VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

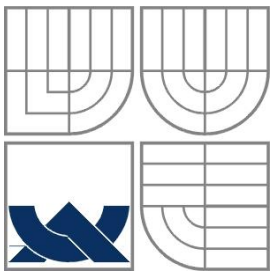
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

OPTIMALIZACE ALGORITMŮ PRO ZPRACOVÁNÍ OBRAZU V C++ POMOCÍ ŠABLON

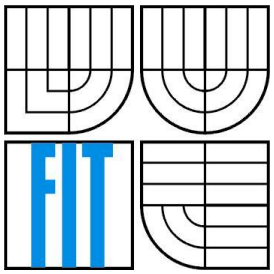
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE RADEK ČEPL
AUTHOR

BRNO 2008



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

OPTIMALIZACE ALGORITMŮ PRO ZPRACOVÁNÍ OBRAZU V C++ POMOCÍ ŠABLON

IMAGE PROCESSING ALGORITHMS OPTIMIZATION USING C++ TEMPLATES

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

RADEK ČEPL

VEDOUCÍ PRÁCE
SUPERVISOR

ING. MICHAL ŠPANĚL

BRNO 2008

Optimalizace algoritmů pro zpracování obrazu v C++ pomocí šablon

Image Processing Algorithms Optimization Using C++ Templates

Vedoucí:

Španěl Michal, Ing., UPGM FIT VUT

Oponent:

Vyskočil Michal, Ing., UPGM FIT VUT

Zadání:

1. Prostudujte základy zpracování obrazu. Zaměřte se na efektivní metody reprezentace a zpracování obrazu.
2. Vypracujte přehled existujících knihoven pro zpracování obrazu v C++. Jaké jsou jejich přednosti a nectnosti.
3. Analyzujte možnosti optimalizace algoritmů pro zpracování obrazu pomocí šablon v jazyce C++.
4. Navrhněte jednoduchou knihovnu, která bude na základních obrazových operacích demonstrovat metody optimalizace.
5. Experimentujte s vaší implementací.
6. Diskutujte dosažené výsledky a možnosti budoucího vývoje.
7. Vytvořte stručný plakát prezentující vaši bakalářskou práci, její cíle a výsledky.

Kategorie:

Počítačová grafika

Implementační jazyk:

C++

Operační systém:

MS Windows, Linux (pokud možno přenositelný kód)

Literatura:

- Dle pokynů vedoucího.

Licenční smlouva

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

Abstrakt

Bakalářská práce se zabývá optimalizací algoritmu AdaBoost pro zpracování obrazu v C++ pomocí šablon. Zaměřuje se především na efektivní vyhodnocení Haarových příznaků pevné velikosti. Je zde porovnána rychlost detekce při klasickém a šablonovém vyhodnocení příznaků. Celá aplikace je vytvořena pomocí jazyka C++ s využitím grafické knihovny OpenCV a knihovny TinyXML a je testována v prostředí operačního systému Windows XP.

Klíčová slova

AdaBoost, Haarovy příznaky, Integrální obraz, Klasifikátor, C++ šablony

Abstract

Bachelor‘s thesis deals with image processing algorithm AdaBoost optimalization using C++ templates. Head aim of this thesis is effective evaluation of Haar Features with constant size. It also compares speed of feature detection on classical and template evaluation. The computer programme was written in C++ programming language using OpenCV graphic library and TinyXML library. Application was created and tested under Windows XP operating system.

Keywords

AdaBoost, Haar features, Integral image, Classifier, C++ templates

Citace

Radek Čepl: Optimalizace algoritmů pro zpracování obrazu v C++ pomocí šablon. Brno, 2008, bakalářská práce, FIT VUT v Brně.

Optimalizace algoritmů pro zpracování obrazu v C++ pomocí šablon

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Michala Španěla. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Radek Čepl
10. května 2008

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Michalu Španělovi za vstřícnost a odborné vedení při zpracování této práce. Poděkování patří také Ing. Romanu Juránkovi za jeho ochotu a užitečné rady.

Obsah

1 Úvod

Počítače se postupem času staly důležitým pomocníkem v nejrůznějších odvětvích. V dnešní době je lze bez problémů používat i k tak náročným úkonům jako zpracování obrazových dat a různé operace s obrazem.

Zpracování obrazu je obor, který je především zaměřen na analýzu obsahu obrazu. Zpracovává získaný digitalizovaný obraz a řeší takové otázky jako: jak nalézt podobné objekty, jak zvýraznit různé částí objektů, jak nalézt pozici obličeje v obraze nebo jak opravit poškozený obraz. Počítačové vidění je úzce spojený obor s právě zmíněným zpracováním obrazu, ale tento obor pomocí technických prostředků usiluje o získání popisu objektu vyskytujícího se v obraze a pokouší se, na rozdíl od zpracování obrazu, získaná data interpretovat. Tímto se, alespoň částečně, snaží pro stroj napodobit lidské vidění.

Detekce obličeje je velmi důležitý a složitý problém z oboru počítačového vidění, který se zatím stále nedaří řešit s úplnou přesností. Zjištění přítomnosti a polohy obličeje v obraze je velmi důležité při kontrole výrazu v obličeji, identity nebo aktivity člověka. Vzhledem k poměru velikosti obrazu a obličeje a rychlosti pohybu obličeje musí být tato detekce provedena co nejrychleji.

Tato práce se bude zabývat zrychlením celého vyhodnocení při výpočtu příznaků pomocí C++ šablon, tím se zrychlí celá detekce obličeje. Celé toto šablonové vyhodnocení je poté porovnáno s „klasickým“ vyhodnocením bez šablon a vyvozen patřičný závěr.

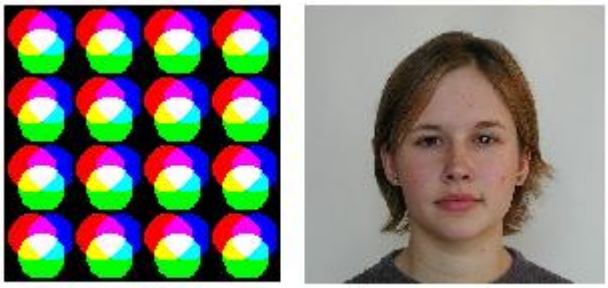
Dokument obsahuje celkem 8 kapitol. První kapitola s názvem **Úvod** obsahuje seznámení s danou problematikou a uvedení do souvislostí. Druhá kapitola vysvětluje důležité pojmy související detekcí. Ve třetí je popsána funkce a důvod použití C++ šablon v tomto problému. Kapitola 4 popisuje vlastní klasifikaci. Pátá kapitola obsahuje návrh a řešení daného problému. V kapitole 6 je popsána výsledná implementace aplikace. Kapitola 7 popisuje postup testování a ukazuje dosažené výsledky. Celá práce je zhodnocena a uzavřena v poslední osmé kapitole s názvem **Závěr.**

2 Vysvětlení pojmů

Tato kapitola obsahuje vysvětlení důležitých pojmů souvisejících s detekcí obličeje.

