirectShow wurde[10].

VfW bot bereits rudimentäre Möglichkeiten zur Anbindung von Webcams und ähnlichen digitalen Datenquellen an multimediale Programme, ist aber heutzutage aufgrund der Nachfolgetechnologie für die Lösung dieser Arbeit nicht mehr von Interesse.
Aufgezeigt wird diese Alternative, da gerade bei der Arbeit mit veralteten Kameratypen oder der Arbeit auf älteren Windowsversionen diese Schnittstelle durchaus noch von Bedeutung sein kann, da zwar DirectShow eine gewisse Abwärtskompatibilität zu VfW anbietet, die Zusammenarbeit aber nicht garantiert werden kann.

5.4 DirectShow

DirectShow[11] ist eine Windows-API und ist (zurzeit) Bestandteil des Microsoft Windows SDK (ehemals Platform SDK). DirectShow ist ein Framework für die Arbeit mit multimedialen Daten und bietet aufgrund des mächtigen Funktionsumfangs mannigfaltige Verwendbarkeit in den Bereichen Audio-/Video-Darstellung, -Verarbeitung, -Aufnahme, Kontrolle über multimediale Hardware und vieles mehr.

Wie weiter oben beschrieben war DirectShow ursprünglich eine Komponente des DirectX-SDKs, wurde aber später aufgrund der Bedeutsamkeit in der multimedialen Windows-Programmierung als eigenständiger Bestandteil in das Windows SDK aufgenommen. Trotz dieser Trennung von DirectX greift DirectShow noch immer auf DirectX-Komponenten zur Videobeschleunigung und Audiosignalverarbeitung zu[12].

Microsoft plant, ab der Betriebssystemversion Windows Vista alle vorhandenen firmeneigenen Multimedia-APIs im neuen Framework Microsoft Media Foundation zusammenzufassen[13] und damit Schritt für Schritt die DirectShow-Technologie zu ersetzen. Da zur Zeit dieser Arbeit noch nicht abzusehen ist, wann DirectShow endgültig durch einen Nachfolger abgelöst wird und auch zu den neu angebotenen Funktionalitäten nicht viel mehr als unvollständige Dokumentationen vorliegen, ist diese Entwicklung für das Thema dieser Arbeit vorerst uninteressant.

[9] http://msdn2.microsoft.com/en-us/library/ms713492.aspx
[10] http://en.wikipedia.org/w/index.php?title=Video_for_Windows&oldid=152526823
[11] https://msdn2.microsoft.com/en-us/library/ms783323.aspx
[12] http://en.wikipedia.org/w/index.php?title=DirectShow&oldid=157191900
[13] http://en.wikipedia.org/w/index.php?title=Media_Foundation&oldid=157330368

5.4.1 generelle Funktionsweise

DirectShow basiert auf dem Component Object Model (kurz COM)[14] der Windows-Plattform, daher sind die in einer Anwendung genutzten Hauptbestandteile der Schnittstelle alle als COM-Objekte angelegt und entsprechend zu erzeugen und verwalten. Die COM-Definition erlaubt keine nachträglichen Änderungen an einer einmalig definierten Schnittstelle[15], daher sind verwendete Schnittstellen auch in einer neueren DirectShow-Version weiterhin gültig und lauffähig.

Die Hauptbaukomponenten in DirectShow sind die so genannten Filter[16], ein Sammelbegriff für Bausteine, die unter anderem eine der folgenden Leistungen zur Verfügung stellen können: eine Datei einlesen, einen Datenstrom einer Video-/Audioquelle einspeisen, Dekodieren eines Video-/Audioformats, Aufsplitten eines Datenstroms in Video- und Audiokomponenten und Weiterreichen von Daten an entsprechende Ausgabehardware, wie Grafik- oder Soundkarten[17].
Trotz dieser heterogenen Funktionsmenge haben alle Filter darin eine Gemeinsamkeit, dass sie einen oder mehrere Pins zur Verbindung untereinander anbieten[18].
Ein Pin stellt dabei die datentechnische Schnittstelle eines Filters nach außen und insbesondere zu anderen Filtern dar. Es wird unterschieden zwischen Input- und Output-Pins, wobei nicht jeder Filter zwingend einen Pin jeder dieser zwei Kategorien anbieten muss (bspw. bietet ein Filter, der Daten aus einer Datenquelle bereitstellt, nur einen oder mehrere Output-Pins, aber keinen Input-Pin an). Zudem ist ein Pin gleichzeitig auch der Anlaufpunkt für das Setzen des Formats des ein- und ausgehenden Datenstroms. So können zum Beispiel am Output-Pin eines Filters, der eine Webcam-Datenquelle repräsentiert, die ausgegebene Bild-Auflösung und der Farbraum der Videodaten eingestellt werden.

Um nun diese Bausteine in einer Anwendung zu einem funktionierenden Ganzen zusammenzufügen, bedarf es dreier Schritte[19]:
Als ersten Schritt muss eine Anwendung eine Instanz des Filter-Graph-Managers erzeugen. Dieser ist wie weiter oben beschrieben ein COM-Objekt und bietet grundlegende Funktionen zum Aufbau und zur Kontrolle eines Filter-Graphen an. In der DirectShow-Terminologie bezeichnet ein Filtergraph eine Ansammlung von Filtern, die alle untereinander verbunden sind, so dass ein Datenfluss von einer Quelle zu einem Zielfilter alle vorhandenen Filter des Graphen in gewünschter Reihenfolge durchläuft.
Der zweite Schritt besteht nun aus dem Aufbau eines Filter-Graphen, wobei zuerst alle für die Anwendung benötigten und im System registrierten Filter aufgefunden und dann dem Filter-Graph hinzugefügt werden. Für fast alle Anwendungen ist dabei ein Startfilter als Datenquelle (z.B. eine geladene Multimedia-Datei) und ein Endfilter (z.B. ein Video-Renderer für Videodaten) obligatorisch. Nachdem alle Filter zusammengesucht und eingefügt wurden,

[14] https://msdn2.microsoft.com/en-us/library/ms786508.aspx
[15] Dunlop (2000) S. 29f
[16] https://msdn2.microsoft.com/en-us/library/ms786509.aspx
[17] Dunlop (2000), S. 511
[18] Bruns & Neidhold (2003), S. 75
[19] https://msdn2.microsoft.com/en-us/library/ms786509.aspx

müssen noch die Pins aller eingesetzten Filter untereinander korrekt verbunden werden. Hierbei können die Verbindungen entweder manuell ausgeführt werden[20] oder es kann ein Mechanismus des Filter-Graph-Managers zur automatischen Verbindung aller Filter genutzt werden (Intelligent Connect[21]). Intelligent Connect lädt bei Bedarf fehlende Filter nach[22] (z.B. einen Standard-Video-Renderer) und verbindet danach die Pins der vorhandenen Filter, wobei für die Anwendung nicht mehr direkt ersichtlich ist, welche Filter genau Verwendung fanden und welche Pins verbunden wurden.
Der letzte Schritt ist die eigentliche Inbetriebnahme des Filter-Graphen. Die Anwendung kann nun Kontrollwünsche, wie das Starten und Stoppen des Datenstroms im Filter-Graphen, an den Filter-Graph-Manager richten, so wie auf eingehende Ereignisse reagieren.

Zur Visualisierung, Simulation und zum Vorabtest eines Filter-Graphen bietet das Windows SDK ein drag-and-drop-Hilfsprogramm namens GraphEdit[23] an. Hier lassen sich bereits alle unter Windows registrierten Filter inspizieren, sowie mit eingeschränktem Funktionsumfang im Zusammenspiel mit den anderen Filter-Graph-Komponenten testen. Ebenso können auch bereits die Konfigurationsmöglichkeiten eines Filters eingesehen werden.

5.4.2 DirectShow mit Fensterkomponenten

Betrachtet man nun diese generelle Beschreibung im Kontext dieser Arbeit, so fallen einige unerwünschte oder gar hinderliche Aspekte ins Auge, die eine direkte Verwendung der DirectShow-Komponenten zur Lösung der Zielsetzung ausschließen.
Das größte Hindernis ist die starke Einbindung der fensterorientierten Windowsprogrammierung unter DirectShow. So ist die Konfiguration einer Bilddatenquelle ein leichtes, wenn die Möglichkeit besteht, den eingebauten Konfigurationsdialog der Hardware über das entsprechende Interface[24] [25] des dazugehörigen Filters aufzurufen. Weiterhin erzeugt der Standard-Video-Renderer[26] und alle anderen DirectX-basierten Renderer automatisch ein Anzeigefenster, das zwar verborgen werden kann, aber dennoch existiert und damit auch korrekt verwaltet werden muss. Da ein Teil der Zielsetzung, die Plattformunabhängigkeit, die Verwendung Windows-spezifischer Fensterklassen verbietet und somit die exklusive Verwendung von Konsolenanwendungen bzw. mitgelieferten Fensterklassen vorschreibt, ist dieser eingeschlagene Weg somit eine Sackgasse.
Als weiteres Argument muss noch die Verwendung der weiter oben beschriebenen automatischen Verbindungsmechanismen aufgegriffen werden. So bedeutet zwar die Automatisierung einen gewissen Komfort bei der Programmierung von DirectShow-Anwendungen, gleichzeitig geht aber auch ein Großteil der Kontrolle über die internen Prozesse verloren. Da dies ebenfalls nicht im Sinne der Zielsetzung ist, muss eine andere Herangehensweise betrachtet werden, die im Folgenden beschrieben wird.

5.4.3 DirectShow ohne Fensterkomponenten

[20] Bruns & Neidhold (2003), S. 74ff
[21] https://msdn2.microsoft.com/en-us/library/ms786503.aspx
[22] Bruns & Neidhold (2003), S. 78
[23] siehe: http://msdn2.microsoft.com/en-us/library/ms787460.aspx
[24] siehe: https://msdn2.microsoft.com/en-us/library/ms783363.aspx
[25] IAMVfwCaptureDialogs, siehe https://msdn2.microsoft.com/en-us/library/ms784351.aspx
[26] https://msdn2.microsoft.com/en-us/library/ms787917.aspx

Gesucht ist nun eine Möglichkeit, einen Filter-Graphen möglichst manuell und kontrolliert zu erzeugen, wobei ausschließlich Komponenten verwendet werden, die auch ohne Fensterkomponenten auskommen, aber trotzdem die gewünschte Funktionalität für die Lösung der gestellten Ziele bieten können.
Es müssen also die Fragen gestellt werden, welche Bausteine (Filter) aus der Komponentenauswahl gewählt werden müssen und wie diese zu verbinden sind.
Als ersten Schritt zur Lösung wird der Endpunkt des Filter-Graphen, der im Normalfall von einem Renderer eingenommen wird, durch einen Filter namens Null-Renderer[27] ersetzt. Dieser erfüllt die Spezifikationen eines Renderers und ist damit legitimer Endpunkt, unterscheidet sich aber stark im Umgang mit den erhaltenen Daten aus dem Datenfluss. So werden die Bilddaten durch den Null-Renderer einfach verworfen, ohne dass sie angezeigt oder für die Anzeige aufbereitet werden. Dies umgeht die Verwendung eines DirectShow-eigenen Anzeigefensters, bringt aber den Nachteil mit sich, dass anders als bei anderen Renderern kein Bild aus dem Datenstrom angefordert werden kann. Um diese essenzielle Funktionalität zu gewährleisten, muss also nach einer weiteren Komponente gesucht werden.

Der zweite Bestandteil muss nun ein Filter sein, der erlaubt, aus dem Datenstrom Bilder anzufordern, um diese dann später intern in PRAGMA weiterzuverarbeiten. Um dies zu realisieren, wird ein Sample-Grabber[28] in den Filter-Graphen eingefügt. Dieser Filter ändert nichts am Datenfluss in einem Filter-Graphen, jedoch kann jederzeit während des Betriebes dem Datenstrom ein aktuelles Bild aus dem Puffer des Filters entnommen werden[29].

Es fehlt nun noch die eigentliche Datenquelle bzw. der Startpunkt des Filter-Graphen. Um nun eine Repräsentation einer angeschlossenen Hardware aufzufinden, wird eine Aufzählung der passenden, unter Windows registrierten Quellen mittels des System-Device-Enumerators[30] erzeugt, die gewünschte Hardware ausgewählt und als Filter ebenfalls in den Filter-Graphen eingefügt.

Zur kontrollierten Verbindung dieser drei Komponenten kann nun die Funktion IGraphBuilder::Connect des vom Filter-Graph-Managers bereitgestellten Interfaces IGraphBuilder[31] zur paarweisen Verbindung zweier vorher bestimmter Pins genutzt werden, so dass die gewünschte Kontrolle über den Datenflussweg erreicht wird.

