OPENCV IN IPHONE: DETEKCIJA OBRAZOV V REALNEM ČASU

Roman Avsec, Peter Peer

GameTeam
Fakulteta za računalništvo in informatiko, Tržaška c. 25, 1000 Ljubljana
E-pošta: roman.avsec@gmail.com, peter.peer@fri.uni-lj.si
URL: http://gameteam.fri.uni-lj.si/

POVZETEK: *V članku bo predstavljenih nekaj adaptacij obstoječih in implementacij novih optimizacijskih tehnik, s katerimi lahko optimiziramo metode detekcije obraza v odprtokodni knjižnici za računalniški vid OpenCV. Na kratko bomo predstavili iPhone procesor ter iPhone razvojno okolje, vanj vključili knjižnico OpenCV in jo poskusili optimizirati do te mere, da se bo detekcija obraza izvršila v realnem času. Celoten postopek bo dokumentiran tako, da bo lahko bralec, ki ga tematika zanima, s pomočjo navodil in izvorne kode tudi samostojno vključil knjižnico v svoj projekt.*

1. UVOD

V okviru skupine GameTeam, ki deluje na Fakulteti za računalništvo in informatiko Ljubljana, smo poleti 2009 za iPhone razvili zabavno aplikacijo "15 seconds of fame" [1], ki iz posnete fotografije s pomočjo OpenCV knjižnice detektira obraz, ga izreže, poveča in transformira v abstrakten portret. Ugotovili smo, da je sicer OpenCV knjižnico relativno enostavno vključiti v iPhone razvijalno okolje, je pa detekcija zelo počasna: na sliki velikosti 400×400 slikovnih elementov se je izvajala od 6 do 10 sekund, odvisno od slike in parametrov detekcije. Zanimalo nas je, ali lahko detekcijo z uporabo OpenCV knjižnice na tej mobilni platformi optimiziramo do te mere, da čas procesiranja obraza skrajšamo v realni čas (real-time).

Na različnih virih po internetu smo naše začetne rezultate primerjali z rezultati ostalih poskusov in ugotovili, da je 6 do 10 sekund čas primerljiv s tistim, ki so ga dosegali drugi posamezniki pri detekciji. Naleteli pa smo na dva zapisa ([2], [3]), kjer poročajo, da so detekcijo na sliki dimenzij iPhone ekrana uspeli skrajšati na 100 – 300ms. Naš cilj je bil poskusiti, če in koliko se lahko temu času približamo z implementacijo predlaganih optimizacijskih tehnik in uvedbo novih.

V pomoč nam je po naključju priskočil Apple, ki je decembra 2009 odobril uporabo do takrat neuradne funkcije UIGetScreenImage(), ki preko zajema slike ekrana omogoča ob primerni pripravi tudi zajem videa s kamere. Pred to odobritvijo te funkcije je bil edini način zajema videa uporaba internih API klicev, ki bi povzročili zavrnitev aplikacije, če bi jo poskusili predstaviti na AppStore.

Video se zajema v obliki posameznih nekompresiranih sličic, celoten zajem ene sličice pa traja cca 50 ms, zato pričakujem, da bo "realni čas" pomenil precej manj kot 20 sličic na sekundo.

2. OPENCV, NAMEN IN UPORABA

OpenCV (Open Computer Vision) je odprtokodna knjižnica z začetki razvoja v podjetju Intel in združuje čez 500 optimiziranih algoritmov za pomoč pri računalniškem vidu in obdelavi slik. Zabeleženih je čez 2 milijona prenosov, uporabna pa je na področjih od splošnega procesiranja računalniških slik, geometričnih opisov, transformacij, strojnega učenja, sledenja in mnogih drugih (povzeto po [5]). Za naše potrebe bomo uporabili funkcije za detekcijo objektov, specifično obrazov, ki v OpenCV knjižnici temeljijo na klasifikaciji Haarovih značilnosti.