2.1 Barevný obraz

Barevný obraz, v případě rastrového resp. digitálního obrazu, je grafická interpretace informace složená z matice pixelů (bodů, indexů). Používá se tří prvkový barevný model a pro každý prvek je přiděleno 8 bitů, čili celkem 24 bitů, pomocí kterých lze zobrazit přibližně 16 miliónů barev. Jako základní model v počítačích je používáno aditivní skládání barev (zvané RGB) pomocí červené, zelené a modré barvy.



obr. 2.1 - Ukázka matice pixelů a barevného obrazu

2.2 Grayscale obraz

Je obraz, který má, na rozdíl od barevného obrazu, zredukovaný barevný prostor pouze na stupně šedi. Pro zobrazení standardních 256 stupňů šedi nám stačí pouze 8 bitů. Pro takovýto převod (z 24 bitů na 8 bitů) se používá empirický vztah:

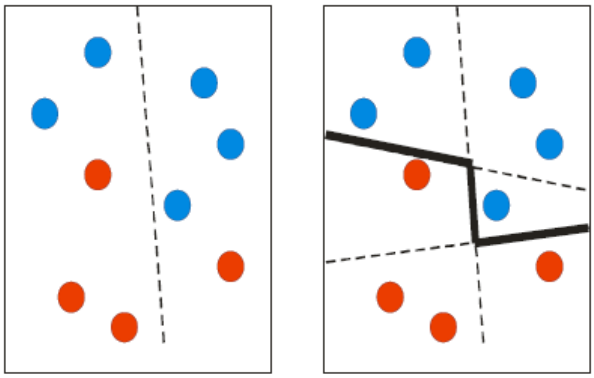
$$I = 0.299R + 0.587G + 0.114B \qquad \text{[vzorec 2.2]}$$

a výsledná hodnota je vždy uložena do všech tří prvků (neplatí v případě jednosložkového obrázku).

obr. 2.2 - Ukázka převodu barevného obrazu na 256 úrovní šedi

2.3 Klasifikátor

Klasifikátor je velmi rychlý, i když ne tolik přesný algoritmus, který je použit na určitá data a snaží se s určitou pravděpodobností rozhodnout, zda splňují danou podmínku. Tato pravděpodobnost musí být větší než náhodný výběr, tedy větší než 50%. Klasifikátory se dělí na **slabé** a **silné**, přičemž silné umějí rozhodovat s větší přesností.



obr. 2.3 - Rozhodování pomocí slabého a silného klasifikátoru

2.4 AdaBoost

AdaBoost znamená „Adaptive boosting“. Takto se nazývá metoda, která spojuje několik slabých klasifikátorů, které mají jako vstup pouze jeden příznak (práh, histogram, rozhodovací strom), do jednoho silného klasifikátoru, který je při rozhodování mnohem úspěšnější než jednotlivé slabé klasifikátory.

2.4.1 Boosting

Jedná se o metodu, jejímž cílem je zlepšit klasifikační přesnost libovolného algoritmu strojového učení. Základem metody je vytvoření více klasifikátorů pomocí výběru vzorků ze základní trénovací množiny.

Boosting je založen na vytvoření počátečního klasifikátoru, jehož přesnost je větší než 50%, a postupném přidávání dalších klasifikátorů se stejnou klasifikační vlastností. Takto je vytvořen soubor klasifikátorů, který má celkovou klasifikační přesnost libovolně vysokou vzhledem ke vzorkům na trénovací množině, tímto je klasifikace posílena (boosted).

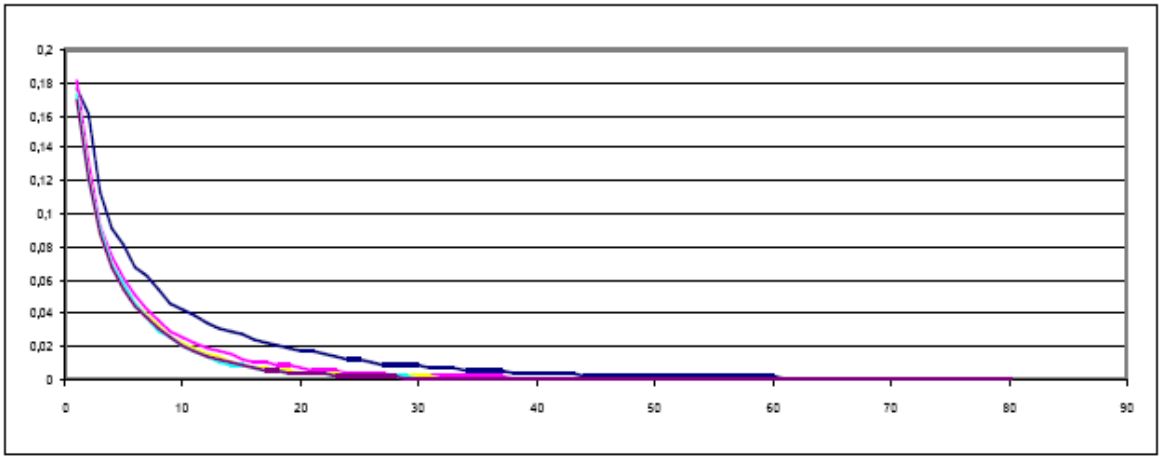
2.4.2 Princip AdaBoostu

Adaboost je v současnosti nejpoužívanější varianta metody boosting.

Byl představen v roce 1995 a jeho největší předností je schopnost exponenciálně snižovat chybu výsledného klasifikátoru na libovolně nízkou úroveň (s danou množinou vzorků a slabých klasifikátorů). Dokáže produkovat v relativně krátkém trénovacím čase klasifikátory s velmi malou chybou jen za použití jednoduchých slabých klasifikátorů. Takové klasifikátory pak mají výhodu, že jsou při použití v reálných podmínkách vysoce přesné a jejich vyhodnocení lze provést ve velice krátkém čase.

Každý trénovací vzorek dostane přidělenu váhu, která určuje pravděpodobnost jeho výběru do trénovací podmnožiny pro jednotlivé klasifikátory z vytvářené skupiny. Je-li vzorek klasifikován přesně, jeho šance na opětovný výběr pro následující klasifikátor klesá nebo v opačném případě roste. Tímto se algoritmus zaměřuje na obtížné vzorky v trénovací sadě.

