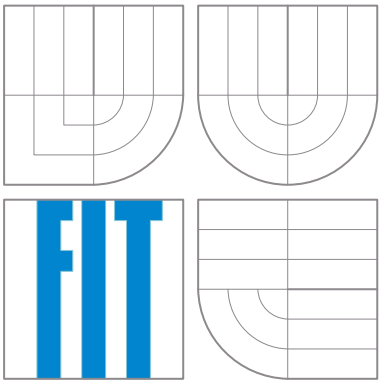


Počítačové vidění

Detekce objektů na GPU, jejich rozpoznávání a sledování

30. prosince 2014

Autor: Pavel Macenauer, xmacen02@stud.fit.vutbr.cz
Jan Bureš, xbures19@stud.fit.vutbr.cz

Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

Obsah

1 Cíl práce

Cílem této práce je naimplementovat detektor objektů na GPU a následně i rozšíření pro rozpoznání obličeje a jeho sledování. Vstupem jsou obrazová data (fotografie, video) a výstupem obrazová data a statistiky s vyznačenými detekcemi odpovídající sledovaným/detekovaným objektům.

2 Detekce objektů

Detektor objektů ke svému běhu používá 2 typy algoritmů.

- Metalgoritmus, který zpracovává celý průběh detekce, tedy skládá jednotlivé slabé klasifikátory v jeden silný a nakonec prohlásí, zda-li se jedná nebo nejedná o hledaný objekt – **Waldboost**
- Algoritmus pro vyhodnocení slabého klasifikátoru – **LBP** (Local Binary Patterns)

2.1 Waldboost

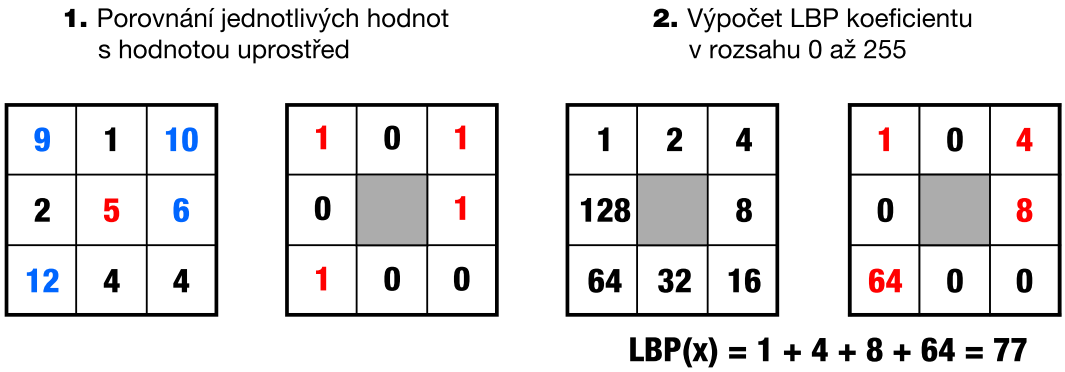
Jak již bylo zmíněno výše, Waldboost je metalgoritmus, který skládá slabé klasifikátory v jeden silný. Vychází z metalgoritmu Adaboost s tím rozdílem, že v každém svém kroku kontroluje, zda-li akumulovaná odezva nepřesáhla určitou mez. V případě, že ano, zařadí jej do odpovídající třídy, jinak pokračuje vyhodnocování dále. Dojede-li na konec, zkontroluje ještě konečnou mez a následně vrátí výsledek. Formálně lze popsat následovně:

Data: $h^{(t)}$, $\theta_A^{(t)}$, $\theta_B^{(t)}$, γ, x
Result: +1, -1
begin
  for 1 **to** T **do**
    if $H_T \geq \theta_B^{(t)}$ **then**
      classify x to the class +1 and terminate
    end
    if $H_T \leq \theta_A^{(t)}$ **then**
      classify x to the class -1 and terminate
    end
  end
  if $H_T > \gamma$ **then**
    classify x to the class +1
  else
    classify x to the class -1
  end
end

Algorithm 1: Algoritmus WaldBoost

2.2 LBP příznaky

Konkrétní odezva slabého klasifikátoru je závislá na použitých příznacích. Detektor používá příznaků LBP (Local Binary Patterns).



