VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

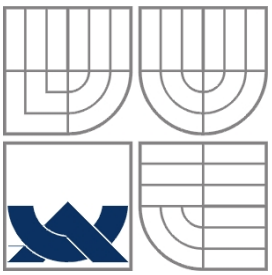
AUTOMATICKÉ ZNAČKOVÁNÍ PREZENTACÍ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

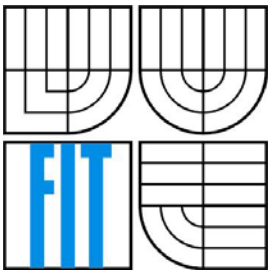
AUTOR PRÁCE
AUTHOR

Bc. MICHAL HUŠKA

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

AUTOMATICKÉ ZNAČKOVÁNÍ PREZENTACÍ

AUTOMATIC TAGGING OF PRESENTATIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MICHAL HUŠKA

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Dr. Ing. JAN ČERNOCKÝ
Mgr. ROBERT BATŮŠEK, Ph.D.

BRNO 2007

Zadání diplomové práce

Téma:

Automatické značkování prezentací

Vedoucí:

Černocký Jan, doc. Dr. Ing., UPGM FIT VUT

Přihlášen:

Huška Michal, Bc.

Zadání:

Úkolem je vyvinout systém, který bude schopen snímat kamerou plátno, na které je promítána prezentace. Pokud dojde na plátně k výměně snímku nebo jiné změně (např. animace), program zaznamená synchronizační značku do souboru. Program má být navržen tak, aby byl schopen provozu na mobilním telefonu a v podmínkách běžné posluchárny.

1. Prostudujte možnosti přístupu ke kameře na mobilních zařízeních.
2. Navrhněte a implementujte algoritmy pro detekci plátna.
3. Navrhněte a implementujte metody pro detekci změny slajdu.
4. Testujte na reálných datech
5. Navrhněte možnosti eliminace vlivu osoby pohybující mezi kamerou a plátnem.

Část požadovaná pro obhajobu SP:

Body 1. a 2. zadání

Kategorie:

Zpracování obrazu

Implementační jazyk:

C++

Operační systém:

Windows Mobile

Literatura:

- podle pokynů vedoucího

Komentář:

Diplomová práce je společným projektem FIT a ANF Data.

Licenční smlouva

Licenční smlouva je uložena v archivu Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

Tato práce se zabývá vývojem aplikace na mobilní zařízení se systémem Windows Mobile. Úkolem aplikace je s využitím vestavěné kamery snímat plátno s probíhající prezentací a ukládat časové značky, informující o změně slajdu či animaci na plátně.
V textu jsou popsány požadavky na systém, je zde provedena analýza, u níž bylo využito jazyka UML, řešení problému na úrovni zpracování obrazu, popis implementace v jazyce C++ a výsledky testování aplikace. V dokumentu je též nastíněna problematika tvorby aplikací pro mobilní zařízení. Velký prostor je věnován práci s multimédii na systému Windows Mobile 5.0, zejména pak problematice spojené s technologií DirectShow.

Klíčová slova

C++, DirectShow, detekce plátna, detekce změny, Pocket PC, Windows Mobile 5.0, zpracování obrazu

Abstract

This thesis deals with a development of application running on mobile devices with Windows Mobile operating system. The main task of this application is observing canvas with running presentation and saving time marks, that inform about slide change or animation.
Description of system requirements, system analysis using UML language, solutions on image processing level, decription of implementation in C++ language and application tests results are described in the text. Problems of mobile device software development are also outlined in the document. A great part is dedicated to work with multimedia on Windows Mobile 5.0 system, especially to problems linked with DirectShow technology.

Keywords

C++, DirectShow, canvas detection, change detection, Pocket PC, Windows Mobile 5.0, image processing

Citace

Michal Huška: Automatické značkování prezentací, diplomová práce, Brno, FIT VUT v Brně, 2007

Automatické značkování prezentací

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. Černockého a Mgr. Batůška, Ph.D.
Další informace mi poskytl Ing. Sumec, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Michal Huška
22.5.2007

Poděkování

Rád bych poděkoval doc. Černockému, Mgr. Batůškovi, Ph.D. a Ing. Sumcovi, Ph.D. za pomoc, vedení a připomínky týkající se této práce.

Obsah

1 Úvod

Obsahem následujícího textu je dokumentace práce na projektu s názvem Automatické značkování prezentací. Cílem této diplomové práce je vytvořit systém, který bude schopen snímat kamerou plátno, na které je promítána prezentace. Systém by měl reagovat na změny obsahu plátna (výměna snímku, animace) a ukládat informace o nich do souboru v podobě synchronizačních značek. Předpokládá se použití na mobilním zařízení (telefon, PDA) v podmínkách běžné posluchárny. Práce je společným projektem Fakulty informačních technologií Vysokého učení technického v Brně a firmy ANF Data.

Motivací k tvorbě takového systému je možnost snadného získání záznamu prezentace uchovaném v malém objemu dat. Snadnost pořízení je dána využitím dnes stále více se rozšiřujících mobilních zařízení s operačním systémem. Jsou-li k dispozici originální slajdy, je možné s pomocí výstupu této aplikace vytvořit záznam ve vysoké kvalitě. Té je dosaženo kombinací originálních slajdů prezentace, uložených snímků a časových značek, určujících jejich synchronizaci se zvukovou stopou. Oproti klasickému videozáznamu jsou uchovávána jen potřebná data, což vede k jejich malému objemu a zjednodušení manipulace s nimi.

Teoretická část textu dokumentuje analýzu projektu, nastiňuje problematiku vývoje software na mobilních zařízeních. Velká část je věnována technologiím využitelným pro zpracování obrazu v tomto prostředí a řešení problému na úrovni zpracování obrazu.

Praktická část se věnuje zejména samotnému popisu implementace, problémům, které při ni nastaly, a jejich řešení. Je zde také popsáno testování výsledné aplikace a jeho výsledky.

2 Popis problému

Ze zadání práce vyplývá řada požadavků. Hlavní funkcí aplikace je značkování. Systém používá vestavěnou kameru telefonu. Ta je namířena na plátno s probíhající prezentací. Pokud dojde k výměně slajdu na plátně, popřípadě je na plátně spuštěna animace, systém by měl toto zaznamenat v podobě časových značek. Současně jsou ukládány snímky z kamery. Toto zpracování by mělo být prováděno v reálném čase. Výstup by pak měl být uložen do XML souboru, díky čemuž jej lze dále zpracovat. Ukládána by měla být též zvuková stopa prezentace. Taková výstupní data pak mohou být využita pro rekonstrukci prezentace. Z XML souboru, zvukového souboru a originálních slajdů je pak možné vytvořit záznam prezentace ve vysokém rozlišení při velmi malém objemu dat v porovnání s klasickým videozáznamem.

Tento systém je určen pro provoz v běžné posluchárně. Předpokládá se mobilní zařízení (telefon nebo PDA) vybavené operačním systémem Microsoft Windows Mobile 5.0. Implementace by měla být provedena v jazyce C++ s využitím vývojového prostředí Microsoft Visual Studio 2005.

Proces, na jehož úspěšném konci je funkční aplikace splňující požadavky zadání, představuje řešení mnoha problémů souvisejících s překonáním obtíží spojených s implementací na mobilní platformě a zvládnutím algoritmů zpracování obrazu.

2.1 Algoritmy zpracování obrazu

Algoritmy zpracování obrazu tvoří jádro funkčnosti celé aplikace. Konkrétně je nutné se zabývat:

- převody mezi barevnými modely,
- detekcí plátna,
- detekcí změny na něm a
- detekcí osob pohybujících se mezi plátnem a kamerou.

Problém převodu barevných modelů vyplývá z toho, že kamery integrované v cílových zařízeních produkují obrazová data ve formátech kódujících obraz s pomocí různých barevných modelů. Ty se navíc mohou lišit od barevného modelu, v němž jsou data dále zpracovávána. Další převod může probíhat při tvorbě výstupu aplikace.

Nalezení plátna ve snímaném obraze pak umožní samotnou detekci změny slajdu či animace na něm. Kvalitu detekce změny je pak možné vylepšit algoritmy detekujícími pohybující se osoby. S využitím informací o takových objektech v obraze lze potlačit jejich vliv na detekci změny a eliminovat tak množství falešných poplachů.

Předpokládá se rovněž, že podoba jednotlivých algoritmů bude ovlivněna faktem, že kamera není statická.

2.2 Implementace

2.2.1 Windows Mobile 5.0

Windows Mobile 5.0 je kompaktní operační systém založený na Win32 API a vybavený řadou základních aplikací pro mobilní zařízení. Cílovou platformou této práce je právě systém Windows Mobile 5.0. Existují důležité odlišnosti mezi vývojem aplikací na tento systém a na jeho desktopovou obdobu Windows XP. Proto je pro vývojáře nutné o Windows Mobile něco vědět předtím, než pro tuto platformu začnou vyvíjet své aplikace.

2.2.1.1 Mobilní verze Windows

Windows Mobile je podmnožinou platforem založených na Windows CE. Windows Mobile (Pocket PC, Smartphone a Portable Media Center) jsou v současnosti hlavní platformy používané na mobilních zařízeních. Každá z těchto platforem obsahuje odlišné komponenty Windows CE v závislosti na hardwarových možnostech cílových zařízení a jejich účelu.

Windows Mobile (Pocket PC) je platforma definovaná Microsoftem pro použití na zařízeních PDA. Existuje množina profilů definujících minimální požadavky na hardware a software, který je podporován (Professional edition, Premium edition). Tyto požadavky jsou přísnější než ty, které jsou kladené na platformy založené na Windows CE.

Na rozdíl od Pocket PC je platforma Smartphone určena pro použití na mobilních telefonech. Uživatelské rozhraní Smartphone předpokládá navigaci pomocí joysticku a tlačítek typických pro mobilní telefony. Na rozdíl od Pocket PC není podporován touchscreen. Mnoho aplikací pro Smartphone a Pocket PC existuje ve verzi pro obě platformy nebo běží na obou platformách s omezenou funkčností.

S příchodem Windows Mobile 5.0 (Pocket PC 2005) rozdíly mezi Pocket PC a Smartphone takřka vymizely. Stejný operační systém běží na obou platformách při vypnutí nebo zapnutí určitých funkcí. Hlavním rozdílem mezi Pocket PC a Smartphone zůstal touchscreen. Většina programů napsaných pro starší verze Pocket PC 2002 nebo 2003 je schopná fungovat i na 5.0. Naopak, software pro Windows Mobile 5.0 není schopen fungovat na zařízeních navržených pro verzi 2002 a 2003.

2.2.2 Hardware a jeho omezení

Zařízení, na kterých má být výsledná aplikace provozována, jsou svým výkonem dost omezená. Dá se očekávat, že se tato skutečnost výrazně projeví v možnostech výběru algoritmů, které budou v aplikaci implementovány, a tedy i v dosažených výsledcích. Současná Pocket PC jsou většinou vybavena procesory o taktovací frekvenci od 200 do 400 MHz (ARM verze 4, Intel XScale CPU, MIPS CPU nebo SH3 CPU). U zařízení typu Smartphone ještě méně, většinou 200 MHz. Příkladem nejvýkonnějšího z dnešních (rok 2007) Pocket PC zařízení může být Dell Axim x51v s procesorem

Intel XScale PXA270 běžícím na 624 MHz. Velikost paměti takových zařízení je momentálně ovlivněna kapacitou dostupných paměťových médií, dnes typicky 1 – 4 GB. V případě aplikace zpracovávající obrazová data z kamery jsou důležitým omezením vlastnosti kamery, jako vstupu dat do aplikace. Výsledky zpracování nasnímaných dat jsou jistě závislé na mnoha parametrech:

- kvalitě snímače (šum),
- kvalitě A/D převodníku,
- obrazového procesoru (rozlišení),
- kvalitě optické soustavy (rozlišení optické soustavy, schopnost zaostřování, nízký přenos kontrastu, chromatická aberace, vinětace, atd.).

Typické rozlišení snímaného videa je 176 na 144, špičková zařízení zvládnou rozlišení 320 na 240.

2.2.3 Analýza

Výsledek každého projektu ve velké míře závisí na kvalitě provedené analýzy. Tento projekt jistě není výjimkou. Jedním z výstupů analýzy je model vyvíjeného softwaru. K vytvoření modelu aplikace bylo s výhodou použito jazyka UML.

2.2.3.1 UML

UML (Unified Modeling Language) je standardní jazyk pro specifikaci, vizualizaci a dokumentaci prvků projektu softwarového vývoje [2]. Je tedy vhodným nástrojem pro tvorbu vizuálních modelů. UML se hodí zejména pro objektově orientovaný vývoj, ale díky zabudovaným rozšiřovacím mechanismům jsou možnosti jeho využití mnohem širší. K zachycení všech podstatných aspektů systému definuje UML následující pohledy:

- pohled případů užití (ostatní pohledy jsou do něj integrovány, používá diagram případů užití a diagram interakce),
- logický pohled (diagram tříd, stavový diagram, objektový diagram),
- pohled procesů (diagram tříd, objektový diagram),
- pohled implementace (diagram komponent),
- pohled nasazení (diagram nasazení).

Nejznámější a nejpoužívanější částí standardu jsou diagramy. Ty lze dále dělit na:

- strukturní diagramy (diagram tříd, objektový diagram, diagram komponent, diagram nasazení, diagram balíčků),
- diagramy chování (diagram případů užití, diagram aktivit, stavový diagram a skupina diagramů zvaná diagramy spolupráce: sekvenční diagram, diagram časování, diagram komunikace).

Hlavním rysem UML je vysoká úroveň srozumitelnosti v něm vytvořených diagramů, což je podstatná výhoda například při jednání se zákazníkem. Návrh UML zohledňuje bezproblémovou implementaci v nástrojích CASE (Computer-Aided Software Engineering) a snadnou interpretaci diagramů. Většina současných CASE systémů jej podporuje.

2.2.3.2 Diagram tříd

Během analýzy, prováděné v této práci, byl s výhodou použit diagram tříd. Je to druh statického strukturního diagramu. Na rozdíl od jiných strukturních diagramů popisuje diagram tříd strukturu systému třídami, jejich atributy a vztahy mezi třídami. Následuje popis konkrétního diagramu tříd (viz příloha A), který byl jedním z výstupů analýzy vyvíjeného systému.

ChangeDetector

Jádrem celé aplikace je třída *ChangeDetector*. Tato část je zodpovědná za řízení algoritmů zpracování obrazu, reaguje na podněty z uživatelského rozhraní, ovládá spuštění záznamu zvukové stopy a taktéž řídí tvorbu výsledného výstupu značkování.

XMLOutput

Tvorba výsledného výstupu je řešena v třídě *XMLOutput*. Právě ona po dokončení zpracovávání obrazových dat zapíše značky do XML souboru.

SoundCapture

Úkolem třídy je záznam zvukové stopy a její uložení do trvalé paměti.

ImageExport

Tato třída se stará o ukládání snímků do trvalé paměti.

AbstractCamera

Abstraktní třída *AbstractCamera* definuje rozhraní pro třídy, jejichž úkolem je dodávat obrazová data aplikaci. Z toho pohledu je nejvýznamnější metoda *SetCallback*, která nastavuje funkci, jenž je volána při získání nového snímku.

OnlineCamera a OfflineCamera.

V UML diagramu jsou znázorněny dvě třídy, které jsou potomky třídy *AbstractCamera*. Jsou to *OnlineCamera* a *OfflineCamera*. Jejich rozdíl je dán situací, ve kterých budou použity. Zatímco *OnlineCamera* je určena do výsledné aplikace a obrazová data získává z fyzického zařízení (kamery mobilního telefonu, PDA), *OfflineCamera* umožňuje testovat správnost implementace a úspěšnost jednotlivých algoritmů zpracování obrazu, které jsou nebo budou v aplikaci použity. Zdrojem dat v tomto případě není reálná kamera, ale soubor s uloženým videem. Při použití referenčního

videozáznamu pak lze objektivně testovat použité algoritmy. Dalším důležitým místem, kde se během vývoje třída *OfflineCamera* může uplatnit, je práce s emulátorem zařízení s Windows Mobile 5.0. Emulátor totiž nenabízí žádný způsob, jak chybějící hardwarovou kameru nahradit.

SlideChangeDetector

Opět se jedná o abstraktní třídu. Účelem je poskytnout jednotné rozhraní pro jednotlivé algoritmy detekující změnu slajdů na plátně či případné animace. Všechny třídy implementující tyto algoritmy pak definují stejné metody a lze tedy mezi nimi snadno přepínat. To je další důležitá vlastnost modelu, usnadňující porovnávání výsledků dosažených různými algoritmy.

