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

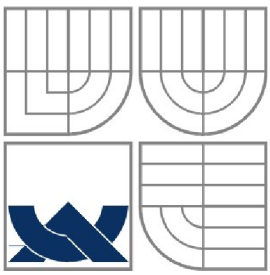
# DETEKCE OBLIČEJŮ V OBRAZE Z KAMERY NA MOBILNÍM TELEFONU S WM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

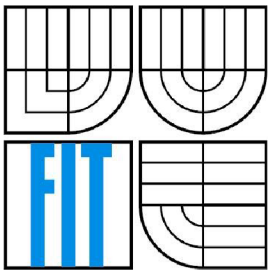
AUTOR PRÁCE
AUTHOR

Bc. MARTIN TUREČEK

BRNO 2010



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

# DETEKCE OBLIČEJŮ V OBRAZE Z KAMERY NA MOBILNÍM TELEFONU S WM

FACE DETECTION IN CAMERA IMAGE ON A MOBILE PHONE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN TUREČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ADAM HEROUT, Ph.D.

BRNO 2010

# Abstrakt

Tato práce se zabývá detekcí obličejů na mobilních telefonech. Konkrétně se zaměřuje na platformu Windows Mobile. Úvod je tedy věnován tomuto operačnímu systému a možnostem práce s kamerou. Další část textu je věnována obecné problematice detekce obličeje v obraze s ohledem na slabý výkon cílových zařízení. Součástí práce je také popis získávání obrazu z kamery pomocí multimediálního frameworku DirectShow a tvorba vlastního transformačního filtru pro detekci obličeje. V závěru jsou shrnuty dosažené výsledky formou testů na několika mobilních zařízení a také jsou zmíněna všechna úskalí, která obnáší vývoj aplikací pro Windows Mobile.



# Abstract

This thesis deals with a face detection on mobile phones. It especially focuses on Windows Mobile platform. The introduction is therefore devoted to this operating system and alternatives of working with the camera. The next part of the text refers to general problems of the face detection in the image considering the weak performance of the target device. Another part of this thesis is a description of the acquisition of images from the camera using DirectShow multimedia framework and creation of a custom transformation filter for the face detection. Achieved results are summarized in the conclusion. It takes a form of tests examining different mobile devices. All difficulties arising during Windows Mobile developing are also mentioned.



# Klíčová slova

detekce obličeje, mobilní telefon, Windows Mobile, DirectShow, AdaBoost, WaldBoost, integrální obraz, klasifikace, příznak, LRD, Haar

# Keywords

face detection, mobile phone, Windows Mobile, DirectShow, AdaBoost, WaldBoost, integral image, classification, feature, LRD, Haar


# Citace

Martin Tureček: Detekce obličejů v obraze z kamery na mobilním telefonu s WM, diplomová práce, Brno, FIT VUT v Brně, 2009

# Detekce obličejů v obraze z kamery na mobilním telefonu s WM

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Martin Tureček
26.5.2010

## Poděkování

Rád bych tímto poděkoval vedoucímu své práce Ing. Adamovi Heroutovi, Ph.D. za jeho ochotu, podnětné konzultace a odborné rady, které mi po celou dobu poskytoval.

# Obsah

# Úvod

Mobilní zařízení jsou ve vyspělých zemích velmi rozšířená (v roce 2008 byl počet aktivních SIM karet v ČR 13 571 000 [1]), přestože se jedná o poměrně mladou část trhu. Je to tedy velmi dynamicky se rozvíjející odvětví s jistou budoucností. Cílem této práce je zjistit, zda je výkon dnešních mobilních zařízení již dostatečný pro algoritmy o velké prostorové a časové náročnosti. Pro tyto účely byla zvolena aplikace, která detekuje obličeje v obraze z kamery, jíž většina moderních mobilních telefonů disponuje. Jistým problémem je, že na trhu je k dispozici celá řada operačních systémů, ať už otevřených nebo uzavřených (např. Windows Mobile, Symbian, Google Android, Openmoko, iPhone OS). To je také jeden z důvodů, proč je vývoj software pro tato zařízení velmi náročný a nákladný a úspěch je závislý na výběru vhodné platformy vzhledem ke zvolenému typu aplikace. Při vývoji je také nutno mít na mysli, že výkon těchto zařízení se nemůže zdaleka rovnat výkonu stolních počítačů. Právě na tento fakt musíme vzhledem ke zvolené problematice neustále pamatovat.

Na základě těchto skutečností byla zvolena platforma Windows Mobile. Tento výběr byl učiněn ze dvou důvodů. Kvůli nezanedbatelnému zastoupení na trhu s mobilními telefony a s vírou, že tato platforma bude nejvhodnější pro zadanou problematiku. Hlavním výrobcem zařízení určených pro tento operační systém je firma HTC, která je známa kvalitním zpracováním a vysokým výkonem svých výrobků.

Tato práce je strukturována tak, aby nejprve čtenáře seznámila obecně s tímto operačním systémem, zejména se zaměří na vývoj softwaru pro tuto platformu. Hlavní pozornost se přitom bude věnovat možnostem získání obrazu z kamery.

V následující části se pak bude zabývat teorií detekce obličeje v obraze. Nejdříve budou uvedeny možné přístupy detekce objektů v obraze a poté se bude podrobněji věnovat problematice detekce obličeje pomocí vybraných metod.

Další kapitola bude věnována návrhu a implementaci aplikace. Postupně se podrobněji seznámíme s frameworkem DirectShow (konstrukce filter graphu, tvorba vlastního transformačního filtru) a poté bude popsána konkrétní aplikace, kterou představuje filtr pro detekci obličejů v obraze.

Poslední kapitola shrne dosažené výsledky. Největší prostor zde bude věnován zejména porovnání vybraných přístupů, experimentům se zvyšováním rychlosti detekce na úkor kvality a dalším experimentům souvisejících s omezením zvolené platformy. Také zde bude prostor pro popis všech platformě specifických potíží, se kterými jsem se během vývoje aplikace setkal.

Právě na základě poslední kapitoly bude učiněn závěr, zda je výkon dnešních chytrých telefonů dostatečný pro náročné aplikace.

# 1 Windows Mobile

Windows Mobile je operační systém, který je vyvíjen firmou Microsoft a je navržen pro smartphony a nejrůznější mobilní zařízení. Jeho předchůdcem je platforma Windows CE. Systém je navržen tak, aby byl ovládáním co možná nejvíc podobný desktopovému systému firmy Microsoft, což až do současné verze (Windows Mobile 6.5.x) vyžaduje používání stylusu. Ovládání je tak sice velmi přesné, ale místy těžkopádné. Společnost Microsoft čelí také velké kritice díky tomu, že uživatelské rozhraní vypadá v porovnání s konkurencí zastarale a naprosto ignoruje moderní styl ovládání (multi-dotykové displeje například nepodporuje vůbec). To je také hlavní příčinou toho, proč Windows Mobile i přes poměrně kvalitní jádro (správa aplikací, multi-tasking) ztrácí podíl na trhu.

Podle uznávané analytické firmy Gartner ztratila platforma Windows Mobile téměř jednu třetinu trhu mezi třetím kvartálem roku 2008 a třetím kvartálem roku 2009 [2]. Nyní ji s 11% podílem na trhu se smartphony patří čtvrté místo (trhu vévodí platforma Symbian – 44,6%, operační systémy firmy RIM se umístily na druhém místě s 20,8% a bronzovou příčku si zajistil iPhone s 17,1%). Propad Windows Mobile je tedy velmi strmý a otázkou je, zda se zastaví, či tato platforma nenávratně zanikne. Firma Microsoft vzhledem k obchodnímu neúspěchu desktopového systému Windows Vista přesunula veškeré své úsilí (finance i pracovní sílu) na vývoj nástupce operačního systému Windows Vista – Windows 7. Učinila tak jistě proto, že na rozdíl od mobilních zařízení, nemá v operačních systémech pro stolní počítače významnějšího tržního konkurenta a ztráta této pozice by měla velmi nepříjemné důsledky. Tento krok ovšem zapříčinil právě to, že Windows Mobile začal k nespokojenosti svých uživatelů zastarávat a ti začali přecházet ke konkurenci.

Podstatnou většinu z oněch výše zmíněných 11% tak připadá na zařízení od tchajwanské firmy HTC Corporation. Možná jen díky ní není operační systém Windows Mobile minulostí. Všechna zařízení této firmy vynikají kvalitním dílenským zpracováním, nadprůměrnou hardwarovou výbavou a přívětivým uživatelským rozhraním. Firma HTC totiž z Windows Mobile zachovala tu kvalitní část (backend) a kompletně předělala uživatelské rozhraní (frontend). Její nejnovější zařízení (HTC Touch HD2 Leo) dokonce umožnilo to, co bylo dlouhou dobu pro uživatele systému Windows Mobile utopií – přinesla multi-dotykové ovládání.

I když tedy HTC radikálně předělalo celý operační systém, vývoj pro tato zařízení to nijak nepostihlo. Kompatibilita se staršími Windows Mobile aplikacemi zůstala a pro vývoj nových je nadále možno používat veškeré vývojové nástroje od firmy Microsoft.

## 1.1 Architektura CPU

V raných verzích (Pocket PC 2000) nebyla standardizována architektura CPU a první zařízení tak běžela na mnoha CPU architekturách (SH-3, MIPS, ARM). V současné době jsou však všechna nová mobilní zařízení založena na architektuře ARM. Ta se ovšem (na rozdíl od WM) neustále vyvíjí a jelikož je Windows Mobile ve své nejnovější verzi (aktuálně 6.5.x) založeno stále na jádře Windows CE 5.2 (z roku 2004), mnoho nových funkcí této architektury není podporováno právě na úrovni jádra operačního systému (například FPU).

## 1.2 Vývoj

K vývoji aplikací pro Windows Mobile je nutno nainstalovat Windows Mobile Software Developmnet Kit, nás konkrétně bude zajímat pouze verze 6. Dostupné jsou dvě varianty tohoto SDK. Verze Standard je určena pro zařízení s Windows Mobile, které nemají dotykovou obrazovku a jsou ovládána výhradně pomocí klávesnice. Verze Professional je naopak určena pro zařízení ovládána pomocí dotykové obrazovky.

Vývoj aplikace je velmi pohodlný zejména díky vývojovému prostředí Visual Studio. To integruje obvyklé nástroje jako jsou editor zdrojového kódu, kompilátor nebo propracovaný debugger. Pro účel této práce je ale hlavní výhodou možnost využít emulátory mobilních zařízení či ladit kód přímo na mobilním zařízení. Naopak velkou nevýhodou je nutnost vlastnit edici Professional, která je pro vývoj na mobilní zařízení podmínkou a není na rozdíl od verze Express zdarma.

Co se samotného vývoje týče, máme na výběr pouze ze dvou relevantních variant. První variantou je psát nativní kód v C++. Druhou alternativou je psát managed kód v C#, který využívá .NET Compact Framework. Tento framework je podmnožinou standardního .NET Frameworku a nabízí spoustu knihoven, které by značně urychlily vývoj. Hlavním problémem ovšem je, že takový program může být spuštěn pouze v režii CLR virtuálního stroje (Common Language Runtime virtual machine), což jej nutně předurčuje k vyšším požadavkům na systémové zdroje. Jelikož se v našem případě jedná o aplikaci, kde je rychlost hlavním cílem, musím se vzdát jisté pohodlnosti, kterou .NET Compact Framework nabízí a vydat se cestou nativního kódu a C++.

## 1.3 Práce s kamerou

V rámci diplomové práce jsem vyzkoušel několik alternativ, jak přistupovat ke kameře mobilního zařízení.

Dokud jsem nezavrhl alternativu vývoje v .NET Compact Framework, pokoušel jsem se zjistit, jaký má tento framework možnosti, co se práce s kamerou týče. Bohužel jedinou dostupnou možností je funkce CameraCaptureDialog, která pouze vyvolá systémovou aplikaci určenou pro práci s fotoaparátem. Pomocí tohoto dialogu tak vývojář pouze dostane cestu k fotografii, či videosekvenci, kterou pomocí aplikace pořídil. Tato alternativa je tedy nepřípustná vzhledem k tomu, že by naše aplikace měla pracovat v reálném čase.

C++ nabízí opět funkci, která stejně jako v předchozím případě vyvolá standardní systémový dialog pro obsluhu fotoaparátu. Jedná se o funkci SHCameraCapture, která je opět pro naše účely nevhodná. Zajímavější ovšem v našem případě je, že Microsoft do svého systému Windows Mobile zahrnul rozhraní COM (Component object model). Na něm je totiž postaven multimediální framework a API DirectShow, který lze též využít na mobilních zařízeních.

Na závěr je ještě třeba zmínit, že ačkoliv plná verze .NET Frameworku přístup ke COM komponentám umožňuje, Compact verze je o tuto možnost ochuzena. Existovala by možnost napsat celou aplikaci jako dynamickou knihovnu v C++, která by řešila všechny kroky nutné k zachytávání a zpracování videa z kamery. Jednotlivé metody by pak bylo možno volat z Compact Frameworku pomocí P/Invoke (Platform Invocation Services). Ta umožňuje volat nativní kód z managed kódu. To by ovšem samotnou detekci zbytečně zdržovalo a význam použití platformy .NET by byl omezen pouze na tvorbu uživatelského rozhraní, což je vzhledem k povaze projektu naprosto zbytečné.

## 1.4 DirectShow

Jak bylo již uvedeno výše, DirectShow je multimediální framework založený na COM. Component Object Model je standard binárního rozhraní, které má za úkol umožnit komunikaci mezi procesy a tvorbu objektů za použití široké palety programovacích jazyků, což umožňuje znovu použít softwarové komponenty v dalších projektech. Veškeré tyto výhody zdědil i DirectShow. Ten dělí komplexní multimediální úkoly do sekvence základních kroků, které se nazývají filtry a ty se sdružují do grafu filtrů (filter graph). Spousta užitečných filtrů je již součástí DirectShow. Programátorovi je ovšem umožněno dopsat si vlastní filtr podle svých vlastních potřeb, který lze pak použít i v jiných aplikacích. Toho bych využil v budoucí práci na projektu, kdy bych rád použil výsledný filtr pro detekci obličeje i pro vytvoření aplikace spustitelné na desktopovém systému Microsoft Windows.

DirectShow je ovšem často kritizován zejména za poměrně náročnou tvorbu vlastních filtrů (podrobněji se k této problematice vrátím ve třetí kapitole).