Obrázek 1: Výpočet LBP příznaku

3 Implementace detektoru na GPU

GPU se oproti CPU liší především tím, že obsahuje mnohonásobně více výpočetních jader. Není vhodné tedy zpracovávat data sekvenčně, kdy by se jedním jádrem, kterých můžou být v případě moderních grafických karet stovky až tisíce, zpracovával celý výpočet, ale paralelně, kdy využijeme všech.

Detektor má 2 základní fáze:

- Vytvoření pyramidového obrazu
- Detekce obličejů na pyramidovém obrazu

3.1 Vytvoření pyramidového obrazu

Vláken se rozeběhne tolik, kolik je pixelů původního obrazu. Následně se z každého pixelu vytváří zmenšeniny.

Jednotlivé zmenšeniny se vytvářejí po oktávách a úrovních. V jednoduchosti lze říci, že zmenšeniny jsou omezeny na maximální a minimální velikost. Vytvoří tolik obrazů kolik je oktáv * úrovní a nevytvářejí se již zmenšeniny, které jsou menší, než určitá mez. Každé vlákno vezme tedy jeden pixel původního obrazu a sekvenčně prochází jednotlivé zmenšeniny dané mírou zmenšení a v nich ho umísťuje.

Výstupem je jeden obraz, ve kterém jsou jednotlivé zmenšeniny původního obrazu. Tento obraz se následně nabinduje jako textura.

3.2 Detekce objektů

Vstupem do samotné detekce je pyramidový obraz uložený jako textura. Vláken se rozeběhne tolik, kolik je pixelů pyramidového obrazu.

Každý pixel je počátkem vzorku daného velikostí, na kterou je natrénován detektor. V našem případě 26x26. Následně prochází jednotlivými stages detektoru a snižuje nebo zvyšuje odezvu. V případě, že odezva spadne pod mez danou konkrétní stagí, je výpočet pro daný vzorek ukončen. V případě, že výpočet projde přes všechny stages, je zkontrolována ještě konečná mez, na odstranění parazitních vzorků s velmi malou odezvou. Pokud je odezva větší i než konečná mez, jedná se o hledaný objekt.

Takovýchto detekcí je tradičně více na jednom místě a to pro jednotlivé zmenšeniny.

3.3 Uspořádání paměti

Podstatné součásti detektoru jsou rozděleny na následující části:

- **Stages** – constant memory
  K jednotlivým stages je přistupováno sekvenčně pro každý pixel (výpočetní vlákno) obrazu. Jsou tak broadcastovány do všech výpočetních jader.
- **Alphas** – texture (global) memory
  Náhodné přístupy do constant memory by dělali přístup k alphám velmi neefektivní, protože by se zbytečně broadcastovaly. Navíc se jedná o data, kterých je více, než-li constant memory na většině grafických karet. Texturovací pamět je optimalizována pro náhodný read-only přístup.
- **Původní obraz** – texture (global) memory Texturovací pamět, krom toho, že je optimalizována pro náhodné read-only přístupy umožňuje bilineární interpolaci 2D dat. Lze tak za pomocí hardwarových jednotek počítat zmenšeniny obrazu
- **Pyramidový obraz** – texture (global) memory
- **Obrazové parametry** – constant memory
  Parametry obrazu, se kterými se hojně pracuje, např. velikost obrazu jsou taktéž ukládány v constant memory. Vzhledem k tomu, že se pro jednotlivá vlákna nemění, je vhodné je broadcastovat a využívat rychlého přístupu.

4 Trackování a rozpoznávání obličejů

Jak bylo uvedeno výše, detektor zpravidla vrací více detekcí na jednom místě pro jednotlivé zmenšeniny. Proto se při trackování nejprve z těchto mnoha detekcí vybere oblast, kde se obličej nalézá. Tento výpočet probíhá tak, že se prochází jednotlivé detekce a zkoumá se, zda nepřekrývají nějakou jinou alespoň z 50%. Pokud ano vybere se detekce s lepší odezvou, jinak není oblast do trackování obličejů zahrnuta.

Celá oblast se poté převede do barevného modelu HSV (Huge, Saturation, Value), který lépe poslouží pro rozpoznávání podobností mezi obličeji. Samotné porovnání se provádí pomocí histogramu vytvořeného na základě obrázku v barevném modelu HSV, kde pro výpočet histogramu postačí pouze hodnoty H a S. Hodnotu V jsme se rozhodli vypustit z důvodu proměnlivých podmínek osvětlení obličeje, například při pohybu člověka od/ke zdroji osvětlení scény (lampa, okno, ...).