Mit diesem Aufbau lassen sich nun bereits Bilder aus dem Datenstrom auslesen und nach Formatumwandlung in PRAGMA weiterverarbeiten. Es fehlt jedoch noch die Möglichkeit der Konfiguration der Einstellungsparameter bei Verzicht auf die Nutzung eingebauter Fensterdialoge. Um auch diese Funktionalität anzubieten, wird auf zwei Schnittstellen zurückgegriffen, die standardgemäß vom Quellenfilter oder dessen Output-Pins angeboten werden.
Die Schnittstelle IAMVideoProcAmp[32] kann am Quellenfilter mittels der von IUnknown[33] vererbten Funktion IUnknown::QueryInterface angefordert werden[34] und bietet dann get- und set-Operationen für eine Anzahl an Parametern (z.B. Helligkeit und Kontrast), die in der Enumeration VideoProcAmpProperty[35] aufgeführt sind. Dabei ist zu beachten, dass nicht alle

[27] https://msdn2.microsoft.com/en-us/library/ms787445.aspx
[28] https://msdn2.microsoft.com/en-us/library/ms787594.aspx
[29] über ISampleGrabber::GetCurrentBuffer
[30] siehe https://msdn2.microsoft.com/en-us/library/ms787871.aspx
[31] siehe http://msdn2.microsoft.com/en-us/library/ms785794.aspx
[32] siehe http://msdn2.microsoft.com/en-us/library/ms784400.aspx
[33] siehe http://msdn2.microsoft.com/en-us/library/ms680509.aspx
[34] Dunlop (2000), S. 30
[35] siehe: http://msdn2.microsoft.com/en-us/library/ms787924.aspx

angebotenen Einstellmöglichkeiten auch von allen Hardwaretypen unterstützt werden. Ebenso ist auch der Einstellbereich eines Parameters von der verwendeten Hardware abhängig, so dass über die Schnittstelle auch ein Wertebereich, eine Standardeinstellung sowie die Schrittgröße der Einstellung abgefragt werden können.
Erfragt man an einem Output-Pin die Schnittstelle IAMStreamConfig[36], so bietet sich hier die Möglichkeit, das Ausgabeformat der Datenquelle abzufragen und neu zu setzen. Darunter fallen Einstellungen zur Bildauflösung, der Framerate (errechnet aus der Anzeigedauer eines Frames) und der Farbkodierung der Bilddaten.

Mit diesen Komponenten kann nun eine programmiertechnische Lösung der Zielsetzung erarbeitet werden.

6 Lösung

Die im nachfolgenden beschriebene programmiertechnische Lösung erfüllt zu Großteilen die Zielsetzung und behält dabei im Auge, dass die zu bearbeitende Klasse Teil eines quelloffenen Projektes sein soll und damit nur frei erhältliche Werkzeuge und SDKs verwendet werden dürfen. Auch ist eine ausreichende Trennung der Funktionalität von der späteren Benutzung gewährleistet, so dass der Nutzer sich weder mit der gewählten Technologie auskennen muss, noch es einen Unterschied machen wird, welches Betriebssystem bei der Arbeit mit der Klasse unter PRAGMA benutzt wird (diese Arbeit stellt die Windows-Komponente dar, während in einer weiteren Arbeit dieselbe Funktionalität für Linux-Plattformen angeboten wird).

6.1 Codebeschreibung

Da der erarbeitete Quellcode dieser Arbeit Teil eines größeren Projekts ist, gelten hinsichtlich des Quelltextes einige Formatierungsrichtlinien. So werden zum Einrücken immer zwei Leerstellen verwendet, wobei niemals achtzig Zeichen pro Codezeile überschritten werden sollen. Dies führt zu eingerückten Parameterlisten bei Funktionsaufrufen, wobei auch hier der Einrückabstand von zwei Leerstellen beachtet wird. Dies bewirkt, dass der Code in allen Texteditoren die gleiche Erscheinungsform hat und sich in der Gestaltung nicht vom Rest der PRAGMA-Bibliothek unterscheidet. Weiterhin soll auf eine sinnvolle Variablenbenennung geachtet werden, die bereits Aufschluss auf die Verwendung der Variablen gibt, wobei durch ein angehängtes Suffix noch der Verwendungsbereich deklariert wird ('G' für global, 'L' für lokal, 'A' für (übergebenes) Attribut etc.).

Bei der Zusammenarbeit mit unbekannter Hardware und auch der COM-Umgebung kann es bedingt durch Hardwarebeschränkungen (Beispiel: die Kamera bietet die Regelung eines Einstellparameters nicht an) oder inkorrekte Funktionsaufrufe (z.B. ungültige übergebene Parameter) häufig zu Fehlern kommen. Alle COM-Funktionen liefern daher als Rückgabewert eine Statusmeldung zurück, die Aufschluss darüber gibt, ob eine Operation erfolgreich abgeschlossen wurde oder ein Fehler aufgetreten ist. Durch eine weitere Auffächerung lässt sich auch die Fehlerquelle in den meisten Fällen unter Zuhilfenahme der Dokumentation

[36] siehe: http://msdn2.microsoft.com/en-us/library/ms784115.aspx

bestimmen. Der generelle Erfolg kann mit den Makros FAILED und SUCCEEDED überprüft werden[37], die speziell auf die Auswertung der Erfolgsmeldung abgestimmt sind.
Im herausgearbeiteten Quellcode muss also besondere Rücksicht darauf genommen werden, ob und wie die aufgerufenen Befehle verarbeitet wurden. So muss auch nach jedem Fehler die saubere Freigabe aller bis zum Auftritt des Fehlers angeforderten Systemressourcen gewährleistet sein. Dies ist insbesondere bei den zahlreichen Funktionsaufrufen in den Konstruktoren eine nicht zu verachtende Problemstellung.

Für den Benutzer der Klassenfunktionen soll der mögliche Erfolg in gebündelter Form ersichtlich sein. Somit erbt die bearbeitete Klasse von der generischen Datenquellenklasse eine Membervariable vom Typ BOOL, die jeweils den Erfolg der zuletzt aufgerufenen Funktion erhält. Mit dem Aufruf der dazugehörigen get-Operation lässt sich so nach jeder Operation feststellen, ob diese fehlerfrei alles Geforderte erfüllt hat oder ein Fehler auftrat. Insbesondere lässt sich auch nach Aufruf des Konstruktors über eben diese Funktion die erfolgreiche Erstellung einer Verbindung zur Kamera nachprüfen.

Um eine durchgehende Definiertheit der Memberfunktionen zu gewährleisten, reicht es aber noch nicht aus, sich auf den Wert der Erfolgs-Variable zu verlassen. Wird beispielsweise nach dem Konstruktoraufruf bei nicht angeschlossener Kamera, wobei die Erfolgsvariable auf pragma::FALSE gesetzt wird, die Funktion zum Ausgeben der Quellenliste in XML aufgerufen, wird diese einwandfrei funktionieren und die Erfolgsvariable daher auf pragma::TRUE setzen. Jede andere Funktion der Klasse kann aber trotz des nun positiven Wertes der Erfolgsvariable nicht erfolgreich aufgerufen werden, da die benötigten Objekte im Konstruktor nicht erzeugt wurden. Daher muss bei jeder Funktion zuerst abgefragt werden, ob alle benötigten Objekte überhaupt im Konstruktor korrekt initialisiert wurden. Dies wird durch eine einfache Abfrage der entsprechenden Zeiger auf den Inhalt "NULL" realisiert und im Fehlerfall jeweils ein Dummy-Wert zurückgegeben. Da diese Abfrage für alle Memberfunktionen zutrifft, wird sie bei den genauen Beschreibungen weiter unten nicht mehr erwähnt.

Bei der Verwendung der COM-Schnittstelle gibt es einige generelle Funktionen, die bei allen Objekten Verwendung finden. So werden alle angeforderten Objekte mit Hilfe der Funktion CoCreateInstance[38] [39] erzeugt. Der so erhaltene Zeiger auf das Objekt repräsentiert eine Schnittstelle des Objekts. Da aber ein Objekt auch eine Vielzahl anderer Schnittstellen anbieten kann, können diese gezielt mit einer von IUnknown vererbten Funktion[40] abgefragt und angefordert werden. Soll ein Objekt wieder freigegeben werden, erfolgt dies ebenfalls mit einer von IUnknown vererbten Funktion namens Release[41].

Um die Kamera im laufenden Betrieb zu verwalten, werden für diese spezifische Lösung einige globale Zeigervariablen der Klasse als Membervariablen hinzugefügt. Diese werden im Verlauf des Konstruktoraufrufs mit Zeigern auf angeforderte Schnittstellen bzw. COM-Objekte belegt und sind für alle Klassenfunktionen von essenzieller Bedeutung. Auf die Speicherung sonstiger Kameradaten wurde dahingegen verzichtet, insbesondere um Dateninkonsistenz zwischen Variableninhalt und tatsächlichem, in der Kamera gespeicherten, Einstellungswert zu vermeiden. So werden Einstellungsparameter, wie die Helligkeit oder der Kontrast, nicht einfach aus einer Variablen ausgelesen, sondern bei jeder Anfrage erst über die zugehörige Schnittstelle aus der Hardware ausgelesen. Der klare Zeitvorteil der

[37] Dunlop (2000), S. 513, Anm.: das Erfolgsmakro hieß in dieser alten Version noch SUCCESS
[38] siehe http://msdn2.microsoft.com/en-us/library/ms686615.aspx
[39] vergleiche Anhang A I
[40] IUnknown::QueryInterface, siehe http://msdn2.microsoft.com/en-us/library/ms680509.aspx
[41] IUnknown::Release, siehe http://msdn2.microsoft.com/en-us/library/ms680509.aspx

Parametervariablen wird also vernachlässigt für eine geringe Menge lösungsspezifischer globaler Variablen.

Um die später angeforderten Bilder aus der Kamera richtig auslesen zu können, bedarf es weiterhin zweier Variablen, in denen zum einen die Farbtiefe, also die Anzahl Bits pro Bildpunkt, gespeichert wird, zum anderen aber auch der Farbraum, in dem das Bild formatiert ist, da hierüber die Anordnung der Bildinformationen festgestellt werden kann.

6.1.1 Konstruktor/Destruktor

Im Sinne der Aufgabenstellung muss der Konstruktor nicht nur die Aufgabe übernehmen, alle Schritte vorzubereiten, die zum Erhalt von Bildern aus der Datenquelle notwendig sind, sondern auch alle benötigten Initialisierungen des COM-Subsystems und die Erstellung der COM-Objekte übernehmen.

Weiterhin muss der Konstruktor auch die Aufgabe erfüllen, bei einem auftretenden Fehler während der Aufbauphase des Filter-Graphen alle bisher angeforderten Ressourcen korrekt freizugeben, um Speicherlecks und Komplikationen bei einem erneuten Zugriff auf nicht korrekt freigegebene Ressourcen zu vermeiden.

Der Destruktor hingegen muss alle vorher erwähnten Schritte korrekt rückgängig machen, so dass alle angeforderten Objekte und Systemressourcen wieder freigegeben werden.

Mit diesen getroffenen Voraussetzungen kann nun eine konkrete Realisierung angegangen werden.

6.1.1.1 Standardkonstruktor

Der Standardkonstruktor soll alle oben erwähnten Aufgaben erfüllen mit der Besonderheit, dass als Datenquelle die erstbeste Videohardware gewählt werden soll, ohne das eine Auswahlmöglichkeit zwischen verschiedenen vorhandenen Quellen besteht.

Der erste Schritt innerhalb des Standardkonstruktors ist wie auch bei allen nachfolgenden Funktionen das Setzen der Erfolgsmitteilung auf den booleschen Wert FALSE. Um später die Initialisierung der globalen Zeigervariablen zu überprüfen, werden diese im Anschluss mit einer Hilfsfunktion alle auf den Wert NULL gesetzt.

Vor der Verwendung der COM-Komponenten muss zuerst das COM-Subsystem initialisiert werden. Als nächsten Schritt werden Instanzen aller benötigten COM-Objekte mittels der Funktion CoCreateInstance erzeugt, wobei als Parameter jeweils ein Klassen-Identifizierer (CLSID[42]) des gewünschten Objektes übergeben wird. Gleichzeitig wird als Ausgabe-Parameter die jeweils zugehörige globale Variable übergeben, die danach einen Zeiger auf das Objekt erhält.
Liegen nun Repräsentationen des Filter-Graph-Managers, des Sample-Grabbers und des Null-Renderers vor, wird im Anschluss eine Schnittstelle des Filter-Graph-Managers erfragt, mit der später der Dateifluss im aufzubauenden Filter-Graphen kontrolliert werden kann[43].