3. IPHONE RAZVOJNO OKOLJE

Razvojno za iPhone okolje se imenuje Xcode in teče na operacijskem sistemu OS X. Avtor okolja je podjetje Apple in ga daje na voljo brezplačno vsem, ki se na njihovi spletni strani http://developer.apple.com registrirajo kot razvijalci. Razvijalci lahko aplikacije testirajo v priloženem iPhone simulatorju, ki pa nima implementiranih funkcij, kot so: pospeškometer, GPS, kamera. Ker je za naš primer kamera ključnega pomena, smo razvoj in testiranje opravljali direktno na napravi iPhone 3G, za kar je potreben *Developer certificate*, ki je sicer pri Applu na voljo proti letnemu plačilu.

4. IPHONE PROCESOR

iPhone poganja procesor iz ARM družine. Starejša modela (2G in 3G) imata vgrajen procesor iz družine ARM 11 z ARMv6 arhutekturo, novejši model (3GS) pa procesor družine Cortex z arhitekturo ARMv7. Oba procesorja sta sicer binarno kompatibilna, med njima pa so razlike, ki jih prikazuje Tabela 1 (povzeto po [8]).

Tabela 1: razlike med procesorjema iPhone 3G in iPhone 3GS

	iPhone 2G, 3G (ARM11)	**iPhone 3GS (ARM Cortex A8)**
Proces izdelave	90nm	65nm
Velikost cevovoda	8-stopenjski	13-stopenjski
Frekvenca ure	412MHz	600MHz
Velikost predpom. L1	16KB I-Cache + 16KB D-Cache	32KB I-Cache + 32KB D-Cache
Velikost predpom. L2	brez	256KB

Vidimo lahko, da je procesor novejšega iPhone-a hitrejši in z več tako prvonivojskega kot drugonivojskega predpomnilnika. Pred starejšim procesorjem ima še veliko prednost, razširitev NEON, ki je hibridna 64/128-bitna SIMD arhitektura, razvita za pospeševanje multimedijskih vsebin, vključno z kodiranjem/dekodiranjem video vsebin, 3D grafike, prepoznavo govora, dekodiranje in sinteza zvoka ter nenazadnje za računalniško obdelavo slik [9].

Zavoljo kompatibilnosti v tej fazi ukazov NEON razširitve ne bomo uporabljali, saj bodo vsi testi delani na napravi iPhone 3G, ki jih ne podpira.

5. INTEGRACIJA OPENCV V IPHONE XCODE PROJEKT

Osnovni jezik iPhone razvojnega okolja je programski jezik Objective-C, ki je subset C-ja, zato ima polno podporo za vključevanje C knjižnic ter prav tako polno podporo za vključevanje C++ izvorne kode. OpenCV tako lahko vključimo v razvojno okolje na več načinov, a predvsem zaradi možnosti, da lahko sami prevajamo in eksperimentiramo z izvorno kodo knjižnice, smo se odločili za postopek, ki je opisan na spletni strani [4] ter uporabo OpenCV 2.0 v iPhone SDK 3.1.2. Po končanem postopku, ki ga v članku sicer ne bomo opisovali podrobneje, se dva nivoja višje od trenutne lokacije ("../../") nahajata mapi s prevedeno knjižnico in potrebnimi header datotekami – opencv_simulator/ ter opencv_device/. Obe mapi prekopiramo v korensko mapo našega Xcode projekta.

Sedaj še v Xcode projektu nastavimo poti, kjer se nahajajo statične knjižnice in pripadajoče header datoteke. To storimo v meniju Project > Edit Active Target in v razdelku Linking > Other Linker Flags. S pomočjo ikone v levem spodnjem kotu () izberemo Add Build Setting Condition in dodamo dva pogoja:

- Any iPhone OS Device / Any Architecture
- Any iPhone OS Simulator / Any Architecture

V pogoj "Any iPhone OS Device / Any Architecture" bomo z dvojnim klikom na polje vrednosti ter nato s klikom na ikono + dodali 4 zastavice in sicer:

- -lstdc++
- -lz
- $(SRCROOT)/opencv_device/lib/libcv.a
- $(SRCROOT)/opencv_device/lib/libcxcore.a

V pogoj "Any iPhone OS Simulator" dodamo enake štiri zastavice, le pri zadnjih dveh mapo "opencv_device" ustrezno spremenimo na "opencv_simulator". Vse, kar preostane je, da v naš projekt vključimo sledečo header datoteko:

#import <opencv/cv.h>

6. OPTIMIZACIJA OPENCV ZA IPHONE PLATFORMO

6.1 Optimizacija knjižnice

S pomočjo orodja Instruments, ki je sestavni del okolja Xcode, smo poskusili ugotoviti, kje aplikacija porabi največ časa. Slika 1 prikazuje funkcije, v katerih se aplikacija zadržuje najdlje; po Amdahlovem zakonu bomo največje pohitritve pridobili, če uspemo pohitriti prav te funkcije.