K výpočtu rozdílnosti histogramu jsme využili metodu zvanou „Bhattacharyya distance“ viz. vzorec č. 1.

$$d(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{\overline{H_1}\overline{H_2}N^2}} \sum_I \sqrt{H_1(I) \cdot H_2(I)}} \quad (1)$$

Obrázek 2: Bhattacharyya distance

Pomocí tohoto vzorce zjistíme rozdílnost histogramu v rozsahu hodnot ¡0;1¿ (0 - stejné histogramy, 1 - histogramy nemají nic společného).

K porovnání dvou obličejů jsme využili výše uvedený postup při porovnání histogramu obličeje, ke kterému jsme ještě přidali vzdálenost oblasti, na které se obličej nacházel při posledním výskytu ve videu (popřípadě v sekvenci obrázků). Předpokládáme, že v sekvenci obrázku se daný obličej nebude přesouvat skokově, neboli vzdálenost od posledního výskytu bude u jednoho obličeje velmi malá. Problém by mohl nastat, pokud se obličej vytratí ze záběru a objeví se později v jiné jeho části (například odchod z místnosti a pozdější návrat jiným vchodem). Z toho důvodu jsme se rozhodli normalizovat vzdálenost do rozmezí ¡0;0,5) a přičíst k hodnotě rozdílnosti histogramu. Na základě součtu hodnocení se pak prochází uložené obličeje a vybere se nejlepší shoda. Pokud hodnocení nepřesáhne daný práh (nastaven na 0,6) prohlásí se obličej za shodný s dříve rozpoznaným. V opačném případě je obličej přidán do seznamu obličejů jako unikátní.

5 Rozdělení práce

Pavel Macenauer detektor objektů na GPU (zároveň diplomová práce), rozpoznání objektů a kostra aplikace

Jan Bureš sledování objektů a jejich rozpoznávání, výstupy z aplikace

6 Ovládání programu

6.1 Překlad

- Závislost na knihovnách: OpenCV 2.4.* (testováno na 2.4.9, 2.4.10), CUDA 6.5
- Přiložen projekt pro Microsoft Visual Studio (nestováno na Microsoft Visual Studio 2013).
- Nejaktuálnější verze je k nalezení na:
  https://github.com/mmaci/vutbr-fit-pov-face-tracking

6.2 Spuštění

Program se spouští přes příkazovou řádku s následujícími parametry:

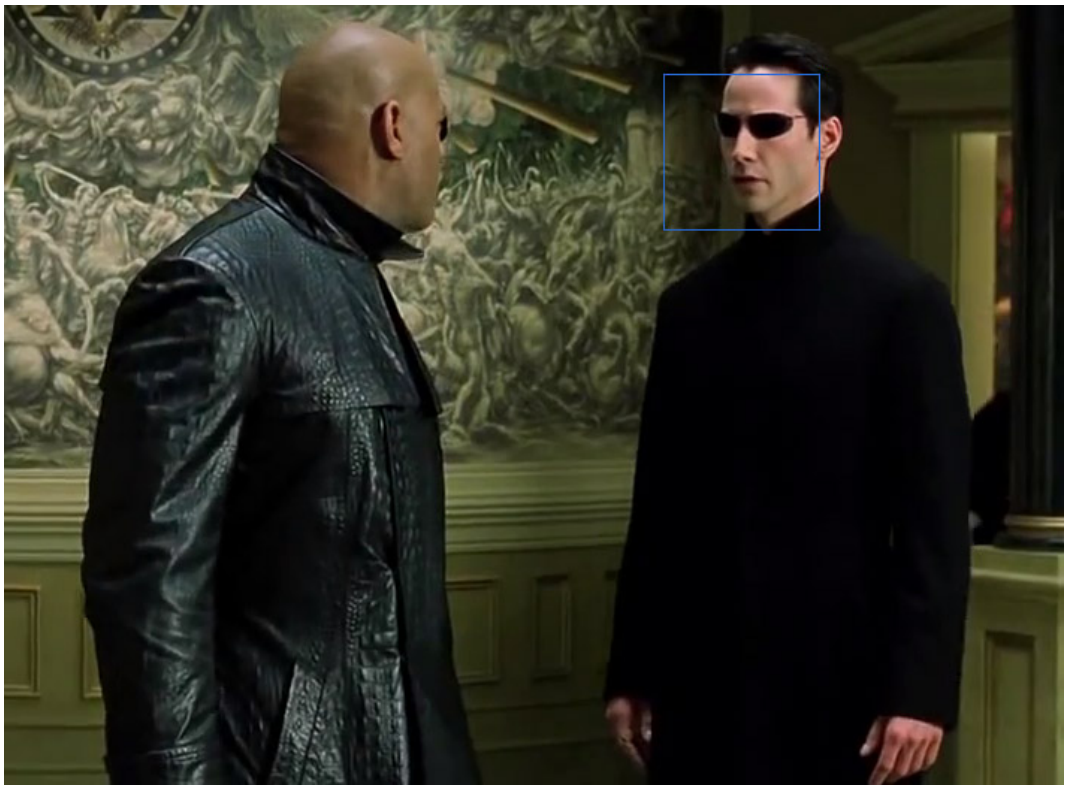
```
-ii [input file] or -di [dataset] or
-iv [input video] and -ot [output track info]
```