## 1.5 Windows Phone 7

V průběhu tvorby této diplomové práce oznámil Microsoft následníka operačního systému Windows Mobile – Windows Phone 7. Oficiální uvedení na trh bylo předběžně oznámeno na konec léta roku 2010. Se svým předchůdcem by neměl mít společného téměř nic – jádro operačního systému je kompletně přepsáno, uživatelské rozhraní připomíná rozhraní hudebního přehrávače Zune HD a Microsoft se chlubí těsnou integrací na své nejúspěšnější služby (Xbox Live, Zune nebo Bing) a sociální sítě. Již nyní lze ale odhadnout, že tato nová platforma se svou filozofií bude maximálně snažit přiblížit konkurenčnímu produktu od firmy Apple. Aplikace bude možné nahrávat jen pomocí aplikace Marketplace a budou muset být schváleny přímo odpovědným oddělením Microsoftu. Také multi-tasking nebude plně podporován – alespoň v takové podobě, jaké byl znám ve starších verzí.

Vývoj aplikací by měl být založený na technologiích .NET, Silverlight a XNA. .NET Compact Framework přestane být podporován úplně. Z toho plyne, že podpora existujících aplikací bude zřejmě mizivá.

Microsoft tedy plánuje za Windows Mobile udělat tlustou čáru a začít znovu na zelené louce. Jak tato snaha dopadne ukáže až čas.

# 2 Detekce obličeje v obraze

Detekce obecně je úloha, která má za úkol zjistit, zda jsou v obraze přítomny objekty zájmu. V našem případě se bude jednat o obličeje. Jedná se o složitou úlohu zejména kvůli velké variabilitě objektů. Výraz obličeje, jeho natočení, velikost, nehomogenní osvětlení, nejrůznější stíny z okolí či různá zkreslení kamery jako jsou šum nebo rozmazání – to jsou jen některé faktory, které detekci velmi znesnadňují, ale kvalitní detektor by si s nimi měl poradit.

## 2.1 Přístupy

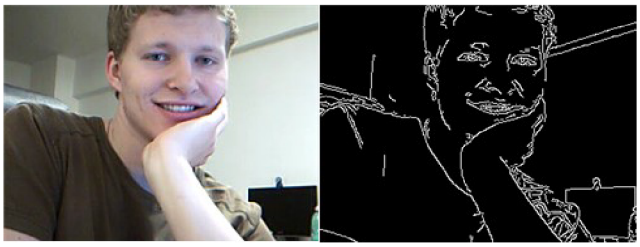
Existuje několik technik používaných k detekci objektů v obraze. Ty se často dělí do čtyř kategorií [3].

**Top-Down**

Techniky z kategorie Top-Down (shora dolů) jsou techniky, které jsou založeny na znalostech o hledaném objektu (knowledge-based) – v našem případě obličeje. Víme tedy například, že obličej se v obraze většinou obsahuje symetricky umístěnou dvojici očí, nos a koutky úst. Vztah mezi těmito příznaky je dán jejich relativní vzdáleností a pozicí. V obraze jsou tedy nejdříve nalezeny tyto příznaky a na základě vytvořených pravidel detekujeme kandidáty – hlavním problémem je právě vytvoření těchto pravidel na základě znalostí o detekovaném objektu. Pokud jsou pravidla příliš striktní, objekty zájmu nejsou často správně detekovány. Pokud jsou pravidla naopak příliš obecná, tak dochází s vysokou pravděpodobností k falešným detekcím. Dále je za použití této techniky obtížná detekce objektu zájmu z různých úhlů pohledu, protože je nutno vytvořit pravidla pro všechny tyto situace.

**Bottom-Up**

Techniky z kategorie Bottom-Up (zdola nahoru) jsou opakem předchozí kategorie, kdy se snažíme najít invariantní příznaky detekovaného objektu. Tato technika je založena na faktu, že člověk dokáže bez významnějších potíží detekovat objekty v různých polohách a za různého osvětlení, takže musí existovat i příznaky, které jsou invariantní vůči těmto změnám. Většina metod patřící do této kategorie nejprve detekuje jednotlivé části objektů a poté na základě statistického modelu, který popisuje vztahy mezi jednotlivými příznaky, určuje přítomnost hledaného objektu. U detekce obličeje jsou těmito příznaky často obočí, nos, oči nebo ústa (které jsou obvykle získávány pomocí hranového detektoru – viz obrázek 1). Problém s těmito příznakově orientovanými algoritmy je, že příznaky mohou být v obraze ovlivněny například stíny. Ty mohou do obličeje zanášet mnoho ostrých hran.



*Obrázek 1 Ilustrace detekce příznaků v obličeji pomocí Cannyho hranového detektoru.*

**Template matching**

Techniky z kategorie Template matching (vyhledávání vzorů) využívají standardní modely hledaného objektu obsahující tvar, barvu a texturu. Tyto modely jsou obvykle definovány ručně nebo na základě parametrizované funkce. V obraze jsou poté vyhledávány jednotlivé prvky objektu, přičemž se určuje míra podobnosti se vzorovými modely. Výhodou tohoto přístupu je, že je velmi jednoduše implementovatelný. Nicméně se také ukázalo [3], že tento přístup není vhodný pro detekci obličeje, protože není schopný se efektivně vyrovnat s různorodostí obličejů (velikost, úhel pohledu atd.)

**Appearance-based**

Do poslední kategorie patří techniky, které nedefinují modely ručně, ale vyvářejí tyto modely automatickým učením z poskytnutých dat. Učení probíhá na základě technik známých ze statistické analýzy a strojového učení, aby se našly relevantní charakteristiky jednotlivých vzorků dat. Naučené charakteristiky jsou ve formě distribučních modelů nebo rozlišujících funkcí, které jsou následně použity pro detekci objektů zájmu. Jelikož se tato práce dále bude zabývat pouze těmito metodami, budou další podrobnosti uvedeny v následujících kapitolách.

## 2.2 Klasifikace

V rámci této práce se tedy budeme zabývat jen appearance-based metodami, které k detekci obličeje využívají klasifikátor. Formálně lze pak úlohu klasifikace popsat následovně:

**Vstup**: trénovací data (dvojice vzorek a jeho ohodnocení – 0=negativní vzorek, 1=pozitivní vzorek)

$$\{(x_1, y_1), \ldots (x_m, y_m)\}, x_i \in X, y_i \in [0,1]$$

**Výstup**: klasifikátor, který danému vzoru přiřadí ohodnocení

$$h: X \to Y$$

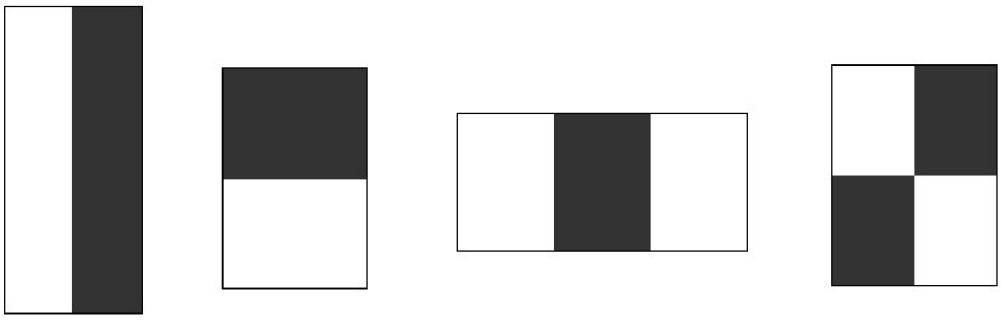
Klasifikační systém se obvykle skládá ze tří částí – získání dat, extrakce příznaků a samotné klasifikace.

## 2.3 Použité příznaky

V úlohách počítačového vidění se často používají obrazové příznaky. Příznakově orientované systémy se oproti pixelově orientovaným systémům používají zejména proto, že jsou daleko rychlejší a flexibilnější.

**Haarovy příznaky**

Haarova báze [9] může být několika typů (viz obrázek 2).



*Obrázek 2: Typy Haarových bází.*

Lze vidět, že Haarovy příznaky jsou rozděleny do několika oblastí (černá a bílá), výpočet konkrétního příznaku je pak realizován jako rozdíl mezi sumou pixelů v bílé a černé oblasti (vzorec 1).

$$f(x) = \sum_{w \in W} x(w) - \sum_{b \in B} x(b) \qquad (1)$$

, kde $x$ je vzorek dat, $W$ a $B$ jsou množiny pixelů příslušející k bílé, respektive k černé oblasti

**Local Rank Patterns**

Local Rank Patterns (dále jen LRP) [6] jsou příznaky, které jsou invariantní vůči nehomogennímu osvětlení, reflektují amplitudu lokálních změn a také poskytují informace o lokálních vzorech.

Pro výpočet LRP uvažujeme skalární obraz $I(x, y) \rightarrow R$ , na němž můžeme definovat následující vzorkovací funkci:

$$S_x^g(u) = (f * g)(x + u) \qquad (2)$$

Parametry této funkce jsou $g$ – jádro konvoluce (konvoluce je provedena před samotným vzorkováním) a vektor $x$, který určuje počátek vzorkování.

Další částí výpočtu LRP je vektor relativních souřadnic, což je vektor dvou-dimenzionálních souřadnic, který definuje okolí libovolného tvaru.

$$U = [u_1 u_2 \dots u_n], u_i \in \mathbb{Z}^2, n \in \mathbb{N} \tag{3}$$

Tento vektor je posléze spolu s vzorkovací funkcí použit pro získání vektoru hodnot popisujících okolí tvaru na pozici x ve skalárním obrazu (maska):

$$M = [S_x^g(u_1)\ S_x^g(u_2) \dots S_x^g(u_n)] \tag{4}$$

Pro každý prvek k v masce M je jeho rank definován jako:

$$R_k = \sum_{i=1}^{n} \begin{cases} 1, if\ M_k < M_i \\ 0, otherwise \end{cases} \tag{5}$$

Stejně jako v případě LRD se jedná o pořadí daného prvku masky v seřazené posloupnosti všech prvků masky. Tato hodnota je opět (stejně jako u LRD) nezávislá na lokální energii v obrázku, což je důležitá vlastnost pro chování LRP příznaku, který je definován:

$$LRP(a, b) = R_a \cdot n + R_b \tag{6}$$

**Local Rank Difference**

Podmnožinou LRP příznaků jsou příznaky Local Rank Differences (dále jen LRD) [7, 8]. Jedná se o příznaky, které jsou založené na lokálním uspořádání hodnot pixelů nebo oblastí.

Pro výpočet LRD opět uvažujeme skalární obraz $I(x, y) \to R$, na němž může být definována následující vzorkovací funkce $(x, y, m, n, u, v, i, j \in \mathbb{Z})$:

$$S_{xy}^{mn}(u, v) = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} I(x + m(u-1) + i, y + n(v-1) + j) \tag{7}$$

Parametry funkce jsou $m$, $n$ – dimenze bloku vzorkování a $x$, $y$ – počátek vzorkování (souřadnice pixelu v obrazu). Na základě této vzorkovací funkce může být následně definována obdélníková maska:

$$M_{xy}^{mnwh} = \begin{bmatrix} S_{xy}^{mn}(1,1) & S_{xy}^{mn}(2,1) & \cdots & S_{xy}^{mn}(w,1) \\ S_{xy}^{mn}(1,2) & S_{xy}^{mn}(2,2) & \cdots & S_{xy}^{mn}(w,2) \\ \vdots & \vdots & \ddots & \vdots \\ S_{xy}^{mn}(1,h) & S_{xy}^{mn}(2,h) & \cdots & S_{xy}^{mn}(w,h) \end{bmatrix} \quad (8)$$

Parametry této masky jsou stejné jako v případě vzorkovací funkce. Další dva parametry (*w, h*) určují dimenzi masky. Experimenty ukázaly [8], že pro detekci obličeje operující s pod-oknem o velikosti 24x24 pixelů jsou dimenze vzorkovacího bloku (*m, n*) 1x1, 2x2, 2x4 a 4x2 dostatečné.

Pro každou pozici v masce je definován rank:

$$R_{xy}^{mnwh}(u,v) = \sum_{i=1}^{w}\sum_{j=1}^{h} \begin{cases} 1, & if\ s_{xy}^{mn}(i,j) < S_{xy}^{mn}(u,v) \\ 0, & otherwise \end{cases} \quad (9)$$

Rank je pořadí daného člena masky v seřazené posloupnosti všech členů masky. Tato hodnota je nezávislá na lokální energii v obrázku, což je důležitá vlastnost pro chování LRD příznaku, který je definován:

$$LRD_{xy}^{mnwh} = (u,v,k,l) = R_{xy}^{mnwh}(u,v) - R_{xy}^{mnwh}(k,l) \quad (10)$$

# 2.4 Viola-Jones

Paul Viola a Michael Jones v roce 2001 ve své práci [4] popsali framework pro detekci objektů, který dosahuje velmi vysoké rychlosti detekce. Ten se skládá ze tří klíčových částí. První z nich je reprezentace obrazu, která se nazývá integrální obraz. Další částí je učící algoritmus, který je založen na AdaBoostu. Posledním částí je metoda kombinování klasifikátorů do kaskády. Hlavním přínosem této metody je rychlé odmítnutí částí obrazu, kde se nachází například pozadí, zatímco „slibné“ části podstupují důkladnější zpracování.

**AdaBoost**

Ve výše zmíněné práci [4] je použita alternativa algoritmu AdaBoost, která se využívá jak pro extrakci příznaků, tak pro trénování klasifikátoru. Ve své původní podobě je tento algoritmus využíván, aby zvýšil výkon jednoduchého učícího algoritmu. Toho je dosaženo lineární kombinací slabých klasifikátorů (klasifikátor, jehož jedinou podmínkou je úspěšnost vyšší než 50% - nejedná se tedy o prostý odhad) do silného (nelineárního) klasifikátoru. Tím je docíleno zvýšení klasifikační přesnosti (boosting).