[42] CLSID – Class Identifier, Subtyp der unter COM gebräuchlichen GUID – Global Unique Identifier
[43] IMediaControl, siehe http://msdn2.microsoft.com/en-us/library/ms785872.aspx

Nun wird der noch fehlende Quellen-Filter aufgesucht, indem zuerst eine Instanz des System-Device-Enumerators erzeugt wird. Mit dieser kann nun eine Aufzählung einer Unterkategorie der Systemressourcen erzeugt werden, in diesem Falle wird nach Video-Eingabe-Geräten gesucht[44]. Da der Standardkonstruktor das erstbeste Gerät verwenden soll, reicht es nun, das erste Element der Aufzählung zu wählen. Es liegt zu diesem Zeitpunkt noch kein Filter-Objekt der Videoquelle vor, daher muss nun über einen eindeutigen Namen[45] das dazugehörige Filter-Objekt angefordert werden. Da die Aufzählung schon Zugriff auf die passende Schnittstelle[46] bietet, kann über die Funktion IMoniker::BindToObject ein Zeiger auf das zugehörige Filter-Objekt angefordert werden.

Nachdem alle Filter erzeugt bzw. aufgesucht wurden, werden sie im nächsten Schritt dem Filter-Graphen hinzugefügt. Die Komponenten sind zu diesem Zeitpunkt noch unverbunden, somit gilt es als nächstes, die passenden Pins der Filter zu verbinden. Hierzu wird eine Hilfsfunktion von DirectShow benutzt, die den ersten freien Pin in der gewünschten Flussrichtung liefert. Wurden jeweils paarweise die entsprechenden Output- und Input-Pins gefunden, werden diese anschließend verbunden. Generell werden diese Pins im weiteren Verlauf nicht mehr benötigt, die einzige Ausnahme stellt jedoch der Output-Pin des Quellenfilters dar, da hier später Konfigurationen des Ausgabeformats vorgenommen werden. Dieser Pin wird daher in einer eigenen globalen Variablen vermerkt. Zu diesem Zeitpunkt kann auch bereits durch Aufruf einer Hilfsfunktion das Ausgabeformat für den späteren Gebrauch vermerkt werden.

Als letzter Arbeitsschritt wird die Schnittstelle des Sample-Grabber-Filters[47] angefordert, um diesen danach zu konfigurieren. Standardgemäß wird der Datenfluss nach Entnahme eines Bildes gestoppt, von daher muss per Funktion ISampleGrabber::SetOneShot das Stoppen des Datenflusses unterbunden werden. Über ISampleGrabber::SetBufferSamples wird das Puffern der durchlaufenden Bilder veranlasst.

Der Filter-Graph ist nun korrekt aufgebaut und verbunden, alle notwendigen Konfigurationen wurden getroffen, so dass der Konstruktor an dieser Stelle mit dem Setzen der Erfolgsmitteilung verlassen werden kann.

6.1.1.2 Auswahlkonstruktor

Neben dem Standardkonstruktor soll auch die Möglichkeit bestehen, gezielt eine Bildquelle zu wählen, daher muss es einen zweiten Weg geben, ein Objekt dieser Klasse zu erzeugen. Die Lösung ist ein zweiter Konstruktor, der als Parameter den eindeutigen Namen der Bildquelle überwiesen bekommt und nach dieser Quelle suchen wird.

Der Auswahlkonstruktor ist zu großen Teilen identisch mit dem Standardkonstruktor. Die abweichenden Codezeilen finden sich in der Suche nach dem Quellenfilter. So reicht es nicht mehr, das erste Objekt der Geräteaufzählung zu wählen, sondern es muss bei allen Elementen der Aufzählung geprüft werden, ob das aktuell betrachtete Element das Gesuchte ist.

[44] der dazugehörige Identifier ist CLSID_VideoInputDeviceCategory
[45] DirectShow verwendet die Terminologie *Moniker* (engl., deutsch: *Name, Spitzname*)
[46] IMoniker, siehe: http://msdn2.microsoft.com/en-us/library/ms679705.aspx
[47] ISampleGrabber, siehe: http://msdn2.microsoft.com/en-us/library/ms786690.aspx

Hierzu wird eine while-Schleife benutzt, die in zwei Fällen terminiert:

1. das Ende der Aufzählung wurde erreicht (*notFound* wird FALSE) 2. das gesuchte Gerät wurde aufgefunden (*reachedEnd* wird TRUE)

Diese beiden Bedingungen sind mit einem logischen UND miteinander verknüpft, wobei die Abbruchbedingung folgendermaßen formuliert ist:

```
while(!reachedEnd && notFound)
```

Um nun die passende Quelle zu finden, wird von jedem Element der sogenannte "friendly name"-String angefordert, der eine für Menschen lesbare Beschreibung des Gerätetyps enthält. Dieser wird nach Umrechnung aus dem Unicode-Format in eine C++-Zeichenkette mit dem übergebenen Parameter verglichen. Liegt ein Treffer vor, wird notFound auf FALSE gesetzt, ansonsten wird zum nächsten Element übergegangen, bei gleichzeitigem Test, ob noch Elemente vorhanden sind (sonst wird reachedEnd auf TRUE gesetzt und die Schleife terminiert).
Nach der Terminierung der Schleife muss nun noch abgefragt werden, ob ein tatsächlicher Treffer vorliegt oder die Suche erfolglos abgebrochen wurde. Liegt ein Treffer vor, wird weiter verfahren wie im Standardkonstruktor, ansonsten endet an dieser Stelle der Konstruktor ohne Erfolg.

6.1.1.3 Destruktor

Der Destruktor ist durch die Verwendung von Hilfsfunktionen sehr kurz gehalten und arbeitet folgende Schritte ab: Zuerst wird überprüft, ob im Filter-Graph ein Datenstrom fließt. Ist dies der Fall, muss dieser zuerst angehalten werden, bevor die weiteren Operationen ausgeführt werden.
Ist der Filter-Graph im Ruhezustand oder gestoppt worden, können im Anschluss alle angeforderten Objekte freigegeben werden. Dies wird mit der von IUnknown vererbten Funktion IUnknown::Release realisiert. Diese Funktion übernimmt gleichzeitig die Referenzenzählung, so dass alle Objekte nach Freigabe automatisch gelöscht werden[48].
Im Anschluss können alle globalen Zeiger auf NULL zurückgesetzt werden.
Als Abschluss beendet der Destruktor die Verwendung des COM-Subsystems.

6.1.2 Leseoperationen

Nachfolgend werden alle Funktionen beschrieben, die in irgendeiner Form Daten auslesen. Hierunter fallen alle Funktionen, die Kameraparameter auslesen, wie auch die Funktion, die das aktuelle Bild anfordert. An Stellen, in denen sich die Umsetzung verschiedener Funktionen nur in wenigen Punkten unterscheidet, werden diese gemeinsam beschrieben.

[48] vergleiche Dunlop (2000), S. 29

6.1.2.1 scanForDevices

Um die eindeutigen Namen der angeschlossenen Gerätetypen zu kennen und im Auswahlkonstruktor zur gezielten Ansteuerung zu verwenden, muss es eine Lese-Operation geben, die eine Liste aller Geräte für den Benutzer zugänglich macht. Da PRAGMA eine eigene Klasse zur Verwaltung von XML-Dokumenten besitzt, liegt es nahe, die Geräteliste als XML-Datei zu erzeugen und auf der Festplatte abzulegen[49].

Nachdem eine interne Repräsentation eines XML-Dokuments mit dem entsprechenden Konstruktor erzeugt wurde, kann nun nach allen passenden Geräten gesucht werden. Hierbei wird genau so vorgegangen, wie bei der Suche nach dem passenden Gerät im Auswahlkonstruktor. Über den System-Device-Enumerator wird also eine Aufzählung der passenden Gerätetypen angefordert, diese dann sequenziell durchlaufen und die erhaltenen Namen nach einer Umwandlung in ein passendes Stringformat als Wert eines neuen Knotens innerhalb des XML-Dokuments abgelegt[50]. Wenn das Ende der Aufzählung erreicht ist, können alle lokal angeforderten Ressourcen wieder freigegeben werden, so dass nach dem Schreiben der XML-Datei auf die Festplatte[51] die Funktion mit einer Erfolgsmeldung verlassen werden kann.

6.1.2.2 type

Diese Funktion gibt bei Aufruf die Art der Datenquelle aus[52] und lag bereits bei Beginn dieser Arbeit fertig vor.

6.1.2.3 imagePointer

Die Funktion *imagePointer* stellt die wichtigste Leseoperation der Klasse dar. Hiermit wird ein Bild von der Kamera angefordert und als interne PRAGMA-Repräsentation (ImagePointer) zurückgegeben. Es wird also nicht nur die Funktionalität des Bildholens von dieser Funktion gefordert, sondern auch die Umwandlung des Formats das die Kamera liefert in die entsprechende Darstellung eines Bildes unter PRAGMA.

Zuallererst muss in der Funktion überprüft werden, ob bereits ein Datenfluss im Filter-Graphen vorliegt[53]. Ohne diesen können keine neuen Bilddaten angefordert werden, daher wird mittels der Schnittstelle IMediaControl[54] zuerst der aktuelle Zustand des Filter-Graphen überprüft[55] und dieser bei Bedarf in den Zustand "Running" geschaltet[56]. Hierbei ist es sehr wichtig, dass der Filter-Graph erst wieder angesprochen werden darf, wenn die Zustandsänderung vollkommen abgeschlossen ist, da es sonst zu Laufzeitfehlern kommt.

[49] vergleiche Anhang A II
[50] mittels XMLDocPointer::newStringNode
[51] mittels XMLDocPointer::write – erhält als Argument den übergebenen Dateipfad/-namen
[52] in diesem Fall pragma::DataSource::DATA_SOURCE_WEB_CAM
[53] vergleiche Anhang A III
[54] http://msdn2.microsoft.com/en-us/library/ms785872.aspx
[55] mit IMediaControl::GetState
[56] mit ImediaControl::Run

Ist nun der Filter-Graph im gewünschten Zustand, kann ein Bild am Sample-Grabber angefordert werden[57]. Dazu muss die Funktion ISampleGrabber::GetCurrentBuffer zweimal aufgerufen werden. Beim ersten Aufruf erhält man die Größe der zu erwartenden Bilddaten in Byte. Vor einem erneuten Aufruf wird mit dieser Information ein entsprechend großer Puffer angelegt, um eine Referenz auf eben diesen als Argument des zweiten Aufrufs zu übergeben. Der Puffer erhält jetzt die Bilddaten in Form des Datenteils einer Device-Independant-Bitmap. In welchem Format diese Daten vorliegen, kann am Mediaformat des Output-Pins des Quellfilters oder per ISampleGrabber::GetConnectedMediaType abgelesen werden[58].

Der erste Schritt, der Erhalt von Bilddaten, ist hiermit abgeschlossen. Um jetzt die Bilddaten in das PRAGMA-Format zu bringen, ist das Wissen von Nöten, in welcher Kodierung die Bilddaten vorliegen. Ist es eine der unkomprimierten RGB-Unterarten, so können die Bildpunkte sequenziell durchlaufen und an einen Image-Pointer überwiesen werden. Liegt jedoch ein Kompressionsformat vor[59], so müssen die Bilddaten korrekt interpretiert und umgewandelt werden.

Um eine Entscheidung zu treffen, welche Methode verwendet wird, um den Bildpuffer auszulesen, greift die Funktion auf die vermerkten Werte des Ausgabeformats zurück. So kann aus dem aktuell verwendeten Farbraum auf die Anordnung der Bildwerte im Puffer geschlossen werden (sequenziell bei RGB, (meist) nach Kanälen getrennt bei YUV). Ist die Anordnung bekannt, fehlt noch das Wissen, wie viele Bits für einen Bildpunkt vorgesehen sind. Diese Information lässt sich aus der vermerkten Farbtiefe gewinnen, wobei noch einmal erwähnt werden muss, dass eine Anzahl von n Bits für einen Bildpunkt nicht die Aussage impliziert, dass diese n Bits auch nacheinander im Bildpuffer liegen. Hierzu ist gerade bei YUV-Komprimierungen eine genaue Studie der Anordnungsdefinition vonnöten.

Wie im Quelltext ersichtlich, wird vor der Zuweisung an den PRAGMA-internen Image-Pointer per zweifacher if-Abfrage eine passende Methode ausgewählt[60].