CanvasDetector

Tato třída obsahuje metody pro detekci plátna v obraze. Informace o poloze plátna jsou dále využity třídami, které detekují změnu slajdu.

2.2.3.3 Sekvenční diagram

Sekvenční diagram zobrazuje interakci mezi objekty a to v závislosti na čase. Tento typ diagramu nachází své využití při popisu částí systémů nebo menších systémů s jednoduchým dynamickým chováním, které se dají popsat sekvencí zpráv mezi malým, konstantním počtem objektů nebo procesů.

Příloha B obsahuje sekvenční diagram hlavního případu užití. Popisuje řízení detekce změny slajdu. Z grafického uživatelského rozhraní je volána metoda *SetCallback* třídy *OfflineCamera*. Tím se nastaví funkce *Callback* jako funkce, která je pak volána pokaždé, když kamera získá další snímek. Z této funkce je volána metoda *RecieveImageData* třídy *OfflineCamera.* V tu chvíli již má objekt třídy *OfflineCamera* k dispozici data aktuálního snímku a může začít jejich zpracování. Z metody *ControlDetection*, která řídí detekci, je volána metoda *Detect* některé z tříd implementujících jeden z možných algoritmů detekce změny slajdu. Pokud je změna detekována, aktualizuje se seznam značek a přidá se aktuální snímek mezi snímky k uložení. Tento cyklus je ukončen voláním metody *StopDetection* třídy *ChangeDetector* z uživatelského rozhraní.

3 Řešení na úrovni zpracování obrazu

3.1 Zpracování obrazu

Podstatná část této práce spočívá ve zvládnutí algoritmů zpracování obrazu. Obraz je ve své podstatě dvourozměrný signál. Z toho důvodu jeho zpracování obvykle představuje aplikaci algoritmů zpracováním signálu. Úlohami typickými pro zpracování obrazu jsou například:

- zpracování barevné informace (nastavení jasu, nastavení kontrastu, převody mezi barevnými modely, …),
- geometrické transformace (změna velikosti, rotace, deformace, …),
- segmentace obrazu,
- filtrace obrazu,
- odstranění šumu,
- kombinace dvou obrazů a další.

Řešení tohoto projektu využívá algoritmy a postupy počítačového vidění, které jsou jednou z mnoha aplikací zpracování obrazu.

3.2 Počítačové vidění

Počítačové vidění, jak název napovídá, se zabývá „napodobováním“ zraku živých bytostí stroji [14]. Tento obor nachází své uplatnění v situacích, kdy je potřeba zpracovávat informace v podobě obrazových dat, v podmínkách, které vylučují využití člověka s jeho zrakem. Příkladem nasazení systému počítačového vidění může být kontrola kvality velkého množství miniaturních součástek na výrobních linkách. Strukturu takového systému pak obvykle tvoří jednotky pro:

- získání dat,
- předzpracování,
- extrakci příznaků,
- detekci objektů,
- vyšší úroveň zpracování.

Rozdělení na výše uvedené jednotky lze pozorovat i v návrhu systému, který je předmětem této práce.

3.2.1 Vstupní signál

Vstupem do realizovaného systému jsou, jak již bylo uvedeno, obrazová data. Ta mohou pocházet buď z kamery nebo videosouboru (v případě testování). Pro dnešní mobilní zařízení je

charakteristická poměrně špatná kvalita obrazu získaného z jejich vestavěné kamery (viz sekce 2.2.2). Vstupní data převedená do digitální podoby pak lze charakterizovat vzorkovací frekvencí, rozlišením jednotlivých snímků a formátem, v němž jsou snímky kódovány. Dříve než začne samotné zpracování dat, je nutné je převést do vhodného formátu.

3.2.1.1 Formáty video dat

Existuje několik používaných formátů kódování videa (bez použití komprese). Ty definují, jak je barevná informace uložena v datech. Jednotlivé formáty se liší strukturou dat, množstvím bitů použitých na popis jednoho pixelu (bits per pixel, bpp). Různé formáty také definují reprezentaci barev v různých barevných modelech. Nejčastěji je použita jedna z variant barevných modelů YUV nebo RGB. K identifikaci datového formátu se běžně používá kód FourCC (Four Character Code), což je vlastně sekvence čtyř bytů, unikátních pro každý formát. Tento způsob pochází z operačních systémů Mac OS, kde vznikl pod názvem OSType. Postupem času se ale rozšířil do mnoha dalších technologií včetně QuickTime a DirectShow. Podrobný popis těchto formátů a jejich FourCC kódů lze nalézt například na www.fourcc.org [10].

3.2.1.2 Barevné modely

Barevný model je abstraktní matematický model popisující způsob, jakým lze reprezentovat barvy pomocí n-tice čísel (obvykle tři nebo čtyři složky) [11]. Pokud je tento model asociován s přesným popisem, jak mají být jednotlivé složky interpretovány, pak je výsledná množina barev nazývána barevným modelem.

3.2.1.3 Barevný model RGB

Složkami tvořícími barvu v modelu RGB jsou červená (R), zelená (G) a modrá (B). RGB je model aditivní, což znamená, že smícháním všech barev získáme bílou. Tato vlastnost předurčuje model RGB k použití u zobrazovacích zařízení, jako jsou například monitory (včetně displaye u mobilních zařízení). V souvislosti s tímto modelem se objevuje i pojem RGBA, což je rozšíření o další složku, kterou je průhlednost (Alpha). Nejčastěji použité videoformáty na mobilních zařízeních používají 1 byte pro každou složku (celkem 24 bitů) nebo 5 bitů pro každou složku nebo 5 bitů pro červenou a modrou a 6 pro zelenou.

3.2.1.4 Barevný model YUV

Model YUV pochází ze světa televizních signálů. Často je použit u vestavěných kamer mobilních zařízení. Existuje mnoho variant (YIQ, YCbCr, ...), jejichž společným rysem je oddělení jasové složky (Y) od barevných informací (U, V). Toto rozdělení umožňuje například využít stejnou jasovou složku v barevných i černobílých televizorech. Této vlastnosti je také využito v mnoha formátech, které tak mohou kódovat barevnou informaci menším počtem bitů než jasovou složku, která je důležitější. Videoformáty postavené na tomto modelu se dělí na dvě skupiny. Jednu skupinu tvoří

takzvané zhuštěné (packed) formáty, kdy jsou všechny tři složky v jednom poli. Druhá skupina takzvaných rovinných (planar) formátů má pro každou složku jedno pole, výsledný obraz je pak rekonstruován z těchto tří polí.

3.2.1.5 Převod YUV na RGB

Výstup z vestavěné kamery bývá u mobilních zařízení v různých formátech, tedy i v různých barevných modelech. Z toho důvodu je v mnoha případech nutné mezi barevnými modely provádět konverzi [12].

Převod RGB na YUV:

$$Y = + 0.299\ R + 0.587\ G + 0.114\ B$$
$$U = - 0.14713\ R - 0.28886\ G + 0.436\ B$$
$$V = + 0.615\ R - 0.51499\ G - 0.10001\ B$$

Převod YUV na RGB:

$$R = Y + 1.13983\ V$$
$$G = Y - 0.39466\ U - 0.58060\ V$$
$$B = Y + 2.03211\ U$$

U je z intervalu <-0.436, 0.436>
V je z intervalu <-0.615, 0.615>
R,G,B, Y jsou z intervalu <0, 255>

Existuje také efektivnější varianta konverze pouze s použitím operací s pevnou řadovou čárkou.

3.2.1.6 Rozlišení snímku

Kamery v mobilních zařízeních ve většině případů nabízí volbu rozlišení, v jakém mají být vzorkovaná obrazová data posílána na výstup. Vzhledem k principu mnoha algoritmů zpracování obrazu lze předpokládat, že výkon řešeného systému bude do značné míry závislý na rozlišení zpracovávaných snímků. To jistě platí například u algoritmů filtrace obrazu, kdy při provádění konvoluce je obraz procházen pixel po pixelu. Výpočetní výkon mobilních zařízení je v současné době velmi limitován, proto by bylo vhodnější zvolit spíše menší hodnoty rozlišení. Příliš malé rozlišení má ovšem negativní vliv na výsledek některých obrazových operací. Výběr správného rozlišení by tedy měl být kompromisem mezi výkonem systému a kvalitou jeho výstupu.

3.2.1.7 Vzorkovací frekvence

Hodnota vzorkovací frekvence vstupních dat je závislá na vlastnostech kamery v zařízení, na němž systém poběží. V současnosti jsou běžné hodnoty kolem 25 snímků za sekundu. Tato hodnota je pak po celou dobu vzorkování konstantní. Vzhledem k výpočetní náročnosti prováděných operací a možnostem hardwaru budou v průběhu zpracování aktuálního snímku další dodávané snímky vynechávány. Výsledná frekvence zpracování tak bude záležet na výkonu konkrétního zařízení. Toto bude platit alespoň v nejbližší době, v budoucnu se bude se zvyšujícím se výkonem mobilních zařízení hodnota frekvence zpracování snímků také zvyšovat. Systém je přitom navržen tak, aby tato frekvence nepřesáhla frekvenci vzorkovací. Minimální vzorkovací frekvenci a frekvenci, s jakou musí systém vstupní data zpracovávat, lze určit na základě jevů, které mají být ve vstupním videosignálu detekovány. Uvažujeme-li detekci změny slajdu, která v reálném případě nepřekročí frekvenci 1 snímek za sekundu, měly by na základě vzorkovacího teorému stačit 2 snímky za sekundu. V případě detekce animace, kdy se jako animace předpokládá změna snímku s frekvencí alespoň 4 snímky za sekundu, pak dle stejné úvahy vychází jako potřebná frekvence 8 snímků za sekundu.

3.2.2 Filtrace obrazu

Filtrace obrazu je operace, při níž dochází ke změně hodnot obrazových bodů. Často jsou používány konvoluční filtry.

Konvoluční filtry jsou charakteristické tím, že mají definovánu filtrační matici hodnot (konvoluční jádro, maska). Tato matice je čtvercová a má obvykle rozměry 3 × 3 hodnot a více. Filtrace pak spočívá v provedení konvoluce, tedy procházení obrazu pixel po pixelu. Konvoluci lze vyjádřit následujícím vzorcem:

$$C[m][n] = \sum_{u}\sum_{v}(A[m+u][n+v] * H[u][v]),$$

ve kterém C je výstupní obraz, A vstupní obraz a H konvoluční jádro, m a n jsou indexy pixelů v obraze, u a v jsou indexy hodnot v konvolučním jádře.

V každém kroku je konvolučním jádrem překryt vstupní obraz a hodnotami jádra vynásobeny odpovídající pixely obrazu. Součet těchto násobení pak tvoří novou hodnotu pixelu. Následuje posunutí jádra o pixel a pokračuje se dalším krokem. Existuje několik druhů filtrů, například vyhlazovací, hranové, derivační, integračně derivační apod. Některé z nich jsou použity i v této práci.

3.2.3 Extrakce příznaků

Extrakce příznaků (feature extraction) je zvláštním případem dimenzionální redukce, tedy jakýmsi zjednodušením informací potřebných pro popis velkého množství dat. Metody spadající do této množiny jsou často užívány pro snížení počtu proměnných popisujících data určená k analýze. Počet těchto proměnných bývá redukován tak, aby byla zachována dostatečná přesnost v popisu dat. Tím

jsou zajištěny menší nároky na paměť, výpočetní náročnost či velikost trénovací množiny u klasifikačních algoritmů. V oblasti zpracování obrazu jsou to například algoritmy detekce hran, detekce pohybu, prahování, detekce geometrických objektů apod. V tomto projektu je například použita detekce přímek s pomocí Houghovy transformace.

3.2.3.1 Detekce hran

Hrany v obraze jsou oblasti s velkou změnou jasu. Ve snímané scéně pak mohou hrany odpovídat důležitým jevům, jako jsou například hranice objektů. Detekce hran tak zaujímá důležité místo v oblasti zpracování obrazu a počítačového vidění (extrakce příznaků). Významnou vlastností metod detekce hran je výrazné snížení množství dat, která nejsou důležitá při analýze struktury obrazu. Tyto metody lze rozdělit na metody založené na [14]:

- první derivaci,
- druhé derivaci,
- parametrickém modelu hran.

Metodami založenými na první derivaci jsou například Sobelův operátor, Kirschův operátor, Robertsův operátor, Robinsonův nebo Prewittův operátor. Společnou charakteristikou těchto metod je příslušnost ke konvolučním filtrům a porovnání výsledku konvoluce s prahem. Z toho vyplývá analýza malého lokálního okolí, citlivost na šum a značná závislost na velikosti objektu jemuž přísluší hrany.

Další skupinu tvoří metody, jež provádí druhou derivaci obrazu a zkoumají průchod nulou. Příkladem z této skupiny je Laplacian. Mnoho z těchto metod opět používá konvoluci. Nevýhodou těchto metod je příliš velké rozmazání či občasná dvojitá odezva na některé hrany.

V poslední skupině jsou metody založené na parametrickém modelu hran. Typickým zástupcem je Houghova transformace. Značnou nevýhodou těchto algoritmů jsou jejich výpočetní nároky, jsou však odolné vůči porušeným datům nebo šumu.

3.3 Návrh řešení

Kvalita výsledků aplikace se odvíjí od schopnosti detekovat změnu slajdů či případnou animaci ve zpracovávaném obraze. Vzhledem k podmínkám, za jakých má systém pracovat, je proto potřeba řešit hned několik druhů problémů:

- problémy vycházející z charakteru prostředí (pohybující se kamera, pohybující se osoby mezi plátnem a kamerou),
- problémy spojené s kvalitou vstupního obrazu (nízké rozlišení, šum, atd.),
- problémy vyplývající z nedostatečného výkonu hardwaru (omezení mobilních zařízení).

3.3.1 Detekce plátna

Pro úspěšné provádění detekce změny slajdu je nejprve nezbytné ve snímaném obraze nalézt plátno, na nějž jsou slajdy promítány. V této práci je použit postup do značné míry inspirovaný [9].

Aby bylo vůbec možné plátno detekovat, bylo nutné stanovit určité předpoklady. Tyto předpoklady pak umožňují odlišit plátno od ostatních objektů ve snímaném obraze. Předpoklady jsou tedy tyto:

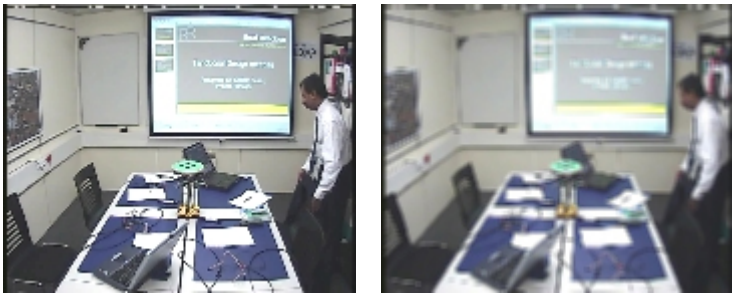
- okraje plátna jsou tvořeny hranami v oblasti úrovní jasu,
- plátno je čtyřúhelník,
- každá dvojice tvořená vertikální a horizontální přímkou ohraničující plátno spolu svírá úhel blízký pravému úhlu,
- nejjasnější oblast obrazu je v oblasti plátna,
- obraz je intuitivně snímán tak, že přímky ohraničující plátno jsou zhruba rovnoběžné s přímkami tvořícími okraj obrazu.

Při výběru jednotlivých metod zpracování obrazu musel být také zohledněn nízký výkon mobilních zařízení. Na výkonnějších platformách je jistě vhodné použít další metody, které by vedly ke kvalitnějšímu výsledku.

Následuje popis jednotlivých kroků detekce plátna.

3.3.1.1 Blur

Blur, neboli rozmazání, je velmi vhodná operace pro předzpracování obrazu, ve kterém mají být následně detekovány hrany. Jde o konvoluční filtr, který každému pixelu obrazu přiřadí průměrnou hodnotu okolních pixelů. V této práci je použita varianta filtru zvaná Box Blur, která používá konvoluční jádro složené z prvků stejné hodnoty. Box Blur je aproximací dokonalejšího filtru Gaussian Blur. Při vícenásobném použití Box Bluru na vstupní obraz lze získat téměř shodný výsledek jako při použití Gaussian Bluru. Výhodou Box Bluru je však fakt, že díky stejným hodnotám v konvolučním jádře jej lze s výhodou optimalizovat. Právě z důvodu menší výpočetní náročnosti je v této práci použit Box Blur. Výsledkem aplikace tohoto filtru na vstupní obraz je potlačení negativního vlivu šumu v obraze na další zpracování.