Takto AdaBoost minimalizuju chybu na trénovacích datech. Pokud se stále daří hledat slabé klasifikátory s pravděpodobností větší než 50%, chyba na trénovacích datech v limitě klesne k nule (viz. obr. 2.4.2).



obr. 2.4.2 - Pokles chyby na trénovacích datech přidáváním klasifikátorů

Zvyšování počtu klasifikátorů ve skupině ale může vést k tzv. přetrénování (ztrátě schopnosti generalizovat vlivem přílišného zaměření klasifikátorů na rozeznávání pouze konkrétních trénovacích dat). K tomuto jevu dochází ale relativně zřídka i pro extrémně vysoké hodnoty.

Pro praktickou aplikaci AdaBoostu platí základní pravidlo:
Boosting zlepšuje klasifikaci pouze tehdy, pokud klasifikátory ve skupině poskytují lepší než jenom náhodné výsledky, což však nelze předem zaručit.

3 Šablony

Tato kapitola vysvětluje, co jsou šablony a jak, kdy a kde se používají.

3.1 Vznik šablon

C++ šablony neboli C++ templates původně vznikly jako nástroj pro tvorbu generických kontejnerů, které jsou řešením pro opětovné použití kódu. Nýbrž takováto implementace nebyla dostačující a do definice jazyka byly přidány další funkce jako parciální specializace šablon objektových typů, přetěžování šablon volných funkcí nebo pravidlo o dvojím čtení definice šablony. Takto vznikl mocný nástroj použitelný v řadě problémů.

3.2 Specializace šablon

Specializaci šablon lze ukázat na tomto příkladě:

```
template<bool b> class Assert;        // Primární šablona
template<> class Assert<true> {};     // Specializace

const int n = 25;
Assert<(n>20)> x;
```

Kde kontrola, zda konstanta n je větší než 20, proběhne už **při překladu,** nikoliv za běhu. Je-li tato podmínka splněna, použije překladač specializaci Assert<true> a vytvoří bezvýznamnou proměnnou. Jestliže podmínka není splněna, zjistí překladač, že nezná specializaci pro hodnotu false. Pokusí se tedy použít obecnou šablonu, ale protože tu také nezná, ohlásí chybu.

Šablony dále umožňují zobrazit jeden typ na jiný, zobrazit celé číslo na typ, vybrat v závislosti na dané podmínce jeden ze dvou typů, kde výběr hodnoty lze řešit např. pomocí podmínkového operátoru ?: a datový typ lze zvolit pomocí mechanizmu specializace šablon objektových typů, a umožňuje i naprogramovat cyklus.

3.3 Rekurzivní specializace šablon

Cykly v šablonách objektových typů definovat nelze – aparát šablon nenabízí analogii příkazu while a dalších, lze je ovšem nahradit rekurzí vzhledem k celočíselným parametrům, neboť definice šablony s celočíselným parametrem se může odvolávat na tutéž šablonu s jinou hodnotou parametru.
Ukončení rekurze předepíšeme pomocí specializace pro hraniční hodnotu. Podle okolností může jít o úplnou nebo částečnou (parciální) specializaci.

Zde je ukázka příkladu pro výpočet hodnoty 3^N pomocí rekurzivní specializace šablon:

```
// primární šablona pro 3^N
template<int N>
class Pow3 {
  public:
    enum { result=3*Pow3<N-1>::result };
};

// specializace pro N == 0 – ukončení rekurze
template<>
class Pow3<0> {
  public:
    enum { result = 1 };
};
```

Po použití
```
cout << Pow3<7>::result << '\n';
```

bude vypsán výsledek 3^7, tedy číslo 2187.
Je zde tedy použito těchto pravidel pro výpočet mocnin:

- $3^N = 3 * 3^{N-1}$
- $3^0 = 1$

Při výpočtu překladač vytváří odpovídající instance Pow3<N>, kde se ovšem odvolává na hodnotu Pow3<N-1>. Tato rekurze skončí, když překladač při vytváření instance Pow3<1> najde odkaz na použití Pow3<0>, neboť zde použije specializaci primární šablony pro tuto hodnotu parametru a tato specializace se již na nic neodvolává.

Překladačem vytvářené instance šablony Pow3<1>, Pow3<2>, atd. nezpůsobují generování žádného kódu, nýbrž překladači pomáhají v **době překladu** vypočítat hodnotu požadované konstanty.

3.4 Optimalizace

Šablonové metaprogramování skutečně vede k efektivnějšímu programu, ať už kvůli znovupoužití různých částí kódu, či rychlejšímu vyhodnocování. Šablony se hodí všude tam, kde pracujeme s pevnými hodnotami a výpočet má být proveden co nejrychleji. Tohoto je dosaženo vyhodnocením šablony už v době překladu, což je rychlejší nežli vyhodnocování při běhu programu.

Při testování rychlosti výpočtu se mohou projevovat i kvality optimalizace v jednotlivých překladačích, ale ve většině případů se šablonové metaprogramování ukázalo jako velmi rychlé a zrychlení může být vyšší než 50%.

4 Klasifikace

Kapitola popisuje, jak je potřeba upravit obraz pro klasifikátor, aby byla klasifikace efektivní a správně provedená, dále použití Haarových příznaků v slabých klasifikátorech, uložení klasifikátorů a použití šablon pro rychlejší vyhodnocování.

4.1 Úprava obrazu pro klasifikátor

4.1.1 Vstupní obraz

Jako vstup programu je použit upravený barevný obraz do formátu vhodného pro detekci. Takto upravený obraz má podobu vyříznutého obličeje a jeho velikost je zmenšena na 24x24 pixelů. Tato velikost je dána nastavenou velikostí klasifikačního okna v použitém detektoru. Jako vstupní obrazy byly použity upravené obrazy z databáze CBCL[12].

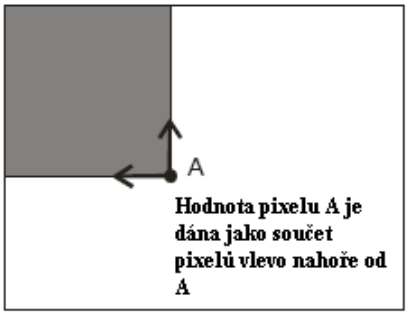
obr. 4.1.1 - Příklady použitých vstupních obrazů (velikost 24x24 pixelů)

4.1.2 Integrální obraz

Integrální obraz je takový obraz, jenž má v každém pixelu uloženu hodnotu, která odpovídá součtu hodnot pixelů vlevo nahoru od tohoto pixelu. Tento obraz se tvoří z Grayscale obrazu a je použit pro efektivní zjištění sumy pixelů v obdélníkových oblastech v Haarových příznacích. Při použití takovéhoto uložení obrazu lze získat součet jakékoliv obdélníkové oblasti v obraze pomocí rohových pixelů tohoto obdélníka.