Im geschriebenen Quellcode wird eine beispielhafte Umwandlung von Bildwerten im YUV-Farbraum in den RGB-Farbraum vorgenommen. Hierzu werden für jeden Bildpunkt die drei Farbkomponenten Y, U und V aus dem Puffer ausgelesen. Danach erfolgt die Umrechnung jedes Bildpunkts in die drei Farbkanäle des RGB-Farbraums, wobei beachtet werden muss, ob die U- und V-Komponenten vorzeichenlos angegeben sind (bspw. ist bei einer Komponentengröße von 1 Byte der Wertebereich nicht [0,255], sondern [-128,127]). Nach der Umrechnung kann mit Minimum- und Maximumfunktionen noch festgestellt werden, ob die errechneten Werte aufgrund von Rundungsfehlern den festgelegten Wertebereich überschreiten (im Beispiel ist dies [0,255]) und können dann jeweils am aktuellen Pixel vermerkt werden.

Der Quellcode beinhaltet auch eine beispielhafte Verwertung von RGB-Bilddaten, wobei hier selbstverständlich die Umwandlung in einen anderen Farbraum entfällt. Beachtet werden muss jedoch, dass bei RGB-Bildern mit 32 Bit Farbtiefe nicht die ganze Bitanzahl auf die drei Farbkanäle entfällt, sondern für gewöhnlich 8 Bit davon entweder unbenutzt oder für die Darstellung von Transparenzeigenschaften genutzt werden.

[57] vergleiche Anhang A IV
[58] http://msdn2.microsoft.com/en-us/library/ms786688.aspx
[59] im Fall dieser Kamera die YUV-Unterart I420
[60] vergleiche Anhang A V

Nach der Umwandlung in eine PRAGMA-Repräsentation der Bilddaten muss abschließen der angeforderte Speicherbereich des Puffers wieder freigegeben werden, um dann mit einer Erfolgsmeldung den PRAGMA-Zeiger zurückgeben zu können.

6.1.2.4 brightness / contrast

Die Funktionen zum Auslesen der Helligkeit und des Kontrasts ähneln sich stark, da sie sowohl dieselbe Schnittstelle, als auch dieselbe Umrechnung verwenden. Ziel ist es, den aktuellen Wert des gewünschten Parameters auszulesen und von dem hardwareabhängigen Intervall der Einstellungsbreite auf ein genormtes Fließkommaintervall [0.0,1.0] abzubilden.

Zuerst wird am Quellfilter auf die übliche Art die passende Schnittstelle[61] angefordert und mit der get-Funktion der entsprechende[62] aktuelle Parameterwert angefordert. Da dieser als schlichter Integer vorliegt, werden zur Umrechnung auch noch die hardwareseitigen Intervallgrenzen benötigt. Minimum und Maximum des Parameters lassen sich mit der Funktion IAMVideoProcAmp::GetRange erfragen, so dass nun mit nachfolgender Rechnung das Ergebnis bestimmt werden kann.

$$Wert_{normiert} = \frac{Wert_{hardware} - Wert_{\min}}{Wert_{\max} - Wert_{\min}}$$

Dieser errechnete Wert wird nun zusammen mit einer Erfolgsmitteilung zurückgegeben.

6.1.2.5 autoWhiteBalance

Ausgabe dieser Funktion soll sein, ob der Weißabgleich automatisch durch die Kamera erfolgt oder der Parameter manuell einzustellen ist. Das Verfahren ist zu großen Teilen identisch mit der Abfrage von Helligkeit und Kontrast, unterscheidet sich aber darin, dass als Rückgabe nur ein boolescher Wert gefragt ist.

Diesmal wird nach der Anforderung der passenden Schnittstelle (s.o.) das Augenmerk nicht auf den aktuellen Einstellungswert, sondern auf das gleich mitgelieferte Flag, das den Zustand der Regelungsart angibt, gerichtet. Verglichen wird der Zustand nun mit einem Element der Enumeration "VideoProcAmpFlags"[63] und dann ein entsprechender boolescher Wert zusammen mit der Erfolgsmitteilung zurückgeliefert.

6.1.2.6 xResolution / yResolution

Die Auflösung in Höhe und Breite hängt fest an dem Ausgabeformat des gewählten Output-Pins, so dass für das Auslesen dieser beiden Parameter nicht wie bisher üblich eine Schnittstelle des Quellfilters, sondern eine des Outputpins angefordert werden muss.

[61] IAMVideoProcAmp, siehe http://msdn2.microsoft.com/en-us/library/ms784400.aspx
[62] wählbar durch die Verwendung einer der Optionen aus der Enumeration VideoProcAmpProperty
[63] in diesem Falle die Option "VideoProcAmp_Flags_Auto" für automatische Regulierung des Weißabgleichs

Ist die passende Schnittstelle[64] erfragt, wird über die Funktion IAMStreamConfig::GetFormat das aktuelle Ausgabeformat abgefragt und in einem Zeiger vermerkt. Das so erhaltene Format ist in der Form einer Struktur des Typs AM_MEDIA_FORMAT[65]. Um aus dieser Struktur den Videoinfoheader[66] auslesen zu können, muss vor Dereferenzierung des entsprechenden Zeigers noch die Gültigkeit der Formatinformationen geprüft werden[67]. Ist passend auf die richtige Struktur gecastet worden, wird aus der erhaltenen Struktur eine weitere herausgesucht (diesmal die Struktur Bitmapinfoheader[68]), die nun das Auslesen des gewünschten Parameters ermöglicht[69] [70]. Nach erfolgreichem Auslesen kann die Funktion den erhaltenen Wert zurückgeben.

6.1.2.7 frameRate

Die Framerate (*engl., dt.:* Bildwiederholrate) kann unter DirectShow nicht direkt abgefragt werden, sondern muss aus der durchschnittlichen Anzeigedauer eines Frames errechnet werden.

Hierzu wird der gleiche Weg eingeschlagen wie bei der zuvor beschriebenen Funktion zur Ausgabe der Auflösung, bis eine Referenz auf den Videoinfoheader vorhanden ist. Jetzt kann der Membervariablen AvgTimePerFrame die Framedauer entnommen werden. Die Umrechnung in die Framerate erfolgt nach der folgenden Formel, wobei zu beachten gilt, dass der erhaltene Wert der Framedauer die Einheit 100 Nanosekunden hat.

$$Framerate = \frac{1 * \frac{10^9 ns}{1s} * \frac{100ns}{100ns}}{AvgTimePerFrame} = \frac{10^7 * 100ns}{AvgTimePerFrame} Hz$$

Das erhaltene Ergebnis kann dann nach einer Umwandlung zum Integer zusammen mit dem Setzen der Erfolgsmeldung zurückgegeben werden.

6.1.2.8 colorImages

Die letzte Leseoperation soll Auskunft darüber geben, ob die benutzte Kamera zurzeit ein Farbbild oder ein Grauwertbild liefert.

Es wird dazu wie beim Auslesen von Helligkeit und Kontrast die Schnittstelle IAMVideoProcAmp angefordert und per get-Funktion der Wert des Parameters "VideoProcAmp_ColorEnable" ausgelesen. Mit einer einfachen if-Abfrage wird nun der erhaltene Wert auf den booleschen Wertebereich abgebildet, wobei ein Wert ungleich null der Option "Farbbild" entspricht. Die Funktion gibt den booleschen Wert dann unter Setzen der Erfolgsmitteilung zurück.

---

[64] IAMStreamConfig, siehe http://msdn2.microsoft.com/en-us/library/ms784115.aspx
[65] siehe http://msdn2.microsoft.com/en-us/library/ms925337.aspx
[66] siehe http://msdn2.microsoft.com/en-us/library/ms787915.aspx
[67] vgl. Anmerkungen unter http://msdn2.microsoft.com/en-us/library/ms925337.aspx
[68] siehe http://msdn2.microsoft.com/en-us/library/ms779712.aspx
[69] entweder biWidth oder biHeight für die X- bzw. Y-Auflösung
[70] vergleiche Anhang A VI

6.1.3 Schreiboperationen

Wie auch bei den Leseoperationen wurden inhaltlich fast identische Funktionen zu einem Kapitel zusammengefasst. Nachfolgend also alle Operationen, die das Setzen bestimmter Parameter erlauben.

6.1.3.1 brightness / contrast

Zum Setzen der Helligkeit oder des Kontrasts wird wie auch beim Auslesen der Parameter auf die Schnittstelle IAMVideoProcAmp[71] zurückgegriffen. Der Übergabeparameter der Funktion sieht einen Wertebereich von [0.0,1.0] in Fließkomma-Zahlen vor, so dass die gewünschte Einstellung nicht direkt in die set-Funktion eingesetzt werden kann. Es wird ein weiteres Mal das Minimum und Maximum des hardwarespezifischen Einstellungsbereichs angefordert und mit folgender Formel eine Umrechnung ausgeführt.

$$Wert_{hardware} = (Wert_{\max} - Wert_{\min}) * Wert_{normiert} + Wert_{\min}$$

Nach dieser Umrechnung und einer Umwandlung zurück in einen Ganzzahlenwert, kann die Funktion IAMVideoProcAmp::set mit dem umgerechneten Wert aufgerufen werden und bei erfolgreicher Statusmeldung die Funktion mit einer Erfolgsmitteilung verlassen werden.

6.1.3.2 autoWhiteBalance

Die Funktion autoWhiteBalance wird die Regulierungsart des Weißabgleichs festlegen. Als Übergabe dient ein boolescher Wert, der den Zustand der automatischen Regulierung repräsentiert. Wie bereits bekannt, läuft die Einstellung über die IAMVideoProcAmp-Schnittstelle, wobei innerhalb dieser Funktion beim Setzen der Einstellung als Besonderheit nicht der Wert, sondern ausschließlich das Flag abgeändert wird, so dass nach Anforderung der Schnittstelle zuerst der aktuelle Einstellungswert ausgelesen und für den späteren Gebrauch vermerkt wird. Danach muss der boolesche Wert in das passende Element der Flag-Aufzählung[72] umgesetzt werden. Mit dem Erfolg eines anschließenden IAMVideoProcAmp::set, bei dem das gewünschte Flag und der vermerkte aktuelle Wert übergeben werden, kann die Funktion verlassen werden.

6.1.3.3 xResolution / yResolution

Im Gegensatz zu den Einstellungen am Filter direkt, wird die Auflösung wie weiter oben beschrieben durch das aktuelle Ausgabeformat des gewählten Output-Pins bestimmt. Dieses kann nicht verändert werden, so lange sich der Filter-Graph nicht im Ruhezustand befindet[73]. Von daher muss vor jedem Änderungsversuch der Filter-Graph über die Kontrollschnittstelle gestoppt werden, sofern er gerade in Betrieb oder pausiert ist.

[71] siehe http://msdn2.microsoft.com/en-us/library/ms784400.aspx
[72] siehe http://msdn2.microsoft.com/en-us/library/ms787923.aspx
[73] http://msdn2.microsoft.com/en-us/library/ms784116.aspx

Jetzt kann am Output-Pin die passende Schnittstelle angefordert und das aktuelle Ausgabeformat ausgelesen und vermerkt werden. Um nun die Auflösungsänderung durchzuführen, wird das soeben erhaltene Format wie gewünscht abgeändert. Hierbei gilt es zu beachten, dass nur gültige Formate wieder an den Output-Pin zurückgereicht werden können, was das Problem aufwirft, dass die X- und Y-Auflösung nicht separat geändert werden können, ohne entsprechend auch die dazugehörige andere Dimension zu verändern. Ist als Beispiel die aktuelle Auflösung 320x240 und soll nun die X-Auflösung auf 640 umgestellt werden, reicht es nicht, innerhalb des Formats die X-Auflösung auf 640 zu setzen, da sich so das Format 640x240 ergibt, was zwar zwei legitimen Auflösungen nahe liegt (640x480 und 320x240), aber selbst nicht legitim ist.

Um dieses Problem zu lösen, wird nun einfach aus der zu setzenden Auflösung die entsprechende andere Dimension errechnet, wobei hier ein Bildverhältnis von 4:3 angenommen wird.

$$Auflösung_X = \frac{4}{3} * Auflösung_Y = Auflösung_Y + \frac{Auflösung_Y}{3}$$
$$Auflösung_Y = \frac{3}{4} * Auflösung_X = Auflösung_X - \frac{Auflösung_X}{4}$$

Ist die gültige Auflösung korrekt berechnet, werden diese beiden Informationen im Bitmapinfoheader (siehe Kapitel 6.1.2.6) gesetzt und danach das gewünschte neue Format an den Output-Pin zurück überwiesen. Der Filter-Graph muss nicht mehr in den ursprünglichen Betriebszustand zurückgesetzt werden, da bereits in der Bildauslesefunktion eine entsprechende Abfrage stattfindet. Somit kann die Funktion an dieser Stelle mit dem Setzen des Erfolgs-Flag verlassen werden.