Vstupem samotného algoritmu je trénovací množina dvojic $(x_i, y_i)$, kde $x_i$ je vzorek a $y_i$ je třída vzorku ( $y_i \in [0,1]$ ) a výstupem klasifikátor $h(x)$

- Je dána množina trénovacích obrázků $(x_1, y_1)...(x_n, y_n)$ , kde $y_i \in [0,1]$ , kde 0 značí negativní vzorek a 1 vzorek pozitivní
- Inicializace vah $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ pro $y_i = 0$ , respektive $y_i = 1$ , kde $m$ je celkový počet negativních vzorků a $l$ je celkový počet pozitivních vzorků
- Pro každé $t = 1, ..., T$ :
  1. Normalizace vah, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$ , takže $w_t$ je rozložení pravděpodobnosti
  2. Pro každý příznak $j$ trénuj klasifikátor $h_j$ , který je omezen použitím právě jednoho příznaku. Chyba je vyhodnocena s ohledem na $w_t$ , $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$
  3. Vyber klasifikátor $h_t$ s nejmenší chybou $\epsilon_t$
  4. Aktualizuj váhy: $w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$ , kde $e_i = 0$ pokud je $x_i$ klasifikováno korektně, jinak $e_i = 1$ a kde $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$
- Výsledný silný klasifikátor je: $h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & jinak \end{cases}$ , kde $\alpha_t = \log \frac{1}{\beta_t}$

*Tabulka 1: Učící algoritmus AdaBoost podle [4]. Celkem je vytvořeno T slabých klasifikátorů, každý za použití právě jednoho příznaku. Výsledný silný klasifikátor je vytvořen lineární kombinací těchto slabých klasifikátorů.*

**Integrální obraz**

Viola-Jones algoritmus předpokládá využití Haarových příznaků, jejichž hodnota se vypočítá jako rozdíl několika sum pixelů v obraze. Aby byl tento výpočet urychlen, je možno si z původního obrazu před-počítat obraz integrální. Ten je definován:

$$ii(x, y) = \sum_{x' \leq x,\, y' \leq y} i(x', y') \tag{11}$$

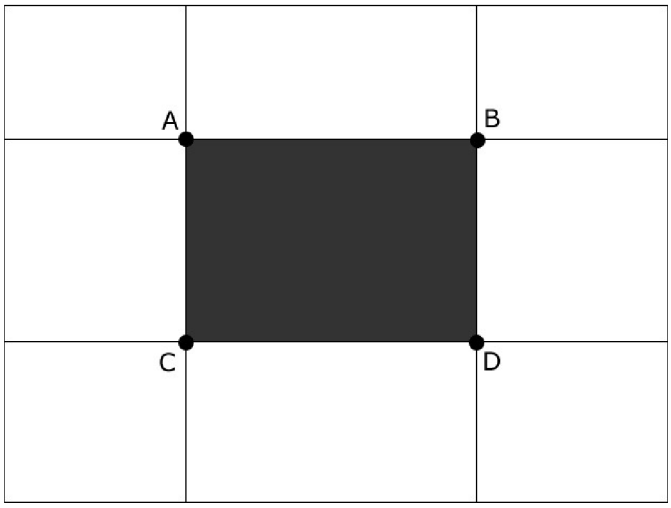
, kde *ii(x,y)* je integrální obraz, *i(x,y)* je původní obraz**.** Výhodou integrálního obrazu je, že jsme schopni jej vypočítat jedním průchodem.

$$\begin{aligned} s(x, y) &= s(x, y-1) + i(x, y) \\ ii(x, y) &= ii(x-1, y) + s(x, y) \end{aligned} \tag{12}$$

, kde *s(x,y)* je kumulativní suma řádku, *s(x,-1)=0* a *ii(-1,y)=0*.


*Obrázek 3: Integrální obraz: hodnota bodu (x, y) je suma všech pixelů nad a vlevo od daného bodu.*

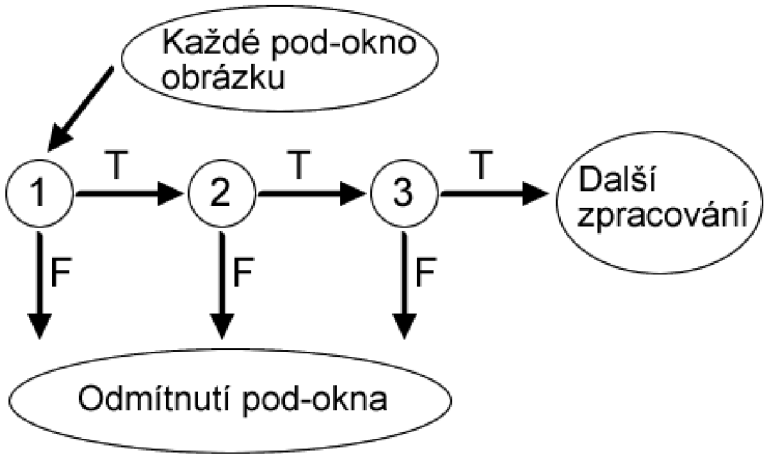
Jakmile máme k dispozici integrální obraz, sumu hodnot pixelů v obdélníkové oblasti můžeme vypočítat pouze pomocí čtyř referencí. Hodnota Haarových příznaků založených na dvou obdélnících tak může být vypočítána s použitím pouhých šesti referencí (dva body jsou pro oba obdélníky společné), osmi referencí pro tří-obdélníkové příznaky a devíti referencí pro čtyř-obdélníkové.




*Obrázek 4: Suma všech pixelů v oblasti šedého obdélníku lze z integrálního obrazu spočítat jako A+D – (C+B).*

**Kaskáda klasifikátorů**

Hlavním důvodem pro použití kaskády klasifikátorů je dosáhnout větší úspěšnosti detekce při snížení výpočetního času. Toho dosáhneme tím, že zkombinujeme několik slabých klasifikátorů (jeho chyba je menší než 0.5, takže se nejedná o náhodné rozhodování) v jeden silný klasifikátor. Většinou bývá nejdříve použita sada několika jednoduchých klasifikátorů, které jsou schopny jednoznačně a hlavně rychle odmítnout většinu negativních pod-oken obrazu, až poté přijdou na řadu komplexnější klasifikátory, které mají za úkol snížit celkovou false-positive hodnotu. Objekt v aktuálně zpracovávaném pod-okně je detekován až v případě, že dané pod-okno prošlo celou kaskádou, aniž by bylo odmítnuto.




*Obrázek 5: Kaskáda klasifikátorů.*

## 2.5 WaldBoost

Jiný přístup zvolila metoda WaldBoost [5], která je kombinací algoritmu AdaBoost a metody Sequential probability ratio test (SPRT). Klasifikátor, který je natrénován touto metodou, nemusí být vyhodnocen celý. Pokud totiž v průběhu klasifikace nastane jistota, že vzorek patří do určité třídy, můžeme na základě této skutečnosti rozhodování ukončit.

SPRT pracuje se dvěma prahy (*A, B*). Během vyhodnocování se v každém kroku provede vždy jedno měření, jehož výsledek je zahrnut do celkové hodnoty $R_t$. Ta je poté porovnávána s jednotlivými prahy. Pokud je hodnota $R_t$ větší nebo rovna hodnotě prahu *B*, pak je klasifikace *S* vyhodnocena +1 (vzorek je úspěšně klasifikován). Pokud je hodnota $R_t$ menší nebo rovna prahu *A*, je klasifikace *S* vyhodnocena jako -1 (pod-okno odmítnuto). V případě, že hodnota $R_t$ leží mezi prahy *A* a *B*, je nutno provést další krok.

$$S\begin{cases}+1, & R_t \le B \\ -1, & R_t \ge A \\ \varphi, & B < R_t < A\end{cases} \tag{13}$$

, kde $R_t$ je likelihood ratio

$$R_t = \frac{p(x_1, \ldots, x_t)|y=-1}{p(x_1, \ldots, x_t)|y=+1} \tag{14}$$

- Vstup: $(x_1, y_1) \ldots (x_l, y_l)$ , kde $x_i \in X$ a $y_i \in \{-1, 1\}$ ; požadovaná cílová $\alpha$ (false negative rate) a $\beta$ (false positive rate)
- Inicializace: Váhy vzorků $w_1(x_i, y_i) = \frac{1}{l}$ , $A = \frac{(1-\beta)}{\alpha}$ a $B = \frac{\beta}{(1-\alpha)}$
- Pro každé $t = 1, \ldots, T$ :
  1. Vyber $h_t$ podle $h^{(T+1)} = \frac{1}{2} \log \frac{P(y=+1 \vee x, w^{(T)}(x, y))}{P(y=-1 \vee x, w^{(T)}(x, y))}$
  2. Určení likelihood ratio $R_t$ podle vzorce 14
  3. Nalezení prahu $\theta_A^{(t)}$ a $\theta_B^{(t)}$
  4. Odstranění vzorků z trénovací množiny, pro které platí $H_t \ge \theta_B^{(t)}$ nebo $H_t \le \theta_A^{(t)}$
  5. Vložení nových dat do trénovací množiny
- Výstup: silný klasifikátor je: $H_T$ a dva prahy $\theta_A^{(t)}$ a $\theta_B^{(t)}$

*Tabulka 2: Algoritmus WaldBoost.*

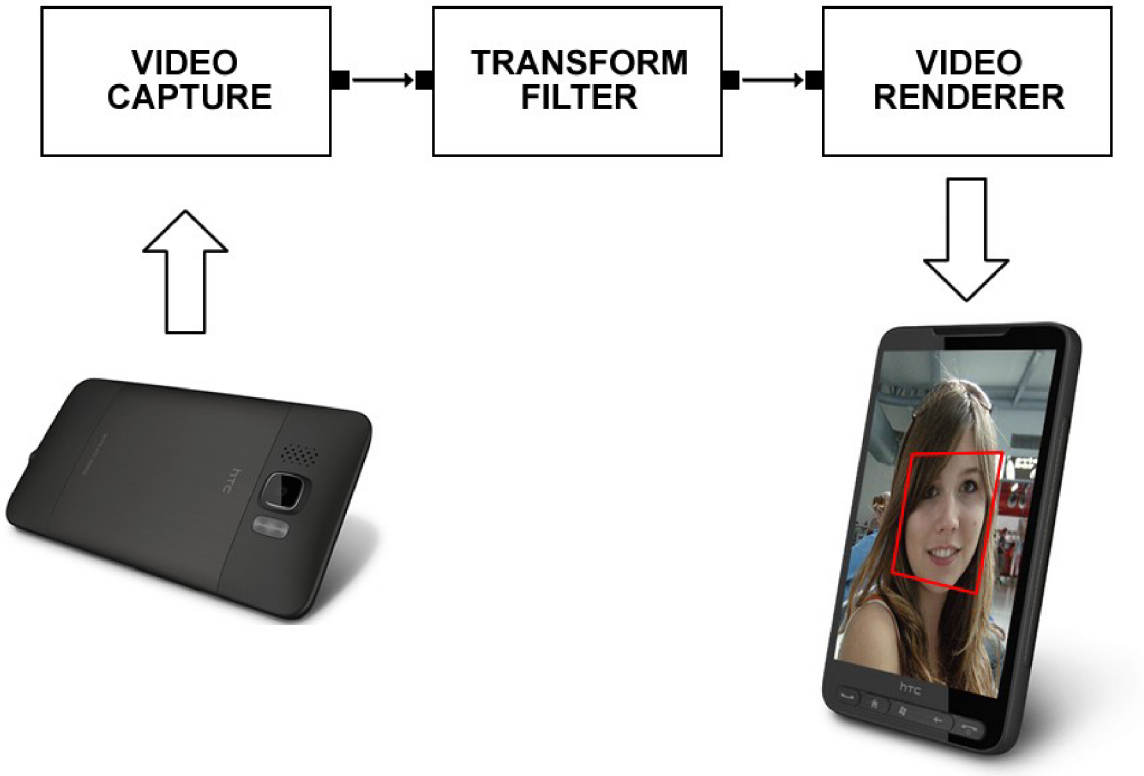
# 3 Návrh a implementace

V rámci diplomové práce jsem navrhl a implementoval mechanismus pro zachytávání videa z kamery na Windows Mobile pomocí frameworku DirectShow a zrealizoval vlastní filtr, který má možnost s těmito daty pracovat. V této kapitole následuje popis, jak jsem postupoval při návrhu a implementaci této úlohy.

## 3.1 Filter graph

Základním stavebním prvkem DirectShow je komponenta, která se nazývá filtr [10]. Filtr je obecně softwarová komponenta, která provádí nějakou operaci nad multimediálním streamem. Rozhraní DirectShow je navrženo tak, že výstup jednoho filtru může být zároveň zdrojem pro vstup jiného filtru. Několik takto spojených filtrů se nazývá filter graph.

Pro účely aplikace vyvíjené v rámci diplomové práce jsem navrhl jednoduchý filter graph, jehož grafické schéma je uvedeno na následujícím obrázku:




Obrázek 6: *Filter graph pro zachytávání videa a následnou detekci obličeje.*

První filtr slouží pro zachytávání videa z kamery. Jeho výstup slouží jako vstup pro vlastní transformační filtr, který v obraze detekuje obličeje. Kolem detekovaného obličeje pak ještě nakreslí čtverec. Takto transformovaný obraz poslouží jako vstup posledního filtru – renderovacího filtru, který má za úkol svůj vstup zobrazit na obrazovce zařízení.

**Konstrukce filter graphu**

Existují tři základní úkoly, které musí provést každá aplikace používající DirectShow [10]:

1. Programátor musí vytvořit instanci filter graph manageru (umožňuje správu filter graphu).
2. Aplikace za pomocí filter graph manageru (vytvořen v předchozím kroku) sestaví filter graph.
3. Aplikace pomocí filter graph manageru ovládá sestavený filter graph. Filter graph během zpracovávání multimediálního streamu také generuje události, na které musí aplikace správně reagovat. (Na základě tohoto bodu se obecně doporučuje, aby Filter Graph nebyl konstruován z hlavního vlákna aplikace - jinak hrozí deadlock [12])

Instance COM objektu se vytvoří pomocí funkce CoCreateInstance, jejíž jediným parametrem je CLSID instancované třídy. Abychom mohli používat filter graph, musím nejdříve vytvořit instanci capture graph builderu (rozhraní ICaptureGraphBuilder2 [14]), který poskytuje metody pro vytváření filter graphu. Až poté můžeme přistoupit k vytvoření samotného filter graph manageru:

```
CComPtr<ICaptureGraphBuilder2> m_pCaptureGraphBuilder;
CComPtr<IGraphBuilder> pFilterGraph;

m_pCaptureGraphBuilder.CoCreateInstance(CLSID_CaptureGraphBuilder);
pFilterGraph.CoCreateInstance(CLSID_FilterGraph);
m_pCaptureGraphBuilder->SetFiltergraph(pFilterGraph);
```

*Tabulka 3: Vytvoření instance filter graph manageru pomocí capture graph builderu.*

Pomocí takto vytvořené instance filter graph manageru můžeme začít se samotnou konstrukcí filter graphu. Jednotlivé filtry se opět instancují pomocí funkce CoCreateInstance. K přidání filtru do filter graphu slouží pak metoda filter graph manageru – AddFilter. Z obrázku 6 je patrné, že prvním filtrem je filtr pro zachytávání videa z kamery. Vytvoření takového filtru je v DirectShow poměrně jednoduché, při jeho konstrukci mu je ovšem nutno poskytnout informace o zařízení, které slouží jako zdroj dat. Tato informace se předává pomocí objektu, jemuž byla přes rozhraní IPropertyBag

poskytnuta možnost ukládat perzistentně svá nastavení [13]. Velmi zjednodušená konstrukce takového filtru pak vypadá následovně:

```
CPropertyBag pPropBag;
CComVariant varCamName  = L"CAM1:";
CComPtr<IPersistPropertyBag> pVideoPropertyBag = NULL;
CComPtr<IBaseFilter> m_pVideoCaptureFilter;

m_pVideoCaptureFilter.CoCreateInstance(CLSID_VideoCapture);
m_pVideoCaptureFilter->QueryInterface(IID_IPersistPropertyBag,
      (void **)&pVideoPropertyBag);
pPropBag.Write(L"VCapName", &varCamName);
pVideoPropertyBag->Load(&pPropBag, NULL);
pFilterGraph->AddFilter(m_pVideoCaptureFilter, L"Video Source");
```

*Tabulka 4: Zjednodušená konstrukce filtru pro zachytávání obrazu z kamery.*

Nejdůležitějším filtrem je transform filter, který se stará o samotnou detekci obličeje v obraze. Jeho implementace bude popsána později. V rámci konstrukce filter graphu jen zmíním, že přidání vlastního filtru se ničím neliší od přidávání ostatních filtrů. Opět je nutné vytvořit hlavně jeho instanci pomocí známého CLSID daného filtru a poté jej přidat do filter graphu.