Matematická reprezentace integrálního obrazu: [vzorec 4.1.2]

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x', y')$$



obr. 4.1.2a - Princip vytvoření integrálního obrazu

V našem detektoru je integrální obraz vytvořen podobným způsobem tak, že je o jeden pixel širší a vyšší než zdrojový obraz a hodnoty pixelů na nultém řádku i sloupci jsou nulové.
Hodnota pixelu na souřadnicích [x, y] je tedy součet hodnot pixelů na souřadnicích [0,0] až [x-1, y-1].

Grayscale obraz

x,y	0	1	2	3
0	23	27	25	58
1	32	23	86	254
2	18	14	15	15
3	89	66	58	57

Integrální obraz

x,y	0	1	2	3	4
0	0	0	0	0	0
1	0	23	50	75	133
2	0	55	105	216	528
3	0	73	137	263	590
4	0	162	292	476	860

obr. 4.1.2b - Převod Grayscale obrazu na integrální obraz

4.1.3 Integrální obraz druhých mocnin

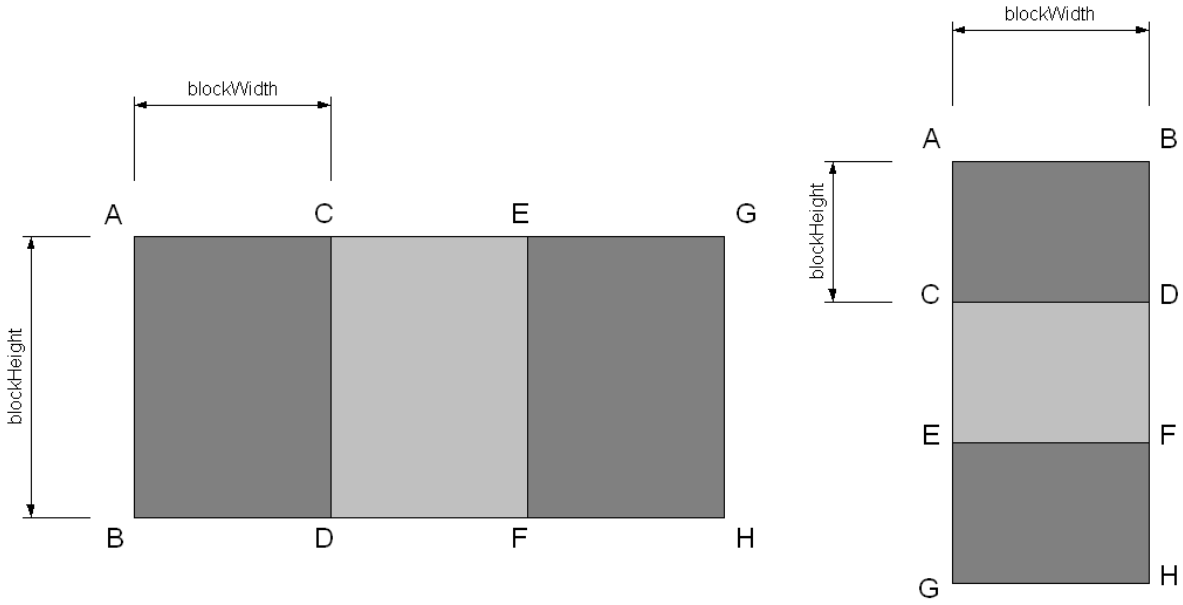
Integrální obraz druhých mocnin má v každém pixelu uloženu hodnotu, která odpovídá součtu druhých mocnin hodnot pixelů vlevo nahoře od tohoto pixelu. Tento obraz se tvoří stejným principem jako integrální obraz a je použit pro efektivní výpočet standardní odchylky v Haarových příznacích.
Matematická reprezentace integrálního obrazu druhých mocnin: [vzorec 4.1.3]

$$ii(x,y) = \sum_{x' \le x, y' \le y} i^2(x', y')$$

I tento obraz je v našem detektoru vytvořen s šířkou i výškou o jeden pixel vyšší a nulovými hodnotami na prvním řádku i sloupci. Hodnota pixelu na souřadnicích [x, y] je určena součtem hodnot druhých mocnin pixelů na souřadnicích [0,0] až [x-1, y-1].

4.2 Haarovy příznaky

Jedná se o spojité obrazové příznaky, které jsou často základem slabých klasifikátorů pro zpracování obrazu. Jako výsledek vracejí reálné hodnoty a jsou určeny především tvarem konvolučního jádra (lze si představit jako tabulku umístěnou na určité místo v obraze), dále jeho pozicí v obraze a velikostí. Námi použité příznaky jsou na obr. 4.2.

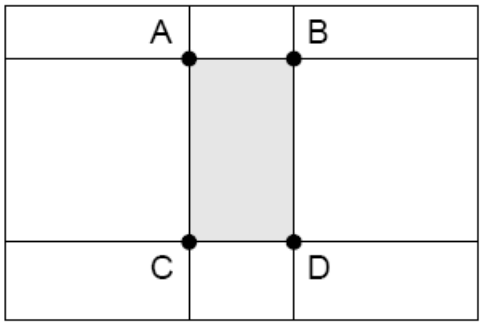


obr. 4.2 - Použité Haarovy příznaky a znázornění jejich bloků

V tomto případě se jedná o horizontální trojitý a vertikální trojitý příznak. A-H jsou rohové pixely jednotlivých bodů, blockWidth a blockHeight jsou šířka a výška jednoho bloku příznaku. Při klasifikaci jsou použity také dvojité příznaky, které jsou určeny pouze rohy A-F.

4.2.1 Použití integrálního obrazu

Při výpočtech je použití příznaků místo pixelů daleko rychlejší a tento přístup je velice vhodný. Protože vyhodnocování sumy pixelů v jednotlivých blocích je děláno opakovaně, je použito integrálního obrazu pro efektivní vyhodnocení těchto oblastí.
Toto vyhodnocení je zobrazeno na obr. 4.2.1.



obr. 4.2.1 - Pro tuto oblast v integrálním obraze (A, B, C, D jsou rohové pixely) bude suma pixelů v šedé oblasti rovna D+A-B-C

4.2.2 Odezva příznaku

Výsledná odezva příznaku f(x) je dána jako rozdíl černých a bílých oblastí. Při použití tohoto vyhodnocení jednotlivých bloků pak platí pro jednotlivé typy příznaků tyto vzorce na výpočet odezvy pro vzorek x:

Double Horizontal Feature (dvojitý horizontální)

$$f(x) = -A + B + 2C - 2D - E + F$$ [vzorec 4.2.2a]

Double Vertical Feature (dvojitý vertikální)

$$f(x) = -A + B + 2C - 2D - E + F$$ [vzorec 4.2.2b]

Ternal Horizontal Feature (trojitý horizontální)

$$f(x) = A - B - 3C + 3D + 3E - 3F - G + H$$ [vzorec 4.2.2c]

Ternal Vertical Feature (trojitý vertikální)

$$f(x) = A - B - 3C + 3D + 3E - 3F - G + H$$ [vzorec 4.2.2d]

Tyto vzorce pro odezvu Haarových příznaků můžou být u jednotlivých typů klasifikátorů různé podle toho, zda se odečítá bílá část od černé nebo naopak a zda se u trojitých příznaků použije (pro vyvážení hodnot) bílá část jednou nebo dvakrát.