Self Run %	Running %	ms Running ▾	Library	Symbol Name
35,6	35,6	7500,0	RT_OpenCV	ARMcvRunHaarClassifierCascade
35,5	35,5	7480,0	RT_OpenCV	ARMcvHaarDetectObjects
14,8	14,8	3120,0	RT_OpenCV	ARMcvSetImagesForHaarClassifierCascade
2	2	430,0	libSystem.B.dylib	mach_msg_trap
1,7	1,7	370,0	libSystem.B.dylib	lrintl
1,4	1,4	310,0	RT_OpenCV	void cv::pyrDown_<cv::FixPtCast<unsigned char, 8>, cv::NoVec>(cv::Mat const&, cv::Mat&)
1,1	1,1	250,0	CoreGraphics	CGSBlendARGB8888toRGBA8888
0,9	0,9	190,0	RT_OpenCV	icvBGRx2BGR_8u_CnC3R(unsigned char const*, int, unsigned char*, int, CvSize, int, int)
0,7	0,7	160,0	CoreGraphics	CGSFillDRAM8by1
0,7	0,7	150,0	RT_OpenCV	dyld_stub___ledf2vfp
0,6	0,6	140,0	RT_OpenCV	dyld_stub___floatsisfvfp
0,5	0,5	110,0	libSystem.B.dylib	sqrt
0,5	0,5	110,0	RT_OpenCV	dyld_stub___extendsfdf2vfp
0,4	0,4	100,0	RT_OpenCV	dyld_stub___muldf3vfp
0,3	0,3	70,0	libSystem.B.dylib	pthread_setspecific
0,3	0,3	70,0	RT_OpenCV	dyld_stub___mulsf3vfp

Slika 1: prikaz funkcij, kjer se aplikacija zadržuje najdlje

Vidimo, da se aplikacija najdlje zadržuje v funkcijah cvRunHaarClassifierCascade ter cvSetImagesForHaarClassifierCascade. Ti dve funkciji sta del implementacije Viola-Jones algoritma za detekcijo objektov. Viola-Jones algoritem uporablja t.i. klasifikatorje, ki so pridobljeni s strojnim učenjem na primerih objektov. Pozitivni klasifikatorji s primeri posameznega objekta (obraz, avto, roka, ...) so po učenju skrčeni na enake dimenzije, največkrat 20×20 slikovnih elementov. Ti klasifikatorji so nato uporabljeni na vhodni sliki na področjih enake velikosti, kot so bili uporabljeni pri učenju. Klasifikatorji se lahko enostavo povečujejo in pomanjšujejo, zato se za detekcijo objekta v določeni interesni regiji (Region of Interest – ROI) velikost iskalnega okna povečuje od začetne izbrane velikosti do velikosti celotne regije [5]. Takšen način je zelo uporaben za iskanje objekta neznane velikosti v določeni regiji, hkrati pa je tudi zelo potraten, saj se mora za vse velikosti iskalnega okna in vse pozicije tega okna v interesni regiji izvesti funkcija cvRunHaarClassifierCascade. To lahko v praksi pomeni tudi do nekaj tisoč klicev funkcije za posamezno sliko.

Pri pregledu teh funkcij smo ugotovili, da je večina aritmetika detekcije v plavajoči vejici, kar je šibka točka ARMv6 arhitekture. Za optmizacijo same knjižnice smo za zgled vzeli primer [1] in izbrali naslednja področja:

- faktor uteži (weight) v strukturi CvHidHaarFeature je bil pretvorjen iz float v integer
- prag tolerance (treshold) v strukturi CvHidHaarTreeNode je bil pretvorjen iz float v integer

- izračuni s pragom tolerance, utežmi in faktorjem normalne variance v funkciji cvRunHaarClassifierCascade so bili iz aritmetike plavajoče vejice pretvorjeni v celoštevilsko aritmetiko
- za izračun celoštevilčnega kvadradnega korena v funkciji cvRunHaarClassifierCascade je bil uporabljen algoritem [10]
- izračun uteži v funkciji cvSetImagesForHaarClassifierCascade je bil iz aritmetike plavajoče vejice pretvorjen v celoštevilsko aritmetiko

Na račun teh omejitev bo sicer natančnost prepoznave slabša, a po vmesnih testih še vedno dovolj dobra za večino aplikacij.