Posledním filtrem je video renderer, jehož úkolem je zobrazit výsledný stream. Tento filtr se vytvoří již spolu s vytvořením instance filter graph manageru, je však nutné jej správně nastavit. Dále je třeba správně propojit výstup a vstup jednotlivých filtrů, k čemuž poslouží metoda capture graph builderu – RenderStream [15]. Pomocí parametrů této metody se definuje, které filtry se mají spojit a jaká kategorie výstupů, respektive vstupů bude použita (v našem případě se jedná o kategorii PIN_CATEGORY_PREVIEW, abychom docílili okamžitého zobrazení vstupu z kamery na displej mobilního zařízení) a také typ média, který je v konkrétním případě relevantní (u nás MEDIATYPE_Video). Jelikož je celá aplikace programována za pomocí Windows API, je dobrým zvykem nastavit okno, které bude vlastníkem video okna. K tomu účelu slouží rozhraní IVideoWindow [16]. Celé nastavení renderování pak může vypadat následovně:

```
CComPtr<IBaseFilter> pVideoRenderer;
CComPtr<IVideoWindow> pVideoWindow;

m_pCaptureGraphBuilder->RenderStream(
      &PIN_CATEGORY_PREVIEW, &MEDIATYPE_Video, m_pVideoCaptureFilter,
      m_pFaceDetectionFilter, NULL);
pFilterGraph->FindFilterByName(TEXT("Video Renderer"), &pVideoRenderer);
pVideoRenderer->QueryInterface(IID_IVideoWindow,
                           (void**) &pVideoWindow);

pVideoWindow->put_Owner((OAHWND)m_hwnd);
// další nastavení - fullscreen, pozice okna, atd.
```

*Tabulka 5:Propojení DirectShow filtrů a nastavení vykreslování.*

Tím jsme úspěšně zkonstruovali filter graph, který odpovídá návrhu na obrázku 5. Rozhraní IMediaControl [17] nám pak umožňuje ovládat tok dat skrze filter graph (např. metody Run a Stop). Spuštění všech filtrů ve filter graphu například jednoduše docílíme:

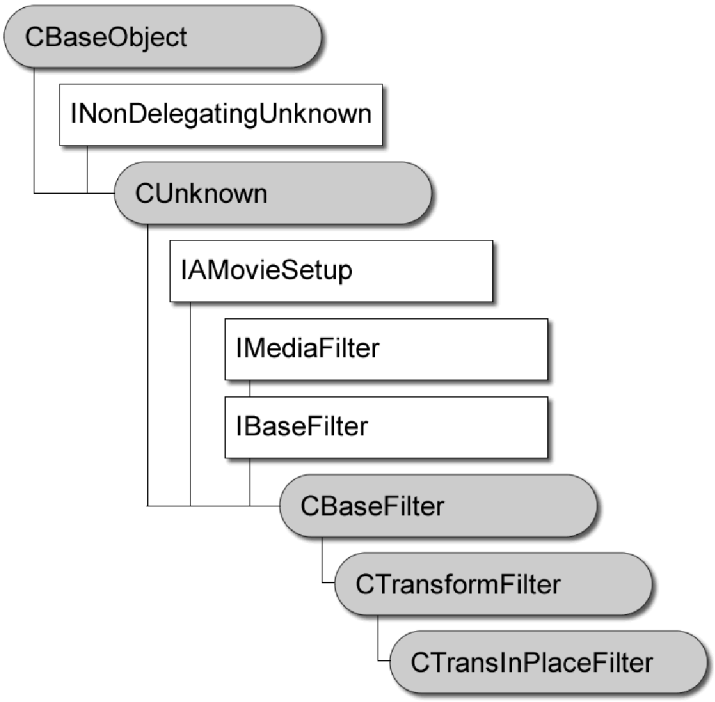
```
CComPtr<IMediaControl> pMediaControl;

pFilterGraph->QueryInterface(IID_IMediaControl, (void **)&pMediaControl);
pMediaControl->Run();
```

*Tabulka 6: Příklad ovládání filter graphu pomocí rozhraní IMediaControl.*

## 3.2 Transformační filtr

Transformační filtr je filtr, který má právě jeden vstup a jeden výstup. Ačkoliv to není nutnou podmínkou, doporučuje se, aby takovýto filtr vždy dědil z bázové třídy [11]. V případě našeho filtru pro detekci obličejů se nabízejí následující dvě bázové třídy:

- CTransformFilter – slouží jako základ transformačnímu filtru, který odděluje buffer pro vstupní a výstupní data. Tento filtr je často nazýván copy-transform filtr – jakmile na vstupu přijme nějaká data, zkopíruje je do výstupního bufferu.
- CTransInPlaceFilter – je navržen tak, aby se veškeré operace prováděli přímo na vstupním bufferu, který je předán na výstup. Tento filtr je potomkem CTransInPlaceFilter.




*Obrázek 7: Hierarchie bázových tříd [20].*

Zásadní nevýhodou bázové třídy CTransInPlaceFilter je, že alokace paměti není v režii samotného filtru, ale externího filtru (většinou rendereru). To může značně omezovat výkon při čtení dat ze vstupního obrazu, který může ve skutečnosti být uložen ve video paměti grafické karty. Filtr postavený na CTransformFilter se musí o alokaci bufferu pro výstup starat sám a vzniká zde tak režie při kopírování vstupního bufferu do bufferu výstupního. Obecně tedy platí, že výhodnější by mělo být vytvářet transformační filtr jako potomka CTransformFilter. To se ale během vývoje aplikace nepotvrdilo a alespoň u konkrétní implementace filtru pro detekci obličejů jsem žádný rozdíl v rychlosti nezaznamenal.

Po výběru bázové třídy je na řadě deklarace třídy. Nejdůležitější částí je vygenerování CLSID (toto GUID jednoznačně určuje filtr v systému a ten může být následně používán i jinými aplikacemi):

```
// {6D9373D2-01F4-405b-857B-34BA40B1B455}
DEFINE_GUID(CLSID_FaceDetectionFilter,
      0x6d9373d2, 0x1f4, 0x405b, 0x85, 0x7b, 0x34, 0xba, 0x40, 0xb1, 0xb4, 0x55);
```

*Tabulka 7:Příklad CLSID, které jednoznačně identifikuje filtr v systému.*

Kdykoliv jsou dva filtry spojeny, oba musí souhlasit s typem média, se kterým se bude pracovat v rámci tohoto spojení. Toto vyjednávání pracuje na úrovni pinů, nicméně CTransformFilter

(a tedy i jeho potomek – CTransInPlaceFilter) implementuje jednoduchý framework, který tuto proceduru značně usnadňuje. V případě CTransformFilteru se jedná o tři čistě virtuální metody – CheckInputType, CheckTransform a GetMediaType – které je nutno implementovat. V případě CTransInPlaceFilter se jedná o ještě jednodušší úlohu, kdy je potřeba implementovat jen funkci CheckInputType, která vrací, zda je filtr schopen pracovat s typem média na vstupu.

Po dokončení vyjednávání o typu média, můžeme v rámci CTransInPlaceFilter přistoupit k tomu nejdůležitějšímu, což je implementace metody Transform. Filtr, jehož výstup má náš transformační filtr na svém vstupu, doručuje jednotlivé snímky voláním metody rozhraní IMemInputPin – Receive. Transformační filtr pak tyto data zpracovává voláním metody Transform (jedná se o čistě virtuální metodu), která zajišťuje hlavní funkčnost filtru. CTransformFilter a CTransInPlaceFilter se pak liší v přístupu k zpracování jednotlivých snímků. V našem případě máme operaci usnadněnou, protože získání ukazatele na obrazová data je velmi jednoduché (viz následující tabulka):

```
BYTE * pImageData = NULL;
pMediaSample->GetPointer(&pImageData);
```

*Tabulka 8: Získání ukazatele na surová data obrázku.*

Význam těchto dat je závislý na vyjednaném typu média. Pokud se například filtry dohodly na formátu RGB565, jsou jednotlivé pixely popsány dvěma bajty.

Posledním krokem je přidání podpory pro COM rozhraní. Není potřeba implementovat metody pro počítání referencí (AddRef a Release), protože tyto metody jsou již implementovány v bázové třídě transformačních filtrů – CUnknown. Jelikož náš filtr kromě rozhraní IBaseFilter (skrze CBaseFilter) neimplementuje žádné jiné rozhraní, není třeba dělat již nic jiného. V případě, že by náš filtr implementoval jiná rozhraní a my bychom rádi umožnili přístup k těmto rozhraním mimo samotný filtr (pomocí metody QueryInterface [18]), je nutné provést dva další kroky – vložit makro DECLARE_IUNKOWN do veřejné deklarační části a přetížit metodu NonDelegatingQueryInterface, která zajišťuje kontrolu IID rozhraní, a v případě úspěchu této kontroly vrací ukazatel na transformační filtr.

Aby bylo možno distribuovat filtr jako dynamicky linkovanou knihovnu, musíme provést několik dalších kroků [19]:

1. Definovat statickou metodu, která vytvoří a vrátí instanci našeho nového filtru. Její jméno může být libovolné, musí ovšem dodržovat definovanou signaturu.

2. Deklarovat globální pole CFactoryTemplate pojmenované jako g_Templates, které obsahuje informace potřebné pro zaregistrování filtru (jméno filtru, CLSID, metoda vytvářející instanci filtru atd.)
3. Definovat globální proměnnou typu integer, která bude pojmenována jako g_cTemplates, jejíž hodnota bude odpovídat délce g_Templates pole z předchozího kroku
4. Posledním krokem je implementace funkcí, které umožňují registrovat (DLLRegisterServer) a odregistrovat (DLLUnregisterServer) DirectShow filtr

Tímto je filtr hotový. Po zkompilování a sestavení filtru je ještě třeba filtr zaregistrovat v systému, aby jeho instance bylo možné vytvářet pomocí CLSID. V rámci diplomového projektu jsem zatím pouze vytvořil malou utilitku v .NET Compact Frameworku, která pomocí P/Invoke zaregistruje, respektive odregistruje daný filtr. V dalším pokračování práce je v plánu registraci zahrnout do instalačního balíčku.

Nyní je možno filtr používat standardním způsobem (vytvoření instance pomocí CLSID a přidání do filter graphu) nejen z vytvořené testovací aplikace, ale i z jakékoliv jiné aplikace využívající DirectShow. Neměl by být problém výsledný filtr přeložit a používat na stolním PC, ale to je až záležitost dalšího pokračování práce.

## 3.3 Filtr pro detekci obličejů v obraze

Na základě předchozí kapitoly byl realizován filtr, jež detekuje ve vstupním obraze obličeje. Úspěšnou detekci pak naznačí nakreslením čtverce kolem pozitivně klasifikovaného pod-okna.

**Barevný prostor**

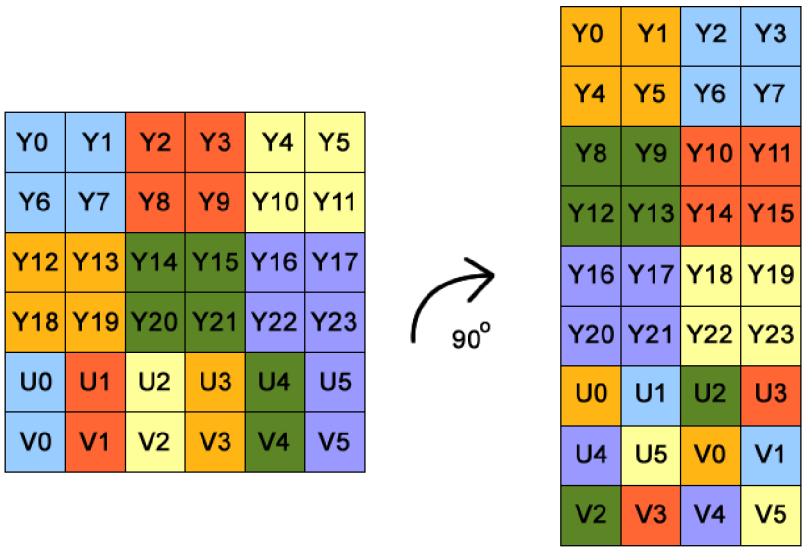
Jak bylo uvedeno v předchozí kapitole, vlastní DirectShow filter musí pracovat s reprezentací obrazu v různých barevných prostorech. Je to dáno širokou variabilitou zařízení, z nichž každé používá jiný barevný prostor (např. RGB24, RGB565, YV12). Filtr pro detekci obličeje byl od začátku vyvíjen pro barevný prostor YV12 [21] (často označován jako YUV420), který se nakonec ukázal jako velice výhodný. Pro získání barevného modelu YUV z RGB24 je definován následující vztah:

$$\begin{aligned} Y &= (0{,}299 * R) + (0{,}587 * G) + (0{,}114 * B) \\ V &= (B - Y) * 0{,}565 \\ U &= (R - Y) * 0{,}713 \end{aligned} \tag{15}$$


*Obrázek 8: Barevný prostor YV12.*

Z obrázku 8 lze vidět, že každý bod je složen ze tří komponent Y (luma – jas), U a V (komponenty nesoucí údaje o barvě) – každá složka zabírá v paměti jeden byte. Složky Y jsou rozděleny do matice 2x2 pixelu, kterým odpovídá vždy jedna složka U a V. Z toho plyne, že jeden obrazový bod je popsán 1,5 bytů. Pro naše účely je tento barevný prostor maximálně výhodný, protože není potřeba vstupní obraz převádět do stupňů šedi. Veškeré potřebné informace obsahuje složka Y, která je navíc na začátku pole s daty vstupního obrazu.

Drobný problém nastal v případě nutnosti vstupní obraz otočit o 90 stupňů (důvody budou popsány později). Rotace obrazu potom není triviální, protože se musí zachovat korespondence barevných složek U a V s maticí 2x2 složek Y.