Obrázek 3.1 původní obraz (vlevo) a po aplikaci filtru Box Blur (vpravo)

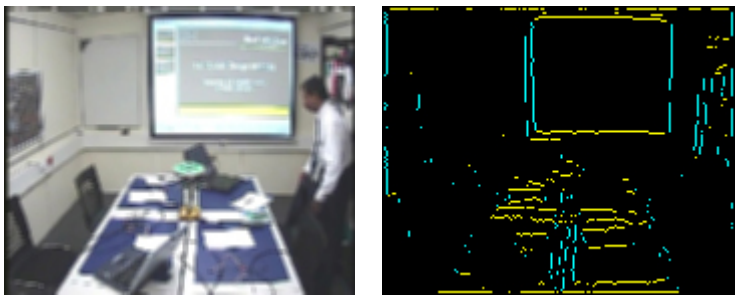
3.3.1.2 Sobelův hranový operátor

Máme-li obraz předzpracovaný rozmazáním, můžeme přistoupit k vyhledání hran. Jako nejvhodnější metoda se ukázal Sobelův hranový detektor. Je to vlastně konvoluční filtr, který používá dvě jádra. Jedno pro detekci horizontálních hran a druhé pro detekci vertikálních hran.

$$\begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix} \qquad \begin{pmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Oproti jiným metodám (Laplacian) má jednu vlastnost, kterou lze hlavně v tomto případě považovat za velmi výhodnou. Je jí detekce orientace hrany v procházeném pixelu. Vycházíme-li z předpokladu, že objekt plátna je v obraze zastoupen čtyřúhelníkem se stranami přibližně rovnoběžnými s osami souřadného systému, můžeme ignorovat ty pixely, které byly algoritmem detekovány jako hranové, ale orientace jim příslušné hrany není v tolerovaném rozmezí od souřadných os.

Sobelův operátor má i jeden nežádoucí efekt. Vzhledem k tomu, že tato metoda aproximuje gradient funkce jasu, je detekovaná hrana většinou v obraze zastoupena pruhem o šířce několika pixelů s jasem odpovídajícím právě gradientu. Pro další zpracování je ovšem vhodné, aby nalezená hrana byla reprezentována co možná nejtenčí linií. Toho lze dosáhnout použitím některých sofistikovaných metod, které ale zvýší náročnost celkového výpočtu. Místo nich můžeme opět využít předpokladu přibližné rovnoběžnosti hran s osami a toho, že známe orientaci hran. Procházíme-li v průběhu detekce obraz shora dolů a zleva doprava, pak stačí nulovat hodnoty jasu (což odpovídá pixelům potencionálně nalezené hrany) nad nalezenou horizontální hranou nebo vlevo od nalezené vertikální hrany. Tím dosáhneme jakéhosi ztenčení hran s prakticky nulovým zvýšením náročnosti výpočtu. V kombinaci s jednoduchým prahováním je pak dosaženo uspokojivého výsledku.



Obrázek 3.2 obraz po aplikaci Box Blur filtru (vlevo) a po aplikaci Sobelova hranového operátoru (vpravo), modře jsou vyznačeny vertikální, žlutě horizontální hrany

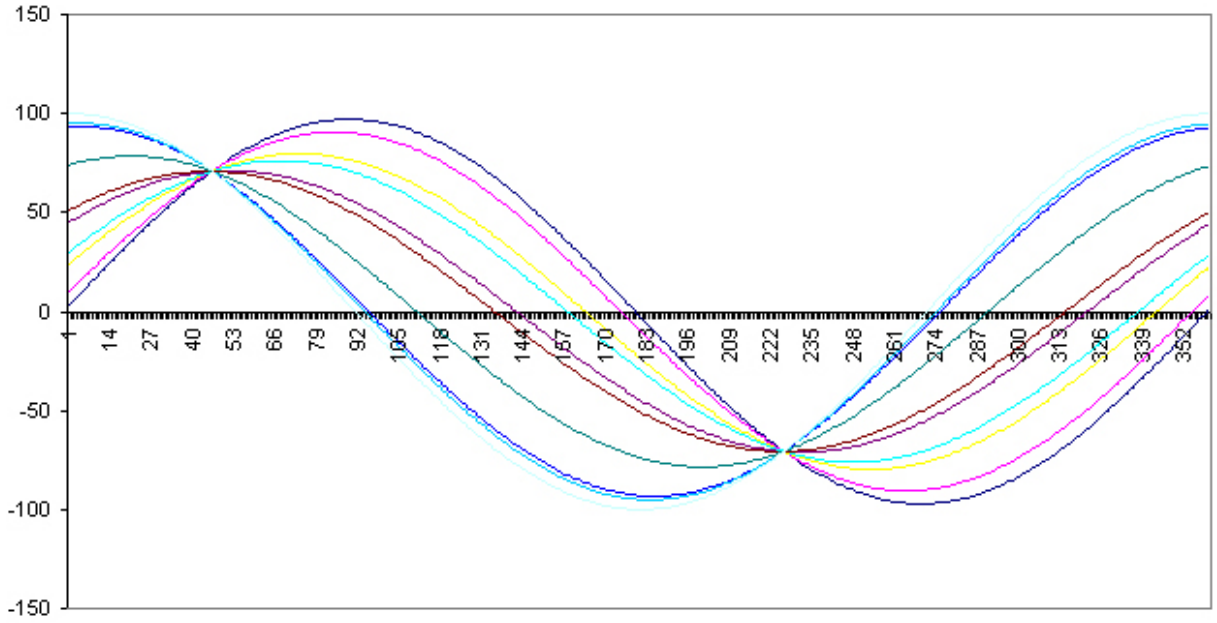
3.3.1.3 Houghova transformace

Nyní již máme dostatečně předzpracovaný obraz, abychom mohli přistoupit k fázi extrakce příznaků. V té je nejlepším řešením nalézt přímky odpovídající okrajům hledaného plátna. Jako nejvhodnější metoda se jeví Houghova transformace. Jde o velmi robustní metodu, která je odolná vůči přerušení čar, což je právě v tomto případě velmi vhodné. Nezanedbatelnou implementační výhodou je, že výsledné informace o nalezeném plátně je možné vyjádřit jako vektor osmi čísel (dvě hodnoty pro každou ze čtyř přímek).

Houghova transformace je metoda, díky níž je možné nalézt v obraze přímky. Jde však zobecnit i na vyhledání jiných geometrických útvarů (kružnice, elipsy). Tato metoda využívá toho, že přímku lze v parametrickém prostoru, kde parametry jsou úhly přímky a její vzdálenost od počátku souřadnic, vyjádřit jako bod. Dále pak pracuje s přímkou v normálovém tvaru:

$$r = x.\cos\theta + y.\sin\theta$$

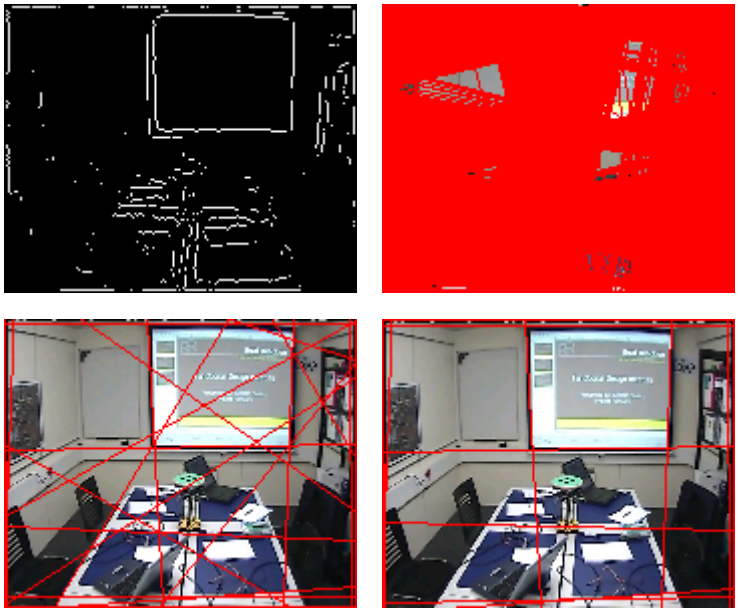
Algoritmus pak nejdříve sestaví dvourozměrnou tabulku, v níž jedním parametrem jsou úhly od 0° do 360° a druhým vzdálenost od počátku souřadnic od 0 do hodnoty rovnající se velikosti úhlopříčky obrazu. Postupně pak prochází body obrazu, tedy body, jež náleží hranám nalezeným v předchozím zpracování. U každého takového bodu inkrementuje hodnoty na souřadnicích přímek v tabulce, ke kterým bod přísluší. V této tabulce se nakonec vyhledají lokální extrémy, které určují nalezené přímky.



Obrázek 3.3 body ležící na jedné přímce, vyjádřeny orientací a vzdáleností od počátku soustavy souřadnic

Z předchozího popisu je zřejmé, že jde o výpočetně velmi náročný algoritmus. I zde je ovšem možné s výhodou využít předpokladu o přímkách příslušících okrajům plátna a úhlu svíraném s osami. Není ovšem nutné procházet u každého bodu celý rozsah 0° až 360°, postačí jen intervaly odpovídající tolerované odchylce od os souřadného systému. Tím je dosaženo výrazného zrychlení. Nadále však platí velká závislost na počtu procházených bodů obrazu. Jejich minimalizace v předchozích krocích zpracování je tedy velmi podstatná.

Výběr jen těch nejvýznamnějších přímek opět zohledňuje výpočetní možnosti cílové platformy. To je v této práci provedeno kombinací prahování se shlukováním přímek. Z tabulky udávající četnosti bodů náležících k přímkám jsou vybrány jen ty přímky, u nichž tato hodnota přesahuje definovaný práh. Dále je procházena tato tabulka a přímky jsou slučovány s těmi, které mají větší hodnotu v tabulce a nejsou v parametrickém prostoru vzdáleny více, než určuje předem definovaný práh.



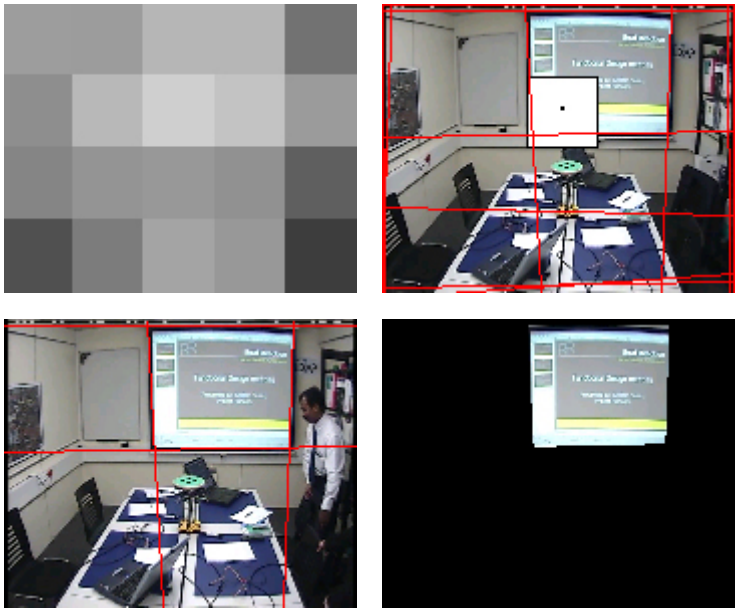
Obrázek 3.4 obraz po vyhledání hran (vlevo nahoře), po aplikaci Houghovy transformace (vpravo nahoře), po shlukování přímek (vlevo dole), po výběru přímek přibližně rovnoběžných s osami (vpravo dole)

3.3.1.4 Určení nejjasnější oblasti

Jak je možné vidět na obrázku 3.4, z několika málo přímek, jež jsou výstupem Houghovy transformace, ještě není jasné, které z nich ohraničují právě plátno. Aby bylo možné vybrat správné čtyři přímky, je nutné využít dalšího předpokladu. A sice, že plátnu odpovídá nejjasnější oblast vstupního obrazu. Proto je vstupní obraz podvzorkován tak, aby oblast odpovídající jednomu rozlišovacímu bodu v miniatuře byla menší, než nejmenší rozumná velikosti plátna v obraze (v tomto případě 1/20 velikosti obrazu). V těchto několika málo hodnotách pak lze rychle vyhledat bod s

největší hodnotou. Promítnutím tohoto bodu zpět do vstupního obrazu je získána oblast s největším jasem.

Vyhledání přímek ohraničujících plátno je poté otázkou nalezení přímek nejbližších k nejjasnější oblasti obrazu. To odpovídá následujícímu postupu. Nejprve je nutné určit střed nejjasnější oblasti. Poté průsečíky všech přímek s rovnoběžkami k osám souřadné soustavy procházejícími tímto středem. Nakonec se vyberou čtyři přímky s nejbližšími průsečíky zleva, zprava, zdola a shora.



Obrázek 3.5 obraz po vyhledání nejjasnější oblasti v obraze (vlevo) a její vyznačení v obraze s nejvýznamnějšími přímkami (vpravo), obraz po výběru přímek ohraničujících plátno (vlevo), po odfiltrování okolí plátna (vpravo)

3.3.1.5 Aktualizace pozice plátna

Protože se předpokládají výchylky kamery ze své pozice, musela by detekce pozice plátna v obraze v ideálním případě probíhat opakovaně v každém snímku. Výpočetní náročnost však tento přístup v současné době značně diskriminuje. Řešením by mohlo být opětovné vyhledávání plátna v obraze, až když je detekována změna právě v oblasti plátna, tedy změna slajdu. Při příliš velkém trvání vyhledání pozice plátna má tento přístup negativní vliv na schopnost detekovat animace. V této práci je realizován právě tento způsob. Další, ovšem výpočetně značně náročnější, možností je porovnávat pozice odpovídajících si významných bodů obrazu.

3.3.2 Detekce změny slajdu

Jádrem funkčnosti systému je detekce změny slajdu. Asi nejjednodušším způsobem, jak změnu zaznamenat, je prosté porovnání dvou po sobě jdoucích snímků. Díky předchozí lokalizaci plátna

v obraze je možné detekci provádět jen v oblasti obrazu, která nás skutečně zajímá. Nabízí se hned několik možností:

- porovnání všech pixelů obou snímků na odpovídajících si souřadnicích s definovaným prahem; výsledkem porovnání je počet odlišných pixelů; změna je detekována pokud tento počet přesáhne určený práh; výsledek algoritmu bude jistě značně ovlivněn výchylkami kamery ze své pozice,
- výpočet rozdílu sum všech pixelů v obou snímcích; použití prahu pro detekci změny; algoritmus však není schopen detekovat změnu pozice objektu na plátně, případně jeho pohyb (jednoduchá animace),
- porovnání histogramů obou snímků; opět algoritmus nerozezná změnu pozice či pohyb objektu,
- detekce na základě korespondence význačných bodů v obraze; imunní vůči výchylkám pozice kamery, ale výpočetně náročná,
- výpočet mediánu histogramu snímku, jenž je výsledkem rozdílu odpovídajících si pixelů dvou následujících snímků; použití prahu pro rozhodnutí o případné změně,
- výpočet mediánu histogramu snímku, jenž je výsledkem rozdílu odpovídajících si pixelů vybraného snímku a snímku vzniklého průměrováním předchozích; použití prahu pro rozhodnutí o případné změně; oproti předchozí metodě je méně citlivá na šum.

3.3.3 Detekce animace

Nejjednodušší detekce animace představuje použití stejných algoritmů jako při detekci změny slajdů, s tím rozdílem, že pro rozhodnutí o výsledku je bráno v potaz větší množství snímků. O tom, zda je detekována animace nebo jen prostá změna slajdu, je v této práci rozhodováno na základě časového intervalu od poslední změny v obraze a nastaveného prahu. Použití sofistikovanějších metod založených na sledování lokálních změn, například s pomocí výpočtu optického toku nebo sledování pohybu objektů, je vzhledem k možnostem hardwaru velmi omezeno.

3.3.4 Detekce osob mezi plátnem a kamerou

Další možnou úpravou je detekce pohybujících se osob v obraze a následné využití informací o jejich pozici a pohybu při detekci změny slajdu. Pohyb přednášejícího nebo dalších osob totiž ovlivňuje většinu algoritmů pro detekci změny. Oproti nejjednodušším metodám detekce pohybujících se objektů musí taková detekce ovšem zohledňovat fakt, že kamera není statická. Výpočet pohybujících se objektů musí být vztažen k nějakému významnému objektu, u nějž lze předpokládat, že je nehybný. Výhodné je využít takového již nalezeného objektu, a to plátna s prezentací. Následuje návrh algoritmu vyhledání těchto osob v obraze, který není v této práci implementován. Postup detekce by mohl být následující:

- předpokládáme, že osoby se pohybují jen ve směru jedné z os soustavy souřadnic; dále že kamera je dostatečně daleko od plátna a pohyb kamery je velmi malý v porovnání s touto vzdáleností,
- s pomocí informací o plátně v dvou po sobě jdoucích snímcích určit geometrickou transformaci obrazu; usnadněním může být předpoklad o pohybu kamery, můžeme tedy brát v úvahu jen translaci,
- se zohledněním transformace hledat změnu mezi obrazy, která se děje mimo oblast plátna,
- určit obdélník obalující oblast s detekovanou změnou; strany obdélníku jsou rovnoběžné s transformovanými osami soustavy souřadnic; obdélník reprezentuje nalezený, pohybující se objekt,
- aktualizovat informace o poloze již dříve nalezených objektů,
- pokud nějaký dříve nalezený objekt nelze přiřadit a byl v blízkosti plátna, pak v případě, že je pod plátnem detekován menší objekt na přibližně shodné pozici (lze určit z předpokladu o pohybu v jednom směru), aktualizovat polohu podle něj, jinak predikcí určit polohu objektu v oblasti plátna (oblast pod plátnem nemusí být součástí záběru),

Úpravou algoritmů pro detekci změny v oblasti plátna je pak možné ignorovat při rozhodování ty části plátna, kde se nachází detekované objekty.

