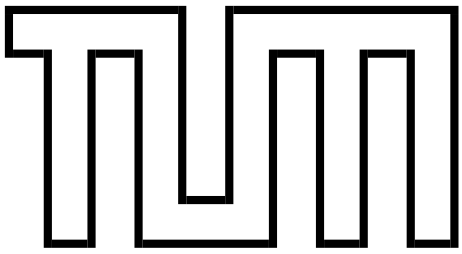

Technische Universität München

Fakultät für Informatik

Forschungs- und Lehreinheit Informatik IX

Video Streaming

Systementwicklungsprojekt

Simon Valentini

Themensteller: Prof. Michael Beetz, PhD

Betreuer: Dipl. Inf. Matthias Wimmer

Abgabetermin: 16. August 2005

Inhaltsverzeichnis

Kapitel 1

Motivation

Smart Environments sind intelligente Räume, die ihre Umgebung wahrnehmen und auf die darin ausgeführten Tätigkeiten reagieren können. Damit sich das System dem Menschen anpassen kann, müssen die benötigten Informationen über gewisse Sensoren (z.B.: Kameras, Mikrofone) aufgenommen werden. [smaom]

Programme, die zur Erkennung der Mimik eines Menschen (Projekt Emotion Detection) oder zur Lokalisierung beliebiger Personen (Projekt Person Tracking) dienen, schöpfen in der Regel die Kapazitäten moderner Computer bereits zur Gänze aus. Deshalb ist es notwendig eine verteilte Anwendung zu entwickeln, wenn mehrere Anwendungen den Videostream einer Kamera nutzen möchten.

VideoStreaming ermöglicht das Übertragen von Videodaten in Echtzeit. Dies ermöglicht einem Rechner, der mit entsprechender Hardware (Kameras) ausgestattet ist, die Videostreams über ein Netzwerk an beliebige Rechner zu verteilen. Auf diesen Rechnern werden dann die rechenintensiven Applikationen ausgeführt.

Ziel dieser Arbeit ist die Entwicklung eines Client-Server-Systems. Der Streaming-Server sendet die Videodaten beliebiger Eingabegeräte (z.B.: FireWire-Kameras, analoge Kameras und diverse andere Geräte, die über DirectShow angesprochen werden können) an einen oder mehrere Teilnehmer. Auf Client-Seite wird eine C++-Bibliothek zur Verfügung gestellt, die in jedes beliebige C++-Projekt integriert werden kann, um die Bilder aus dem Videostream wieder auslesen und weiterverarbeiten zu können. Außerdem wird ein kleiner TestClient erstellt, der im Prinzip als Viewer verwendet werden kann.

Kapitel 2

Verwendete Techniken

Im folgenden Abschnitt wird auf sämtliche Technologien, Geräte und Bibliotheken kurz eingegangen, die in dem Projekt Einfluss haben.

2.1 OpenCV

OpenCV ist eine von Intel entwickelte freie (Open-Source) Bibliothek, welche Funktionalität für den Bereich Computer Vision bereitstellt [Suddf]. Speziell in Echtzeitanwendungen findet sie Einsatz, da unter anderem viele Funktionen aus dem Gebiet der 3D-Rekonstruktion und der Bewegungsanalyse enthalten sind. Da OpenCV auf auf der Image Processing Library (IPL) basiert, werden dieselben Datenstrukturen verwendet (z.B.: IplImage).

2.2 Netzwerk-Kommunikation

Eine der Aufgaben des Streaming Servers ist es, eine sehr große Anzahl von Clients gleichzeitig mit den selben Informationen zu versorgen. Da bei Video-Streaming-Anwendungen prinzipiell eine sehr hohe Bandbreite benötigt wird, würde die Kapazität eines Netzwerkes gesprengt werden, wenn jeder Client seinen eigenen Datenstrom bekommen würde.

Um mehrere Clients mit nur einem einzigen Datenstrom (aus Sicht des Servers) versorgen zu können, benötigt man einen sogenannten Multicast-Stream. Die Clients befinden sich dabei in Gruppen, deren IP-Adressraum zwischen 224.0.0.0 und 239.255.255.255 liegt. Der Multicast-Server kann dabei über eine beliebige Unicast-Adresse angesprochen werden, und verteilt die empfangenen Informationen dann an die Clients weiter.

Damit ein Client einen Multicast-Stream empfangen kann, muss er einer Multicast-Gruppe beitreten. Diese Authentifizierung und die generelle Koordination läuft dabei über das

IGMP (Internet Group Management Protocol). Genaueres dazu findet sich unter [igmtm].