*Obrázek 9: Rotace obrazu v barevném prostoru YV12.*

**Reprezentace obrazu**

Reprezentace obrazu, kterou používají klasifikátory, se liší podle použitých příznaků. Klasifikátor využívající LRD příznaky používá pro reprezentaci obrazu pole bytů, kde každý prvek pole odpovídá hodnotě intenzity jednoho pixelu obrazu. Tato třída je odvozena ze třídy AbstractImage, která mimo vlastní data obrazu uchovává i další nezbytná data pro další zpracování (výška, šířka, relativní velikost oproti původnímu obrazu z kamery).

Klasifikátor využívající Haarovy příznaky používá pro reprezentaci obrazu integrální obraz. Jedná se o obraz, který byl popsán v předchozí kapitole a umožňuje rychlý výpočet sumy pixelů v dané části obrazu. Kromě klasického integrálního obrazu obsahuje třída IntegralImage i kvadratický integrální obraz, který je při výpočtu odezvy jednotlivých slabých klasifikátorů používán pro rychlý výpočet standardní odchylky. Pomocí této odchylky se potlačují vlivy nehomogenního osvětlení.

**Obrazová pyramida**

Aby byl detektor obličejů invariantní vůči vzdálenosti obličeje od kamery, je nutno vytvořit několik pod-vzorkovaných obrazů vstupního obrazu z kamery. Díky tomu jsme schopni detekovat obličeje, které jsou velmi blízko kamery a na které by byla standardní velikost pod-okna (typicky 24x24 pixelů) nedostatečná.

Bázovou třídou pro detekci obličejů je třída AbstractImagePyramid. Pro potřeby aplikace z ní byly odvozeny dvě třídy – ImagePyramid (pro klasifikátor natrénovaný nad LRD příznaky) a IntegralImagePyramid (pro klasifikátor natrénovaný nad Haarovými příznaky). V rámci diplomové práce byla také implementována metoda využívající Haarovy příznaky, která obrazovou pyramidu nepotřebuje a různou vzdálenost obličeje od kamery řeší zvětšováním detekčního okna.

**Pod-vzorkování obrazu**

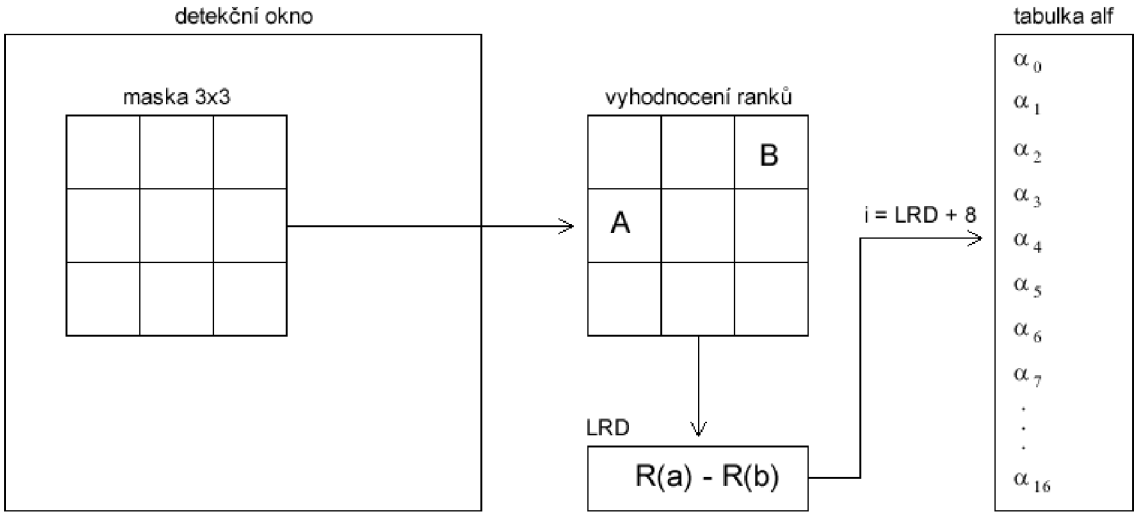
Pro pod-vzorkování obrazu byl zvolen algoritmus nearest-neighbour. Je to algoritmus, který je velmi jednoduchý a jeho výsledkem je obraz, který není moc kvalitní. Výsledek je ale dostatečný pro detekci obličeje, protože většina hlavních rysů zůstává zachována. Hlavní výhodou je tak rychlost tohoto algoritmu oproti sofistikovanějším metodám (bilineární, bikubická apod.).

**Příznaky**

Konkrétní implementace příznaků jsou odvozeny od třídy AbstractFeature. Tato třída obsahuje jednu čistě virtuální funkci.

```
virtual double evaluate(int index, int offsetX, int offsetY, AbstractImage* image) = 0;
```

Tato funkce vrací odezvu pod-okna (daného parametry offsetX a offsetY), které jsou akumulovány a porovnávány s prahy aktuálního slabého klasifikátoru (odpovídá parametru index). V rámci diplomové práce byly implementovány dvě třídy, které jsou potomky této bázové třídy. První třídou je třída LRDFeature, která umožňuje výpočet LRD příznaků. Vstupem je instance třídy Image – tedy obraz v odstínech šedi. Schéma algoritmu pro samotné vyhodnocení LRD příznaku pak může vypadat následovně :




*Obrázek 10: Blokové schéma vyhodnocení LRD příznaku (převzato z [6])*

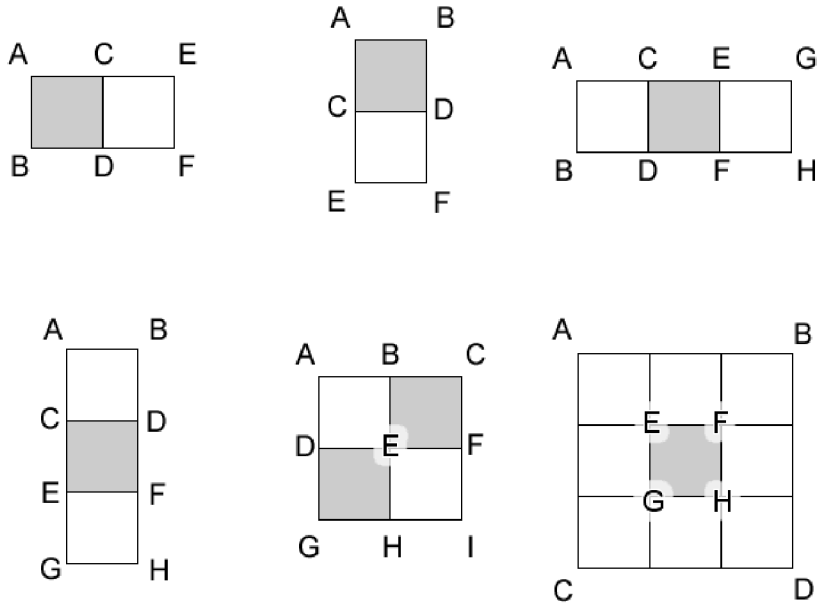
Maska je v našem případě matice 3x3 bloků, kde každý blok je sumou intenzit oblasti (typicky o velikosti 1x1, 1x2, 2x1, 2x2).

Druhou třídou je třída implementující výpočet Haarových příznaků - HaarFeature. V první fázi výpočtu je třeba určit standardní odchylku aktuálního pod-okna – potlačí se tím vlivy nehomogenního osvětlení na výsledek detekce.

$$s = \sqrt{\frac{\sum_{i=1}^{n} I(i)^2}{n} - \left(\frac{\sum_{i=1}^{n} I(i)}{n}\right)^2} \qquad (16)$$

V předchozím výrazu považujeme pixely vzorku za jednorozměrné pole *I* o délce n. Výpočet obou sum je velmi rychlý, díky před-počítanému integrálnímu a kvadratickému integrálnímu obrazu.

Dále se pomocí integrálního obrazu vypočítá odezva požadovaného typu Haarova příznaku. V našem případě je použito celkem 6 typů těchto příznaků:




*Obrázek 11 Typy Haarových příznaků použité ve filtru pro detekci obličejů – horizontal double, vertical double, horizontal ternal, vertical ternal, diagonal a surround.*

Díky integrálnímu obrazu je výpočet těchto příznaků velmi rychlý:

| Příznak | Výpočet | Počet referencí |
|---|---|---|
| Horizontal double | -A + 2C – E + B – 2D + F | 6 |
| Vertical double | -A + B – 2C – 2D – E + F | 6 |
| Horizontal ternal | +A – 3C + 3E – G – B + 3D – 3F + H | 8 |
| Vertical ternal | +A – B – 3C + 3D + 3E – 3F – G + H | 8 |
| Diagonal | +A- 2B + C – 2D + 4E – 2F + G -2H + I | 9 |
| Surround | +A – B – C + D – 9(+E – F – G + H) | 8 |

*Tabulka 9: Počet referencí do pole s obrazovými daty u jednotlivých typů příznaků.*

Například u diagonálního Haarova příznaku nás výpočet stojí celkem 9 referencí do pole s obrazovými daty. Kdybychom nepoužívali integrální obraz, ale obyčejný, tak by bylo nutné pro výpočet stejného příznaku 48 referencí do pole obrazových dat (uvažujeme velikost bloku 4x3).

**Data klasifikátoru**

Data natrénovaného klasifikátoru jsem obdržel ve formátu xml. Jeden slabý klasifikátor v kaskádě klasifikátorů natrénované algoritmem WaldBoost (LRD příznaky) může vypadat například následovně:

```
<stage posT="1e+50" negT="-0.136475">
   <HistogramWeakHypothesis predictionValues="-1.46935 -1.75678 -1.70577 -1.62235
-1.51736 -1.26519 -1.03946 -0.761181 -1.8172 -0.463974 -0.241405 -0.120997
0.175624 0.354821 0.601864 0.808101 1.16909">
     <LRDFeature positionX="7" positionY="4" blockA="2" blockB="3" blockWidth="2"
blockHeight="2" blocksX="3" blocksY="3"/>
  </HistogramWeakHypothesis>
</stage>
```

*Tabulka 10: Ukázka dat slabého klasifikátoru ve formátu xml.*

Každý slabý klasifikátor má dva prahy (posT, respektive negT), které slouží metodě SPRT pro rychlé přijetí, respektive zamítnutí pod-okna. Dále jsou zde uvedeny predikční hodnoty a údaje, které určují vlastnosti příznaku (pozice příznaku v rámci pod-okna, rozměry bloku, počet bloků apod.)

Aplikace tedy mohla přímo načítat data klasifikátoru z tohoto souboru. Vzhledem k relativní náročnosti parsování souboru ve formátu xml (zvláště na platformě WM) jsem se nakonec rozhodl, že data klasifikátoru transformuji ze souboru xml do hlavičkového C++ souboru, který bude součástí filtru. K tomuto účelu vznikla aplikace s grafickým uživatelským rozhraním, která tuto konverzi maximálně zjednodušuje (viz příloha A).

**Klasifikátor**

Samotný klasifikátor je implementován ve třídě Classifier. Konstruktor této třídy očekává dva parametry – obrazovou pyramidu a vstupní obraz z kamery, do kterého na konci vykreslí výsledné odezvy klasifikátoru.

Alternativním konstruktor pak místo obrazové pyramidy přijímá ukazatel na instanci třídy integrálního obrazu. Tento konstruktor je dostupný jen při použití direktivy preprocesoru HAAREXT. Detekce pak neprobíhá nad jednotlivými úrovněmi obrazové pyramidy, ale k vyhodnocování dochází přímo z jednoho integrálního obrazu, přičemž se zvětšuje pouze detekční pod-okno.

```
class Classifier
{
public:
      Classifier(AbstractImagePyramid* pyramid, CameraImage* cameraImage);
      Classifier(IntegralImage* integralImage, CameraImage* cameraImage);
      ~Classifier(void);
      int evaluate();
private:
      AbstractFeature* feature;
      AbstractImagePyramid* pyramid;
      IntegralImage* integralImage;
      CameraImage* cameraImage;

      int evaluateHaarExt(IntegralImage* image, float scaleFactor,
         int startLevel, int levelCount, int skipPixels,
         Detection** detections, int* count);
      void drawDetection(Detection* detection);
      void drawDetections(Detection** detections, int count);
      void initFeature();
};
```

*Tabulka 11: Třída Classifier.*

Třída obsahuje pouze jednu veřejnou metodu. Metoda evaluate provede samotnou detekci ve všech úrovní obrazové pyramidy (alternativně přímo z jednoho integrálního obrazu). Jaké příznaky se budou používat je určeno již při překladu filtru nastavením direktivy preprocesoru HAAR (popřípadě HAAREXT), respektive LRD. Každé pod-okno je vyhodnoceno kaskádou klasifikátorů. Pokud projde celou kaskádou (vyhodnocení není předčasně ukončeno), pak je ještě akumulovaná hodnota

porovnána s uživatelem nastavenou hodnotou prahu. Pokud je výsledná hodnota větší, než je hodnota prahu, je obsah pod-okna klasifikován jako obličej a přidá se do vektoru pozitivních detekcí.

Po dokončení detekce je k dispozici vektor pozitivních odezev, jejichž šablona je dána třídou Detection

```
class Detection
{
public:
	Detection(int x, int y, int height, int width, double response);
	~Detection(void);

	int getX() { return x; }
	int getY() { return y; }
	int getWidth() { return width; }
	int getHeight() { return height; }
	double getResponse() { return response; }
private:
	int x, y, width, height;
	double response;
};
```

*Tabulka 12 Třída Detection.*

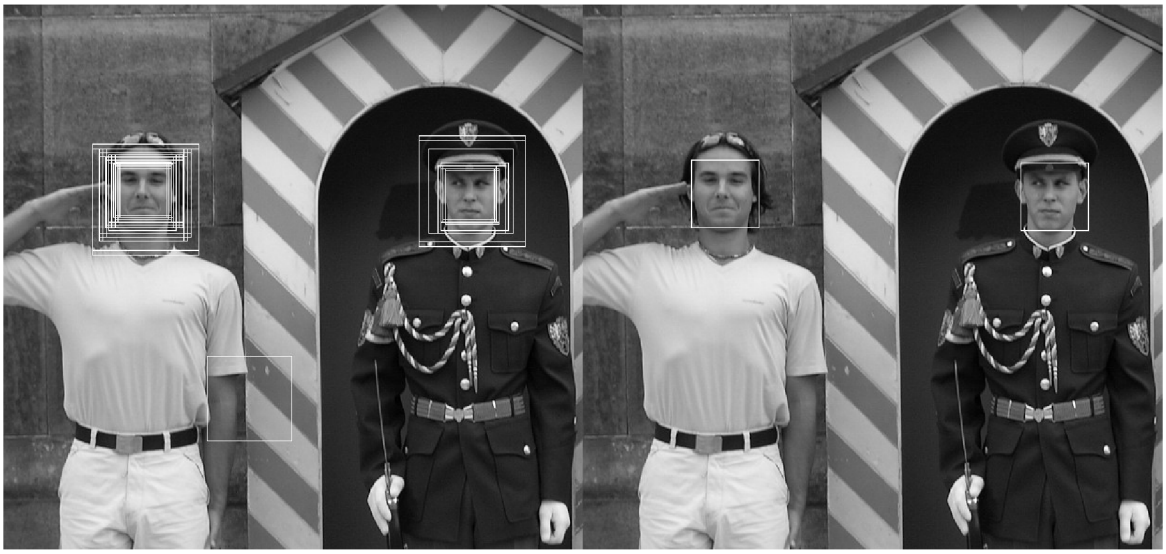
Tato třída obsahuje všechny potřebné údaje k následnému potlačení ne-maxim, popřípadě vykreslení (pozice, výška, šířka a odezva klasifikátoru).