6.2. Otimizacija postopkov

Pri detekciji v realnem času delamo načeloma z video vhodom in se zato lahko praviloma zanašamo na dva faktorja, ki nam pomagata odločitvi optimizacije:

- velika verjetnost je, da pozicija obraza ne bo veliko odstopala glede na prejšnjo lokacijo;
- velika verjetnost je, da velikost obraza ne bo veliko odstopala glede na prejšnjo velikost.

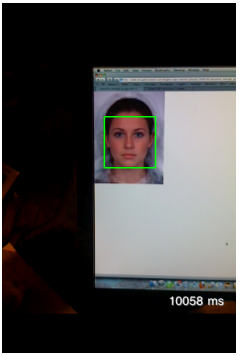
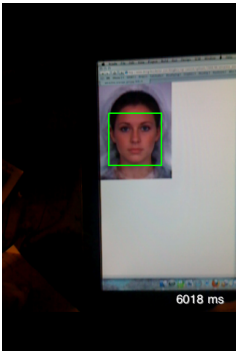
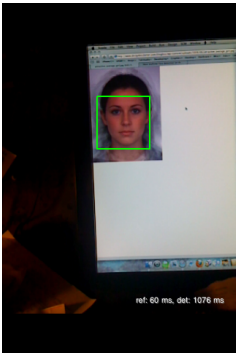
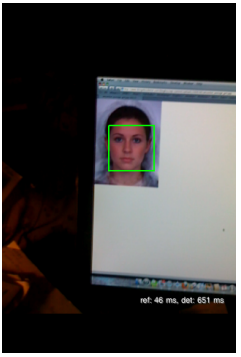
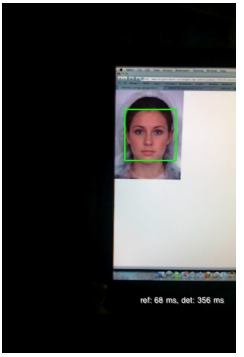
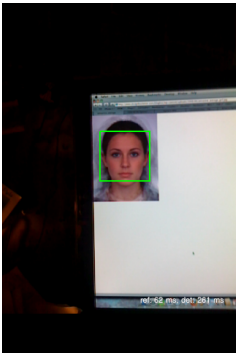
V duhu teh predvidevanj smo postopek priredili tako, da smo uporabili podatke prejšnje detekcije in sicer:

1. interesno področje (Region of Interest - ROI) sedaj izračunamo glede na pozicijo obraza v prejšnji detekciji, zraven pa vključimo še nekaj točk tolerance v vsaki smeri, da dovolimo odstopanja od prejšnje pozicije zaradi premikov. Odvisno od velikosti obraza in časa posamezne detekcije bi to pomenilo, da se objekt v oddaljenosti 3m lahko premika s hitrostjo cca 1m/s, pa bo algoritmu še vedno uspela hitra detekcija s pomočjo predhodne regije;
2. začetna velikost iskalnega okna ni konstantna, temveč jo v vsaki iteraciji izračunamo znova iz velikosti obraza prejšnje detekcije in vključimo nekaj točk za odstopanje.

6.3. Preverjanje vpliva posamezne tehnike (benchmark)

Čas posamezne detekcije je odvisen od več faktorjev – velikosti vhodne slike, velikosti obraza, pozicije obraza ter parametrov detekcije, kot so velikost iskalnega okna, faktor povečevanja iskalnega okna itd. Za potrebe primerjave smo izbrali generični obraz v velikosti cca 100×100 slikovnih elementov na sredini slike. Tabela 2 prikazuje čas, ki je bil potreben za detekcijo tega obraza v vsaki stopnji optimizacije.