6.1.3.4 frameRate

Ziel dieser Funktion soll es sein, die Bildwiederholrate auf den gewünschten übergebenen Wert einzustellen. Hierzu muss erneut das Ausgabeformat des Output-Pins abgeändert werden, was nur im Ruhezustand des Filter-Graphen möglich ist. Zu Beginn der Funktion muss also der Filter-Graph wenn nötig angehalten werden. Danach kann am aktuellen Output-Pin die zuständige Schnittstelle[74] angefordert werden. Ist das zurzeit eingestellte Ausgabeformat ausgelesen, kann die Änderung der Bildwiederholrate angegangen werden. Hierzu wird aus der Format-Struktur der Videoinfoheader ausgelesen. In dieser Struktur befindet sich das Element AvgTimePerFrame, das nun passend abgeändert werden muss, um die Bildwiederholrate auf den gewünschten Wert zu setzen. Hierzu muss wiederum beachtet werden, dass die Dauer eines Frames in der Einheit 100 Nanosekunden anzugeben ist, womit sich folgende Umrechnung ergibt:

$$AvgTimePerFrame = \frac{1 * \frac{10^9\, ns}{1s} * \frac{100ns}{100ns}}{Framerate} = \frac{10^7\, Hz}{Framerate} * 100ns$$

Ist die Framedauer korrekt berechnet und im Videoinfoheader vermerkt worden, kann abschließend das veränderte Ausgabeformat an den Pin zurückgegeben werden.

[74] IAMStreamConfig, siehe http://msdn2.microsoft.com/en-us/library/ms784115.aspx

Der Pin wird nun automatisch ein gültiges Ausgabeformat wählen, dass dem angegebenen so nahe wie möglich ist, da es durch Rundungsfehler bei der Umrechnung immer zu kleinen Abweichungen von der tatsächlichen Framedauer kommen kann. Die Funktion kann nun mit einer Erfolgsmeldung verlassen werden.

6.1.3.5 colorImages

Ergebnis dieser Funktion soll die Einstellung des Betriebsmodus der Kamera auf Farbbild- oder Graubildlieferung sein. Von daher muss das übergebene boolesche Argument passend in die Darstellung der Schnittstelle umgesetzt werden. Wie auch beim Auslesen dieses Parameters wird also zuerst die Schnittstelle IAMVideoProcAmp des Quellfilters angefordert. Da der verlangte Parameter nur zwei verschiedene Werte annehmen kann, diese aber nicht zwingend identisch sein müssen mit den Zahlwerten 0 und 1, wird über IAMVideoProcAmp::GetRange das Minimum und Maximum des Einstellbereichs angefordert. Das Minimum entspricht hierbei dem passenden Wert zu "Graubildwiedergabe", das Maximum dementsprechend dem der Farbbildwiedergabe. Mit einer einfachen if-Abfrage wird nun der übergebene boolesche Wert auf diese zwei Werte umgewandelt und per set-Funktion wieder an die Hardware gereicht. Ist das Flag für Erfolg gesetzt, kann die Funktion verlassen werden.

6.1.4 Hilfsfunktionen

Da die oben beschriebenen Klassenfunktionen meist einen kompakten Umfang haben, wird nur sehr wenig Funktionalität in Hilfsfunktionen ausgelagert. Die verwendeten Hilfsfunktionen entstammen dabei teilweise der MSDN und dienen zumeist dem Aufräumen angeforderter Systemressourcen. Alle Hilfsfunktionen werden nachfolgend kurz in ihrer Funktionsweise erklärt:

DeleteMediaType (MSDN[75])
Diese Funktion bekommt den Zeiger auf eine AM_MEDIA_TYPE Struktur überwiesen und gibt den davon belegten Speicherbereich unter Aufruf der nachfolgenden Funktion korrekt frei.

FreeMediaType (MSDN[76])
Diese Funktion wird von *DeleteMediaType* aufgerufen und gibt den durch einen früheren Aufruf einer COM-Funktion allokierten Speicher wieder frei.

cleanUpReferences
Die Hilfsfunktion testet jede globale Zeigervariable, ob diese belegt ist und gibt bei Belegung das vom Zeiger vermerkte Objekt frei.

setGPointersNull
Mit dieser Hilfsfunktion werden alle globalen Zeigervariablen auf NULL gesetzt. Ein Aufruf sollte sowohl vor der ersten Zuweisung an eine der Zeigervariablen (also im Konstruktor)

[75] http://msdn2.microsoft.com/en-us/library/ms783299.aspx
[76] http://msdn2.microsoft.com/en-us/library/ms783692.aspx

geschehen, als auch nach der Freigabe aller angeforderten Objekte (im Destruktor nach dem Aufruf von cleanUpReferences).

GetUnconnectedPin (MSDN[77])
Diese Funktion bekommt einen Zeiger auf einen beliebigen Filter, sowie eine gewünschte Datenflussrichtung übergeben und speichert in den ebenfalls übergebenen Zielzeiger den ersten freien Pin des Filters, bei dem die angegebene Flussrichtung passt und der zudem aktuell nicht verbunden ist. Als Rückgabewert liefert die Funktion wie alle anderen COM-Operationen eine Statusmeldung im HRESULT-Format.

saveOutputFormat
Diese Hilfsfunktion liest am übergebenen Pin das aktuelle Ausgabeformat aus und verarbeitet die Informationen, so dass sie passend in der Image-Funktion benutzt werden können, um den Bilddatenpuffer korrekt zu interpretieren. Hierbei sind das verwendete Farbmodell und die Farbtiefe von Bedeutung. Auf das Farbmodell kann über den Media-Subtyp des Ausgabeformats geschlossen werden. Hierzu wird der vorliegende Subtyp mit diversen Subtyparten verglichen und bei einem Treffer ein passendes Element aus der Farbraum-Enumeration an die globale Variable übergeben[78]. In der vorliegenden Implementierung sind alle typischen Subtypen des RGB- und YUV-Farbraums vertreten[79], es gibt aber noch weitere, teilweise undokumentierte Subtypen.
Die Farbtiefe selbst kann am Bitmapinfoheader auf demselben Weg wie auch die horizontale und vertikale Bildauflösung erfragt werden. Ist auch diese Information in der passenden globalen Variable abgelegt, kann die Funktion verlassen werden.

6.2 Kamerabeschreibung

Im Zuge dieser Arbeit wurden alle Programmierergebnisse mit einer Kamera der Firma Philips[80] getestet. Da die verwendete Hardware immer auch eine Beschränkung des angebotenen Funktionsumfanges einer generischen Schnittstelle bedeutet, folgen hier die Spezifikationen der Kamera, um im späteren Verlauf nicht lösbare Probleme im Zusammenspiel mit diesem Kameratypen aufzuzeigen.

Die verwendete Kamera verfügt über einen 1/3" VGA CCD Sensor. Die maximale Video-Auflösung entspricht daher auch dem VGA-Standard von 640x480 Bildpunkten bei einer maximalen Framerate von 60 Bildern pro Sekunde. Es wird auch ein Einzelbildmodus angeboten mit einer maximalen Auflösung von 1280x960 Bildpunkten, wobei aufgrund des verwendeten Sensors von Interpolation ausgegangen werden darf.
Die Kamera wird über USB an das PC-System angeschlossen und lässt sich unter Windows XP ohne Installation direkt verwenden (plug&play). Für die Betriebssysteme Windows 98SE, 2000 und Me ist die vor Benutzung zu verwendende, beigefügte Treiber-CD von Nöten, die auch zusätzliche Dienstprogramme enthält.
Ein mehrseitiger fensterbasierter Konfigurationsdialog wird kameraseitig angeboten. Dieser bietet manuelle Kontrolle über typische Bildparameter, sowie die automatisierte Übernahme der Parameterregelung.

[77] http://msdn2.microsoft.com/en-us/library/ms783680.aspx
[78] vergleiche Anhang A VII
[79] siehe http://msdn2.microsoft.com/en-us/library/ms787271.aspx
[80] ToUcam PRO II PCVC 840k

Zur Vollständigkeit sei erwähnt, dass die Kamera auch über ein eingebautes digitales Mikrofon verfügt.

7 Probleme und Hindernisse

Im Verlauf dieser Arbeit taten sich einige Probleme und Hindernisse auf, die nachfolgend beschrieben werden. Dabei wird unterschieden in Probleme, die zur Zeit der Beendigung der Arbeit bereits gelöst sind und solche, die auch nach Beendigung dieser Arbeit weiterhin bestehen. Zudem wird unterschieden, ob es rein programmierorientierte Probleme sind, oder das Problem explizit im Zusammenspiel mit der verwendeten Hardware auftaucht.

7.1 gelöste Probleme

7.1.1 Code

Windows-spezifische Lösung einer betriebssystemunabhängigen Programmkomponente
Wie bereits in der Zielsetzung beschrieben, ist die PRAGMA-Bibliothek plattformunabhängig zu halten. Hierbei wird das Problem aufgeworfen, wie Windows-spezifischer Quellcode, wie er sich bei dieser Arbeit ergab, auf anderen Betriebssystemen kaschiert werden kann, so dass diese Programmkomponenten einfach ignoriert werden und nur die zum Betriebssystem passenden Komponenten benutzt werden. Dieses Problem lässt sich sehr leicht mit der Präprozessor-Direktive #IFDEF (und dem schließenden #ENDIF) lösen, da so ein bedingtes Kompilieren veranlasst werden kann. Dabei werden einfach die eingeklammerten Quellcodezeilen ignoriert. In der Lösung wurden also konsequent alle Windows-spezifischen Quellcodezeilen in #IFDEF-Blöcke gesetzt, mit der Prüfung, ob es sich um eine Win32-Plattform handelt, damit sie nur auf Windows-Plattformen kompiliert werden.

Zugriff auf globale Erfolgsflag
Die vererbte Membervariable successE hat in der bearbeitenden Klasse das Schreibrecht "protected" und kann damit innerhalb von Memberfunktionen mit einem neuen Wert belegt werden. Sind diese jedoch wie im vorliegenden Fall als konstant deklariert, erzeugt das Überschreiben der Variablen einen Compiler-Fehler.
Um dies zu umgehen, wird die Variable mit dem Schlüsselwort "mutable" belegt, damit innerhalb jeder Funktion die konstante Variable überschrieben werden kann.

korrektes Auslesen der Bilddaten
Die Dokumentation des Sample-Grabbers enthält die Information, dass die Funktion zum Anfordern des gepufferten Bilds eine Device-Independant-Bitmap (DIB) liefert. Die Spezifikationen dieses Formats sprechen hauptsächlich von Bilddaten im RGB-Format[81] bei sequenzieller Anordnung der Bildpunkte im Puffer und einer entsprechenden Anzahl Bits pro Pixel. Dies kann zu der irreführenden Annahme führen, dass die Bild-Funktion bereits in RGB umgewandelte Bilddaten liefert. Fasst man jedoch die erhaltenen Daten fälschlicherweise als RGB auf, wird man im günstigsten Fall ein Graubild zu sehen

[81] siehe Petzold (2005), S. 687ff

bekommen (nämlich den Y-Anteil einer YUV-Komprimierung), in den meisten Fällen jedoch nur bewegte Schlieren.
Um ein korrektes Auslesen der Bilddaten zu garantieren, muss erwähnt werden, dass DIBs neben RGB auch in zahllosen weiteren Komprimierungsformen vorliegen können. Die Komprimierung kann dabei aus dem Bitmapheader ausgelesen werden oder über die Bitzahl pro Pixel "erraten" werden.
Mit dieser Information ist es möglich, die von der verwendeten Kamera gelieferten Bilddaten korrekt als eine YUV-Unterart zu identifizieren und die Bildpunkte korrekt aus dem Puffer auszulesen und danach in RGB umzurechnen.

7.1.2 Kamera

Weißabgleich
Viele Kameraparameter lassen sich (je nach Hardware) automatisch regeln oder über die passende Schnittstelle manuell regeln. Der wichtige Parameter Weißabgleich jedoch wird von diesem Kameratyp nicht zur Regelung freigegeben[82].
Um trotzdem mit der Kamera arbeiten zu können, nutzt man ausweichend vor Aufruf der Kamera unter PRAGMA den eingebauten Einstellungsdialog der Kamera, um sowohl die automatische Regelung des Weißabgleichs auszuschalten, als auch den Parameter selbst zu bestimmen. Diese etwas umständliche Problemlösung führt aber zu weiteren Problemen (*Einstellungsdialog*, siehe unten).