Da Multicast ein verbindungsloses Protokoll als Grundlage benötigt, wird das User Datagram Protocol (UDP) verwendet. Bei UDP wird nicht garantiert, dass ein einmal gesendetes Paket ankommt oder dass Pakete in der gleichen Reihenfolge ankommen, in der sie gesendet wurden. Es liegt kein explizites Handshakeverfahren vor. Die Kommunikationspartner können nicht feststellen, ob Pakete verloren gingen. Auch eine Vervielfältigung von Paketen kann auftreten.

Deshalb muss die Anwendung daher gegenüber verloren gegangenen und umsortierten Paketen unempfindlich sein und selbst eine entsprechende Korrekturmaßnahme bereitstellen.

2.3 Qt

Qt ist eine von der norwegischen Firma Trolltech entwickelte C++ Klassenbibliothek zur Gestaltung von graphischen Benutzeroberflächen (GUI ...Graphical User Interface). Die erstellten Anwendungen lassen sich plattformübergreifend nutzen, da Qt mit sogenannten Widgets arbeitet, die von den verschiedenen Betriebssystemen (MS Windows, MacOS, Linux, etc.) emuliert und durch das entsprechende Design dargestellt werden. [BS04]

Außerdem enthält Qt diverse Tools wie zum Beispiel den Qt-Designer, mit dem graphische Oberflächen per WYSIWYG-Verfahren (What You See Is What You Get) sehr schnell erstellt werden können. Die Funktionalität kann nachträglich in C++ hinzugefügt werden.

Die Ereignisbehandlung wird in Qt über ein Signal-Slot-System gehandhabt. Näheres dazu unter [Troml].

2.4 DirectShow

DirectShow ist ein Bestandteil von DirectX. Diese Komponente ist eine Schnittstelle für das Wiedergeben und Aufnehmen von multimedialen Streaming-Daten. Die darin enthaltenen DirectShow-Komponenten, die auch unter dem Begriff Filter bekannt sind, werden im System registriert und stehen dann sämtlichen Applikationen zur Verfügung. Bei der Verarbeitung der Multimedia-Streams sind mehrere Filter beteiligt, die eine Kette bilden.

Jeder Filtergraph besteht somit aus Filtern folgender Kategorien:

- Quellfilter, die die Signale liefern,
- Filter zum Bearbeiten (z.B.: Dekodieren, Format-Transformationen, etc.) und
- Senken (z.B.: ein Renderer, welcher die Bilder in einem Fenster ausgibt oder in eine Datei schreibt).

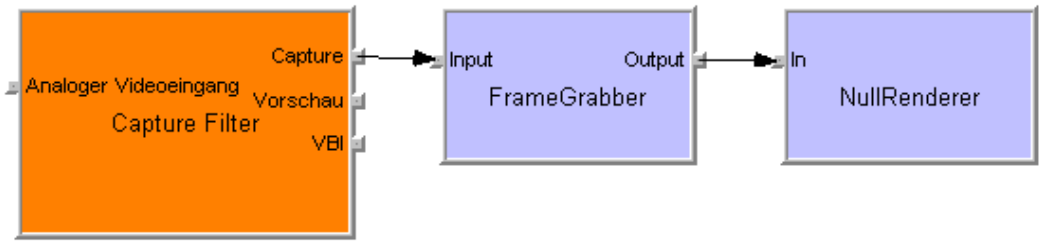

Abbildung 2.1: Direct Show Graph

Der im Server verwendete Graph hat die Zusammensetzung aus Abbildung 2.1. Der Capture-Filter liefert den Datenstrom vom Eingabegerät an den FrameGrabber-Filter, welcher bei jedem empfangenen Bild eine Callback-Methode aufruft, in der die Daten dann ausgelesen werden können. Um einen vollständigen und funktionierenden Graphen zu erhalten, muss noch eine Senke als letzter Filter in die Kette eingefügt werden. Da die Bilder über DirectShow weder am Monitor dargestellt noch in eine Datei geschrieben werden, wird der sogenannte NullRenderer-Filter verwendet, der wie der Name schon sagt, keine Funktionalität enthält.

2.5 FireWire Kameras

Als Eingabegeräte können neben DirectShow-Geräte auch Kameras, die über die Firewire-Schnittstelle angesprochen werden, verwendet werden. Während unserer Arbeit verwendeten wir das Modell Merlin (siehe Abbildung 2.2) der Firma Allied Vision Technologies.