4 Vývojové prostředky

4.1 DirectShow

Ještě donedávna nenabízely systémy Windows Mobile vývojářům žádné API (Application Programming Interface), které by umožňovalo sofistikovanější práci s kamerou na mobilním zařízení. Od verze 5.0 je však systém vybaven DirectShow API. To v současné době představuje prakticky jedinou možnost, jak zpracovávat video z kamery na systému Windows Mobile. Protože DirectShow umožňuje přistupovat k jednotlivým snímkům z kamery, je vstup dat do systému řešeného v této práci realizován právě tímto API.

DirectShow (zkracováno jako DS, DShow) je rozhraní pro zpracování multimediálních dat, konkrétně zvukových a obrazových datových toků (streamů). Toto API je vyvíjené firmou Microsoft. Jeho účelem je zjednodušení práce s multimediálními aplikacemi na platformě Microsoft Windows, izolací aplikací od složitosti datových transportů, hardwarových odlišností a synchronizace. Hlavní využití je tedy v realtime přehrávání, nahrávání a editaci multimediálních dat, dále v ovládání zařízení pro snímání obrazu používajících WDM (Windows Driver Model) a VFW (Video For Windows), podporovány jsou též zařízení jako TV karty, kamery, videorekordéry a další. Podporuje též množství multimediálních formátů a kodeků, jako například `avi` (Audio-Video Interleaved), `mpeg` (Motion Picture Experts Group), `wmv` (Windows Media Video), `asf` (Advanced Streaming Format), `wav` (Waveform audio format), `mp3` (MPEG-1 Audio Layer 3). Vzhledem k původu DirectShow, je jeho použití omezeno výhradně na systémy Microsoft Windows. DirectShow vzniklo v dobách systému Windows 3.0 z technologie ActiveMovie jako konkurence k QuickTime firmy Apple Computers, v té době nejpoužívanější multimediální platformě. Původně bylo vyvíjeno pod kódovým jménem Quartz, hlavní dynamická knihovna se dodnes jmenuje quartz.dll. Z pohledu operačního systému tvoří DirectShow vrstvu mezi aplikací a hardwarem. Pokud je dostupná hardwarová akcelerace, DirectShow ji využije. DirectShow je objektově orientované a využívá objektového modelu COM. V současnosti má DirectShow širokou podporu mezi programovacími jazyky, například C++, C#, Java, Visual Basic, Pascal.

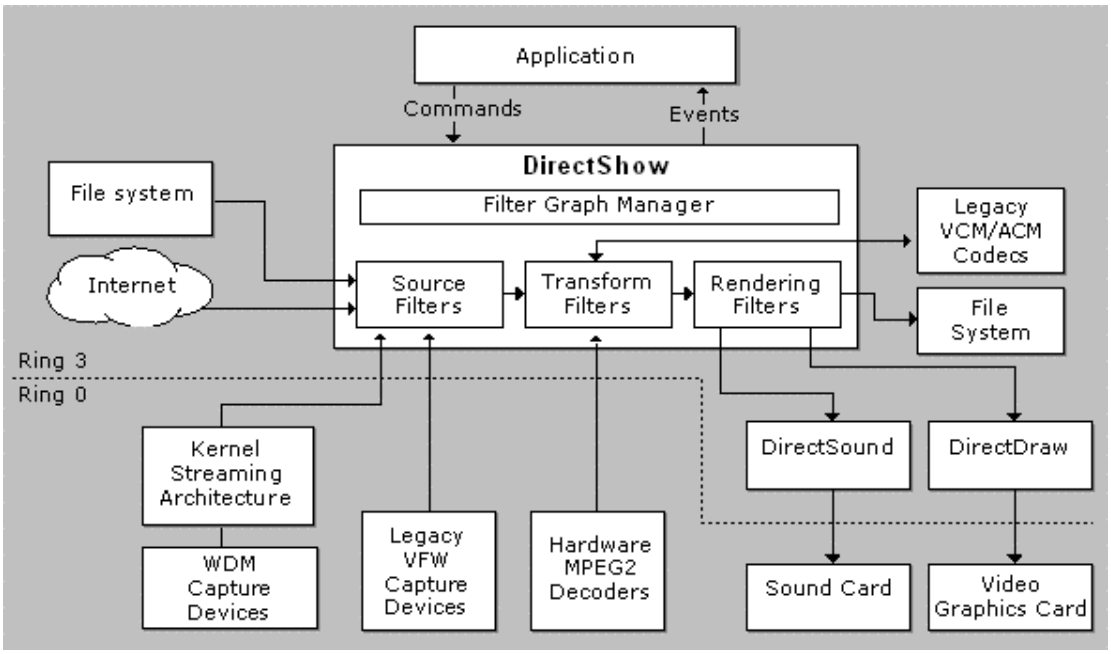
4.1.1 Historie

V roce 1998 bylo Active Movie definitivně přejmenováno na DirectShow a distribuováno jako součást DirectMedia SDK. Po té se stalo součástí kolekce multimediálních API zvané DirectX (bylo to v době verze 7). V dubnu 2005 bylo DirectShow přesunuto do Microsoft Platform SDK. V roce 2006 bylo ale odstraněno i z Microsoft Platform SDK. Až do současné doby existuje jediný způsob jak jej získat, a to nainstalovat Platform SDK, verzi z dubna 2005.

DirectShow je nástupcem dřívějšího rozhraní Video for Windows. S příchodem operačního systému Windows Vista se objeví nástupce DirectShow, rozhraní nazvané Media Foundation, které je opět založeno na technologii COM. Hlavními vylepšeními by měla být podpora ochrany obsahu, vyšší kvality audia a videa, technologie Digital Rights Management (DRM) a další.

4.1.2 Princip práce s DirectShow

V DirectShow existuje několik základních pojmů. Jsou to: graf filtrů, filtr a pin. Graf je tvořen navzájem propojenými filtry a určuje cestu, kterou projdou data od svého zdroje do cíle.

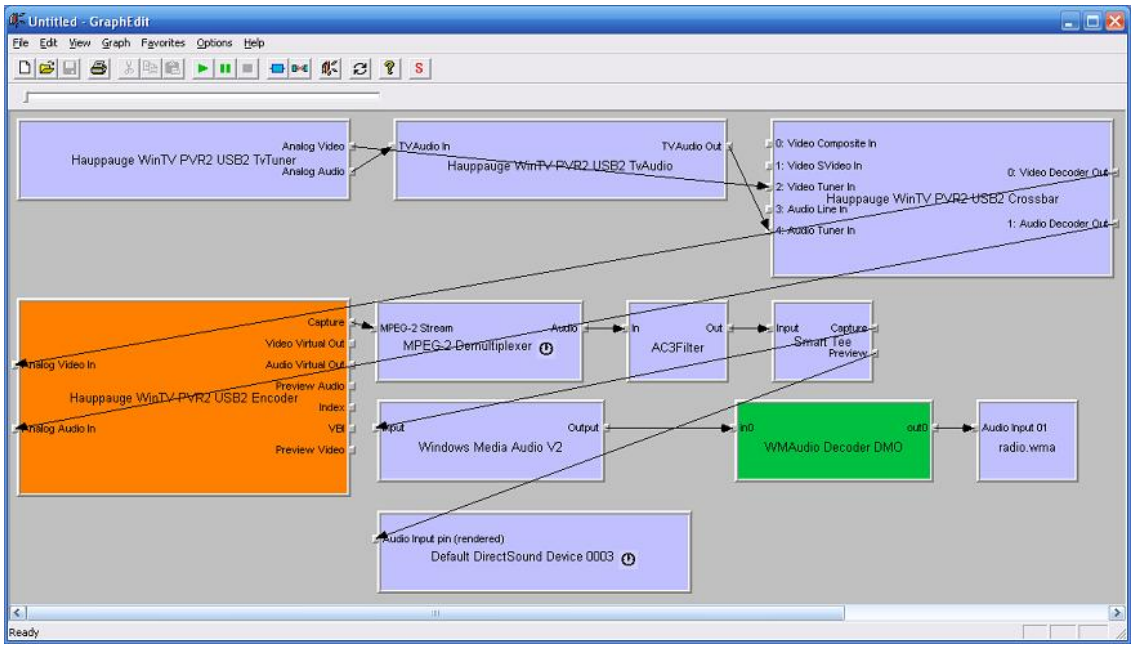


Obrázek 4.1 vztah mezi aplikací, DirectShow komponentami a hardwarovými a softwarovými komponentami, které DirectShow podporuje (převzato ze stránek msdn.microsoft.com).

4.1.3 Graf

Graf musí začínat vstupním filtrem a končit výstupním. Pokud je graf korektně zapojen, prochází data postupně ze zdrojového filtru skrz další zapojené filtry až do výstupních filtrů. Toto se děje souběžně. Data tedy jakoby protékala grafem, proto pojem datový tok (stream). Graf může tímto způsobem zpracovávat více datových toků současně. Příkladem může být souběžné nahrávání obrazových dat z webkamery a zvukových dat z mikrofonu. Data v tomto případě pochází ze dvou na sobě nezávislých zdrojových filtrů a s pomocí jiného filtru jsou spojována dohromady do jednoho audio/video streamu. Struktura grafu může být měněna na základě kódu aplikace, nezpracovává-li graf právě nějaká data. Jednotlivé filtry grafu mohou být odstraněny, nahrazeny jinými nebo může být změněno jejich vzájemné propojení. Typicky je graf sestavován během inicializace aplikace. Graf se

může nacházet v několika stavech. Nejprve je graf sestaven, spuštěn, poté může být pozastavován a znovu pouštěn a na konci ukončen.



Obrázek 4.2 ukázka grafu filtrů zobrazeného aplikací GraphEdit

(převzato z www.mperfect.net/mceDirectShow/)

4.1.4 Filter Graph Manager

Pro správu filtrů v grafu existuje v DirectShow objekt zvaný Filter Graph Manager. Ten obsahuje metody pro sestavení a používání grafu. Práce s grafem pak představuje volání příslušných metod objektu nebo metod rozhraní, jež Filter Graph Manager implementuje, a obsluhu událostí, prostřednictvím kterých tento objekt komunikuje s aplikací. Existují například metody pro přidávání filtru do grafu, odebírání filtru z grafu, propojování a rozpojování filtrů, práci s piny, spuštění či zastavování grafu. Sestavení grafu lze do jisté míry zjednodušit s použitím mechanizmu Intelligent Connect. Ten využívá několik algoritmů, díky nimž sám vybere a spojí příslušné filtry do grafu. Vybírá mezi filtry již obsaženými v grafu, případně přidá filtry zajišťující potřebnou konverzi dat mezi jinými dvěma. Případné využití tohoto mechanizmu v sobě ovšem skrývá nevýhody vyplývající z programátorovy neznalosti struktury takto vzniklého grafu.

4.1.5 Filtry

Během cesty grafem procházejí data právě filtry. Filtr je tedy jakýsi objekt, který má svou specifickou funkci z pohledu práce s datovým tokem. Filtry v sobě ukrývají své vnitřní operace. Programátor nepotřebuje vědět, co se uvnitř děje s daty a je tak odstíněn od řešení problémů spojených s konkrétními datovými formáty a operacemi nad nimi. Filtr může být zdrojem dat, data zpracovávat

nebo je prezentovat na nějakém zařízení. Filtr může modifikovat data ve streamu, nemůže však měnit strukturu grafu. Rozeznáváme tyto druhy filtrů:

- zdrojové,
- transformační,
- výstupní (renderovací),
- oddělovací (splitter),
- spojovací (mux).

Úkolem zdrojových filtrů je vstup dat do grafu, může se jednat o vstup z kamery, souboru, sítě apod. Renderovací filtry jsou zakončením grafu a jejich funkcí může být například prezentace zpracovávaných dat. Tyto filtry často využívají jiných technologií. Například přehrávání zvuku je uskutečněno prostřednictvím DirectSound (součást DirectX). Pro zobrazování videa bylo nejdříve použito GDI (Graphics Device Interface), ve verzi 6.0 se objevil Overlay Mixer využívající DirectDraw 6 (rovněž součást DirectX), s příchodem Windows XP přibyl Video Mixing Renderer 7 (VMR-7) využívající DirectDraw 7 a spolu se vznikem DirectX verze 9 lze zobrazovat i pomocí Video Mixing Renderer 9 (VMR-9) řešící vykreslování s pomocí Direct3D. Transformační filtry jsou zapojeny mezi vstupními a renderovacími. Jejich typickou úlohou je komprese nebo dekomprese dat. Splitter filtry rozdělují vstup na několik výstupů, příkladem může být rozdělení na audio a video data. Naopak Mux filtry mají opačný úkol – spojení dat z několika vstupů do jednoho výstupu, například spojení audio a video dat a zapsání do souboru. Všechny filtry musí mít určité společné vlastnosti, jako například schopnost přijímat tři základní zprávy: start, stop, pause, díky nímž lze kontrolovat provádění grafu.

4.1.6 Piny

Všechny filtry jsou také opatřeny vstupními a výstupními piny. Každý filtr musí mít alespoň jeden pin. Vstupní filtry nemají vstupní piny a renderovací zase nepotřebují výstupní piny. Pin je prostředkem pro propojení dvou filtrů. Každý pin akceptuje určitý typ dat (media type). Dva filtry lze propojit, pouze pokud vstupní pin akceptuje typ dat výstupního pinu.

4.1.7 Sample Grabber

Zvláštním filtrem, který je důležitý v aplikacích zpracovávajících obrazová data je Sample Grabber. Je to filtr, který umožňuje přijímat a zpracovávat vzorky dat (samples) při jejich průchodu grafem filtrů. Protože je to transformační filtr, má jeden vstupní a jeden výstupní pin. Tento filtr samotná data nemění, proto je možné jej bez obav zapojit do grafu. Aplikace je pak schopna přijímat jednotlivé vzorky dat voláním metod rozhraní *ISampleGrabber*, které filtr implementuje. Existují dva způsoby, jak vzorky získat. Jedním z nich je volání metody *GetCurrentBuffer* a získat tak přímo data nebo

voláním metody *SetCallback* nastavit funkci, které budou data předána při každém přijetí vzorku dat filtrem Sample Grabber.

4.2 Tvorba vlastního filtru

Spolu s DirectShow je distribuována řada filtrů, s jejichž pomocí dokáže programátor multimediální aplikace řešit většinu běžných úloh. Přesto je občas nucen přidat do své aplikace funkcionalitu, které by s žádným existujícím filtrem nedosáhl (například vlastní formát dat nebo efekt). Proto je velmi vítanou vlastností DirectShow možnost tvorby filtru vlastního. Z pohledu programátora multimediálních aplikací je filtr ve skutečnosti dynamickou knihovnou (konkrétně DLL), která má definovaný vstupní bod a podporuje některá COM rozhraní.