- **Input file** – vstupní obrázek

- **Dataset** – textový soubor obsahující seznam obrázků
- **Input video** – vstupní video
- **Output track info** – soubor pro textový výstup

Výstupem je textový soubor s výsledky trackování a výřezy jednotlivých obličejů. Vše je uloženo ve stejné složce, ve které je spuštěna aplikace.

Obrázek 3: Příklady detekovaných obličejů



Obrázek 4: Ukázka výstupu na videu

7 Závěr a vize do budoucnosti

Naimplementovaný detektor objektů je funkční a na mobilní grafické kartě Nvidia Quadro K1000M zvládá detekovat obličeje na HD obraze průměrně za 100ms, což odpovídá 10 FPS.

Na zvýšení rychlosti by bylo třeba omezit výpočet např. zredukováním počtu vyhodnocovaných stagí z 2048 na 1/2 nebo 1/4, čímž snížíme jeho přesnost, ale snížíme délku výpočtu pod úroveň FPS v běžných videích.

Výpočetní náročnost trackování obličejů je vzhledem k detektoru zanedbatelná. Pro porovnání obličeje je využit histogram oblasti, na které se obličej nachází a vzdálenost mezi současnou a předchozí pozicí obličeje. Zkoušeli jsme i porovnání na základě extrakce klíčových bodů, ale tento způsob nevykazoval dobré výsledky.

Další možné optimalizace do budoucnosti jsou:

Zpřesnění generování zmenšenin Aktuálně jsou sice generovány z texturovací paměti, ale v ní jsou uložena obrazová data jako unsigned char, což vede ke zmenšeninám pomocí alg. Nearest neighbour. Převod na float by umožnil bilineární interpolaci pomocí hardwaru.

Rozdělení vyhodnocování stagí na více kernelů a reorganizace vláken Aktuálně probíhá vyhodnocování stagí od začátku do konce v rámci jednoho kernelu. Většina vzorků je však zamítnuta již na začátku. Šlo by tak zjistit, pro které vzorky stále ještě běží výpočet a spustit nový kernel pouze pro ně.

Optimalizace zmenšenin na velikost videa Velikost videa je často standardizována na velikost a poměr stran (16:9, 4:3, HD, FULL HD, ...). Šlo by tak generovat zmenšeniny pro daný standard, kde by se využilo maximum obrazové plochy, zmenšila její velikost a výpočet by šlo urychlit rozbalením cyklů.

Optimalizace rozpoznání obličejů Do budoucna bychom chtěli přidat další možnosti porovnání obličejů, protože v současnosti je velká váha přiložena právě vzdálenosti mezi předchozí a současnou pozicí obličeje.

Literatura

[1] Grabner, H. ; Sochman, J. ; Bischof, H. ; Matas, J. Training sequential on-line boosting classifier for visual tracking.
http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4761678, 2008.

[2] Herout, A. ; Josth, R. ; Juranek, R. ; Havel, J. ; Hradis, M. ; Zemcík, P. Real-time object detection on CUDA.
http://link.springer.com/article/10.1007%2Fs11554-010-0179-0, 2011.

[3] Sochman J. ; Matas J. WaldBoost - learning for time constrained sequential detection .
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1467435, 2005.

[4] Zemcik, P. ; Juranek, R. ; Musil, P. ; Musil, M. ; Hradis, M. High performance architecture for object detection in streamed videos.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6645559, 2013.