Vektor pozitivních odezev lze buď přímo vykreslit nebo nejprve potlačit ne-maximální hodnoty (viz dále). Pro samotné vykreslení slouží privátní metoda drawDetection, která vykreslí přímo do vstupního obrázku pozitivní detekce. Tento obrázek je poté vykreslen na displej mobilního zařízení pomocí odpovídajícího DirectShow filtru.

**Potlačení ne-maxim**

Výsledkem detekce je n-rozměrný vektor, který obsahuje všechna pod-okna, jenž klasifikátor pozitivně vyhodnotil. V oblasti obrazu, kde se skutečně nachází obličej (tedy pozitivní detekce) je počet pozitivních odezev často velmi velký. Pro výstup na obrazovku mobilního zařízení je však výhodnější zobrazit kolem detekovaného obličeje jen jednu odezvu (ideálně tu nejsilnější). Pro tyto účely je v aplikaci použito potlačení ne-maxim (non-maxima supression), které nejprve shlukuje jednotlivé odezvy do skupin na základě jejich vzájemné vzdálenosti v obraze a z nich poté vybere jen nejsilnější odezvu, která se nakonec vykreslí.

Na následujícím obrázku (12) lze vidět, že kolem obličejů vzniklo spousta pozitivních odezev. Navíc je zde vidět také jedná falešná detekce. Podobné ojedinělé falešné detekce lze eliminovat tím, že určíme, jaký je minimální počet odezev ve shluku. V tomto konkrétním případě byl minimální počet nastaven na dvě pozitivní odezvy.



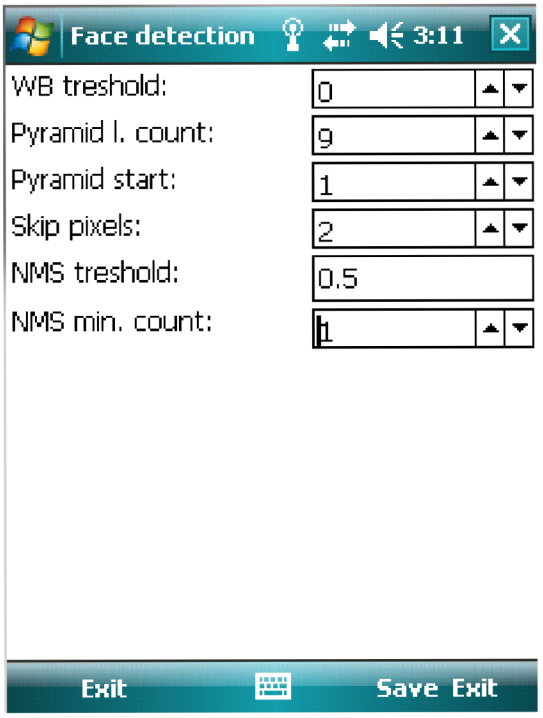
*Obrázek 12: Potlačení ne-maxim.*

**Nastavení aplikace**

Pro jednoduché a uživatelsky příjemné nastavení filtru vznikla v rámci diplomové práce aplikace, která umožňuje pomocí uživatelského rozhraní nastavit parametry detekce. Pomocí ní lze nastavit 6 parametrů – práh pro přijmutí pod-okna vyhodnoceného metodou WaldBoost, počet úrovní obrazové pyramidy, počáteční úroveň pyramidy, dále kolik pixelů se má přeskakovat při přikládání detekčního pod-okna (minimální hodnota je 1 – pod-okno se bude přikládat každý obrazový bod). Poslední dva parametry se týkají metody potlačení ne-maxim. První určuje práh minimální vzdálenosti dvou detekovaných pod-oken a na základě této hodnoty se přidají do stejného shluku. Druhý udává minimální počet pod-oken, které musí být ve shluku, aby se pod-okno s největší pozitivní odezvou vykreslilo.

Aplikaci by v budoucnosti bylo vhodné rozšířit i o další nastavení – nabízí se například volba rozlišení kamery. Vzhledem k tomu, že se tato práce snažila spíše zabývat experimenty s různými druhy příznaků a podrobněji zmapovat problematiku vývoje aplikací pro Windows Mobile, nebylo této aplikaci věnováno příliš času, a proto nakonec umožňuje nastavovat jen parametry detekce.

Data se ukládají do souboru *faceDetection.ini*, z kterého si filtr pro detekci obličejů vyzvedne zadané hodnoty. Pokud tento soubor není nalezen, použijí se výchozí hodnoty. Aplikace byla napsána v .NET Compact Frameworku 3.5.




*Obrázek 13: Jednoduchá aplikace umožňující nastavit parametry detekce DirectShow filtru pro detekci obličejů.*

# 4 Dosažené výsledky

Tato kapitola popisuje výsledky, kterých se v rámci diplomové práce dosáhlo. Hlavním cílem bylo prozkoumat výkon dnešních nejrychlejších mobilních zařízení za použití aplikace implementující algoritmy náročné na výkon. Budou zde proto uvedeny naměřené výsledky, které byly zaměřeny zejména na celkovou rychlost, rychlost jednotlivých částí výpočtu (tvorba obrazové pyramidy, samotná klasifikace atd.) a porovnání LRD a Haarových příznaků. Ještě před tím je ale třeba zmínit všechny problémy, na které se během vývoje aplikace narazilo. Bohužel jich nebylo málo.

## 4.1 Problematické části vývoje

**Rozhraní DirectShow**

Aplikace byla od začátku testována na zařízení HTC HD. Už na tomto zařízení jsem se setkal s problémy ve frameworku DirectShow. Ačkoliv zařízení disponuje kamerou o rozlišení 5 megapixelů, nebylo možné nastavit rozlišení kamery na preview pinu filtru pro zachytávání obrazu z kamery. Rozlišení tedy bylo neměnné a to 144x176 obrazových bodů, což bylo vzhledem k parametrům zařízení velmi málo.

V průběhu vývoje bylo stávající testovací zařízení vyměněno za novější model – HTC Touch HD2 Leo. Problémy však nezmizely, spíše naopak. Ačkoliv už bylo možné nastavit rozlišení kamery na preview pinu (celkem 13 rozlišení od 80x48 až po 800x480, což odpovídá nativnímu rozlišení displeje telefonu), tak obraz z tohoto pinu byl otočený o 90 stupňů. Marně jsem se snažil tento problém vyřešit – zkoušel jsem různá fóra, kde jsem se akorát dozvěděl, že nejsem sám, kdo tento problém má. Dále jsem kontaktoval technickou podporu firmy HTC, která mě nakonec odkázala zpět na neoficiální fórum. Nakonec mi nezbylo nic jiného než vyřešit problém hrubou silou a správnou orientaci obrazu si zajistit sám. Tuto funkčnost jsem nakonec pro zjednodušení zahrnul přímo do filtru pro detekci obličeje, i když by v budoucnosti měla tato funkce přijít do samostatného vlastního filtru. Jistou potíží je, že tato operace není úplně triviální a bohužel výslednou aplikaci zpomaluje.

**Zastaralé jádro operačního systému**

Nejmodernější procesory postavené na architektuře ARM disponují nejen vysokým taktem (1 GHz), ale obsahují třeba i Floating Point Unit. Jak bylo ale zmíněno v první kapitole, jádro systému Windows Mobile je zastaralé a nové vlastnosti procesorů nepodporuje. Veškeré instrukce v plovoucí řádové čárky jsou tak emulovány a jsou proto velmi náročné. Konkrétně v případě Haarových příznaků je absence přímého využití těchto instrukcí hodně znát (výpočet standardní odchylky atd.).

Vliv nepřítomnosti podpory FPU lze demonstrovat díky programu FPUEnabler. Jedná se o experimentální nástroj, který záplatuje přímo jádro operačního systému v paměti a zpřístupní tak některé instrukce FPU přímo, bez složitého emulování. Tento program je primárně vyvíjen pro Samsung Omnia II a vývojáři tvrdí, že na HTC Touch HD2 nedosahuje zdaleka tak dobrých výsledků. Po záplatování jádra operačního systému provede program rovnou jednoduchý benchmark, který otestuje zrychlení přímého využití jednotlivých instrukcí oproti emulaci:

| Instrukce | Zrychlení HD2 | Zrychlení Omnia II | Instrukce | Zrychlení HD2 | Zrychlení Omnia II | Instrukce | Zrychlení | Zrychlení Omnia II |
|---|---|---|---|---|---|---|---|---|
| negs | 39% | 71% | led | 50% | 72% | itos | 73% | **130%** |
| eqs | 23% | 70% | ltd | 29% | 74% | itod | 56% | **100%** |
| ges | 43% | 70% | adds | 63% | **139%** | utos | 87% | **127%** |
| gt | 43% | 69% | subs | 95% | **156%** | utod | 55% | 98% |
| les | 28% | 71% | muls | 93% | **142%** | stoi | 50% | **109%** |
| lts | 48% | 69% | divs | **140%** | **178%** | stou | 65% | **111%** |
| negd | 24% | 75% | addd | **160%** | **299%** | stod | 51% | **108%** |
| eqd | 49% | 71% | subd | **168%** | **315%** | dtoi | 84% | **130%** |
| ged | 48% | 74% | muld | **152%** | **292%** | dtou | 65% | **106%** |
| gtd | 48% | 72% | divd | **402%** | **511%** | dtos | 62% | **117%** |

*Tabulka 13: Zrychlení jednotlivých instrukcí po záplatě programem FPUEnabler pro zařízení HTC HD2 a Samsung Omnia II.*

Z tabulky lze vidět, že záplatovat se u zařízení HTC Touch HD2 vyplatí pouze 5 instrukcí. U zařízení Samsung Omnia II je výsledek dle očekávání daleko lepší – záplatovat se vyplatí hned 17 instrukcí. U obou zařízení lze vidět největší zrychlení pro instrukce divd – u HTC Touch HD2 402% a u Samsung Omnia II dokonce 511%. Jaký to bude mít vliv na výkon ukážeme v další kapitole v jednom z experimentů.

## 4.2 Výsledky testů

Oba klasifikátory, které jsou v následujících experimentech použity, byly trénovány algoritmem WaldBoost. Oba s cílovou chybou 20%. Liší se jen v použitých příznacích – první z nich je trénován pomocí Haarových příznaků, druhý používá příznaky LRD.

K dispozici byly celkem 4 testovací přístroje, které se lišily zejména svou výkonností. Bohužel už v zárodku se ukázalo, že výkonnostně nejslabší telefon (HTC TyTN) je velmi pomalý i při

obyčejném zachytávání videa z kamery a následném zobrazení na displej (rychlost videa byla čistě subjektivním odhadem 0,5 fps). Proto bylo z dalších testů toto zařízení vynecháno.

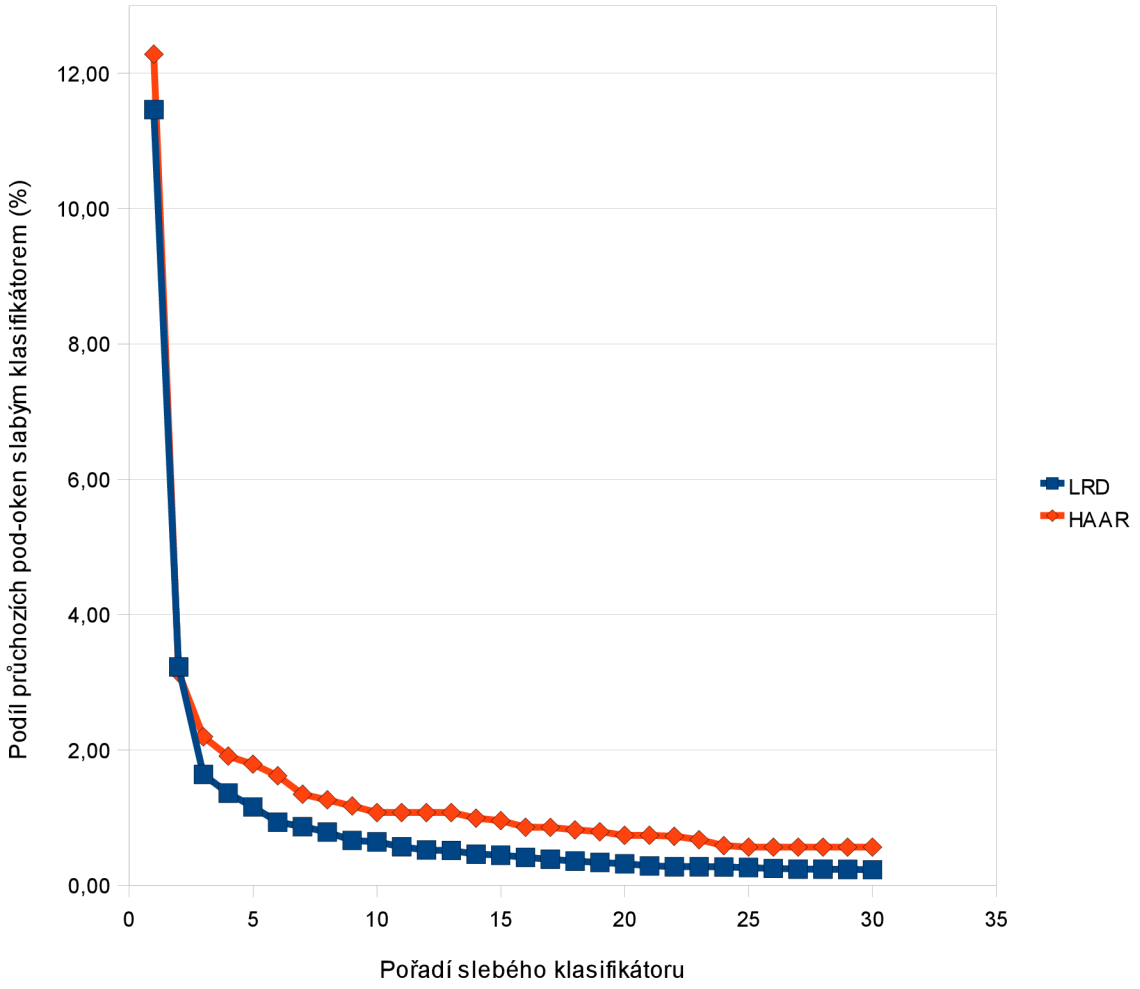
|  |  |  |  |
|---|---|---|---|
| **HTC Touch HD2 (Leo)** | **Samsung Omnia II** | **HTC Touch HD (Blackstone)** | **HTC TyTN (Hermes)** |
| Qualcomm QSD8250 1 GHz | Samsung S3C6410 800 MHz | Qualcomm MSM7201 528 MHz | Samsung S3C2442 400 MHz |
| 800x480 (WVGA) | 800x480 (WVGA) | 800x480 (WVGA) | 320x240 (QVGA) |
| 5 Mpx | 5 Mpx | 5 Mpx | 2 Mpx |

*Tablka 14: Přehled testovacích zařízení – ilustrační fotografie, název, procesor, rozlišení displeje, rozlišení fotoaparátu.*

Pro účely testování byly použity celkem 3 obrázky. Obrázek **test01** byl použit z důvodu velkého počtu obličejů (celkem 29). Jeho rozlišení je 800x480 bodů, což odpovídá nativnímu rozlišení displeje primárního testovacího zařízení (HTC Touch HD2 Leo). Druhý obrázek, **test02**, byl použit, protože obsahuje pouze dva obličeje a jeho velká část je monochromatické pozadí – mělo by být vyhodnoceno podstatně méně slabých klasifikátorů. Posledním testovacím obrázkem je obrázek **test03,** jehož rozlišení je 144x176 bodů. Všechny tyto obrázky jsou součástí přílohy C.