4.2.3 Standardní odchylka

Haarovy příznaky potřebují po svém vyhodnocení normalizovat standardní odchylkou hodnot ve zkoumaném okně. Pro určení odchylky je proto potřeba ze vstupního obrazu vytvořit integrální obraz a integrální obraz druhých mocnin, pomocí kterých se dá standardní odchylka snadno a efektivně vypočítat.

Pro výpočet je použit vzorec:

$$\sigma^2 = \frac{1}{N}(E - F - G + H) - [\frac{1}{N}(A - B - C + D)]^2 \qquad \text{[vzorec 4.2.3]}$$

Kde

- A, B, C, D jsou hodnoty pixelů v rozích okna integrálního obrazu
- E, F, G, H jsou hodnoty pixelů v rozích okna integrálního obrazu druhých mocnin
- N je počet pixelů v okně

4.2.4 Výsledná hodnota příznaku

Výslednou hodnotu příznaku r(x) po zjištění standardní odchylky a odezvy příznaku lze vypočítat: [vzorec 4.2.4]

$$r(x) = \frac{\mathrm{f(x)}}{\mathrm{blockSize} * \mathrm{stddev}}$$

Přičemž

- f(x) je odezva příznaku na vzorek
- blockSize je velikost jednoho bloku příznaku (blockWidth * blockHeight)
- stddev je standardní odchylka klasifikačního okna

4.3 Odezvy slabých klasifikátorů

Výsledná hodnota příznaku je důležitá pro určení odezvy slabého klasifikátoru h(x).

Výsledná hodnota odezvy:

- je-li hodnota příznaku f(x) větší nebo rovna než klasifikační práh threshold, je dána jako součin hodnoty parita a alpha
- je-li hodnota příznaku f(x) menší než klasifikační práh threshold, je dána jako součin hodnoty minus parita a alpha

Vyjádřeno vzorci:

$$r(x) \geq threshold \quad \rightarrow \quad h(x) = parity * alpha \qquad \text{[vzorec 4.3a]}$$

$$r(x) < threshold \quad \rightarrow \quad h(x) = -parity * alpha \qquad \text{[vzorec 4.3b]}$$

4.4 Výsledná klasifikace

Po vypočítání všech hodnot jednotlivých slabých klasifikátorů je výsledná klasifikace dána sumou všech hodnot: [vzorec 4.4]

$$H(x) = \sum_{i=0}^{N-1} h_i(x)$$

Pokud hodnota výsledné klasifikace $H(x) > 0$, byl detekován obličej.

4.5 Reprezentace klasifikátoru

Výsledný klasifikátor je reprezentován natrénovanou sadou slabých klasifikátorů.

```
<AdaBoostBinaryClassifier sizeX="24" sizeY="24">
  <DecisionStumpWeakHypothesis threshold="-0.542373" parity="-1" alpha="0.901443">
    <HaarHorizontalTernalFeature positionX="6" positionY="4" blockWidth="4" blockHeight="6"/>
  </DecisionStumpWeakHypothesis>
  <DecisionStumpWeakHypothesis threshold="-0.40678" parity="-1" alpha="0.651073">
    <HaarVerticalTernalFeature positionX="6" positionY="6" blockWidth="3" blockHeight="4"/>
  </DecisionStumpWeakHypothesis>
  <DecisionStumpWeakHypothesis threshold="-0.542373" parity="-1" alpha="0.566967">
    <HaarHorizontalDoubleFeature positionX="14" positionY="0" blockWidth="4" blockHeight="5"/>
  </DecisionStumpWeakHypothesis>
  <DecisionStumpWeakHypothesis threshold="0.135593" parity="1" alpha="0.562937">
    <HaarVerticalDoubleFeature positionX="10" positionY="8" blockWidth="8" blockHeight="2"/>
  </DecisionStumpWeakHypothesis>
                Další slabé klasifikátory...
</AdaBoostBinaryClassifier>
```

obr. 4.5 - Uložení klasifikátoru (složeného z jednotlivých slabých klasifikátorů) v XML souboru

Celý klasifikátor je uložen v XML souboru pro lepší přenositelnost a přístup k datům.

Námi použitý klasifikátor pracuje s klasifikačním oknem o velikosti 24x24 pixelů.

Jednotlivé parametry slabých klasifikátorů jsou:

- threshold – klasifikační práh
- parity – směr vyhodnocení (hodnota +1 nebo -1)
- alpha – váha klasifikátoru

Parametry v nich použitých příznacích:

- positionX – X-ová souřadnice pozice příznaku (příznakového rohu A)
- positionY – Y-ová souřadnice pozice příznaku (příznakového rohu A)
- blockWidth – šířka jednoho bloku příznaku
- blockHeight – výška jednoho bloku příznaku

Přičemž počáteční souřadnice [0,0] se nachází v levém horním rohu klasifikačního okna, X určuje jednotlivé sloupce a Y řádky pixelů, a blokem příznaku je myšlena např. oblast A-D (viz. obr. 4.2).

4.6 Použití šablon v klasifikátoru

V klasifikátoru jsou šablony použity pro snahu optimalizovat vyhodnocování příznaků, protože toto vyhodnocování je při detekci prováděno opravdu hodně krát a jakékoliv malé zrychlení vyhodnocení jednoho příznaku se ve výsledku může značně projevit.

Šablony lze v klasifikátoru velmi dobře použít, jelikož klasifikační okno má pevnou velikost 24x24 pixelů a jednotlivé parametry slabých klasifikátorů i Haarových příznaků jsou také už konkrétní natrénované hodnoty.

5 Návrh řešení

Tato kapitola se zabývá vlastním návrhem celé aplikace a rozdělením detekce na jednotlivé podproblémy.

Samotná aplikace je složena z několika částí, které na sebe navazují a dohromady vytvoří požadovaný výsledek. Výsledné aplikaci bude předložen vstupní obraz obsahující popř. neobsahující obličej pro případnou detekci a výsledkem bude výpis hodnoty celkové klasifikace a doba trvání této klasifikace.