4.2.1 DLL

DLL (Dynamic Link Library) je implementací konceptu sdílené knihovny od společnosti Microsoft. Jde o knihovnu, tedy o jakýsi funkční logický celek, který poskytuje služby pro programy (sbírka procedur, funkcí a datových typů, či při objektově orientovaném přístupu sada tříd) a který je uložen v jednom diskovém souboru. Konkrétně se jedná o knihovnu, která je dynamicky propojována s programem na platformě Microsoft Windows. Programy spojené s dynamickou knihovnou jsou menší (ve spustitelném souboru se nachází jen výkonný kód programu), výhodou je možnost jednoduše zaměnit požadovanou dynamickou knihovnu za její novější verzi. Kód v dynamicky linkované knihovně je také za běhu sdílen mezi všemi aplikacemi, které jej používají, což šetří operační paměť. Tyto knihovny mají obvykle příponu `dll`, `ocx`, `drv` nebo `ax` v případě DirectShow filtrů. Formát souboru je stejný jako v případě spustitelných `exe` souborů, DLL tedy mohou obsahovat kód, data i zdroje (resources). Zvláštním případem pak jsou knihovny obsahující pouze zdroje, tzv. resource libraries, například knihovny ikon (přípona `icl`) nebo knihovny fontů (přípona `fon` nebo `fot`). Kromě výrazné úspory paměti je významnou vlastností DLL jejich modularita. V knihovně lze do jisté míry dělat změny, aniž by nastala nutnost pozměnit kód aplikace, která ji využívá. Knihovnu lze nahradit její novější verzí, to s sebou ale přináší problém zvaný „DLL hell“. K tomu dochází, pokud je v systému více aplikací využívající jednu knihovnu, přičemž každá vyžaduje jinou verzi. Tento problém se většinou řeší zkopírováním potřebné verze do adresáře s aplikací. Microsoft tuto často kritizovanou vlastnost řeší v platformě .NET přidáním možnosti koexistence dvou různých verzí jedné knihovny v systému. Kód DLL je v paměti přítomen jen jednou a je sdílen všemi procesy, které používají tuto DLL. V systému jsou DLL organizovány na sekce. Každá sekce má své atributy, jako zapisovatelná, pouze ke čtení, vykonavatelná (kód) nebo nevykonavatelná (data), atd. Datové sekce jsou většinou privátní, tzn. každý proces má svou kopii dat. Pokud je ale datová sekce sdílená, pak nastává nežádoucí bezpečnostní riziko, kdy jeden proces

může měnit chování jiných procesů změnou sdílených dat. Protože se uživatelská omezení nevztahují na použití sdílené paměti DLL, je o to nebezpečnější toto sdílení používat. Další důležitou vlastností DLL je možnost připojení za běhu aplikace (explicit runtime linking). V prostředí Microsoft Visual C++ existuje řada rozšíření ke standardnímu C++, která umožňují definovat u funkcí, zda budou importovány nebo exportovány a to přímo v C++ kódu. Tato rozšíření používají atribut *__declspec* v deklaraci funkce. Pokud externí symbol podléhá pojmenovávacím konvencím jazyka C, musí být deklarovány jako *extern "C"*. Asi nejkomplexnějším příkladem tvorby a použití DLL je technologie COM. Každá COM dynamická knihovna musí exportovat čtyři funkce, *DllMain*, *DllCanUnloadNow*, *DllRegisterServer* a *DllUnregisterServer*. *DllMain* je základní funkce všech DLL. Volá se v okamžiku, kdy se DLL nahrává do paměti. Obsahuje různé inicializační rutiny pro globální a statické objekty. *DllCanUnloadNow* vrací informaci (*S_OK* nebo *S_FALSE*) o tom, zdali je či není knihovna používána, aby mohla být případně vymazána z paměti. *DllRegisterServer* zapisuje informace o komponentách do systémových registrů, je volána aplikací REGSVR32. Funkce *DllUnregisterServer* naopak maže informace o komponentách z registrů (REGSVR32 s parametrem /u).

4.2.2 COM

Jak již bylo zmíněno, DirectShow staví na objektovém modelu COM. Psaní vlastního filtru znamená mimo jiné implementaci COM objektu. Z toho důvodu je pro vznik vlastního filtru nutné, aby programátor rozuměl alespoň základům této technologie.

4.2.2.1 Koncept COM

COM (Component Object Model) je platforma pro dynamickou komunikaci mezi procesy a dynamickou tvorbu objektů v libovolném programovacím jazyce, který tuto technologii podporuje. Její tvůrce, společnost Microsoft, ji představila v roce 1993, ve známost se však dostala až v roce 1997. COM umožňuje znovupoužití objektů bez toho, aby programátor znal jejich vnitřní implementaci. To je umožněno důsledným oddělením rozhraní od implementace. Zatímco v klasických objektově orientovaných jazycích je rozhraní součást objektu, v COM je to samostatná třída. V C++ je rozhraní abstraktní třída, která má definovány čistě virtuální funkce. Neobsahuje proměnné ani implementaci funkcí. Objekt je třída odvozená od rozhraní. Teprve v něm jsou ony virtuální metody přetíženy a implementovány. Objekt může nabízet i více rozhraní. Objekty jsou též samy zodpovědné za své vytvoření a zrušení s využitím počítadla referencí. Bezpečné přetypovávání mezi různými objekty je zajištěno funkcí *QueryInterface*(). Přestože je COM implementováno mnohými platformami, nejvyužívanější je na systémech Windows. COM je specifikace, nikoliv implementace. Definuje pravidla, která musí komponenty splňovat. Uplatnění těchto pravidel je ponecháno na programátorovi. V budoucnu se očekává nahrazení této technologie novější, zvanou Windows Communication Foundation (WCF). Od COM byly později odvozeny technologie COM+ a DCOM.

4.2.2.2 Práce s COM

Práce s COM je složena z používání COM komponent a jejich tvorby. Každý typ komponent má svůj unikátní identifikátor zvaný CLSID, který byl vytvořen metodou GUID (Globally Unique IDentifiers). Komponenta nabízí svou funkčnost skrze jedno nebo více rozhraní, která implementuje. I tato rozhraní jsou identifikovatelná svým IID (Interface ID, opět GUID). Každá komponenta musí implementovat alespoň standardní rozhraní *IUnknown*, od kterého jsou odvozena všechna ostatní. Toto rozhraní obsahuje tři metody:

- *AddRef()*,
- *Release()*,
- *QueryInterface()*.

První dvě zajišťují práci s počítadlem referencí a třetí s využitím IID specifikuje, na která rozhraní lze vrátit referenci, tedy která rozhraní komponenta implementuje. Třída v COM je nazývána „coclass“. Jde o definici třídy nezávislou na jazyku. Taková třída pak nahrazuje konkrétní implementaci jednoho nebo více rozhraní. S COM také souvisí pojem typových knihoven, což jsou metadata reprezentující COM typy. Tyto typy jsou popsány textovým Interface Definition Language (IDL). IDL soubory jsou poté kompilovány MIDL compilerem pro použití v konkrétních programovacích jazycích. V systémech Windows jsou informace o COM třídách, rozhraních a typových knihovnách ukládány v registrech. Tyto informace pak využívají COM knihovny k vyhledání správné lokální knihovny pro každý COM objekt. Aby Microsoft podpořil práci s COM, vytvořil ATL (Active Template Library), což je množství C++ šablon zjednodušujících právě práci s COM.

4.2.2.3 Vytváření instancí komponent

Vytváření instance COM objektu je standardizováno použitím Class Factory, což je též COM objekt. Jde o poměrně složitý proces, který probíhá následujícím způsobem. Nejprve je vytvořena instance objektu class factory zavoláním funkce *CoGetClassObject*. *CoGetClassObject* získá ukazatel na rozhraní class factory zavoláním *DllGetClassObject*, definované v DLL. Poté je zavolána metoda *CreateInstance* vytvořené instanece class factory. Ta teprve vytvoří instanci komponenty a vrátí ukazatel na požadované rozhraní. Celý tento proces se skrývá za voláním funkce *CoCreateInstance*. Class factory je objekt určený k vytváření jiných COM objektů. Každý objekt class factory vytváří jeden typ objektů. Každý objekt class factory je instancí třídy *CclassFactory*. Každá class factory také obsahuje ukazatel na tzv. factory template, což je struktura obsahující informace o specifické komponentě (CLSID a ukazatel na funkci vytvářející komponentu). V DirectShow je *DllGetClassObject* již implementována. Tato funkce hledá template s odpovídající CLSID, pokud najde, vytvoří class factory s ukazatelem na tento template. Globální pole s factory template je nutné deklarovat v DLL. Když je volána metoda *CreateInstance* rozhraní *IClassFactory*, objekt class factory zavolá funkci pro vytvoření instance komponenty definované v template. Výhodou tohoto

složitého procesu je, že programátorovi stačí vytvořit jen malé množství kódu specifického pro jeho komponentu a nemusí implementovat celou class factory.

4.2.3 Výběr rodičovské třídy

Protože tvorba vlastního DirectShow filtru je poměrně složitou záležitostí a je potřeba implementovat velké množství kódu, většina programátorů přistupuje k jeho odvození od existujících základních filtrů. Tyto základní filtry jsou součástí balíku kódu zvaného Base Classes a distribuovaného spolu s DirectShow. V DirectShow jsou všechny objekty odvozeny od těchto Base Classes. Konstruktory a metody těchto tříd zajišťují většinu práce s COM, jako například zachování konzistence počtu referencí. Odvozením vlastního filtru od těchto tříd je dosaženo zdědění funkcionality těchto tříd. Existuje několik základních tříd, od kterých se obvykle odvozují nové vlastní filtry. Jsou to:

- *CSource*,
- *CTransformFilter*,
- *CTransInPlaceFilter*,
- *CVideoTransformFilter*,
- *CBaseRenderer*.

CSource je základní třída pro odvozování zdrojových filtrů, které produkují souvislý tok dat. *CTransformFilter* je třída navržená pro implementaci transformačního filtru s jedním vstupním a jedním výstupním pinem. *CTransInPlaceFilter* je odvozen od *CTransformFilter* a je navržen jako základ pro filtry, které transformují data na místě, na místo toho, aby kopírovali data mezi buffery. *CVideoTransformFilter* je též odvozen od *CTransformFilter* a je primárně navržen jako základní třída pro AVI dekompresory. Předpokládá, že buffer bude určen pro video data. Hlavní vlastností této třídy je možnost kontroly kvality (vypouštění některých snímků za určitých podmínek). *CBaseRenderer* je základ pro tvorbu renderovacích (výstupních) filtrů. Je opatřena jedním vstupním pinem a schopností synchronizace. Výběr rodičovské třídy je tedy závislý na funkci zamýšleného filtru v grafu filtrů a typu zpracovávaných dat.

4.2.4 Registrace filtru v systému

Jak již bylo zmíněno v kapitole o COM, k práci s COM knihovnami je využíváno informací uložených v systémových registrech. Když aplikace volá *CoCreateInstance*, aby vytvořila filtr, COM knihovna použije záznamů v registru k lokalizaci potřebné DLL. Nový filtr je proto nejprve nutné zaregistrovat v operačním systému, než je možné jej začít používat v aplikacích. Registrace filtru představuje přidání informací o něm do systémových registrů. To je možné provést dvěma způsoby. Informace o filtru je možné zapsat manuálně přímo do systémových registrů nebo provést dynamickou registraci s pomocí Regsrv32.exe nebo RegsvrCE.exe. Druhý způsob vyžaduje deklaraci několika struktur s informacemi o filtru. Jsou to:

- AMOVIESETUP_FILTER,
- AMOVIESETUP_PIN,
- AMOVIESETUP_MEDIATYPE.

Dále je nutné uložit ukazatel na první z nich do factory template a předefinovat metodu *GetSetupData*. Předpokládá se též implementace dříve zmíněných metod *DllRegisterServer* a *DllUnregisterServer*.

4.3 XML

Výstupem systému by dle specifikovaných požadavků měl být soubor se značkami. Vzhledem k charakteru výstupních dat je pro další zpracování nejvhodnější ukládat data ve formátu XML. XML (eXtensible Markup Language) je obecný značkovací jazyk. Tento jazyk byl vyvinut a je standardizován konsorciem W3C. XML je určeno zejména pro výměnu dat mezi aplikacemi a pro publikování dokumentů. Umožňuje popsat strukturu dokumentu z hlediska věcného obsahu jednotlivých částí, nezabývá se sám o sobě vzhledem dokumentu nebo jeho částí. Prezentace dokumentu (vzhled) se definuje připojeným stylem. Mezi jeho přednosti patří vytváření konkrétních značkovacích jazyků pro různé účely, široké spektrum různých typů dat, mezinárodní podpora, vysoký informační obsah, snadná konverze do jiných formátů, automatická kontrola struktury dokumentu a další.

S XML je spojeno mnoho dalších pojmů, jedním z nich je DTD (Document Type Definition). Pod tímto označením se skrývá definice typu dokumentu. DTD určuje prostřednictvím množiny deklarací třídu XML dokumentů. Určuje tedy povolený obsah elementů a atributů, jejich typ a výchozí hodnoty. DTD pro XML výstup realizovaného systému vypadá následovně:

```
<?xml version="1.0" encoding="windows-1250"?>
<!ELEMENT Tags (Slide)>
<!ELEMENT Slide>
<!ATTLIST Slide
   src CDATA
   frame NMTOKEN
   time NMTOKEN
   width NMTOKEN
   height NMTOKEN
   canvasAngleLeft NMTOKEN
   canvasAngleTop NMTOKEN
   canvasAngleRight NMTOKEN
   canvasAngleBottom NMTOKEN
   canvasDistanceLeft NMTOKEN
```

canvasDistanceTop NMTOKEN
canvasDistanceRight NMTOKEN
canvasDistanceBottom NMTOKEN
type NMTOKEN
length NMTOKEN >

Element *Tags*, který je v XML výstupu elementem kořenovým, obsahuje elementy *Slide*. Každý element *Slide* představuje jednu změnu slajdu nebo animaci. Tento element obsahuje atributy, v nichž je uchována informace o detekované změně. Atribut *src* uchovává název souboru se snímkem z kamery, *frame* obsahuje číslo zpracovaného snímku, *time* obsahuje čas změny (relativní k začátku značkování), *width* a *height* obsahují rozměry snímku, *canvasAngleLeft*, *canvasAngleTop*, *canvasAngleRight*, *canvasAngleBottom* obsahují hodnoty úhlů jednotlivých přímek ohraničujících plátno, *canvasDistanceLeft*, *canvasDistanceTop*, *canvasDistanceRight*, *canvasDistanceBottom* obsahují hodnoty vzdáleností přímek od počátku soustavy souřadnic, *type* určuje, zda jde o animaci (1) nebo prostou změnu slajdu (0), *length* uchovává informaci o času trvání animace.

4.4 Záznam zvuku

Jedním z nejdůležitějších výstupů vyvíjeného systému je zvukový záznam zpracovávané prezentace. Systém Windows Mobile 5.0 nabízí hned dvě API, s jejichž pomocí lze záznam zvuku realizovat. Prvním z nich je již zmiňované DirectShow API a druhou možnost představuje Waveform Audio API. V případě DirectShow je postup obdobný jako při nahrávání video záznamu, do grafu je nutné přidat a propojit potřebné filtry a nastavit potřebné parametry. V mnoha případech však v zařízeních s Windows Mobile 5.0 chybí potřebné filtry, jak pro kompresi, tak i pro samotný zápis do čistě zvukového souboru, a proto bylo pro realizaci vybráno rozhraní Waveform Audio.

4.4.1 Waveform Audio API

Toto rozhraní pro práci se zvukem je tvořeno množinou funkcí, zpráv a datových struktur. Waveform Audio API umožňuje programátorovi plnou kontrolu nad virtuálními zařízeními zpracovávajícími zvuk. Ty se zde rozdělují na vstupní a výstupní. Vstupní slouží pro záznam zvuku a výstupní pro přehrávání zvuku. Rozhraní také umožňuje zjišťovat přítomnost těchto zařízení v systému a jejich možnosti.

Před použitím musí být každé virtuální zařízení otevřeno. To se provede s pomocí funkcí *waveInOpen* (vstupní zařízení) a *waveOutOpen* (výstupní zařízení), které vracejí handle na otevřené zařízení. Jeden z jejich parametrů určuje, jakým způsobem bude operační systém komunikovat s funkcemi rozhraní. Možnosti jsou:

- s pomocí mechanismu událostí,

- voláním callback funkcí,
- zasíláním zpráv mezi vlákny,
- zasíláním systémových zpráv (Windows messages),
- bez jakékoliv signalizace.

Dalšími parametry těchto funkcí jsou vzorkovací frekvence, počet kanálů, počet bitů na vzorek, typ kódování, atd.

Přehrání zvuku představuje nejprve připravení potřebných struktur, které se provede voláním funkce *waveOutPrepareHeader*. Po každém naplnění bufferu je možné volat funkci *waveOutWrite* s parametrem ukazujícím na buffer, který se má přehrát. Přehrávat se začíná po zavolání funkce *waveOutRestart*.

Pro tuto práci je ovšem důležitý záznam zvuku. Ten začíná připravením důležitých struktur s parametry nahrávání a s ukazatelem na již alokovaný buffer, do nějž mají být uložena nahrávaná data. K tomuto slouží funkce *waveInPrepareHeader*. Připravený buffer se přidá do systémových bufferů voláním *waveInAddBuffer*. Nahrávání začíná použitím funkce *waveInStart*. Jakmile je buffer naplněn daty, je toto signalizováno aplikaci s pomocí některé z výše uvedených možností. Příprava bufferu a přidání bufferu mezi systémové buffery pak může být opakováno ve smyčce, aby bylo dosaženo souvislého nahrávání.