Abbildung 2.2: Fire Wire Kamera Merlin der Firma Allied Vision Technologies

Für diese speziellen FireWire Kameras existiert ein SDK, mit dem wesentlich mehr Parameter der Kamera eingestellt werden können, als wenn die Kamera über den DirectShow-Treiber konfiguriert wird. Die Framerate, die Bildgröße, Farbtiefe sowie der Farbraum und zahlreiche andere Parameter der Kamera lassen sich so wesentlich komfortabler einstellen.

2.6 Framegrabber

Die Framegrabber-Karten der Firma IDS gehören inzwischen zu den weltweit meistverkauften Videokarten für den professionellen Einsatz. Diese Framegrabber-Karten der Falcon-Familie (siehe Abbildung 2.3), die zum Beispiel im Projekt sHome[Trö05] zu finden sind, bieten Anschlussmöglichkeiten für Monochrom- und Farbkameras. Zur Erfassung von Bildern in hoher Qualität lassen sich eine SVHS-Videoquelle und zwei CVBS Signale anschließen. Die maximal mögliche Auflösung beträgt dabei 768 x 576 Bildpunkte.

Abbildung 2.3: IDS FrameGrabber Falcon

Kapitel 3

Software-Design

In diesem Kapitel wird näher auf die implementierte Software und deren Bedienung eingegangen.

3.1 Protokoll

Wie bereits im Kapitel 2.2 erwähnt wurde, dient zur Kommunikation zwischen dem Server und den Clients ein UDP-Multicast-Streaming. Auf einen Videocodec wird an dieser Stelle bewusst verzichtet, da die Dekodierung der Bilder auf Client-Seite die Rechenleistung stark verringern würde. Außerdem will man bewusst die Qualität des Videostreams nicht herabsetzen.

Deshalb werden die Bilder serialisiert und über ein speziell für diese Anwendung entworfenes Protokoll übertragen und auf Client-Seite entsprechend deserialisiert. Im folgenden wird der Ablauf etwas genauer beschrieben.

3.1.1 Versenden der Bilder

Am Anfang jedes Bildes wird zunächst ein Header mit Meta-Information versendet, der sich folgendermaßen zusammensetzt:

- ein hardcodierter String, der den Anfang eines neuen Bildes signalisiert
- die Breite des Bildes in Pixel
- die Höhe des Bildes in Pixel
- die Farbtiefe in Bit
- die Anzahl der Farbkanäle

- der Ursprung des Bildes (z.B.: links-unten)
- die Ausrichtung der Bildzeilen
- die Größe des Bildes in Bytes

Danach wird jede Bildzeile einzeln versendet, wobei auch der Index der Zeile mit geschickt wird, um das Bild wieder korrekt zusammenbauen zu können. Bei UDP müssen die Pakete nicht zwangsläufig in der gleichen Reihenfolge ankommen, wie sie abgesendet wurden.

3.1.2 Empfangen der Bilder

Zuerst wird in dem erhaltenen Datenstrom nach dem hardcodierten String gesucht, um den Anfang eines Bildes zu erhalten. Danach werden die Bild-spezifischen Informationen ausgelesen und das IPL-Image initialisiert. Anschließend wird in einer Schleife über die Anzahl der Bildzeilen jede Zeile einzeln empfangen und an die richtige Stelle im Bildspeicher kopiert.

3.2 Server

Die Aufgabe des Streaming-Servers ist die Verbreitung des aufgenommenen Videostreams im Netzwerk mithilfe des in Kapitel 3.1 beschriebenen Protokolls. Dabei werden die Bilder des ausgewählten Eingabegerätes serialisiert und mit Hilfe eines Multicast-fähigen Routers für beliebig viele Clients im Netz zur Verfügung gestellt.

In Abbildung 3.1 ist die graphische Oberfläche, die mit dem Toolkit Qt (siehe Kapitel 2.3) erstellt wurde, dargestellt.

Die Anwendung besteht im Prinzip aus zwei Teilen:

- Input-Panel
- Output-Panel

Im Input-Panel kann der Nutzer zwischen drei Eingabegeräten wählen:

- FireWire-Cams
- IDS Falcon Framegrabber
- DirectShow

Über die Schaltfläche Preview kann in einem separaten Fenster eine Vorschau des selektierten Videostreams angezeigt werden.

Über das Output-Panel kann der Benutzer die IP-Adresse und den Port des Empfängers angeben. Standardmäßig ist der aktuelle Rechner mit Port 1234 voreingestellt.