Graubildfehler
In den Testläufen der Kamera kam es vereinzelt zu dem Phänomen, dass nur noch Grauwertbilder geliefert wurden, gleich welche Einstellung am Farbparameter vorgenommen wurde. Dies trat auch in allen anderen Anwendungen, die das Bild der Kamera darstellen können, auf. Die Ursache dieses Fehlers ist bis jetzt noch nicht geklärt worden, jedoch lässt sich das Problem beseitigen, in dem die Kamera auf Werkseinstellung zurückgesetzt wird[83].

7.2 nicht lösbare Probleme

7.2.1 Code

Globale Variable für den Zustand des Filter-Graphen
Wie im Quelltext ersichtlich, sollte der Betriebs-Zustand des Filter-Graphen ursprünglich in einer booleschen Variable festgehalten werden, um eine schnelle Abfrage der Bereitschaft innerhalb der Image-Funktion zu gewährleisten. Da Bilder nur bei Datenfluss im Graphen abgenommen werden, ist eine Abfrage vor Anforderung eines Bildes unerlässlich.

Dieser Ansatz birgt jedoch zwei Mängel, die das Verwenden einer solchen Variablen zumindest für die Bildfunktion ausschließen. Zum einen besitzt ein Filter-Graph eine durch eine boolesche Variable nicht abdeckbare Menge an Zuständen. Neben "stopped" und "running" existiert noch der dritte Zustand "paused", der auf diese Weise nicht repräsentiert werden kann. Zum anderen birgt die Verwendung einer Variablen, die mit dem Filter-

[82] vergleiche Anhang B II
[83] dies ist eine Option im Einstellungsdialog der Kamera

Graphen selbst nicht in Verbindung steht, die Gefahr, dass eine Inkonsistenz zwischen Variablenwert und tatsächlichem Zustand des Filter-Graphen entsteht. Somit ist nicht mehr garantiert, dass die Variable den tatsächlichen Zustand des Graphen angibt, woraus wiederum Laufzeitfehler entstehen können.

Als Lösung dieses Problems wird der Zustand des Filter-Graphen vor jeder Bildanforderung am Filter-Graph-Manager erfragt und wenn nötig geändert. Dies erfüllt zwar die gleiche Funktion wie die boolesche Variable und ist überdies auch noch eine verlässliche Quelle für den tatsächlichen Zustand, bringt aber zugleich den Nachteil einer längeren Rechenzeit mit sich, da die Abfrage einer booleschen Variable deutlich weniger Zeit in Anspruch nimmt als die Abfrage am Filter-Graph-Manager über eine Schnittstelle.

Da gerade die Bildholfunktion eine zeitkritische Aufgabe ist, bleibt die Findung einer effizienten Methode ungelöst.

Wahl der richtigen Puffer-Auslese-Methode
Wie bei der Bildhol-Funktion beschrieben, muss der mit Bilddaten gefüllte Puffer korrekt interpretiert werden. Hierzu ist das Wissen über die Anzahl Bits, die auf jeden Bildpunkt entfallen, sowie das Komprimierungsverfahren notwendig. Das Problem dieser Auswahl konnte grob gelöst werden, in dem mittels einer Hilfsfunktion zumindest der Farbraum (und damit die Anordnung der Werte eines Pixels im Puffer) ermittelt wird. Faktisch ist die getroffene Einteilung aber nicht fein genug, da zumindest für die diversen YUV-Formate fast für jedes Format eine andere Pixelwertanordnung definiert ist. Hierzu müsste die Enumeration der Farbraumtypen verfeinert werden, um danach in der Bildfunktion für jedes Format eine eigene Methode zu implementieren. Da jedoch weder diese Methoden getestet werden können, noch eine statistische Einschätzung der bei Webcams gebräuchlichen YUV-Formate vorgenommen werden kann, ist nur eine beispielhafte Methode für das YUV-Format der Testkamera implementiert worden.

Der Quellcode enthält auch eine einfache Methode zur Übetragung von RGB-basierten Bildpuffern, jedoch fehlen auch hier noch weitere Methoden für RGB-Formate mit geringeren Farbtiefen, die zudem teilweise noch auf Farbpaletten zugreifen müssen.

7.2.2 Kamera

Einstellungsdialog
Die Kamera bietet einen umfangreichen eingebauten, fensterbasierten Einstellungsdialog für typische Webcam-Anwendungen an. Dieser umfasst einige Parameter, die nicht über fensterlose Methoden substituiert werden können (unter anderem *Weißabgleich*, siehe oben). Wie weiter oben beschrieben, können diese Parameter vorab über den Aufruf des Einstellungsdialogs verändert werden (z.B. unter GraphEdit, das als Option das Anzeigen eines Filtereinstelldialogs anbietet), da alle Änderungen in der Kamera vermerkt werden und somit auch unter PRAGMA weiterhin ihre Gültigkeit haben.
Der mehrseitige Dialog verschwand jedoch in den meisten Fällen der Benutzung nach ungefähr drei Sekunden ohne Meldung eines etwaigen Fehlers, ohne dass dem Benutzer die Zeit bleibt, die Parameter zu suchen, einzustellen und durch Bestätigung an die Kamera weiterzureichen. Dieser Fehler tritt unabhängig vom aufrufenden Programm auf, sowohl bei reinen Anwendungen (Skype), bei DirectShow-Werkzeugen (GraphEdit), als auch beim auf der Treiber-CD mitgelieferten Konfigurationsprogramm der Kamera (Philips VLounge), so

dass eine genaue Bestimmung der Fehlerursache nicht möglich ist. Anzunehmen ist, dass entweder ein Konflikt mit dem verwendeten Arbeits-PCs besteht oder der Dialog generell einen Programmierfehler beinhaltet.

Weißabgleich per Schnittstelle
Wie weiter oben bereits angesprochen, besteht keine Möglichkeit, mit der Parameter-Schnittstelle von DirectShow den Weißabgleich der verwendeten Kamera zu regulieren. Eine weitere Schnittstelle ermöglicht den Zugriff auf Kameraparameter[84], inklusive der Verfeinerung der Einstellmöglichkeiten (z.B. lässt sich der Weißabgleich getrennt in Rot- und Blau-Komponente regeln), ist aber in der benutzten Ausgabe des Windows SDKs noch nicht vollständig implementiert. Aufgrund dieser Tatsache ist es zum Zeitpunkt dieser Arbeit noch nicht möglich, zu testen, ob diese Schnittstelle eine funktionierende Alternative zur Benutzten darstellt. Da der Einstelldialog der Kamera ebenfalls den Weißabgleich in Rot- und Blauanteil aufteilt, ist dies ein möglicher Hinweis, dass die alternative Schnittstelle mehr Erfolg verspricht im Umgang mit den Kameraparametern.

Belichtung
Bei Benutzung der Kamera mit manuell geregeltem Weißabgleich und ebenfalls manuell eingestellter Helligkeit fällt auf, dass weiterhin starke Helligkeitsschwankungen im Bild auszumachen sind, sobald im Bildbereich Lichtquellen dazukommen oder entfernt werden. Dies beruht auf einer ebenfalls automatisch vorgenommenen Regelung der Belichtung der Kamera.
Diese lässt sich wie gewohnt im Kameraeinstellungsdialog abschalten, jedoch bietet DirectShow auch eine Schnittstelle[85] für eine weitere Gruppe an Kameraparametern (umfasst unter anderem Kameraausrichtung, Zoom, Fokus etc.) an.
Diese Schnittstelle wird jedoch nicht vom Quellfilter der verwendeten Kamera unterstützt, so dass hier weiterhin das Problem besteht, die Belichtung ohne Verwendung von Fensterdialogen regeln zu können

freie Auflösungswahl
Wie weiter oben erwähnt, lässt sich die Bildauflösung am Output-Pin nur paarweise ändern. Da die PRAGMA-Schnittstelle jedoch getrennte Einstellmöglichkeiten anbietet, wurde als provisorische Lösung in dieser Arbeit die jeweils nicht anzutastende Auflösungsdimension trotzdem passend im Verhältnis 4:3 mit verändert, um ein legitimes Ausgabeformat zu erzielen. Sollte zukünftig nur eine Dimension verändert werden, so gilt zu überlegen, ob dies über ein entsprechend abgeändertes Auslesen der Bilddaten (Auslassung oder Interpolation von Bildpunkten) realisiert werden kann.

8 Fazit und Ausblick

8.1 Fazit

[84] IKsPropertySet, siehe http://msdn2.microsoft.com/en-us/library/ms785843.aspx
[85] IAMCameraControl, siehe http://msdn2.microsoft.com/en-us/library/ms783833.aspx

Im Zuge dieser Arbeit gelang es, einen Quellcode zu entwickeln, der die gestellte Zielsetzung zu großen Teilen erfüllt. Hierzu wurde die umfangreiche, aktuelle Technologie DirectShow eingesetzt. Einige entwickelte Komponenten konnten aufgrund technischer Einschränkungen der verwendeten Test-Kamera nur unzureichend überprüft werden, sollten aber aufgrund strenger Beachtung der Dokumentation bei anderen Kameratypen und entsprechender Unterstützung dieser Funktionalitäten wie der Rest fehlerfrei arbeiten.

Aufgrund der verwendeten Technologie ist die Komponente nicht nur auf USB-basierte Webcams und sogar nicht einmal auf Kameratypen der Art "Webcam" beschränkt. Jedwede Kamerahardware, die einen entsprechenden DirectShow-kompatiblen Treiber zur Verfügung stellt und korrekt unter Windows registriert ist, sollte ansprechbar und nutzbar sein.

Durch die aktuellen Umstrukturierungen des Windows SDKs, insbesondere im Funktionsbereich Multimedia, muss abgewartet werden, ab wann eine Überarbeitung der entstandenen Komponente von Nöten ist. Eine grobe Abschätzung ist gegeben, indem die Marktdurchsetzung und -verbreitung des neuen Betriebssystems Windows Vista als Maß genommen wird. Sobald ein genügend großer Anteil den Vorgänger Windows XP abgelöst hat, wird die Zusammenfassung der Multimediakomponenten (unter anderem auch DirectShow) zum neuen Framework Media Foundation fortschreiten und damit DirectShow ersetzen.

8.2 Erfolge und Misserfolge

Die Zielsetzung der Arbeit konnte zu großen Teilen erfüllt werden. Funktionale Einschränkungen beruhen dabei größtenteils auf der verwendeten Kamerahardware, die nur bedingt eine fensterlose Verwendung unterstützt.
Die entstandene programmiertechnische Lösung ist dabei ohne Kenntnis der dahinter liegenden Technologie bedienbar, zur Anpassung einzelner Funktionen muss jedoch zumindest das in dieser Arbeit vermittelte Wissen über die Prinzipien und Funktionalitäten von DirectShow bekannt sein.
Die Lösung ist, wie in der Zielsetzung gefordert, strikt getrennt von Implementierungen für andere Plattformen und so gut wie möglich dem Stil der restlichen PRAGMA-Bibliothek angepasst.
Es werden von der Kamera zeitnahe Bilder geliefert, die zudem ohne Weiterverarbeitung angezeigt eine für das menschliche Auge flüssige Videowiedergabe ergeben. Somit sind auch die Anforderungen hinsichtlich zeitkritischer Verarbeitung erfüllt.
Die Misserfolge dieser Arbeit sind im entsprechenden Kapitel detailliert ausgeführt und beruhen größtenteils auf dem Zusammenspiel von Implementierung und verwendeter Kamerahardware. Keines der aufgeführten Probleme ist dabei so einschneidend, dass die Funktionalität und damit der Gesamterfolg stark eingeschränkt wird.

8.3 eigener Eindruck

Die Aufgabenstellung bot ein angemessen komplexes Problem, das mit den gegebenen Mitteln jedoch gut in Eigenarbeit gelöst werden konnte. Die Arbeit an dieser Problemstellung hat nicht nur das Mitwirken an einem größeren Projekt (PRAGMA) geschult, sondern auch

profunde Kenntnisse über die Umsetzung multimedialer Programmieraufgaben unter Windows vermittelt. In Hinblick auf die Weiterverwendung der entstandenen Programmkomponente als Teil einer größeren Funktionsbibliothek ist auch die Bedeutsamkeit der Arbeit im zufrieden stellenden Maße erfüllt. Somit stellt diese Arbeit eine sinnvolle und sehr lehrreiche Beschäftigung des Verfassers mit den Themen des Studiengangs dar.