Nakonec je vždy nutné zastavit přehrávání či nahrávání zavoláním funkcí *waveOutReset*, *waveInReset*, které uvolní systémové buffery, dále uvolnit zařízení funkcemi *waveOutClose*, *waveInClose* a s pomocí *waveOutUnprepareHeader*, *waveInUnprepareHeader* umožnit uvolnění dříve alokovaných bufferů pro data.

Z principu, jakým je prováděn záznam zvuku vyplývá, že existuje riziko ztráty synchronizace při zpomalení aplikace (například z důvodu nedostatečného výkonu).

4.5 Ukládání obrazu

Při zpracování, v případě detekované změny na plátně, se ukládá celý snímek do trvalé paměti. Je sice možné ukládat jen výřez s plátnem, ale vzhledem k podmínkám nemusí detekce dopadnout vždy dobře. Proto je třeba zachovat celý obraz, informace o plátně jsou ovšem obsaženy jako součást výstupu v XML. Při dalším zpracování mohou být tyto informace dále využity (extrakce slajdu) nebo může být provedeno přesnější vyhledání plátna. To může probíhat například na výkonnějším PC bez omezení požadavkem na realtime zpracování. Ukládání snímku do trvalé paměti je poměrně náročná operace, která má za následek dočasné zpomalení celé aplikace. To může mít špatný vliv na kvalitu v situaci, kdy dochází ke změnám na plátně rychle po sobě. Aby byla minimalizována šance, že z tohoto důvodu systém nestihne změny zaznamenat, uchovávají se nejprve snímky v paměti. Teprve při dosažení nastaveného limitu dochází k uložení snímků do trvalé paměti. Snímky jsou ukládány ve formátu BMP. Ten je vhodný pro jednoduché zpracování. Na platformě Windows Mobile lze

s pomocí Imaging API ukládat obrázky do formátu JPEG a dalších. Nevýhodou některého z těchto úspornějších formátů je náročnost jejich kódování, která na výpočetně omezeném zařízení negativně ovlivňuje průběh souběžného zpracování obrazu.

4.6 Nástroje

Práce na jakémkoliv systému se neobejde bez příslušných vývojových nástrojů. Pro vývoj softwaru, který má být provozován na mobilním zařízení, je jejich výběr poměrně omezen a závisí do značné míry na volbě cílové platformy.

4.6.1 Visual Studio 2005

Visual Studio je balík nástrojů a technologií určených pro vývoj software na stolní počítače. Autorem tohoto produktu je firma Microsoft a je určen především pro tvorbu programů, webových stránek, aplikací, či služeb běžících na platformách od této společnosti. Součástí balíku jsou technologie Visual C++, Visual C#, Visual J# a Visual Web Developer. V této práci bylo využito Visual C++ ve verzi 2005, neboť ke starším verzím nelze nainstalovat nástavbu v podobě Pocket PC SDK ve verzi pro Windows Mobile 5.0.

4.6.2 Pocket PC SDK

Aby bylo možné vyhovět požadavkům na provoz systému na mobilním telefonu s Windows Mobile, je nutné nainstalovat na počítač, na němž je systém vyvíjen, kromě Visual Studia 2005 ještě další nástroje. Jedním z nich je právě Pocket PC SDK (Software Development Kit). Jde o rozšíření Visual Studia o možnosti vývoje software na mobilní zařízení typu Pocket PC. Instalační balík obsahuje mimo jiné hlavičkové soubory, knihovny, IDL soubory, ukázkový kód, dokumentaci a užitečné nástroje, jako například emulátor platformy Windows Mobile.

4.6.3 Smartphone SDK

Vzhledem k výpočetním nárokům implementovaných algoritmů zpracování obrazu se sice předpokládá použití spíše na výkonnější platformě Pocket PC, nicméně v průběhu práce došlo i na testování na zařízení typu Smartphone.

4.6.4 ActiveSync

Pro práci jak s emulátorem zařízení, tak i s reálným zařízením používá Visual Studio 2005 aplikaci ActiveSync. Je to standardní nástroj pro synchronizaci, přenos nastavení a souborů mezi stolním počítačem a zařízeními s Windows Mobile. Tuto aplikaci je možné použít jen v systému Windows

XP, v novějších Windows Vista už se používá nástupce ActiveSyncu, Windows Mobile Device Center.

4.6.5 Jazyk

Programátor mobilních zařízení si může vybrat hned z několika jazyků, v nichž bude svůj software tvořit. Kromě nativního C++ se nabízí managed C++ (řízené a prováděné .NET virtuálním strojem), C#, Java či Visual Basic. Požadované vlastnosti výsledné aplikace, jež je cílem této práce, však značně omezují výběr. Současná verze DirectShow API obsaženého v SDK pro Windows Mobile neumožňuje použití jiného jazyka než je nativní C++. Tento jazyk je také nejvhodnější z pohledu rychlosti výsledné aplikace implementující výpočetně náročné algoritmy zpracování obrazu.

5 Implementace

Zdrojový kód celé práce je rozčleněn do tří samostatných projektů. Prvním projektem je tzv. Sample Grabber filtr, neboli filtr pro přístup k datům snímku v DirectShow grafu. Druhý projekt tvoří zdrojové kódy dalšího DirectShow filtru - transformačního filtru, převádějícího obrazová data z formátu YV12 do formátu RGB24. Výsledkem překladu a sestavení těchto dvou projektů jsou dvě dynamické knihovny. Jejich přítomnost v systému je vyžadována aplikací vzniklou z třetího, hlavního projektu.

Je třeba podotknout, že multimediální aplikaci běžící na systémech Windows Mobile není téměř možné v současné době implementovat natolik obecně, aby pracovala správně na jakémkoliv zařízení tohoto typu a s tímto systémem. Důvodem není jen různost použitého hardwaru (například množství různých formátů výstupních dat kamery), ale i přístup výrobců těchto zařízení, kteří produkují ovladače různé kvality. Díky tomu vzniká mnoho odlišností, které jsou specifické pro jednotlivá zařízení, a které je nutné zohledňovat při implementaci.

5.1 Řešené problémy

Během implementace bylo řešeno hned několik neočekávaných problémů. Až s prvními pokusy o vývoj na platformu Windows Mobile se objevily skutečnosti, které nebyly popsány v dokumentacích a manuálech. V takových situacích se pak staly velmi vítanými informace z diskusních skupin a blogů, kam často přispívali i samotní vývojáři systému Windows Mobile. Díky těmto informacím bylo možné odstranit většinu zaznamenaných překážek. Následuje popis některých z těchto zajímavých problémů a postup jejich řešení.

5.1.1 Absence Sample Grabber filtru

Následující problém vznikl při práci na třídě *OfflineCamera*. Protože návrh systému počítal s použitím standardního filtru Sample Grabber pro získávání snímků z kamery, vyvstal poměrně zásadní problém při zjištění, že v DirectShow obsaženém v systému Windows Mobile 5.0 tento filtr zcela chybí. Vzhledem k tomu, že v tomto sytému není jiná možnost jak zpracovávat obrazová data z kamery než s pomocí DirectShow, bylo třeba vytvořit vlastní Sample Grabber filtr. Další postup při tvorbě filtru odpovídá popisu ze sekce 4.2.

5.1.2 Registrace filtru

Po zdařilém stvoření vlastního Sample Grabber filtru se nečekaným zádrhelem stala registrace filtru v emulátoru Windows Mobile a tedy i otestování jeho funkčnosti a následné používání. Registrace je navíc nutná při každé změně rozhraní filtru, k usnadnění vývoje bylo tedy potřeba nalézt nějaké

jednoduché a rychlé řešení. Protože standardní instalace Windows Mobile nenabízí žádné administrační nástroje, díky nimž by šlo registraci provést, pomohlo až nalezení týmového blogu vývojářů multimediální části systému Windows Mobile [13]. Na tomto webu byl mimo jiné vystaven ke stažení nástroj Device Command Shell. Ten se instaluje jako vylepšení příkazové řádky v použitém vývojovém nástroji Visual Studio 2005 a umožňuje komunikovat se zařízením nebo jeho emulátorem nebo spouštět administrační nástroje. Jedním z těchto administračních nástrojů je i aplikace *regsrv*, s jejíž pomocí se registrace filtru stala triviální záležitostí.

5.1.3 Callback funkce a objektový model

Následující problém se objevil během implementace tříd *OfflineCamera* a *ChangeDetector* a jejich komunikace, konkrétně získání snímku z kamery. Aby nebylo nutné komplikovat implementaci tvorbou samostatného vlákna s časovačem, které by získávalo v určitém časovém intervalu snímky z kamery, bylo přistoupeno k použití callback funkce. Jde o mechanismus, kdy během inicializace je definován ukazatel na funkci, která je pak volána z jiné funkce při výskytu nějaké události. Použití tohoto principu při komunikaci mezi třídou *OfflineCamera* a *ChangeDetector* však narazilo na určité překážky vyplývající z odlišnosti volání standardní funkce jazyka *C* a metod v jazyce *C++*. Každá metoda totiž při svém volání obdrží i skrytý parametr obsahující ukazatel na instanci své třídy. Z toho důvodu se liší deklarace funkce a metody, a tedy i ukazatelů na ně. Teoretickým východiskem z této situace je deklarovat metodu, jenž má sloužit jako callback funkce, jako statickou metodu. Jiným řešením je namísto ukazatele na metodu (funkci) předávat právě ukazatel na instanci třídy a pak při výskytu události místo callback funkce volat metodu této instance. Právě tento postup byl v práci použit.

5.1.4 Formát dat YV12

Jak již bylo zmíněno ve sekci 3.2.1.1, vestavěné kamery mobilních zařízení produkují svůj výstup ve formátech založených na barevných modelech RGB a YUV. Je proto nutné vybrat jeden z těchto formátů a v něm data zpracovávat. V případě odlišného formátu na výstupu kamery přichází na řadu konverze. Protože vstup z videosouboru (v případě použití testovací třídy *OfflineCamera*) je vždy v RGB, a protože pro potřeby výstupu realizovaného systému je vhodnější barevný model RGB, byl tento vybrán jako vstup do zpracování obrazu.

Zařízení, na kterém byl tento systém také vyvíjen, je vybaveno kamerou, jejíž výstupní data jsou kódována ve formátu YV12. Jde o jeden z formátů postavených na barevném modelu YUV. Je to formát planární, jasová složka je uložena v poli osmibitových hodnot, za nímž následují dvě pole o čtvrtinové velikosti, obsahující zbývající dvě složky. Ty jsou ovšem vzorkovány ve čtyřikrát menším rozlišení. Tento formát je velmi rozšířený, zejména v kompresních kodecích typu MPEG.

Pro převod do 24-bitového RGB formátu požadovaného filtrem Sample Grabber bylo nutné implementovat transformační filtr. Prakticky všechny algoritmy zpracování obrazu použité v této práci pracují pouze nad jasovou složkou obrazu, informace o barvě není pro další zpracování potřebná. V tomto případě není nutná ani konverze z YUV do RGB modelu, popsaná v sekci 3.2.1.5. S výhodou tedy bylo využito toho, že YV12 formát má data jasu oddělena od ostatních a toho, že jas jednoho pixelu je dokonce kódován v jednom bytu. Jasovou složku YV12 formátu tedy stačí zkopírovat do všech složek modelu 24-bitového RGB. Tím se také ušetří množství výpočetního času.

5.2 Optimalizace zpracování obrazu

Při řešení této práce nastaly situace, kdy se použití výpočetně náročných algoritmů nedalo vyhnout. V takových chvílích bylo nutné různými technikami zmenšit počet kritických operací na minimum. Příkladem může být předpočítání jasových hodnot pixelů namísto jejich získávání až v průběhu každé z mnoha operací nad obrazem. Dalším příkladem může být tabelování hodnot goniometrických funkcí, které jsou běžnou součástí všech geometrických výpočtů použitých při zpracování (například práce s přímkami). Podstatného zrychlení je možné dosáhnout eliminací operací s plovoucí řadovou čárkou, použitím celočíselných variant některých algoritmů či dalšími technikami souvisejícími s implementačním jazykem.

5.2.1 Eliminace operací s plovoucí řadovou čárkou

Absence hardwarové podpory výpočtů s plovoucí řadovou čárkou u většiny procesorů použitých v mobilních zařízeních nutí programátora toto zohlednit při implementaci. To se projevuje jak při volbě datových typů, tak při výběru algoritmů pro zpracování obrazových dat. Všude, kde to je možné, je proto použito výhradně celočíselných hodnot a typů. Nahrazení některých výpočtů jejich celočíselnou variantou se projevuje snížením rozlišení a tedy i přesnosti výpočtu. V případě implementovaných operací je v drtivé většině případů dosažená přesnost zcela postačující. Názorným příkladem je vyjádření hodnot goniometrických funkcí v celých číslech, kde se jako uspokojivý ukázal rozsah 256 hodnot.

5.2.2 Optimalizace rozhodnutí příslušnosti bodu k plátnu

Z informací popisujících, kde v obraze se nachází detekované plátno, je hned v několika případech nezbytné určit, které pixely vstupního obrazu plátnu náleží. Mezi tyto případy patří detekce změny slajdu, zobrazení vyhledaného plátna v obraze a generování výstupních dat realizovaného systému.

Intuitivně je možné tento problém řešit následovně. Při procházení jednotlivých pixelů obrazu se z přímek ohraničujících plátno vypočítá y-ová souřadnice horního a dolního okraje plátna z x-ové souřadnice procházeného pixelu a x-ová souřadnice levého a pravého okraje plátna z y-ové

souřadnice procházeného pixelu. Poté se určí, zda procházený pixel leží mezi vypočítanými body. Neustálé dosazování do rovnice přímky je ovšem velmi neefektivní.

Protože informace o plátně jsou přímky, je možné problém řešit s pomocí některého z algoritmů rasterizace přímky. Jako nejefektivnější se vzhledem k výhradnímu použití operací s pevnou řadovou čárkou jeví Bresenhamův algoritmus rasterizace úsečky. Všechny algoritmy, které rozhodování o náležitosti pixelu do oblasti s plátnem vyžadují, jsou ale založeny na použití dvou zanořených cyklů, jednom pro procházení řádků a druhém pro procházení sloupců. Proto je nutné Bresenhamův algoritmus modifikovat tak, aby jej šlo využít i v takové situaci.

5.2.3 Globální proměnné

Značné úspory výpočtů lze docílit definováním často používaných dynamický alokovaných proměnných, zejména polí, jako globálních. Operace alokace a rušení paměti představují velké zpomalení a v mnoha případech je možné je nahradit pouhým vynulováním dané části paměti.

6 Data, experimenty a výsledky

Uvažujeme-li pouze změnu slajdu, mohou při detekci této události nastat následující situace:

- změna je správně detekována (True Positive, TP),
- změna není detekována (False Negative, FN),
- změna je chybně detekována (False Positive, FP).

Z pohledu kvality výstupu je daleko závažnější chybou nezaznamenání změny na plátně, než detekce změny ve chvíli, kdy k žádné nedošlo. Dá se také předpokládat, že při citlivějším nastavení detekce změny v obraze je sice větší množství chybných detekcí, ale také větší šance, že se zaznamená každá změna na plátně. Proto je rozumnějším řešením nastavit parametry detekce spíše na vyšší citlivost. Příliš vysoká citlivost ale zvyšuje nároky aplikace na prostor, neboť je nutné ukládat větší množství snímků do trvalé paměti. Také se zvyšuje šance „propásnutí“ změny na plátně v době, kdy je aplikace zaneprázdněna ukládáním snímků.

Kvalita výstupu je kromě nastavení parametrů detekce změny výrazně ovlivněna kvalitou detekce plátna. Algoritmy řešící tento problém jsou závislé na množství parametrů. U detekce hran je to práh, určující, které pixely náleží hranám a tedy vstupují do dalšího zpracování. Houghova transformace je zase ovlivněna nastavením intervalů úhlů, v nichž jsou vyhledávány přímky. Při hledání hranic plátna je podstatné nastavení prahu nejlépe hodnocených přímek a parametry jejich shlukování. Nastavení každé z těchto hodnot má vliv na kvalitu detekce plátna a pro nejlepší dosažený výsledek je nutné individuální nastavení parametrů pro každou scénu.