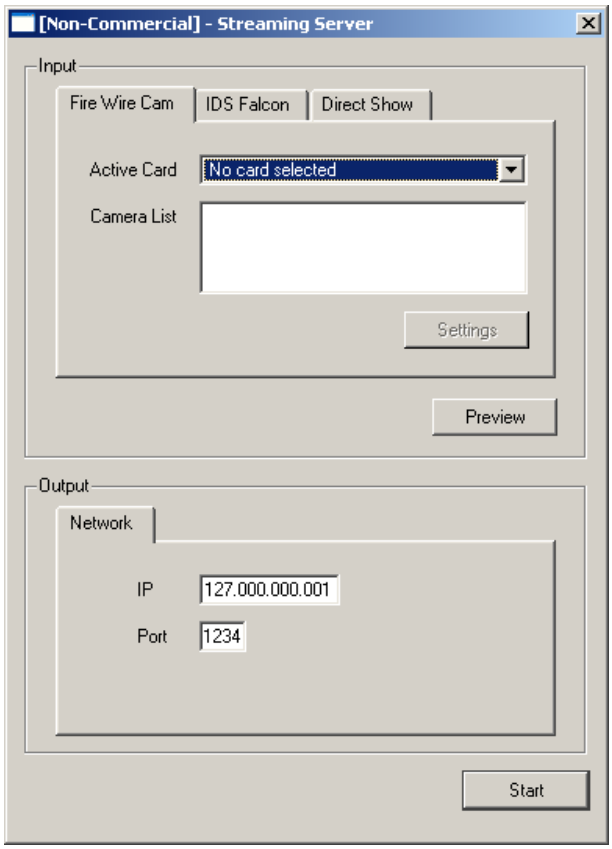

Abbildung 3.1: Die Oberfläche des Servers

Durch Drücken der Start-Schaltfläche am unteren Rand der Applikation kann der Streaming-Vorgang gestartet werden. Dabei wird der Dialog aus Abbildung 3.2 eingeblendet, der eine Schaltfläche für das Anzeigen bzw. Verstecken der Vorschau und eine Schaltfläche zum Beenden des Streaming-Vorganges bereitstellt.

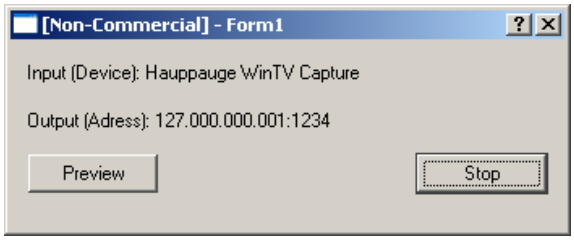

Abbildung 3.2: Streaming Dialog des Servers

3.3 Client

Der Test-Client aus Abbildung 3.3 basiert auf der entwickelten Bibliothek IPL-Receiver. In Kapitel 4 wird erklärt, wie diese Bibliothek in ein Projekt eingebunden werden kann.

Um die Bilder vom Server empfangen zu können, muss die IP-Adresse und der Port des Servers bzw. die des Multicast-Routers, der den Stream bereitstellt, angegeben werden. Durch Drücken der Start- bzw. Stop-Schaltfläche kann der Vorgang dann gestartet bzw. beendet werden.

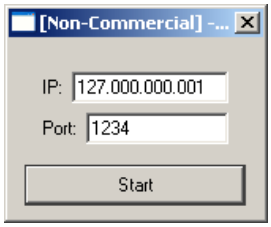

Abbildung 3.3: Die Oberfläche des Test-Clients

Kapitel 4

Integration in andere Projekte

Das folgende Tutorial soll zeigen, wie die IPLReceiver Bibliothek in ein bestehendes Projekt eingebunden werden kann.

4.1 Voraussetzungen

Die Bibliothek basiert auf Intels ComputerVision-Bibliothek OpenCV. Eine korrekte Installation von OpenCV ist deshalb Vorraussetzung für den nachfolgenden Teil.

Zuerst müssen ein paar Einstellungen bei den Projekteigenschaften vorgenommen werden.

Dazu müssen zuerst folgende Include-Verzeichnisse in die Projekteigenschaften aufgenommen werden:

- $(OPENCVROOT)\cv\include
- $(OPENCVROOT)\cxcore\include
- $(OPENCVROOT)\otherlibs\highgui
- $(IPLRECEIVERROOT)\include

$(OPENCVROOT) enthält den Pfad zum Verzeichnis der OpenCV-Installation (z.B.: C:\Programme\OpenCV). Hinter $(IPLRECEIVERROOT) verbirgt sich das Verzeichnis des IPLReceivers.