**Procento pod-oken v jednotlivých úrovních kaskády klasifikátorů**

Jak bylo uvedeno v předcházejících kapitolách, výhodou použití kaskády slabých klasifikátorů je rychlé zamítnutí pod-oken, v nichž se obličej zcela jistě nenachází (např. pozadí scény). V rámci tohoto testu jsem změřil, kolik procent z celkového počtu klasifikovaných pod-oken se dostane do jednotlivých úrovní kaskády.




*Obrázek 14: Procento pod-oken, která projdou jednotlivými kaskádami klasifikátoru (testovací obrázek test01).*

Klasifikátory, které aplikace používá, obsahují poměrně velký počet slabých klasifikátorů (1000 slabých klasifikátorů v klasifikátoru používající LRD příznaky, 300 v případě klasifikátoru používající Haarovy příznaky), přičemž drtivé procento z nich neprojde do vyšších úrovní. Proto je uvedený graf omezen jen na prvních 30 slabých klasifikátorů z kaskády. Z grafu lze také vidět, že procenta průchozích pod-oken se v jednotlivých úrovních kaskády liší maximálně o půl procenta mezi klasifikátory natrénovanými Haarovými a LRD příznaky.

**Vliv emulování instrukcí v plovoucí řádové čárce**

V předešlé části kapitoly bylo uvedeno, že ačkoliv jsou moderní procesory vybavené FPU, zastaralé jádro operačního systému ji neumí využívat. Vliv této skutečnosti by se měl projevit zejména u klasifikace pomocí klasifikátoru trénované s Haarovými příznaky. Test byl prováděn tak, že se

pomocí programu FPUEnabler záplatovalo jádro operačního systému, které tak začalo přímo využívat některé FPU instrukce procesoru a výsledky tohoto měření se porovnaly s výsledky bez použití FPUEnableru. Test bylo možné provést pouze na mobilech HTC Touch HD2 a Samsung Omnia II, které byly jediné z testovacích zařízení podporovány.

| Příznaky | Obrázek | Počet vyhodnocených slabých klasifikátorů | Rychlost (s) | Rychlost (FPUEnabler) (s) | Zrychlení |
|---|---|---|---|---|---|
| HAAR | test01 | 6347195 | 69,55 | 42,76 | **38,51%** |
| HAAR | test02 | 2948210 | 24,99 | 16,02 | **35,91%** |
| HAAR | test03 | 360317 | 3,14 | 1,99 | **36,60%** |
| LRD | test01 | 2690853 | 5,93 | 5,37 | **9,36%** |
| LRD | test02 | 1673825 | 3,68 | 3,43 | **6,69%** |
| LRD | test03 | 143296 | 0,30 | 0,28 | **7,90%** |

*Tabulka 15: Vliv emulování instrukcí v plovoucí řádové čárce – HTC HD2 Leo.*

U zařízení HD2 je v případě klasifikace obrazu klasifikátorem používající Haarovy příznaky zrychlení v průměru 37,00%, v případě LRD je to pak 7,98%. V předešlé kapitole bylo navíc uvedeno, že program FPUEnabler nefunguje pod mobilním zařízením HTC HD2 moc dobře (i tak je zrychlení víc než zarážející).

| Příznaky | Obrázek | Počet vyhodnocených slabých klasifikátorů | Rychlost (s) | Rychlost (FPUEnabler) (s) | Zrychlení |
|---|---|---|---|---|---|
| HAAR | test01 | 6347195 | 141,77 | 56,84 | **59,91%** |
| HAAR | test02 | 2948210 | 63,95 | 26,30 | **58,88%** |
| HAAR | test03 | 360317 | 7,66 | 3,06 | **60,07%** |
| LRD | test01 | 2690853 | 13,30 | 11,82 | **11,14%** |
| LRD | test02 | 1673825 | 7,99 | 7,63 | **4,49%** |
| LRD | test03 | 143296 | 0,67 | 0,60 | **10,95%** |

*Tabulka 16: Vliv emulování instrukcí v plovoucí řádové čárce – Samsung Omnia II.*

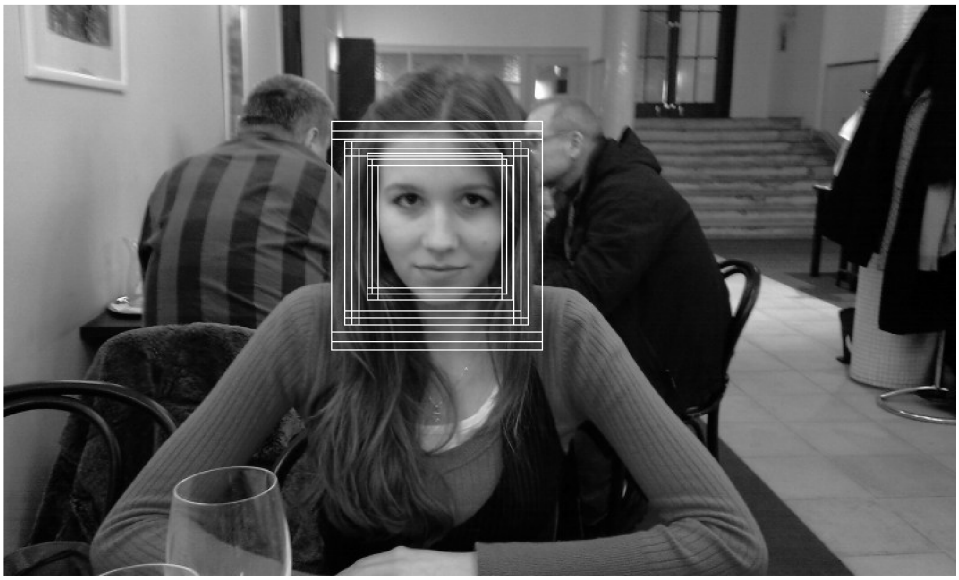
Jelikož je program FPUEnabler vyvíjen pro Samsug Omnia II, dalo se čekat, že zde bude zrychlení ještě znatelnější. Detekce za pomoci Haarových příznaků byla urychlena v průměru o 59,62%, v případě LRD pouze o 8,86%.

Zrychlení je tedy opravdu znatelné a opět poukazuje na zastaralost jádra operačního systému Windows Mobile. Z tohoto i celkových výsledků vyplývá, že je daleko výhodnější použít LRD

příznaky, jejichž vyhodnocení z větší části probíhá jen za pomoci celých čísel, na rozdíl od Haarových příznaků, které počítají spíše s čísly v plovoucí řádové čárce.

**Rychlost na úkor kvality detekce**

Odezva klasifikátoru v místě, kde se nachází obličej je často velmi velká. Nabízí se tak možnost, zda neomezit kvalitu detekce, díky čemuž můžeme dosáhnout vyšší rychlosti detekce. Na obraz se totiž typicky přikládá detekční okno bod po bodu. Výsledek pak v reálné aplikaci vypadá následovně:



*Obrázek 15 Velký počet pozitivních odezev v místě, kde se nachází obličej.*

Kolem obličeje je několik pozitivních odezev. Nabízí se proto experiment, zda není dostatečné přikládat detekční okno každý druhý nebo dokonce čtvrtý bod. Detekce se tak urychlí čtyřnásobně, respektive šestnáctinásobně.

Další možností, jak celou detekci významně urychlit, je vynechání první úrovně pyramidy (tedy obrázku v rozlišení 1:1 s obrazem původním). Za předpokladu, že je detekční okno malé (typicky 24x24), nemůže být rozlišení mobilní kamery, která je často navíc rozostřená, dostatečné pro detekci obličejů ve větší vzdálenosti od kamery. Odstraníme tak část pyramidy, která zabere nejvíc výpočetního času.

| Příznaky | Obrázek | Bez optimalizace* | Optimalizace 1** | Optimalizace 2*** | Zrychlení 1 | Zrychlení 2 |
|---|---|---|---|---|---|---|
| HAAR | test01 | 6347195<br>69,54s | 918493<br>**8,09s** | 222122<br>**2,05s** | **88,36%** | **97,05%** |
| HAAR | test02 | 2948210<br>24,99s | 429960<br>**3,76s** | 103469<br>**1,04s** | **84,95%** | **95,85%** |
| HAAR | test03 | 360317<br>3,14s | 58377<br>**0,51s** | 13874<br>**0,13s** | **83,86%** | **95,98%** |
| LRD | test01 | 2690853<br>5,93s | 431361<br>**0,99s** | 106409<br>**0,30s** | **83,28%** | **94,91%** |
| LRD | test02 | 1673825<br>3,68s | 256772<br>**0,63s** | 64196<br>**0,22s** | **82,86%** | **94,1%** |
| LRD | test03 | 143296<br>0,30s | 21059<br>**0,05s** | 3153<br>**0,02s** | **84,01%** | **93,38%** |

* První řádek – počet vyhodnocených slabých klasifikátorů, druhý řádek – doba detekce

** Optimalizace1 – Přikládání detekčního okna jen každé dva pixely, přeskočení první úrovně pyramidy

*** Optimalizace 2 – Přikládání detekčního okna jen každé čtyři pixely, přeskočení první úrovně pyramidy

*Tabulka 17: Zrychlení detekce na úkor kvality – HTC Touch HD2 Leo.*

Zrychlení detekce je v případě vynechání první úrovně obrazové pyramidy vždy více než 82%. V případě druhé optimalizace je zrychlení vždy více než 93%. Nárůst výkonu je tedy velmi velký a určitě se vyplatí v aplikaci použít i za cenu horších detekčních schopností. Výsledky testů ostatních testovacích zařízení jsou uvedeny v příloze B.

**Jednotlivé části aplikace**

Cílem tohoto testu bylo určit, které části aplikace jsou výpočetně nejnáročnější. Po pečlivém zvážení jsem se rozhodl separovat celou detekci na tři části, které považuji za nejnáročnější. Jedná se o tvorbu pyramidy obrazů, samotnou detekci a potlačení ne-maxim.

| | HAAR | LRD |
|---|---|---|
| Tvorba obrazové pyramidy | 0,59% | 0,31% |
| Klasifikace | 96,83% | 96,76% |
| Potlačení ne-maxim | 1,71% | 1,19% |
| Ostatní | 0,87% | 0,31% |

*Tabulka 18: Porovnání náročností jednotlivých fází detekce obličeje.*

Z výše uvedené tabulky lze vidět, že jednoznačně nejnáročnější fází výpočtu je samotná klasifikace – v obou případech se blíží 97%. Aby se celý proces detekce zrychlil, je třeba se zaměřit zejména právě na tuto část. Nepotvrdily se tak domněnky, že tvorba obrazové pyramidy by zabírala neúměrnou část času a i potlačení ne-maxim je poměrně rychlé v porovnání se samotnou klasifikací.

**Porovnání různých implementací klasifikátoru využívající Haarovy příznaky**

Jak bylo uvedeno v předcházející kapitole, v rámci aplikace byly implementovány dva přístupy pro detekci obličeje pomocí klasifikátoru využívající Haarovy příznaky. První metoda vytvořila obrazovou pyramidu integrálních obrazů a poté postupně tyto obrázky procházela skenovacím pod-oknem. Druhá metoda místo pod-vzorkovávání vstupního obrazu postupně měnila velikost skenovacího okna. Druhá metoda by tak měla být výkonnější vzhledem k odpadnutí režie související s tvorbou obrazové pyramidy.

Po výsledcích předchozího testu, kdy se ukázalo, že tvorba obrazové pyramidy není zdaleka tou nejnáročnější částí detekce, vznikla obava, že rozdíl mezi těmito dvěma přístupy bude minimální.

| Příznaky | Obrázek | Bez optimalizace | Optimalizace 1 | Optimalizace 2 |
|---|---|---|---|---|
| HAAR | test01 | 6347195<br>69,54s | 918493<br>8,09s | 222122<br>2,05s |
| HAAR | test02 | 2948210<br>24,99s | 429960<br>3,76s | 103469<br>1,04s |
| HAAR | test03 | 360317<br>3,14s | 58377<br>0,51s | 13874<br>0,13s |
| HAAR bez obrazové pyramidy | test01 | 6855271<br>66,83s | 680547<br>6,78s | 166589<br>1,71s |
| HAAR bez obrazové pyramidy | test02 | 3021831<br>27,61s | 298863<br>2,97s | 72326<br>0,76s |
| HAAR bez obrazové pyramidy | test03 | 419482<br>3.89s | 50343<br>0,49s | 8154<br>0,11s |

*Tabulka 19: Srovnání dvou různých implementací klasifikátoru využívající Haarovy příznaky – HTC Touch HD2 Leo.*

Z tabulky 19 lze vidět, že rychlost vyhodnocení jednoho obrázku záleží hlavně na počtu vyhodnocených slabých klasifikátorů (jejich počet se lišil v závislosti na použité optimalizaci). Je tedy zřejmé, že tvorba obrazové pyramidy v našem konkrétním případě celou detekci nijak výrazně nezpomaluje a je třeba se zaměřit hlavně na zrychlení výpočtu příznaků.

# 5 Závěr

Cílem práce bylo seznámit se podrobně s platformou Windows Mobile, zejména se zaměřit na možnosti vývoje, a prozkoumat výkon dnešních mobilních zařízení. Pro tyto účely byla implementována aplikace, která detekuje obličeje v obraze z kamery mobilního zařízení.