6.1 Offline testy kvality detekce

Testování bylo provedeno s pomocí třídy *OfflineCamera*, která umožňuje použít videosoubor jako vstup dat namísto snímků z kamery. Aby bylo porovnání objektivní, je vždy vstupem stejný soubor. Použité testovací video pochází z testovacích dat projektu AMI [1] a zachycuje konferenční místnost s lidmi a plátnem, na nějž je promítána prezentace. Plátno zabírá asi 1/4 plochy obrazu. Délka videa je 29 minut a 20 sekund, pro testování byla použita nekomprimovaná verze videa bez zvukové stopy. Video bylo zmenšeno do rozlišení odpovídajícího rozlišení kamery běžného zařízení, 176 na 144 obrazových bodů. Vzorkovací frekvence 25 snímků za sekundu odpovídala možnostem reálného zařízení. V průběhu videa se na plátně mění snímky prezentace a místy jsou prováděny úkony v prostředí operačního systému, nutné pro spuštění další prezentace. Ty svou povahou, při daném rozlišení a vzorkovací frekvenci obrazu, lze pro potřeby testování považovat za události ekvivalentní změně slajdu. Použitým zařízením byl Smartphone Qtek 8500 s procesorem Texas Instruments OMAP 850 (195 MHz).

Výstup systému v podobě časových značek je porovnán s referenčními značkami, čímž je umožněno ohodnotit kvalitu metody při daných parametrech. Výsledek testování je jistě ovlivněn kvalitou souboru s referenčními značkami, nicméně vzájemné srovnání různých metod to neovlivní. Parametry jednotlivých detektorů vycházejí z empirických informací a do velké míry jsou závislé na použitém testovacím záznamu.

Třídy realizující detekci změny jsou ve své podstatě binární klasifikátory. Jejich úspěšnost je tedy také možné popsat pomocí dvou druhů chyb, změna je detekována v situaci, kdy k žádné nedošlo (chyba prvního typu, False Positive) a nezaznamenání změny (chyba druhého typu, False Negative). K vyjádření vlastností klasifikátorů se běžně užívá tzv. ROC křivky (Receiver Operating Characteristic). Je to graf zobrazující podíl TPR (True Positive Rate) a FPR (False Positive Rate). TPR udává míru, s jakou klasifikátor rozhoduje správně o daném jevu jako o pozitivním (v tomto případě o změně slajdu). TPR lze vypočítat jako podíl TP a všech pozitivních případů. FPR zase udává míru, s jakou je o tomto jevu jako o pozitivním rozhodnuto chybně (falešný poplach). Tuto hodnotu lze určit jako podíl FP a všech negativních případů. Dalšími hodnotami charakterizujícími detekční metodu jsou TNR (True Negative Rate), která v tomto případě vypovídá o schopnosti správně nereagovat, když nedochází ke změně. FNR (False Negative Rate) vyjadřuje míru, kdy chybně není detekována změna. TNR lze vyjádřit jako podíl TN a všech negativních případů. FNR je podíl FN a všech pozitivních případů.

6.1.1 Výsledky testování třídy SimpleDetector

Třída *SimpleDetector* kontroluje změnu mezi dvěma snímky s pomocí rozdílu sum pixelů v těchto snímcích. Pozitivní výsledek pak závisí na překročení nastaveného prahu.

	prahová hodnota (v pixelech)									
	50	85	100	300	500	600	700	1000	2000	5000
TPR	0,9875	0,9500	0,9250	0,7375	0,6875	0,6875	0,6500	0,6125	0,4625	0,3125
FPR	0,3078	0,1681	0,1330	0,0165	0,0068	0,0042	0,0031	0,0023	0,0010	0,0004
TNR	0,6922	0,8319	0,8670	0,9835	0,9932	0,9958	0,9969	0,9977	0,9990	0,9996
FNR	0,0000	0,0125	0,0375	0,0500	0,0500	0,0500	0,0500	0,0625	0,1125	0,1500

Tabulka 6.1 výsledky testování třídy *SimpleDetector*

Jako nejrozumnější práh se z pohledu požadavku na co možná největší míru TRP (důležité je propásnout co nejméně změn) jeví hodnota 50 pixelů. Při této hodnotě však třída již dosahuje vysoké míry false positive případů. Tato prahová hodnota je specifická pro konkrétní testovací případ, neboť závisí na absolutní velikosti plátna detekovaného v obraze.

6.1.2 Výsledky testování třídy HistogramDetector

Detekce implementovaná v třídě *HistogramDetector* je založena na porovnání histogramu dvou po sobě jdoucích snímků. Prahová hodnota představující maximální změnu počtu obrazových bodů stejné úrovně jasu pak určuje výsledek testu na změnu obrazu.

	prahová hodnota (v pixelech)									
	500	600	700	800	1000	1500	2000	2500	3000	3500
TPR	1,0000	0,9875	0,9250	0,6875	0,4875	0,3250	0,2625	0,2250	0,1750	0,1000
FPR	0,7271	0,3604	0,1928	0,0707	0,0026	0,0007	0,0003	0,0002	0,0002	0,0001
TNR	0,2729	0,6396	0,8072	0,9293	0,9974	0,9993	0,9997	0,9998	0,9998	0,9999
FNR	0,0000	0,0125	0,0750	0,3125	0,5125	0,6750	0,7375	0,7750	0,8250	0,9000

Tabulka 6.2 výsledky testování třídy *HistogramDetector*

Vhodným prahem v tomto případě je hodnota 600, při níž je ale opět zaznamenána vysoká hodnota FPR

6.1.3 Výsledky testování třídy HistogramAvgDetector

Třída *HistogramAvgDetector* je modifikací předchozího detektoru. Na rozdíl od něj porovnává histogram aktuálního snímku s průměrným histogramem snímků předchozích. Prahová hodnota má stejný význam jako v předešlém případě.

	prahová hodnota (v pixelech)									
	15	20	30	40	50	60	70	80	90	100
TPR	1,0000	0,9875	0,8875	0,8250	0,6875	0,6250	0,5500	0,4500	0,3750	0,3500
FPR	0,7130	0,3648	0,0927	0,0288	0,0125	0,0059	0,0039	0,0022	0,0014	0,0010
TNR	0,2870	0,6352	0,9073	0,9712	0,9875	0,9941	0,9961	0,9978	0,9986	0,9990
FNR	0,0000	0,0125	0,1125	0,1750	0,3125	0,3750	0,4500	0,5500	0,6250	0,6500

Tabulka 6.3 výsledky testování třídy *HistogramAvgDetector*

Jako nejvhodnější práh se zde jeví hodnota 20 pixelů, i zde však odpovídá vysoké hodnotě FPR. V porovnání s předchozí třídou zajistilo průměrování lepší charakteristiku v oblasti nízkých FPR.

6.1.4 Výsledky testování třídy DifferenceHistogramDetector

Třída *DifferenceHistogramDetector* pracuje s rozdílovým snímkem dvou posledních snímků. Porovnáván je pak medián histogramu rozdílového snímku s nastaveným prahem.

	prahová hodnota (v pixelech)									
	8	16	24	32	48	64	96	112	128	160
TPR	1,0000	0,9875	0,9875	0,9625	0,8750	0,7500	0,5625	0,1750	0,1250	0,0000
FPR	0,5802	0,0984	0,0728	0,0653	0,0528	0,0458	0,0350	0,0307	0,0267	0,0188
TNR	0,4198	0,9016	0,9272	0,9347	0,9472	0,9542	0,9650	0,9693	0,9733	0,9812
FNR	0,0000	0,0125	0,0125	0,0750	0,1250	0,2500	0,5625	0,8250	0,8750	1,0000

Tabulka 6.4 výsledky testování třídy *DifferenceHistogramDetector*

Hodnota prahu, při němž je již dosahováno uspokojivé míry TP případů, je 24. FPR při něm tvoří 7 %, což je v porovnání s předchozími případy dobrý výsledek.

6.1.5 Výsledky testování třídy DifferenceHistogramAvgDetector

Třída *DifferenceHistogramAvgDetector* pracuje obdobně jako třída *DifferenceHistogramDetector*, s tím rozdílem, že rozdílový snímek je vytvářen z aktuálního snímku a snímku vzniklého průměrováním snímků předchozích.

	prahová hodnota (v pixelech)									
	8	16	20	24	32	48	64	96	128	160
TPR	1,0000	0,9875	0,9875	0,9875	0,9625	0,9000	0,7750	0,4500	0,0625	0,0125
FPR	0,6982	0,1076	0,0859	0,0785	0,0679	0,0567	0,0487	0,0372	0,0286	0,0206
TNR	0,3018	0,8924	0,9141	0,9215	0,9321	0,9433	0,9513	0,9628	0,9714	0,9794
FNR	0,0000	0,0125	0,0125	0,0125	0,0375	0,1000	0,2250	0,5500	0,9375	0,9875

Tabulka 6.5 výsledky testování třídy *DifferenceHistogramAvgDetector*

Třída *DifferenceHistogramAvgDetector* má velmi podobné chování jako její jednodušší verze. I zde je vhodným prahem hodnota 24 pixelů.

6.1.6 Výsledky testování třídy DifferenceDetector

Podstatou detekce implementované v třídě *DifferenceDetector* je výpočet rozdílu jasu odpovídajících si pixelů v posledních dvou snímcích. Prahem je jejich maximální rozdíl.

	prahová hodnota (v pixelech)									
	2000	3000	4000	6000	8000	10000	15000	20000	30000	50000
TPR	1,0000	0,9750	0,8750	0,6375	0,5750	0,5250	0,4750	0,4625	0,4125	0,2250
FPR	0,4204	0,2901	0,1309	0,0391	0,0205	0,0153	0,0087	0,0048	0,0007	0,0002
TNR	0,5796	0,7099	0,8691	0,9609	0,9795	0,9847	0,9913	0,9952	0,9993	0,9998
FNR	0,0000	0,0250	0,1250	0,3625	0,4250	0,4750	0,5250	0,5375	0,5875	0,7750

Tabulka 6.6 výsledky testování třídy DifferenceDetector

Nejvhodnější práh je v tomto případě hodnota 3000 pixelů. Té ale odpovídá vysoká hodnota FPR. V tomto případě je velikost prahu opět specifická pro konkrétní testovací případ, obecné určení prahu by muselo zohledňovat velikost plátna v obraze.

6.1.7 Výsledky testování třídy DifferenceCountDetector

DifferenceCountDetector je modifikací předchozí třídy. Pracuje však s počtem rozdílných pixelů. Výsledek detekce je opět dán porovnáním s prahovou hodnotou.

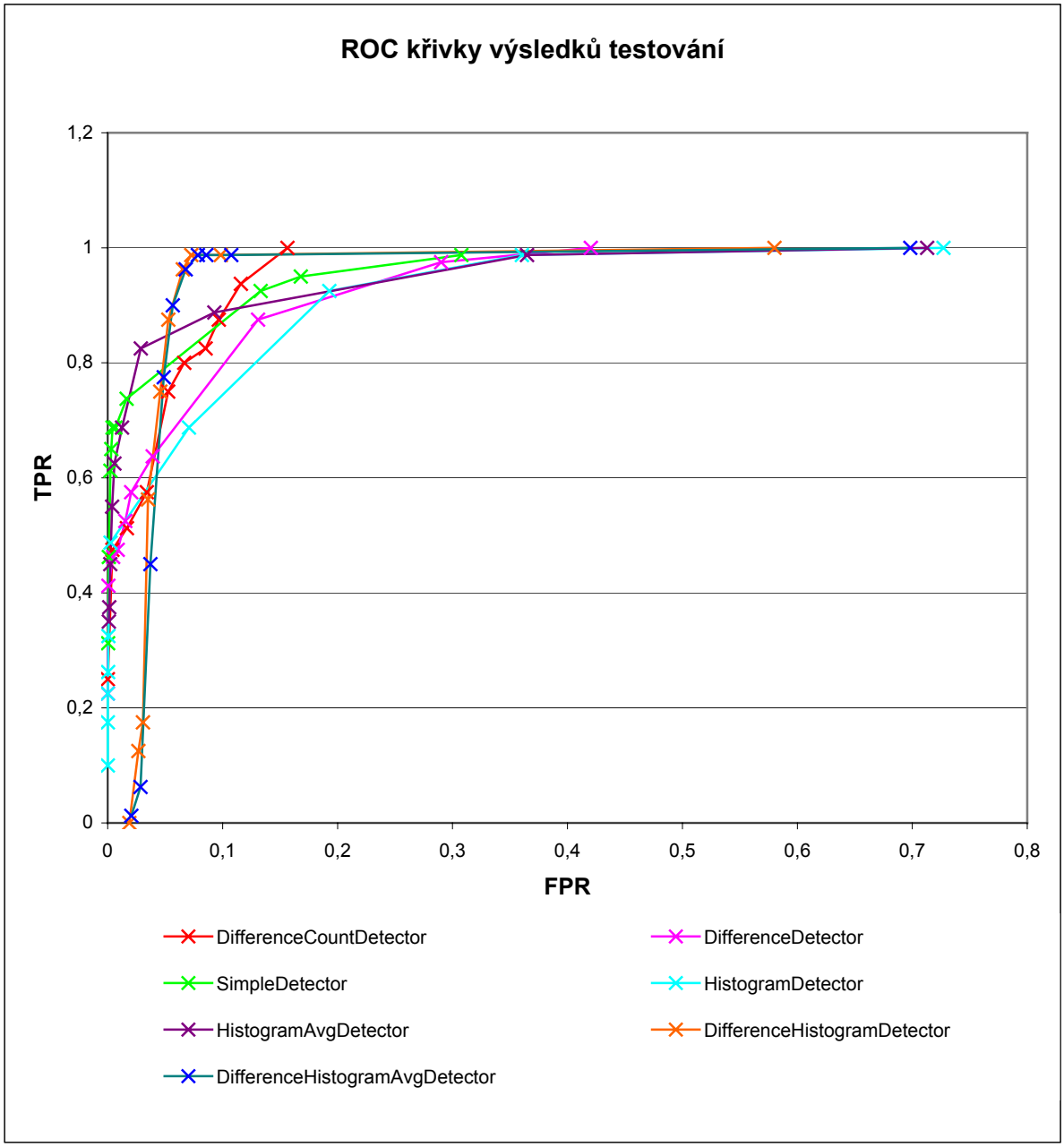
	prahová hodnota (v pixelech)									
	5	10	15	20	30	40	80	200	400	1000
TPR	1,0000	0,9375	0,8750	0,8250	0,8000	0,7500	0,5750	0,5125	0,4750	0,2500
FPR	0,1564	0,1159	0,0967	0,0852	0,0666	0,0527	0,0340	0,0167	0,0041	0,0001
TNR	0,8436	0,8841	0,9033	0,9148	0,9334	0,9473	0,9660	0,9833	0,9959	0,9999
FNR	0,0000	0,0625	0,1250	0,1750	0,2000	0,2500	0,4250	0,4875	0,5250	0,7500

Tabulka 6.7 výsledky testování třídy *DifferenceCountDetector*

Rozumným prahem je u tohoto způsobu detekce hodnota 10 pixelů. V porovnání s předchozí podobnou třídou je při tomto prahu dosahováno menší hodnoty FPR. Práh je opět závislý na rozměrech plátna v pixelech.

6.1.8 Hodnocení offline testů

Na grafu 6.1, znázorňujícím ROC křivky výsledků testování jednotlivých metod, je jasně vidět, že žádná z metod nedosahuje stoprocentního úspěchu bez určité míry falešných poplachů.



Graf 6.1 ROC křivka znázorňující výsledky testování tříd detekujících změnu

Dále je zde vidět, že v oblasti nízkých FPR vykazují nejlepší výsledky třídy *HistogramAvgDetector* a také *SimpleDetector*. Se vzrůstající hodnotou FPR ovšem o prvenství přichází. V oblasti vyšších FPR už jsou jednoznačně nejlepší třídy *DifferenceHistogramDetector* a *DifferenceHistogramAvgDetector*, které vykazují ze všech tříd nejstrmější růst a nejdříve tak dosáhnou hodnot TPR velmi blízkých stu procent. Zatímco u tříd *HistogramDetector* a *HistogramAvgDetector* je jasně vidět pozitivní vliv průměrování, u tříd *DifferenceHistogramDetector* a *DifferenceHistogramAvgDetector* tato technika již žádné zřetelné zlepšení nepřináší. Protože pro výstup aplikace je důležitější dosáhnout co nejvyšších hodnot TPR než co nejnižších FPR, vychází z tohoto porovnání nejlépe třídy *DifferenceHistogramDetector* a *DifferenceHistogramAvgDetector*.

Je potřeba zmínit, že velké množství případů, kdy je mylně detekována změna, je do značné míry způsobeno pohybem přednášejícího mezi plátnem a kamerou. Snížení tohoto počtu je otázkou zapojení algoritmů detekce pohybujících se objektů v obraze, případně detekce člověka v obraze.

Implementované způsoby detekce změny byly vybrány tak, aby jejich hardwarové nároky byly co nejmenší. U většiny použitých metod představuje detekce jeden průchod obrazem, přesněji jeho částí odpovídající plátnu, s pomocí upraveného Bresenhamova algoritmu. Jen v třídách, které pracují s histogramem, je nutné navíc projít tento histogram. Během testování rychlosti metod detekce změny byly naměřeny hodnoty FPS (Frames Per Second, snímků za sekundu) uvedené v tabulce 6.8. Při testování bylo vypnuto ukládání jakéhokoliv výstupu, aby bylo dosaženo co nejobjektivnějších podmínek měření (což je v prostředí Smartphonu velmi relativní).