Weiters müssen noch folgende Bibliotheksverzeichnisse angegeben werden:

- $(OPENCVROOT)\lib
- $(OPENCVROOT)\cv\include
- $(OPENCVROOT)\cxcore\include
- $(OPENCVROOT)\otherlibs\highgui

- $(IPLRECEIVERROOT)\lib

Damit der Linker auch zukünftig fehlerfrei arbeiten kann, müssen ihm folgende Bibliotheken mitgeteilt werden:

- cvd.lib
- highguid.lib
- cxcored.lib
- wsock32.lib
- IplReceiver.lib

4.2 Programmablauf

Mit folgender Zeile Code erhält man die Instanz der Klasse CIplReceiver:

```
CIplReceiver* receiver = CIplReceiver::getInstance();
```

Nur zwei Parameter müssen gesetzt werden. Die IP und der Port des Senders:

```
receiver->setIP("127.0.0.1");
receiver->setPort(1234);
```

Danach kann das Empfangen der Bilder mit folgendem Befehl gestartet werden:

```
receiver->start();
```

Um an die einzelnen übertragenen Bilder zu kommen, hat man zwei Möglichkeiten. Entweder man holt sich immer das aktuelle Bild mittels folgendem Befehl:

```
IplImage* image = receiver->getImage();
```

Oder man gibt eine Callback-Methode an:

```
receiver->setCallBack((void(*)(void*))callback);

void callback(void* _image)
{
  IplImage* image = (IplImage*)_image;
  cvvShowImage("Received Image", image );
}
```

Diese Methode wird immer dann aufgerufen, wenn ein neues Bild empfangen wurde.

Um das Programm wieder zu stoppen muß folgende Methode aufgerufen werden:

```
receiver->stop();
```

4.3 Beispielprogramm

Nachfolgend ist ein kleines Programm aufgelistet, dass den vollständigen Ablauf noch einmal wiedergibt. Zu beachten ist, dass beide Möglichkeiten die Bilder zu empfangen (über eine Callback-Methode und direkt über einen Methodenaufruf) implementiert sind. Bei der Integration in ein Projekt sollte allerdings nur eine der beiden Möglichkeiten verwendet werden.

```
#include <IplReceiver.h>
#include <highgui.h>

void callback(void* _image)
{
  IplImage* image = (IplImage*)_image;
  cvvShowImage("Received Image", image );
}

int main(int argc, char* argv[])
{
  cvNamedWindow("Received Image", 0);

  CIplReceiver* receiver = CIplReceiver::getInstance();

  receiver->setCallBack((void(*)(void*))callback);
  receiver->setIP("127.0.0.1");
  receiver->setPort(1234);
  receiver->start();

  while(true)
  {
    IplImage* image = receiver->getImage();
    cvvShowImage("Received Image", image );
    cvReleaseImage(&image);
  }

  receiver->stop();
  cvDestroyWindow("Received Image");

  return 0;
}
```

Kapitel 5

Resumee und Ausblick

Mit Hilfe des erstellten Client-Server-Systems ist es möglich, die selben Videodaten einer Kamera auf mehreren Rechnern gleichzeitig auswerten zu können. Dabei ist speziell auf Client-Seite die Bibliothek so gestaltet worden, dass extrem wenig Rechenleistung und Resourcen benötigt werden.

Zukünftig könnte noch ein Videocodec in die Applikation integriert werden, um die Auslastung des Netzwerkes zu verringern bzw. Videostreams in einer höheren Auflösung übertragen zu können. Der Preis dafür wäre jedoch eine höhere Auslastung des Prozessors und somit würden der eigentlichen Applikation weniger Resourcen zur Verfügung stehen.

Literaturverzeichnis

[BS04] Jasmin Blanchette and Mark Summerfield. *C++ GUI Programming with Qt 3.* Prentice Hall PTR, 1. edition, 2004.

[igmtm] *IGMP, Internet Group Management Protocol.* http://www.networksorcery.com/enp/protocol/igmp.htm.

[smaom] *Smart Rooms.* http://vismod.media.mit.edu/vismod/demos/smartroom.

[Suddf] Gunther Sudra. *Intel OpenCV.* http://wwwiaim.ira.uka.de/Teaching/SeminarMedizin/Ausarbeitungen/SS2002/01_OpenCV.pdf.

[Trö05] Markus Tröscher. *Grid-basierte Multiagenten-Plattformen für sensorbasierte intelligente Umgebungen.* http://wwwradig.in.tum.de/people/wimmerm/lehre/da_troesche/da_troesche.pdf, 2005.

[Troml] Trolltech. *Signals and Slots.* http://doc.trolltech.com/3.3/signalsandslots.html.

Abbildungsverzeichnis

Index