8.4 Ausblick

Der erstellte Quelltext bietet bereits ein umfangreiches Grundgerüst für elementare Funktionen im Umgang mit Webcam-Quellen, jedoch kann es von Nöten sein, noch weitere Elemente hinzuzufügen. Unter anderem fehlt noch eine Funktion, mit der der Wert für den Weißabgleich ausgelesen beziehungsweise manuell eingestellt werden kann. Dies ist aber aufgrund der verwendeten Kamera, die diese Option von vornherein nicht unterstützt, erst einmal nicht von Interesse.

Ein weiteres Erweiterungsgebiet stellt die Funktion *imagePointer* dar. Hier wird spätestens bei der Verwendung anderer Kamerahardware eine mögliche Anpassung der Methoden zum Bildpufferauslesen wichtig. Sofern die neue Hardware eine andere Kompressionsart verwendet, als bisher durch die vorhandenen Methoden abgedeckt wurde, muss auch das Auslesen der Bilddaten angepasst werden. Hierzu ist wiederum das Wissen über die Anordnung der Werte für die einzelnen Bildpunkte ausschlaggebend, um den Puffer korrekt auszulesen und möglicherweise weitere Umwandlungen in andere Farbräume vorzunehmen.

Ein letzter Ausblick soll der bereits angesprochenen Umgestaltung der Multimedia-Programmierschnittstelle unter Windows Vista gelten. Es ist zu diesem Zeitpunkt noch nicht abzusehen, in welcher Form Änderungen an der jetzigen DirectShow-Version vorgenommen werden. Zwei Szenarien sind denkbar: Zum einen kann im Zuge des Zusammenschlusses aller multimedial orientierten Programmierschnittstellen zur Microsoft Media Foundation eine einfache Eingliederung von DirectShow in die überspannende Gesamtstruktur erfolgen, so dass mit einigen Anpassungen der Verknüpfung zum SDK der Quellcode übernommen werden kann. Zum anderen ist auch eine Neuaufauflage der DirectShow-Funktionalität als neues, vergleichbares Framework mit vollkommen neuen Funktionsbezeichnungen und Strukturen möglich. Dies würde einige Code-Instandhaltungsarbeiten nach sich ziehen, wobei davon ausgegangen werden darf, dass die grundlegenden Prinzipien von DirectShow weiterhin bestehen bleiben werden, so dass eine Neuimplementierung gerade im Zusammenhang mit dieser Ausarbeitung wesentlich weniger Aufwand erfordert, als eine komplette Neugestaltung auf anderer Grundlage.

Anhang A: Codebeispiele

(Codeversion vom 08.10.2007, "..." zeigt Auslassungen an)

I typischer Aufruf von CoCreateInstance inklusive Erfolgsabfrage

```
// create filter graph
hr = CoCreateInstance
    (
     CLSID_FilterGraph,
     NULL,
     CLSCTX_INPROC,
     IID_IGraphBuilder,
     (void **) &pGraphBuildG
    );
if(SUCCEEDED(hr))
{
  …
```

II Aufbau der XML-Struktur

```
// create a XML document
pragma::XML::XmlDocPointer xmlDocL =
                        pragma::XML::XmlDocPointer("datasources");
… (Auffinden der Quellen und Speichern der Namen in resultL)
      // got the name, save to XML
      std::string defaultL = "video_source";
      xmlDocL->newStringNode( xmlDocL->root(),defaultL,resultL );
…
// write XML to disc
xmlDocL->write(targetXmlFileA);
```

III bedarfsmäßiges Starten des Filter-Graphen

```
// get the current state of the filter graph
OAFilterState aStateL;
pMediaControlG->GetState( INFINITE, &aStateL );
if(aStateL != State_Running)
{
 pMediaControlG->Run();
 graphRunningG = true;
 do
 {
  // wait for completion
  Sleep(1000);
  pMediaControlG->GetState( INFINITE, &aStateL );
 }
 while(aStateL != State_Running);
}
```

IV Anfordern eines Frames am Sample-Grabber

```
// grab the current frame
// find the required buffer size.
long cbBufferL = 0;
```

```
hr = pIGrabberG->GetCurrentBuffer( &cbBufferL, NULL );
if(FAILED(hr))
{
  return NULL;
}
// create a buffer with the required size
BYTE *pBufferL = new BYTE[cbBufferL];
if (!pBufferL)
{
  return NULL;// Out of memory. Return an error code.
}
hr = pIGrabberG->GetCurrentBuffer( &cbBufferL, (long*)pBufferL );
if(FAILED(hr))
{
  return NULL;
}
```

V zweifach gestaffelte Abfrage für die passende Bildverarbeitungsmethode

```
// now get the output format and choose the adequate method for
// transforming the buffer into a PRAGMA image pointer
if(colorModelG == COLORMODEL_YUV) // color model is YUV
{
  if(colorDepthG == 12) // color depth is 12 (bits per pixel)
  {
    …
```

VI Auslesen der Auflösung (hier Breite) aus dem aktuellen Ausgabeformat

```
  // retrieve the current output format
  AM_MEDIA_TYPE *pMediaTypeL = NULL;
  if(pStreamConfigL->GetFormat(&pMediaTypeL) == S_OK)
  {
    int xResL = 0;
    // inspect the video info header
    if (pMediaTypeL->cbFormat >= sizeof(VIDEOINFOHEADER))
    {
      VIDEOINFOHEADER *pVihL =

        reinterpret_cast<VIDEOINFOHEADER*>(pMediaTypeL->pbFormat);
      // get the x-resolution from the bitmap header
      xResL = pVihL->bmiHeader.biWidth;
    }
  …
```

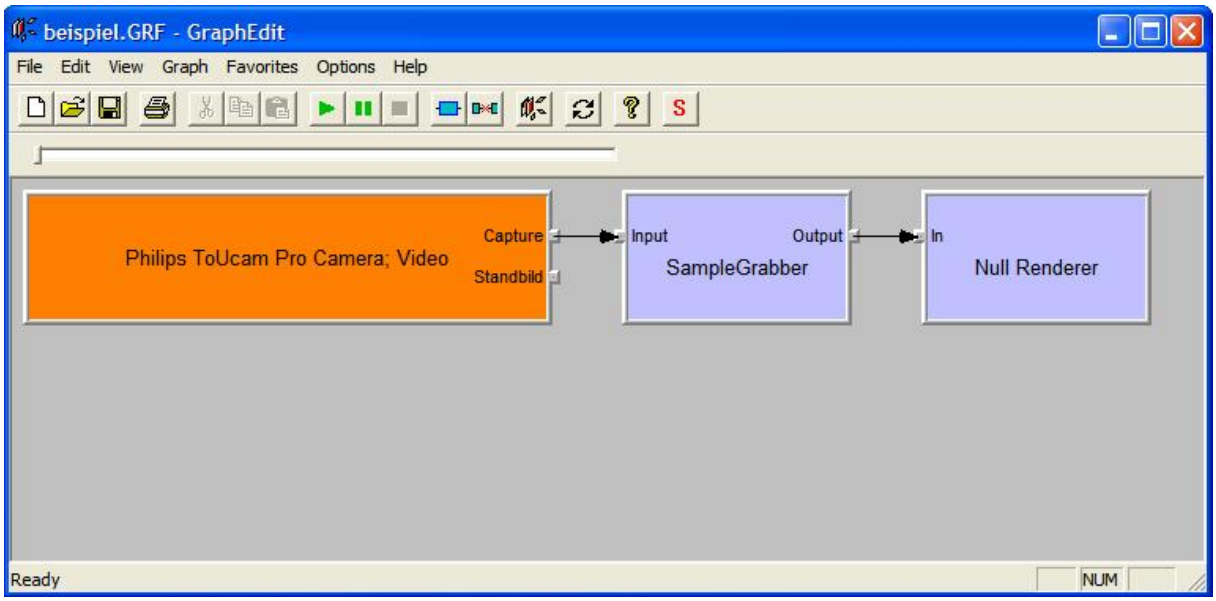
VII Feststellen und Speichern des Ausgabeformats

```
  // retrieve the current output format
  AM_MEDIA_TYPE *pMediaTypeL = NULL;
```

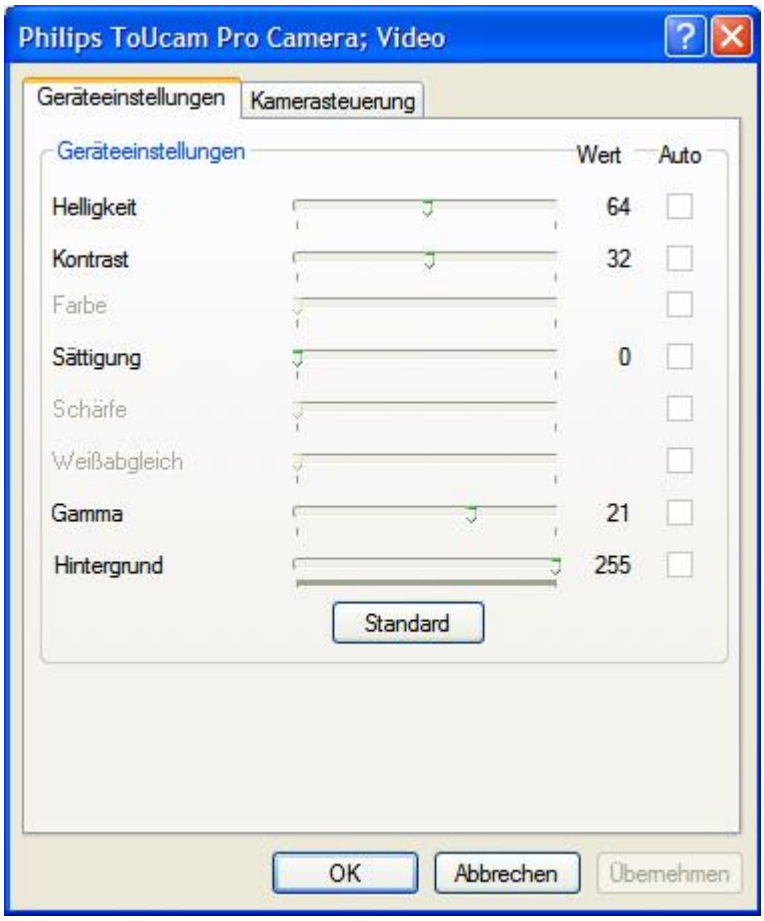
```
if(pStreamConfigL->GetFormat(&pMediaTypeL) == S_OK)
{
 // try to guess the color model by inspecting the media subtype
 // set default:
 colorModelG = COLORMODEL_YUV;
 if
 ( // color model is one of the following numerous RGB types
      pMediaTypeL->subtype == MEDIASUBTYPE_RGB1
  || pMediaTypeL->subtype == MEDIASUBTYPE_RGB4
  || pMediaTypeL->subtype == MEDIASUBTYPE_RGB8
  || pMediaTypeL->subtype == MEDIASUBTYPE_RGB555
  || pMediaTypeL->subtype == MEDIASUBTYPE_RGB565
  || pMediaTypeL->subtype == MEDIASUBTYPE_RGB24
  || pMediaTypeL->subtype == MEDIASUBTYPE_RGB32
  // these are all uncompressed RGB types
 )
      {
  // assume RGB model
  colorModelG = COLORMODEL_RGB;
 }
 else if
 … (nächsten Typ abfragen)
```

Anhang B: sonstiges

I Screenshot von GraphEdit mit Beispielgraph



II Screenshot des Einstelldialogs (DirectShow-basiert) der Webcam unter NetMeeting



III Einrichtung der Entwicklungsumgebung

Folgende Arbeitsschritte sind notwendig, um die im Verlauf der Arbeit entwickelte Kameraschnittstelle unter Windows korrekt kompilieren und nutzen zu können (Unterpunkte, die mit einem Stern versehen sind, sind möglicherweise bereits bei der Installation erledigt worden):

1. Installation der frei erhältlichen Visual C++ Express Edition
2. Installation des Microsoft Windows SDKs (Platform SDK Release 2005)
    a. * unter Visual C++ Pfade zum SDK einrichten (bin,lib…)
3. Installation des DirectX-SDKs (Release April 2007)
    a. * unter Visual C++ Pfade zum SDK einrichten (lib…)
4. Installation des aktuellen PRAGMA-Releases
    a. Verzeichnis einrichten
    b. Umgebungsvariable setzen
    c. 3rdParty-DLLs ins Systemverzeichnis kopieren
    d. PRAGMA-Bibliotheken kompilieren
        i. hierzu ist möglicherweise das Hinzufügen von Bibliotheken, wie "Quartz.lib", "Strmiids.lib", "ole32.lib" und "oleaut32.lib" notwendig
5. eine Visual C++ Projektdatei generieren lassen mit genvcproj4exe.exe (enthält alle benötigten Verknüpfungen zu Laufzeitbibliotheken)