Třída	FPS
SimpleDetector	13,98
DifferenceDetector	14,08
DifferenceCountDetector	14,10
HistogramDetector	14,15
HistogramAvgDetector	14,54
DifferenceHistogramDetector	14,05
DifferenceHistogramAvgDetector	13,97

Tabulka 6.8 výsledky testování rychlosti metod detekce změny

6.2 Výkon aplikace

Při testování s rozlišením 176 na 144 stíhá aplikace zpracovat 15 snímků za sekundu. Ve chvíli, kdy je detekováno plátno se rychlost sníží na 5 snímků za sekundu. K velkému zpomalení také dochází při ukládání snímků do trvalé paměti, to se však děje jen při dosažení nastaveného limitu paměti pro jejich dočasné uchovávání. Tyto hodnoty byly naměřeny na testovacím Smartphonu Qtek 8500, který představuje výkonnostně nejnižší třídu zařízení s operačním systémem Windows Mobile 5.0. Dá se očekávat, že na zařízeních typu Pocket PC, které jsou vybaveny výkonnějšími procesory, jsou tyto hodnoty vyšší.

7 Závěr

7.1 Shrnutí

Úkolem práce bylo vyvinout systém pro mobilní zařízení, který provádí automatické značkování prezentací v podmínkách běžné posluchárny. Autorovi se podařilo proniknout do problematiky spojené se zpracováním obrazu v omezeném prostředí mobilních zařízení, provést analýzu a navrhnout systém, který je v budoucnu snadno rozšiřitelný.

Povedlo se též implementovat funkční aplikaci a testovat úspěšnost použitých metod detekce. Aplikace je v současné podobě schopna rozpoznávat změnu slajdu na plátně či animaci, ukládat příslušné snímky z kamery a zaznamenávat zvukovou stopu prezentace. K dalšímu vývoji aplikace lze využít možnosti získávat obrazová data z videosouboru. Je nutné podotknout, že kvalita výstupu je závislá na množství parametrů, které jsou pro danou situaci specifické.

Pro řešitele bylo největším přínosem práce seznámení se s vývojem software pro platformu odlišnou od běžného PC. Dále pak porozumění některým významným technologiím společnosti Microsoft a v neposlední řadě prohloubení znalostí a zkušeností s programováním v jazyce C++. Hodnotnou byla i příležitost vyzkoušet si aplikaci zpracování obrazu při řešení praktického problému.

7.2 Možná rozšíření

Automatické nastavení parametrů aplikace na základě vlastností snímané scény by mohlo být předmětem další práce na projektu. Stejně tak by byla vhodným rozšířením implementace metod umožňujících minimalizovat, či zcela eliminovat vliv pohybu přednášejícího mezi kamerou a plátnem, vylepšení uživatelského rozhraní aplikace či rozšíření funkčnosti aplikace na další zařízení.

8 Literatura

[1] AMI: Augmented Multi-party Interaction. Dokument dostupný na URL http://www.amiproject.org/ (květen 2007)

[2] Arlow, J., Neustadt, I.: UML a unifikovaný proces vývoje aplikací. Brno, Computer Press, 2003, 408 s., ISBN 80-7226-947-X

[3] Beneš, B., Felkel, P., Sochor, J., Žára, J.: Moderní počítačová grafika. Brno, 1. vydání, Computer Press, 2004, 612 s., ISBN 80-251-0454-0

[4] Eckel, B.: Myslíme v jazyku C++. Praha, Grada Publishing, 2000, 556 s., ISBN 80-247-9009-2

[5] MSDN Library: DirectShow. Dokument dostupný na URL http://msdn2.microsoft.com/en-us/library/ms940098.aspx (květen 2007)

[6] Pesce, M.: Programming Microsoft DirectShow for Digital Video and Television. Redmond, Microsoft Press, 2003, 451 s., ISBN 0-7356-1821-6

[7] Rábová, I.: Diagramy UML a Rational Rose. Brno, MZLU v Brně, 2000, 32 s.

[8] Sonka, M., Hlavac, V., Boyle, R.: Image Processing, Analysis, and Machine Vision. 2nd edition, Thomson-Engineering, 1998, 800 s., ISBN 0-534-95393-X

[9] Sumec, S.: Extrakce doplňkových materiálů ze záznamů přednášek. Praha, 2006

[10] Video Codecs and Pixel Formats. Dokument dostupný na URL http://www.fourcc.org/ (květen 2007)

[11] Wikipedia, the free encyclopedia: Color Models. Dokument dostupný na URL http://en.wikipedia.org/wiki/Color_models (květen 2007)

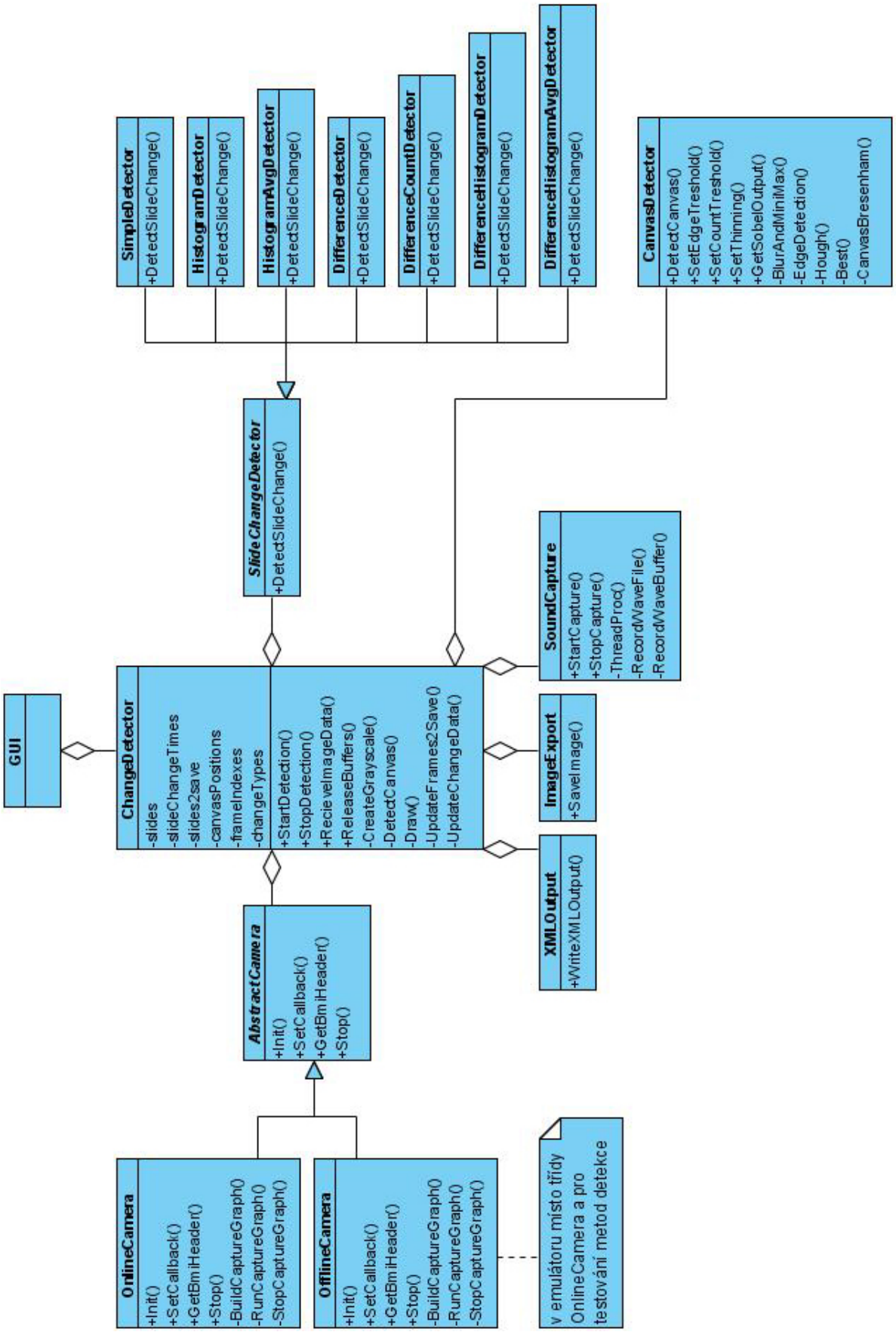
[12] Wikipedia, the free encyclopedia: YUV. Dokument dostupný na URL http://en.wikipedia.org/wiki/YUV (květen 2007)

[13] Windows Mobile Team Blog. Dokument dostupný na URL http://blogs.msdn.com/windowsmobile (květen 2007)

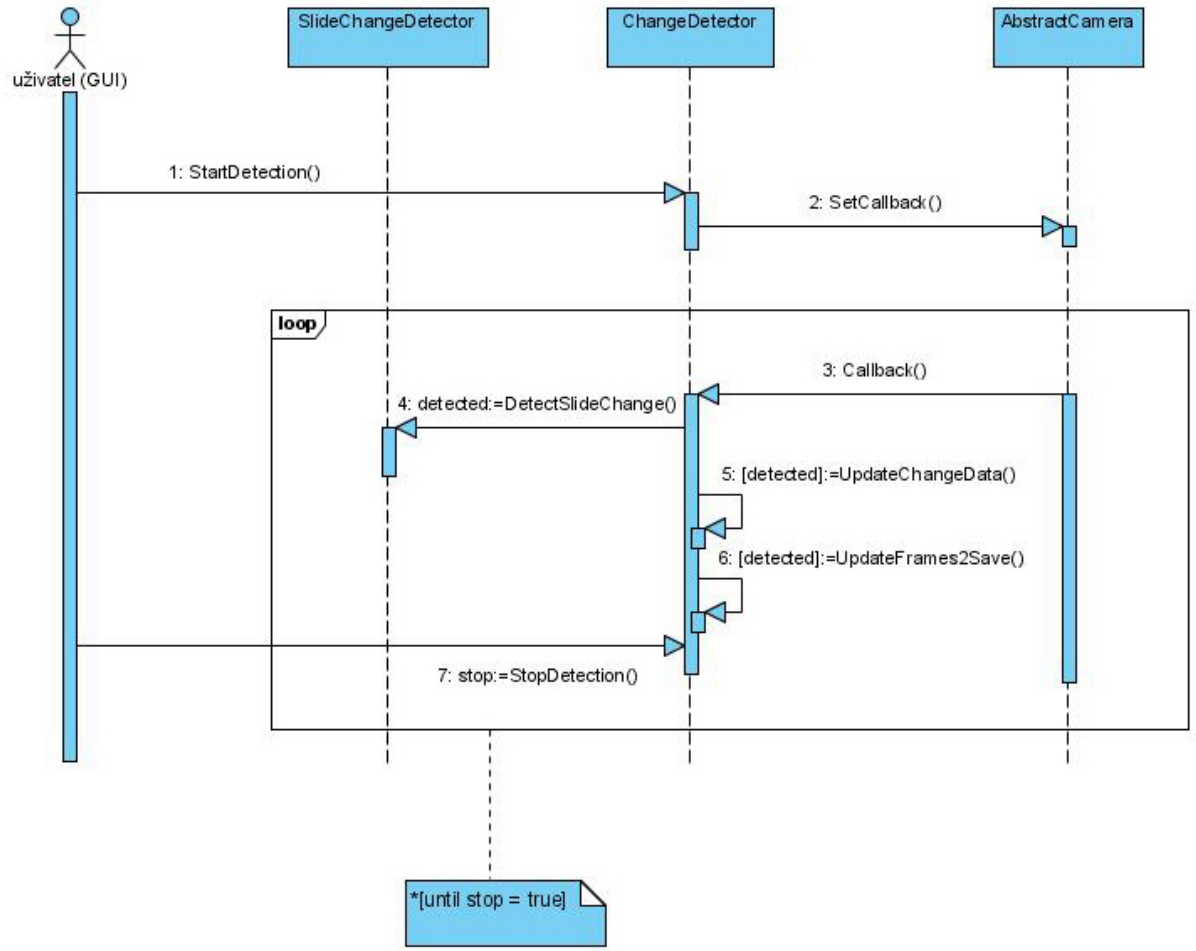
[14] Zemčík, P.: Počítačové vidění - Úvod, motivace, základní principy, aplikace. Dokument dostupný na URL https://www.fit.vutbr.cz/study/courses/POV/private/lectures/POV-Uvod.pdf (květen 2007)

Seznam příloh

Příloha 1. Diagram tříd



Příloha 2. Sekvenční diagram hlavního případu užití



Příloha 3. Sestavení, instalace a spuštění aplikace

Sestavení aplikace

Pro sestavení aplikace ze zdrojových kódů je zapotřebí vlastnit licenci a mít nainstalován nástroj Visual Studio 2005. V něm pak lze otevřít projektové soubory:

- *zdrojove_kody\znackovani\znackovani.sln*,
- *zdrojove_kody\sampleFilter\sampleFilter.sln* (v tomto adresáři je verze filtru funkční pro offline zpracování videosouboru, pro online zpracování je nutné použít filtr z adresáře *zdrojove_kody/ sampleFilter_online_qtek8500, který byl vytvořen a testován pro zařízení Qtek 8500*),
- *zdrojove_kody\transformFilter\transformFilter.sln*

a jednotlivé projekty zkompilovat a sestavit (nabídka *Build – Build Solution*).

V adresáři zdrojove_kody\znackovani\znackovani\Windows Mobile 5.0 Pocket PC SDK (ARMV4I)\Release\ tak vznikne soubor znackovani.exe.

V adresáři *zdrojove_kody\transformFilter\transformFilter\Windows Mobile 5.0 Pocket PC SDK (ARMV4I)\Release* vznikne soubor *ftransform.ax*.

V adresáři *zdrojove_kody\sampleFilter\sampleFilter\Windows Mobile 5.0 Pocket PC SDK (ARMV4I)\Release* vznikne soubor *fsample.ax*.

Instalace aplikace

Před spuštěním aplikace je nutné nejprve připojit zařízení k počítači, zkopírovat soubory do zařízení a zaregistrovat vzniklé filtry *ftransform.ax* a *fsample.ax* v systému Windows Mobile 5.0.

Připojení zařízení se provádí s pomocí usb kabelu nebo nějaké z bezdrátových technologií. V případě použití emulátoru je nutné tento nejprve spustit například ve Visual Studiu s pomocí nabídky *Tools – Device Emulator Manager*. Důležité je mít v systému nainstalován nástroj Microsoft ActiveSync v případě WindowsXP nebo Windows Mobile Device Center v případě Windows Vista (tyto nástroje lze získat zdarma na http://www.microsoft.com/windowsmobile/activesync/default.mspx).

Dalším krokem instalace je přenesení souborů *znackovani.exe, ftransform.ax* a *fsample.ax* do zařízení nebo emulátoru s pomocí nástrojů ActiveSync nebo Windows Mobile Device Center. Soubory *ftransform.ax* a *fsample.ax* musí být v kořenovém adresáři zařízení (jinou lokaci je nutné zohlednit při následné registraci filtrů).

Registraci je možné provést s pomocí nástroje Device Command Shell, který je možné zdarma stáhnout z http://www.gotdotnet.com/workspaces/releases/viewuploads.aspx?id=50618f79-c7b1-4588-9c0a-cf4ddae8092a. Po úspěšné instalaci nástroje lze ve Visual Studiu zvolit nabídku *View – Other Windows – Command Window*. Následně se objeví okno, ve kterém je možné komunikovat se systémem Windows Mobile 5.0 v zařízení nebo emulátoru. Jako první příkaz je nutné zadat

*ce.connec*t následující řetězcem, který určuje cílové zařízení nebo emulátor (v případě připojeného fyzického zařízení – například Smartphonu je to *ce.connect.020:Windows_Mobile_5.0_Smartphone_Device*). Samotná registrace se pak provede příkazy *ce.regsvr fsample.ax* a *ce.regsvr ftransform.ax*.

Spuštění aplikace

Pokud předchozí sestavení a instalace proběhly úspěšně, lze spustit aplikaci *znackovani.exe*. Samotné značkování se spustí v nabídce *Run – Start*. V nabídce *Run* je též možné měnit některé parametry aplikace. Skončení značkování je potřeba potvrdit nabídkou *Run – Stop*. Pro offline zpracování je nutné v zařízení umístit do adresáře *Storage Card*\ testovací videosoubor pojmenovaný *test.avi* (v avi formátu bez komprese). Výstup aplikace *output.xml* je též vytvářen v adresáři *Storage Card*\, snímky jsou ukládány do adresáře *Storage Card\images*\ a zvukový záznam je uložen v souboru *Storage Card\output.wav*.