Aplikace je rozdělena na tyto podproblémy:

- Načtení vstupního obrazu
- Vytvoření pomocných obrazů
    1. Vytvoření Grayscale obrazu
    2. Vytvoření integrálního obrazu
    3. Vytvoření integrálního obrazu druhých mocnin
- Použití dat z XML souboru
    1. Načtení dat z XML souboru
    2. Vytvoření souboru obsahujícího volání funkcí (dle implementace šablon, popř. metod), které mají jako parametry hodnoty načtené z XML souboru
- Vyhodnocení klasifikátoru
- Změření času vyhodnocení
- Výpis času a výsledku klasifikace

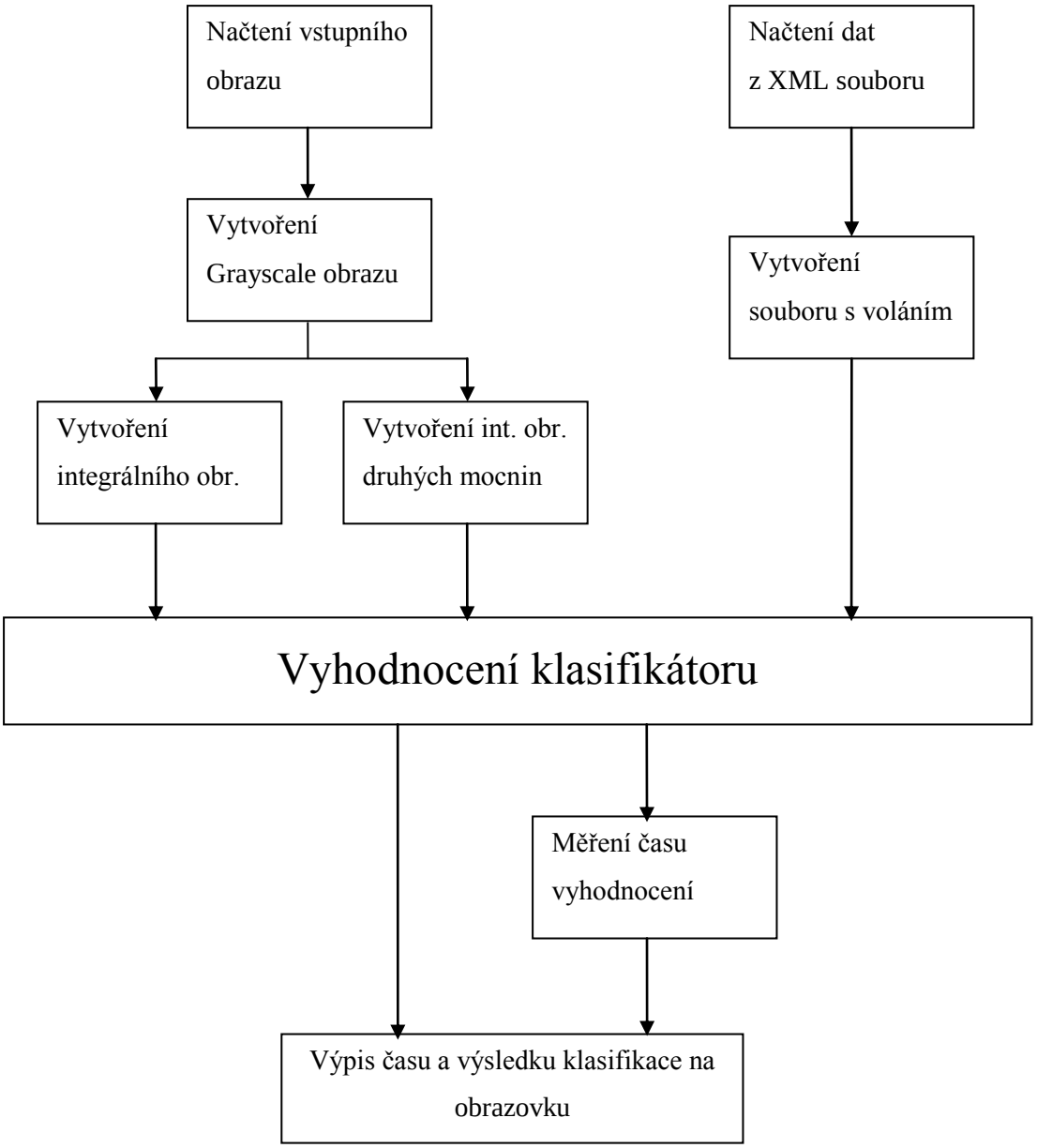
Uvedené podproblémy musejí být vyřešeny co nejefektivněji, snadno pochopitelným způsobem a s co nejmenšími nároky na paměť.

Vstupním obrazem je obraz ve standardních formátech bmp, jpeg nebo jiných, které jsme schopni pomoci funkcí knihovny OpenCV načíst a zpracovat. Obličej ve vstupním obraze by měl být vyříznut tak, aby ho bylo schopné detekovat a neměl by být ani příliš nakloněný, z části rozmazaný a jiné neduhy, kvůli kterým nemusí dojít k detekci.

Jako klasifikátor je vybrán AdaBoost, který je hojně používaný, lze ho dále zlepšovat a nabízí dobré výsledky klasifikace.

K měření času je využito časovače z knihovny cudaUtils, toto měření bude aplikováno pouze na hlavní vyhodnocení klasifikátoru.

obr. 5 - Blokové schéma postupu práce detektoru



Načtení parametrů natrénovaných slabých klasifikátorů a vytvoření souboru s už hotovými voláními funkcí s těmito parametry je univerzální a umožňuje použít jiný natrénovaný klasifikátor. Námi použitý klasifikátor obsahuje pouze 50 slabých klasifikátorů, což není mnoho, ale pro naše potřeby měření času to bohatě postačí.

6 Implementace

V této kapitole je podrobněji popsána vlastní implementace programu. Budou zde vysvětleny použité postupy a algoritmy.

6.1 Knihovna OpenCV

OpenCV neboli Open Computer Vision Library je optimalizovaná grafická knihovna vyvíjená společností Intel. Obsahuje mnoho funkcí pro práci s obrazem, grafickými formáty, ale i pro tvorbu uživatelských rozhraní atd.
V projektu je použita pro práci s obrazem, zejména při načítaní obrazu.

6.2 Knihovna TinyXML

TinyXML je volně dostupná knihovna pro práci s XML soubory. Je velmi malá a jednoduchá. Umožňuje parsování (analýzu syntaxe), čtení a zápis do XML souboru a pracuje s DOM modelem.
V projektu je použita pro načítání slabých klasifikátorů z XML souboru, konkrétně druh klasifikátoru, jeho parametry a parametry v něm obsažených příznaků.

6.3 Knihovna cudaUtils

Jedná se o knihovnu, která je primárně určená pro podporu GPGPU (General-purpose computing on graphics processing units) Cuda společnosti Nvidia. Knihovna ale poskytuje i řadu dalších funkcí nezávisle na použití v GPGPU, např. analýzu argumentů příkazového řádku, vytváření časovačů, hlídání chybových hlášení atd. V projektu je použit časovač pro měření trvání úseku kódu, konkrétně kódu vyhodnocujícího hlavní klasifikaci vstupního obrazu.