Text této práce postupně mapuje vývoj této aplikace. První kapitola je věnována analýze platformy Windows Mobile. Jsou zde zmíněny důvody pro výběr této platformy, její klady a nedostatky a možnosti vývoje. Krátký odstavec je také věnován již jisté budoucnosti této platformy, kterou představuje operační systém Windows Phone 7. Tento nový produkt by měl přinést potřebnou revoluci tomuto poměrně zastaralému a málo se vyvíjejícímu systému.

Druhá kapitola se věnuje problematice rychlé detekce obličejů v obraze. Hlavním cílem této části práce bylo podrobněji se seznámit s teorií, která poté bude využita při samotné implementaci. Zvláštní pozornost je věnována učícím algoritmům AdaBoost a WaldBoost. Cílem práce bylo porovnat i chování alespoň dvou druhů příznaků. Výběr nakonec padl na Haarovy příznaky a LRD příznaky, které jsou v této kapitole také rozebrány.

Třetí kapitola se podrobně zaměřuje na implementaci celé aplikace. První část se věnuje problematice vývoje multimediálních aplikací pomocí multimediálního frameworku DirectShow. Čtenář je seznámen s tvorbou filter graphu pro zachytávání videa z kamery zařízení a jeho následnou transformaci. Ta je realizována pomocí vlastního transformačního filtru. Tento krok je nejprve popsán teoreticky a posléze je popsána realizace transformačního filtru, který ve vstupním obraze detekuje obličeje.

Závěrečná kapitola obsahuje popis dosažených výsledků. Jsou zde zmíněny všechny problémy, se kterými jsem se během vývoje aplikace setkal a které dokazují zastaralost platformy Windows Mobile. Aby bylo možno vyhodnotit výkon mobilních zařízení, je zde také uvedena celá řada testů, které poukazují na silné a slabé stránky nejen mobilních zařízení, ale i cílové platformy.

Aplikace se povedla zrealizovat a po určitých optimalizacích, které většinou spočívaly v jistých ústupcích kvalitě detekce, bylo dosaženo hlavního cíle - detekce obličejů v obraze v reálném čase (tedy alespoň 15 fps) na mobilním zařízení. Práce tímto dokázala, že výkon dnešních mobilních zařízení je již dostatečný i pro velmi náročné aplikace.

Bohužel se také potvrdilo, že platforma Windows Mobile, jejíž jádro pochází z roku 2004, nedokáže potenciál dnešních moderních procesorů využít naplno. Zejména experimenty s výpočty v plovoucí desetinné čárce byly velkým překvapením. Ačkoliv moderní procesory založené na architektuře ARM již obsahují Floating Point Unit, veškeré instrukce jsou stále emulovány na úrovni jádra operačního systému. Díky tomuto zjištění se také ukázalo, proč je výhodnější využití příznaků LRD, než Haarových příznaků.

Co se týče pokračování práce, nabízí se hned několik možností. V první řadě by bylo třeba provést další optimalizace, které by samotnou detekci urychlily. Je třeba se zaměřit hlavně na místa, která se ukázala jako nejproblematičtější, tj. všechny operace s čísly v plovoucí řádové čárce.

Zajímavé by také bylo porovnat výkon s dalšími platformami. Prioritou je hlavně srovnání s platformou iPhone a stále oblíbenějším Google Android. Asi nejdůležitější zjištění by pak přineslo srovnání zařízení HTC Desire (Google Android) s hlavním testovacím přístrojem HTC Touch HD2 Leo. Tyto mobilní zařízení totiž obsahují stejný procesor (Qualcomm Snapdragon), takže by bylo možno srovnat platformy přímo. Začátkem podzimu roku 2010 by také měl vyjít nástupce Windows Mobile 6.5.x – Windows Phone 7. Zde by se nabízelo porovnání, jak si dokázala firma Microsoft poradit s problémy stávající platformy a také jak se změnil vývoj.

Přínosné by také mohlo být srovnat výkon mobilního zařízení se stolním počítačem. To by nemělo být náročné vzhledem k poměrně jednoduchému převedení již existujícího DirectShow filtru.

# Literatura


[1] Český statistický úřad: Mobilní telefonní síť, 2009
[Online; navštíveno 2009-12-11]
URL: http://www.czso.cz/csu/redakce.nsf/i/mobilni_telefonni_sit

[2] ZDNet UK: Windows Mobile loses nearly a third of market share, 2009
[Online; navštíveno 2009-12-25]
URL: http://news.zdnet.co.uk/communications/0,1000000085,39877964,00.htm

[3] Yang M., Kriegman D. J., Ahuja N.: Detecting Faces in Images: A Survey.
IEEE Trans, Pattern Analysis and Machine Intelligence 2002

[4] Viola P., Jones M.: Robust Real Time Object Detection,
Statistical and Computational Theoresis of Vision, Vancouver, 2001

[5] Šochman J., Matas J.: WaldBoost – Learning for Time Constrained Sequential Detection,
2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition
(CVPR'05) – Volume 2

[6] Hradiš M., Herout A., Zemčík P.: Local Rank Patterns – Novel Features for Rapid Object
Detection, Proceedings of ICCVG 2008, Heidelberg, 2008

[7] Zemčík P., Hradiš M., Herout A.: Local Rank Differences – Novel Features for Image
Processing, Proceedings of SCCG 2007, Budmeřice, 2007

[8] Polok L., Herout A., Zemčík P., Hradiš M., Juránek R., Jošth R.:
„Local Rank Differences“ Image Feature Implemented on GPU, 2008
Proceedings of ACIVS 2008

[9] Juránek R.: Pattern Recognition in Image Using Classifiers, Diplomová práce,
VUT Brno 2007

[10] MSDN: Introduction to DirectShow Application Programming, 2009
[Online; navštíveno 2009-09-21]
URL: http://msdn.microsoft.com/en-us/library/ms786509%28VS.85%29.aspx

[11] MSDN: Writing Transform Filter, 2009
[Online; navštíveno 2009-10-01]
URL: http://msdn.microsoft.com/en-us/library/ms788165%28VS.85%29.aspx

[12] MSDN: Filter Graph Manager, 2009
[Online; navštíveno 2009-09-21]
URL: http://msdn.microsoft.com/en-us/library/dd375786%28VS.85%29.aspx

[13] MSDN: IPropertyBag Interface, 2009
[Online; navštíveno 2009-10-03]
URL: http://msdn.microsoft.com/en-us/library/aa768196%28VS.85%29.aspx

[14] MSDN: ICaptureGraphBuilder2 Inteface, 2009
[Online; navštíveno 2009-10-02]
URL: http://msdn.microsoft.com/en-us/library/aa924828.aspx

[15] MSDN: ICaptureGraphBuilder2::RenderStream, 2009
[Online; navštíveno 2009-10-03]
URL: http://msdn.microsoft.com/en-us/library/aa930715.aspx

[16] MSDN: IVideoWindow Interface, 2009
[Online; navštíveno 2009-10-03]
URL: http://msdn.microsoft.com/en-us/library/dd377276%28VS.85%29.aspx

[17] MSDN: IMediaControl Interface, 2009
[Online; navštíveno 2009-10-04]
URL: http://msdn.microsoft.com/en-us/library/dd390170%28VS.85%29.aspx

[18] MSDN: IUnkown::QueryInterface Method, 2009
[Online; navštíveno 2009-10-04]
URL: http://msdn.microsoft.com/en-us/library/ms682521%28VS.85%29.aspx

[19] MSDN: Writing Transform Filters: Add Support for COM, 2009
[Online; navštíveno 2009-10-04]
URL: http://msdn.microsoft.com/en-us/library/ms787698%28VS.85%29.aspx

[20] MSDN: CTransInPlaceFilter Class, 2009
[Online; navštíveno 2010-01-01]
URL: http://msdn.microsoft.com/en-us/library/aa920705.aspx

[21] Jack K.: Video Demystified. 5th Edition.
[s.l.] : Newness, 2007. 944 s. ISBN 978-0750683951

# Seznam příloh

A Návod na ovládání aplikace pro převod dat klasifikátoru ze souboru xml do C++ hlavičkového souboru

B Ostatní testy

C Testovací obrázky

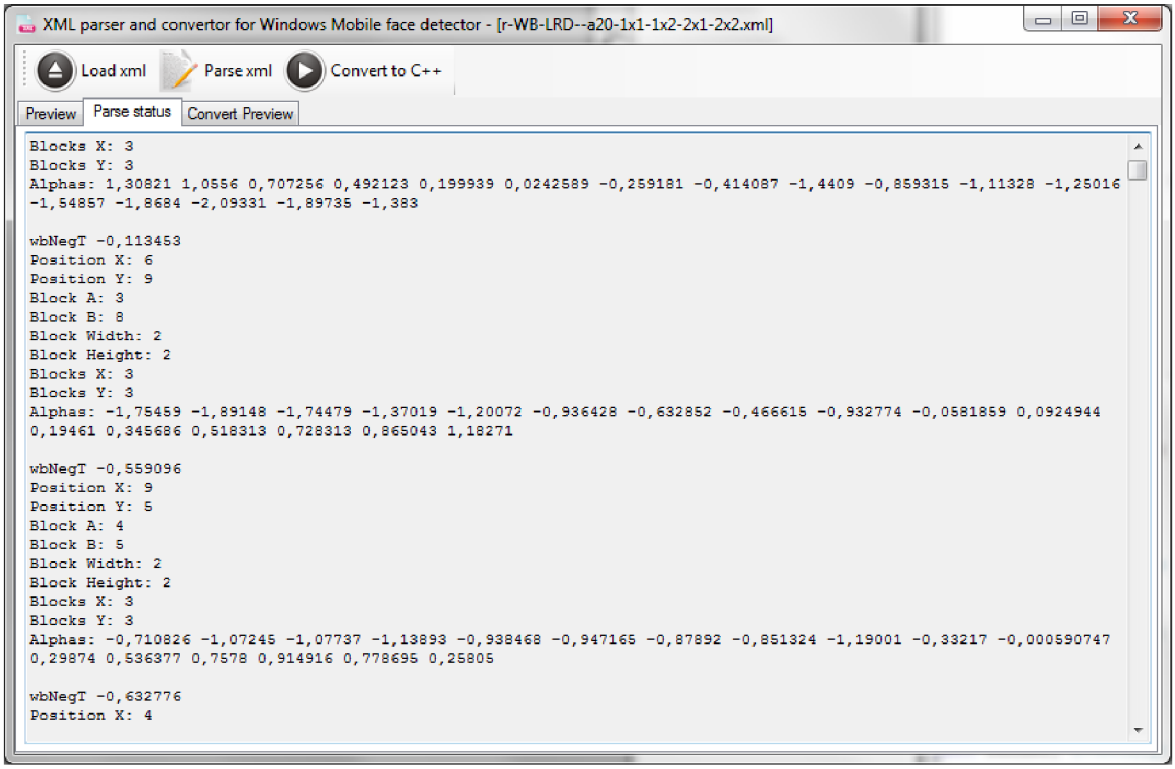
D Datový nosič (CD) s výslednou aplikací, zdrojovými texty práce a tímto textem v elektronické podobě

E Plakát pro prezentování projektu

# Příloha A

Program XmlParser slouží k jednoduchému převodu dat natrénovaného klasifikátoru z formátu xml do C++ hlavičkového souboru. Program zvládá převod klasifikátorů natrénovaných pomocí Haarových a LRD příznaků. Při překladu je použitý typ příznaků potřeba určit direktivou preprocesoru (HAAR, respektive LRD).

Aplikace je ovládána pomocí jednoduchého GUI. Tlačítko *Load xml* vyvolá standardní systémový dialog pro načtení souboru typu xml s daty klasifikátoru. Další tlačítko, *Parse xml*, převede zdrojový soubor do vnitřní reprezentace.




*Obrázek 16: Stav aplikace po parsování xml souboru.*

Posledním tlačítkem je tlačítko *Convert to C++*, které vygeneruje finální hlavičkový soubor. Ten je poté možné použít přímo v DirectShow filtru pro detekci obličeje.

# Příloha B

| Příznaky | Obrázek | Bez optimalizace | Optimalizace 1 | Optimalizace 2 | Zrychlení 1 | Zrychlení 2 |
|---|---|---|---|---|---|---|
| HAAR | test01 | 6347195<br>141,76s | 918493<br>**20,36s** | 222122<br>**5,13s** | **85,64%** | **96,38%** |
| HAAR | test02 | 2948210<br>63,95s | 429960<br>**9,67s** | 103469<br>**2,51s** | **84,88%** | **96,07%** |
| HAAR | test03 | 360317<br>7,66s | 58377<br>**1,28s** | 13874<br>**0,27s** | **83,35%** | **96,51%** |
| LRD | test01 | 2690853<br>13,30s | 431361<br>**2,13s** | 106409<br>**0,60s** | **84,00%** | **95,50%** |
| LRD | test02 | 1673825<br>7,99s | 256772<br>**1,27s** | 64196<br>**0,37s** | **84,07%** | **95,36%** |
| LRD | test03 | 143296<br>0,68s | 21059<br>**0,10s** | 3153<br>**0,02s** | **84,71%** | **97,10%** |

*Tabulka 20: Zrychlení detekce na úkor kvality – Samsung Omnia II*

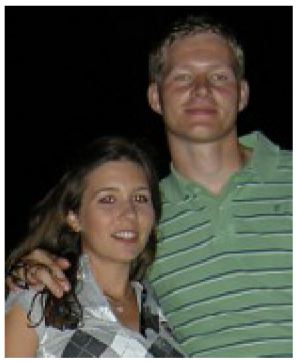
| Příznaky | Obrázek | Bez optimalizace | Optimalizace 1 | Optimalizace 2 | Zrychlení 1 | Zrychlení 2 |
|---|---|---|---|---|---|---|
| HAAR | test01 | 6347195<br>164,60s | 918493<br>**24,46s** | 222122<br>**6,25s** | **85,14%** | **96,20%** |
| HAAR | test02 | 2948210<br>73,59s | 429960<br>**11,67s** | 103469<br>**3,18s** | **84,14%** | **95,68%** |
| HAAR | test03 | 360317<br>8,60s | 58377<br>**1,39s** | 13874<br>**0,34s** | **83,87%** | **96,08%** |
| LRD | test01 | 2690853<br>20,11s | 431361<br>**3,22s** | 106409<br>**0,84s** | **84,01%** | **95,80%** |
| LRD | test02 | 1673825<br>11,73s | 256772<br>**1,90s** | 64196<br>**0,54s** | **83,85%** | **95,41%** |
| LRD | test03 | 143296<br>1,06s | 21059<br>**0,17s** | 3153<br>**0,03s** | **84,09%** | **97,38%** |

*Tabulka 21: Zrychlení detekce na úkor kvality – HTC Touch HD (Blackstone)*

# Příloha C

Obrázek 17 Testovací obrázek test01 – 800x480

Obrázek 18 Testovací obrázek test02 – 800x480



Obrázek 19 Testovací obrázek test03 – 144x176