Glossar

Bildwiederholrate – Anzahl von →Frames innerhalb einer Sekunde eines Videos

COM – kurz für: Component Object Model, Technologie von Microsoft, bietet Funktionen zum Erzeugen und Benutzen von sprachunabhängigen[86] Komponenten (Objekten)

Datenfluss – im Kontext dieser Arbeit Bezeichnung für Bilddaten, die einen →Filter-Graphen durchlaufen

Datenquelle – im Kontext dieser Arbeit die Bezeichnung für eine →Webcam und der daraus gelieferten Bilddaten

Datenstrom – synonym zu →Datenfluss

Dialog – im Kontext dieser Arbeit eine fensterbasierte Benutzerschnittstelle, hauptsächlich genutzt zur Einstellung von Programmparametern oder Eingabe von Daten

DirectShow – Multimediaprogrammierschnittstelle unter Windows, Teil des Windows → SDKs, basiert auf →COM

Filter – abstrakter Baustein unter →DirectShow; kann unter anderem eine →Datenquelle, → Datenstromverarbeitung oder Hardwareausgabe als Funktion realisieren

Filter-Graph – Gerüst für Verkettung und Inbetriebnahme einer Anzahl →Filter, deren verbundene →Pins einen →Datenfluss ermöglichen

Filter-Graph-Manager – Verwaltungsapparat für →Filter-Graphen, bietet Funktionen zu dem Aufbau, der Inbetriebnahme und Steuerung der Filter-Graphen

Format – im Kontext dieser Arbeit meist ein Videoformat, also ein Verbund aus den Parametern Auflösung, →Bildwiederholrate und Kodierung der Farbinformationen

Frame – Einzelbild aus einem Videostream

Framerate – engl., siehe →Bildwiederholrate

GraphEdit – →DirectShow-Werkzeug zur grafischen Erstellung von →Filter-Graphen, bietet auch rudimentäre →Filter-Konfigurations- und Test-Möglichkeiten

Input-Pin – engl., Eingang des →Datenstroms an einem →Filter, siehe →Pin

Interface – engl., siehe →Schnittstelle

Output-Pin – engl., Ausgang des Datenstroms an einem →Filter, siehe →Pin

Pin – Ein- und Ausgang des Datenstroms bei einem →Filter

PRAGMA – umfangreiche, plattformunabhängige Bildverarbeitungsbibliothek, zur Zeit dieser Arbeit noch im Aufbau befindlich

[86] Dunlop (2000), S. 28

Schnittstelle – Funktionssammlung zur Kommunikation mit einem Objekt, zum Beispiel Parameterabänderung oder Aufruf von Objektfunktionen

SDK – kurz für: Software Development Kit, Sammlung von Werkzeugen für die Programmierung auf einer bestimmten Plattform oder einer bestimmten Hardware etc.

Webcam – vornehmlich für Webanwendungen gestaltete Kamerahardware, für den breiten Markt ausgelegte Bedienbarkeit

Weißabgleich – Anpassung einer Kamera an die Farbtemperatur des Lichts

white balance – engl., siehe →Weißabgleich

YUV – Farbraum, wird zur Kodierung/Komprimierung von Bilddaten genutzt

Literaturverzeichnis

Buchquellen:

Bruns & Neidhold (2003): Bruns, Kai & Neidhold, Benjamin: *Audio-, Video- und Grafikprogrammierung.* Fachbuchverlag Leibzig im Carl Hanser Verlag 2003

Dunlop (2000): Dunlop, Robert L.: *DirectX 7 Programmierung in 21 Tagen*. Markt+Technik Verlag, München 2000

Petzold (2005): Petzold, Charles: *Windows-Programmierung*. 5. Auflage. Microsoft Press Deutschland, Unterschleißheim 2005

Alle nachfolgenden Internetquellen in der Form: URL *Seitentitel* Abrufdatum/-zeit

Webseiten - einzelne:

http://libusb-win32.sourceforge.net/
LibUsb-Win32, zugegriffen am 11. September 2007, 12:11 Uhr

http://www.golem.de/0009/9753.html
Logitech stellt kostenloses SDK für QuickCam-Kameras vor, zugegriffen am 11. September 2007, 11:17 Uhr

Webseiten – Domain en.wikipedia.org:

http://en.wikipedia.org/w/index.php?title=DirectShow&oldid=157191900
DirectShow, zugegriffen am 12 September 2007, 18:00 Uhr, Permalink

http://en.wikipedia.org/w/index.php?title=Media_Foundation&oldid=157330368
Media Foundation, zugegriffen am 12. September 2007, 18:20 Uhr, Permalink

http://en.wikipedia.org/w/index.php?title=Video_for_Windows&oldid=152526823
Video for Windows, zugegriffen am 12. September 2007, 17:03 Uhr, Permalink

http://en.wikipedia.org/w/index.php?title=Windows_Image_Acquisition&oldid=149638866
Windows Image Acquisition, zugegriffen am 11. September 2007, 10:20 Uhr, Permalink

Webseiten – Online-Präsenz der MSDN-Hilfe:

http://msdn2.microsoft.com/en-us/library/ms925337.aspx
AM_MEDIA_TYPE, zugegriffen am 15. September 2007, 15:07 Uhr

http://msdn2.microsoft.com/en-us/library/ms779712.aspx
BITMAPINFOHEADER Structure, zugegriffen am 15. September 2007, 15:19 Uhr

http://msdn2.microsoft.com/en-us/library/ms686615.aspx
CoCreateInstance, zugegriffen am 20. September 2007, 11:01 Uhr

http://msdn2.microsoft.com/en-us/library/ms783299.aspx
DeleteMediaType, zugegriffen am 24. September 2007, 11:52 Uhr

https://msdn2.microsoft.com/en-us/library/ms783323.aspx
DirectShow, zugegriffen am 12. September 2007, 17:42 Uhr

https://msdn2.microsoft.com/en-us/library/ms783363.aspx
Displaying a Filter's Property Pages, zugegriffen am 13. September 2007, 17:35 Uhr

http://msdn2.microsoft.com/en-us/library/ms783680.aspx
Find an Unconnected Pin on a Filter, zugegriffen am 24. September 2007, 11:54 Uhr

http://msdn2.microsoft.com/en-us/library/ms783692.aspx
FreeMediaType, zugegriffen am 24. September 2007, 11:53 Uhr

http://msdn2.microsoft.com/en-us/library/ms783833.aspx
IAMCameraControl Interface, zugegriffen am 20. September 2007, 16:20 Uhr

http://msdn2.microsoft.com/en-us/library/ms784115.aspx
IAMStreamConfig Interface, zugegriffen am 14. September 2007, 11:28 Uhr

http://msdn2.microsoft.com/en-us/library/ms784116.aspx
IAMStreamConfig::SetFormat, zugegriffen am 17. September 2007, 15:53 Uhr

https://msdn2.microsoft.com/en-us/library/ms784351.aspx
IAMVfwCaptureDialogs Interface, zugegriffen am 13. September 2007, 16:52 Uhr

http://msdn2.microsoft.com/en-us/library/ms784400.aspx
IAMVideoProcAmp Interface, zugegriffen am 14. September 2007, 11:28 Uhr

http://msdn2.microsoft.com/en-us/library/ms785794.aspx
IGraphBuilder Interface, zugegriffen am 14. September 2007, 11:18 Uhr

http://msdn2.microsoft.com/en-us/library/ms785843.aspx
IKsPropertySet Interface, zugegriffen am 20. September 2007, 16:04 Uhr

http://msdn2.microsoft.com/en-us/library/ms785872.aspx
IMediaControl Interface, zugegriffen am 14. September 2007, 16:27 Uhr

http://msdn2.microsoft.com/en-us/library/ms679705.aspx
IMoniker, zugegriffen am 14. September 2007, 16:37 Uhr

https://msdn2.microsoft.com/en-us/library/ms786503.aspx
Intelligent Connect, zugegriffen am 13. September 2007, 14:29 Uhr

https://msdn2.microsoft.com/en-us/library/ms786508.aspx
Introduction to DirectShow, zugegriffen am 13. September 2007, 13:31 Uhr

https://msdn2.microsoft.com/en-us/library/ms786509.aspx
Introduction to DirectShow Application Programming, zugegriffen am 13. September 2007, 13:29 Uhr

http://msdn2.microsoft.com/en-us/library/ms786690.aspx
ISampleGrabber Interface, zugegriffen am 14. September 2007, 16:49 Uhr

http://msdn2.microsoft.com/en-us/library/ms786688.aspx
ISampleGrabber::GetCurrentBuffer, zugegriffen am 6. Oktober 2007, 19:52 Uhr

http://msdn2.microsoft.com/en-us/library/ms680509.aspx
IUnknown (COM), zugegriffen am 14. September 2007, 11:31 Uhr

http://msdn2.microsoft.com/en-us/library/ms629896.aspx
IWiaVideo Interface, zugegriffen am 11. September 2007, 10:54 Uhr

http://msdn2.microsoft.com/en-us/library/ms787271.aspx
Media Types, zugegriffen am 7. Oktober 2007, 11:44 Uhr

https://msdn2.microsoft.com/en-us/library/ms787445.aspx
Null Renderer Filter, zugegriffen am 13. September 2007, 17:19 Uhr

http://msdn2.microsoft.com/en-us/library/ms787460.aspx

Overview of GraphEdit, zugegriffen am 14. September 2007, 12:00 Uhr
https://msdn2.microsoft.com/en-us/library/ms787594.aspx
Sample Grabber Filter, zugegriffen am 13. September 2007, 17:22 Uhr

http://msdn2.microsoft.com/en-us/library/ms629905.aspx
TakePicture Method (IWiaVideo), zugegriffen am 11. September 2007, 10:26 Uhr

https://msdn2.microsoft.com/en-us/library/ms787871.aspx
Using the System Device Enumerator, zugegriffen am 13. September 2007, 17:40 Uhr

http://msdn2.microsoft.com/en-us/library/ms713492.aspx
Video for Windows, zugegriffen am 12. September 2007, 17:33 Uhr

https://msdn2.microsoft.com/en-us/library/ms787917.aspx
Video Mixing Renderer Filter 7, zugegriffen am 13. September 2007, 16:54 Uhr

http://msdn2.microsoft.com/en-us/library/ms787915.aspx
VIDEOINFOHEADER Structure, zugegriffen am 15. September 2007, 15:14 Uhr

http://msdn2.microsoft.com/en-us/library/ms787923.aspx
VideoProcAmpFlags Enumeration, zugegriffen am 27. September 2007, 15:43 Uhr

http://msdn2.microsoft.com/en-us/library/ms787924.aspx
VideoProcAmpProperty Enumeration, zugegriffen am 14. September 2007, 11:34 Uhr

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wia/wia/overviews/startpage.asp
WIA, zugegriffen am 11. September 2007, 10:21 Uhr

http://msdn2.microsoft.com/en-us/library/ms630343.aspx
Windows Image Acqusition (WIA), zugegriffen am 11. September 2007, 10:21 Uhr

weiterhin wurden folgende Hilfsmittel verwendet:

Microsoft Visual C++ 2005 Express Edition

Microsoft Platform SDK for Windows Server 2003 R2

Microsoft DirectX SDK (April 2007)

Microsoft GraphEdit 9 (zur Visualisierung des Filter-Graphen)

Microsoft Netmeeting 3.01 (zur Visualisierung eines fensterbasierten Parameterdialogs)

Microsoft Paint 5.1 (für die Aufbereitung der Screenshots)

Microsoft Word 2002 (zum Verfassen dieser Ausfertigung)

proton Code Editor V3.2 (zum Setzen des Quellcodes)

http://dict.leo.org/ (für Übersetzungen)

Erklärung

Ich erkläre, dass das Thema dieser Arbeit nicht identisch ist mit dem Thema einer von mir bereits für ein anderes Examen eingereichten Arbeit.
Ich erkläre weiterhin, dass ich die Arbeit nicht bereits an einer anderen Hochschule zur Erlangung eines akademischen Grades eingereicht habe.

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen der Entlehnung kenntlich gemacht. Dies gilt sinngemäß auch für gelieferte Zeichnungen, Skizzen und bildliche Darstellungen und dergleichen.

_____ _____

Datum Unterschrift

Stichwortverzeichnis