6.4 Načtení vstupního obrazu

Jako vstup je očekáván barevný obraz velikosti 24x24 pixelů. Tento obraz je načten funkcí cvLoadImage z knihovny OpenCV do struktury IplImage, která si uchovává o obrazu spoustu informací, jako například druh barevného modelu, hloubku barev, šířku a výšku obrazu, vlastní obrazová data a mnoho dalších. Tato funkce je použita s hodnotou druhého parametru 0, čili obraz je načten rovnou ve stupních šedé (Grayscale). Dále je vytvořen objekt myImage šablonové třídy Image s pevnými parametry udávajícími pevnou výšku a šířku obrázku. Metodou loadData načteme do pole pixValues ze struktury IplImage obrazová data s kterými budeme dále pracovat.

Šablonová třída Image si tedy uchovává informace o výšce, šířce a celkové velikosti obrazu a také vlastní obrazová data. Dále obsahuje kromě metody loadImage také metodu getPixel, použitou pro načtení hodnot pixelu z pole pixValues dle udaných souřadnic x a y.

6.5 Vytvoření integrálního obrazu a integrálního obrazu druhých mocnin

Poté jsou vytvořeny další dva objekty třídy Image, a to myIgImage a myPowIgImage reprezentující integrální obraz a integrální obraz druhých mocnin. Oba tyto objekty jsou vytvořeny s pevnou šířkou a výškou 25 pixelů. Dále použijeme šablonové funkce makeIgImage a makePowIgImage pro vytvoření těchto obrazů. Obrazy jsou vytvářeny cyklem procházejícím vstupní obrázek po řádcích od souřadnice 0,0 (levý horní roh obrazu). Při tomto procházení jsou počítány jednotlivé sumy pixelů ležících doleva nahoru od počítané pozice v integrálním obraze, popř. integrálním obraze druhých mocnin a ukládány na počítanou pozici.

6.6 Načtení dat z XML souboru a vytvoření souboru s voláním šablon

Při načítání z XML souboru je nejdříve vytvořen soubor features.h, který je součástí šablonového vyhodnocení. Poté jsou načteny pomocí funkcí TinyXML jednotlivé slabé klasifikátory.
Nejdříve je zjištěno o jaký druh klasifikátoru se jedná (HaarHorizontalDoubleFeature, HaarVerticalDoubleFeature, HaarHorizontalTernalFeature, HaarVerticalTernalFeature). To ovlivní, jaká šablona pro vyhodnocení bude použita. Poté jsou načteny příslušné parametry slabého klasifikátoru (threshold, parity, alpha) a parametry Haarova příznaku v něm obsaženého (positionX, positionY, blockWidth, blockHeight). Tyto parametry jsou pak použity jako parametry pro volání šablony pro vyhodnocení slabého klasifikátoru.

6.7 Postup klasifikace

Při klasifikaci je volána šablonová funkce evalFtrs, která je vytvořena v souboru features.h. Tato funkce má jako parametry integrální obraz a integrální obraz druhých mocnin. Poté jsou postupně volány jednotlivé vyhodnocení slabých klasifikátorů podle příznaků v nich obsažených. Hodnoty příznaků jsou uchovávány a postupně sumarizovány pro získání hodnoty výsledné klasifikace. Tato hodnota je nakonec vrácena jako hodnota funkce.

Šablony jednotlivých slabých klasifikátorů jsou volány s parametry načtenými ze zdrojového XML souboru. Definice klasifikátorů se nacházejí v souboru templates.h. Šablony obsahují všechny parametry načtené z XML souboru, a také integrální obraz a integrální obraz druhých mocnin pro pozdější počítání standardní odchylky.

V definicích šablon slabých klasifikátorů jsou nejdříve pomocí funkce getPixel získány hodnoty pixelů v rozích příznaků z integrálního obrazu. Souřadnice rohu A v příznaku je určena pomocí parametrů ftrPosX a ftrPosY. Další rohy jsou získány pomocí souřadnic rohu A s patřičně přičtenými násobky hodnot blockWidth nebo blockHeight.

Poté je vynásobením hodnot blockWidth a blockHeight vypočtena velikost jednoho bloku příznaku a také dle příslušného vzorce (vzorce 4.2.2a – 4.2.2d) vyčíslena odezva příznaku. Pomocí velikosti bloku, odezvy příznaku a standardní odchylky je následně určena výsledná hodnota příznaku (vzorec 4.2.4).

Standardní odchylka je vyhodnocována samostatnou šablonou. Jako parametr ji jsou předávány souřadnice pixelů v rozích okna (je počítána z integrálních obrazů, čili souřadnice [0,0], [24,0], [0,24] a [24,24]) a oba integrální obrazy. Pro výpočet standardní odchylky je nejdříve zjištěn počet pixelů v klasifikačním okně, následuje získání hodnot pixelů v rozích integrálního obrazu a integrálního obrazu druhých mocnin. Vlastní hodnota odchylky je spočítána podle vzorce 4.2.3. Tato hodnota je vrácena jako hodnota šablony.

Vypočtená hodnota příznaku je porovnána s parametrem threshold a poté vrácena hodnota slabého klasifikátoru podle podmínek ve vzorcích 4.3a a 4.3b.

Nakonec jsou všechny hodnoty slabých klasifikátorů sečteny a určena výsledná hodnota klasifikace (vzorec 4.4).

6.8 Prostředí a struktura programu

Program byl vytvořen na operačním systému Microsoft Windows XP Professional SP2 pomocí programu Microsoft Visual Studio 2005 Professional Edition. Jako programovací jazyk byl použit jazyk C++ s jeho standardními knihovnami.

Pro načítání dat ze souboru XML byla použita knihovna TinyXML, pro načítání vstupního obrazu knihovna OpenCV a pro měření času knihovna cudaUtils.
Tyto knihovny byly připojeny k hlavním projektům pro výsledné přeložení celé aplikace.

Výsledná aplikace obsahuje 3 hlavní projekty, projekt XMLLoader pro načtení dat z XML souboru a vytvoření souboru s voláním slabých klasifikátorů, projekt bak1 vyhodnocující klasifikaci vstupního obrazu pomocí šablon a projekt bak2 vyhodnocující klasifikaci vstupního obrazu bez použití šablon.

7 Testování a výsledky

Tato kapitola popisuje postup testování a shrnuje dosažené naměřené výsledky.

7.1 Popis testování

Rychlost a správnost klasifikace jednoho klasifikačního okna jsou pro nás hlavní faktory při detekci obličeje v obraze.

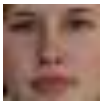
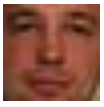
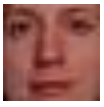
Rychlost klasifikace jednoho klasifikačního okna musí být co nejvyšší, protože toto okno je při hledání obličejů ve velkém obraze (fotografii) posouváno přes celý obraz a vyhodnocováno i několika set tisíckrát. Úspěšnost výsledné detekce závisí na použitém detektoru a jeho natrénování.

Při testování jsou použity upravené obrazy velikosti 24x24 pixelů z databáze CBCL[12] a další vytvořené obrazy obličej neobsahující.

Čas vyhodnocení klasifikačního okna je získán zprůměrováním času změřeného při 100000 opakování vyhodnocení tohoto okna.

K měření času je použito časovače z knihovny cudaUtils a všechny naměřené časy jsou vypsány v mikrosekundách.

7.2 Tabulka výsledků

Vstupní obraz	Průměrná doba klasifikace [µs]		Výsledek klasifikace
	S šablonami	Bez šablon	
	15,4129	20,812	0,629728
	15,4664	20,8096	1,81905
	16,0266	20,8774	2,43034
	15,478	20,6688	2,12361
	15,6289	20,8637	-1,18783
	15,5637	21,094	-0,871664
	15,4212	20,6183	-9,44992
	15,5669	20,4378	-5,26526
	15,366	20,5283	-6,3704
	15,333	20,5631	-5,96693

7.3 Výsledek testování

Na testování bylo použito celkem 10 různých vstupních obrazů. Pro všechny byly určeny průměrné časy klasifikace s použitím šablon a bez použití šablon. Také byla určena výsledná klasifikace, která ve většině příkladů dopadla správně. Chyba klasifikace nastala pouze v jednom případě, kdy u obrazu s obličejem místo kladné hodnoty vyšla záporná hodnota -1,18783. Toto je způsobeno druhem klasifikátoru a chybu lze napravit použitím jiného a lepšího klasifikátoru. Ale na měření času klasifikace tato chyba nemá žádný vliv.

Časy klasifikace s použitím šablon byly naměřeny okolo 15 mikrosekund, klasickým způsobem vyhodnocení bez šablon vyšly okolo 20 mikrosekund. To ve výsledku znamená, že bylo dosaženo zrychlení okolo **27,5%**. Což určitě není zanedbatelná hodnota.

Při použití dalších optimalizací může být šablonové vyhodnocení ještě daleko rychlejší než klasický způsob.

Ačkoliv se dá celá klasifikace zrychlit i dalšími postupy, aplikace šablon je z časového hlediska výhodná a určitě má aplikace šablon pro takovéto vyhodnocení klasifikačního okna pevné velikosti smysl.

Testování proběhlo na notebooku MSI Megabook L735 (AMD Athlon TK-53 64bit Dual-Core 1,7GHz, 2GB RAM) s nainstalovaným operačním systémem Windows XP Professional SP2.

Výsledná aplikace byla přeložena překladačem obsaženým v programu Microsoft Visual Studio 2005.

8 Závěr

Závěr shrnuje výsledky a poznatky celé práce a je to ukončení celé práce.

Detekce obličeje a vyhodnocování pomocí klasifikátorů je v dnešní době hojně řešený problém. Tento postup je rychlý a umožňuje úspěšnou detekci závislou na kvalitě detektoru. Zvyšování rychlosti je ale vždy potřebné, a proto se v tomto problému nabízela možnost aplikace C++ šablon jakožto nástroje stvořeného k optimalizacím.

Práce se zabývala optimalizací vyhodnocení klasifikačního okna pro detekci obličeje v obraze. Splňuje zadání v požadovaném rozsahu a dosažené výsledky jsou uspokojující.

Tyto výsledky jsou obsaženy a zhodnoceny v kapitole 7. Pro praktické použití mají vysokou hodnotu a šablonové vyhodnocení lze ihned aplikovat.

Použití šablon pro vyhodnocování klasifikačního okna pevné velikosti je velmi vhodné a výslednou detekci v celém obraze může nemálo zrychlit. Toto zrychlení může být na jiných překladačích různé. V kombinaci s jinými optimalizacemi a technikami lze ale dosáhnout ještě dalších zrychlení celé klasifikace.

Během přípravy a vlastního návrhu jsem se hlouběji seznámil s problematikou detekce obličeje, blíže pochopil vyhodnocování pomocí klasifikátorů a funkci Haarových příznaků. Také jsem si prohloubil znalosti o různých druzích optimalizace a o technikách práce s obrazovými daty používaných v dnešní době.

Celá aplikace se dá vylepšit použitím lepšího klasifikátoru a dalších metod optimalizace. Pro praktické použití by bylo potřeba aplikaci rozšířit o použití na velký obraz (fotografii) a změřit čas klasifikace celého obrazu. K pohodlnější používání lze poté vytvořit grafické uživatelské rozhraní umožňující přesnější nastavení a lepší zobrazení výsledků naměřených časů a detekovaných obličejů.

Literatura

[1] Bc. Roman Juránek, Rozpoznávání vzorů v obraze pomocí klasifikátorů, Diplomová práce, Fakulta informačních technologií VUT v Brně, 2007

[2] Ing. Přemysl Kršek, Ph.D., Základy počítačové grafiky, Studijní opora, verze 0.7, Fakulta informačních technologií VUT v Brně

[3] Doc. Dr. Ing. Pavel Zemčík, Ing Michal Španěl, Zpracování obrazu, Studijní opora, Fakulta informačních technologií VUT v Brně, listopad 2007

[4] Doc. Ing. Jan Žižka CSc., Navrhování skupin klasifikátorů pomocí opakovaného výběru trénovacích dat, Fakulta informatiky MUNI v Brně, 2005

[5] Addison Wesley, C++ Templates: The Complete Guide, ISBN 0-201-73484-2, 2002

[6] Miroslav Virius, Fascinující svět šablon v C++, Katedra softwarového inženýrství, FJFI ČVUT Praha

[7] Paul Viola, Michael Jones, Robust Real-time Object Detection, Mitsubishi Electric Research Labs, Compaq CRL, Vancouver, Kanada, 2001

[8] Intel Corporation, Open Source Computer Vision Library, Reference Manual, 2001

[9] TinyXml Documentation,
URL: http://www.grinninglizard.com/tinyxmldocs/index.html

[10] Wikipedie, Otevřená encyklopedie,
URL: http://cs.wikipedia.org/

[11] Wikipedia, The Free Encyklopedia,
URL: http://en.wikipedia.org/

[12] CBCL Face Recognition Database,
URL: http://cbcl.mit.edu/software-datasets/heisele/facerecognition-database.html

Seznam obrázků

Seznam vzorců

Seznam příloh

Příloha 1. CD