Tabela 2: prikaz časovne zahtevnosti detekcije po posameznih stopnjah optimizacije

 Brez optimizacij: 10s	 Enkratna incializacija spremenljivk in deljen cache: 6s	 Uporaba pozicije obraza prejšnje detekcije: 1s
 Uporaba velikosti iskalnega okna prejšnje detekcije: 650ms	 Uporaba aritmetike s celimi števili: 350ms	 Nastavitev parametrov detekcije: 260ms

Vidimo lahko, da sta imeli največji doprinos enkratna incializacija deljenih spremenvljik ter uporaba pozicije obraza prejšnje detekcije. Z drugo tehniko smo uspeli ROI (Region of Interest) zmanjšati s celotnega ekrana na le del originalnega področja, kar je precej zmanjšalo število klicev funkcije cvRunHaarClassifierCascade, za katero smo ugotovili, da je funkcija, v kateri aplikacija preživi največ časa. Velik odstotek pohitritve je prispeval tudi prehod na aritmetiko s celimi števili, saj ARMv6 arhitektura v osnovi ne podpira hitrega računanja s plavajočo vejico.

Pod optimizacijo parametrov detekcije je zajeta izbira datoteke s haar-ovimi kaskadami, začetna velikost iskalnega okna, faktor povečave iskalnega okna ter metoda detekcije. Po

empiričnih ocenah zadovoljivih rezultatov so bili izbrani parametri za prikazani primer naslednji:

- datoteka s haar-ovimi kaskadami: haarcascade_frontalface_alt.xml
- začetna velikost iskalnega okna: 30×30 slikovnih elementov
- faktor povečave iskalnega okna: 1.4
- metoda detekcije: CV_HAAR_FIND_BIGGEST_OBJECT in CV_HAAR_DO_ROUGH_SEARCH

Pri izbrani metodi detekcije CV_HAAR_FIND_BIGGEST_OBJECT funkcija najde in vrne le največji objekt v sceni. Z metodo CV_HAAR_DO_ROUGH_SEARCH, ki se uporablja le s prejšnjo metodo iskanja največjega objekta, funkcija ne išče več manjših kandidatov za obraz po tem, ko že enkrat najde primernega kandidata. Ker je bila uporabljena metoda iskanja enega objekta, po mnogih priporočilih nismo uporabil metode CV_HAAR_DO_CANNY_PRUNNING. Le-ta z uporabo detekcije robov ovrže nekatere segmente, kar načeloma pohitri detekcijo, razen v primeru metode iskanja enega objekta.

Z uporabljenimi parametri se sicer zmanjša zanesljivost detekcije, hitrost pa, kot je razvidno iz zgornje tabele, ustrezno poveča.

7. MOŽNOSTI IMPLEMENTACIJE

Detekcija obrazov v realnem času s knjižnico OpenCV je na iPhone-u že skoraj realnost. Kljub temu pa bo zaradi trenutnega stanja – bolj groba detekcija, počasnejše delovanje, kamera le na zadnjem delu telefona – verjetno doma bolj v zabavnih aplikacijah. Praktični primeri uporabe bi lahko zajemali fotoaparat z detekcijo nasmeškov, prepoznava gest, portretni način fotoaparata z avto fokusom na obraz (če Apple objavi API za avtofokus), aplikacije z nadgrajeno resničnostjo (augmented reality) in drugi. Če bo Apple kdaj dodal kamero na sprednji del telefona, bi bili lahko primer uporabe stabilizacija slike pri video konferenci, nadzor aplikacije s premiki glave in podobno.

Celotna koda primera uporabe sledenja obrazu, ki nadgradi resničnost z zelenim kvadratkom okoli osebe, se nahaja na spletnem naslovu:

http://gameteam.fri.uni-lj.si/code/rt_opencv_iphone.zip

8. SKLEP

Z uporabo relativno enostavnih tehnik optimizacije smo časovno zahtevnost detekcije obrazov na iPhone 3G platformi zmanjšali za faktor skoraj 40. Prvotnemu cilju pa smo se s tem vseeno le približali, saj bi si za detekcijo v realnem času želel vsaj 10 sličic na sekundo, kar pomeni zmanjšanje časa na 100 ms.

Kot možnost nadaljne optimizacije se ponuja uporaba SIMD razširitve ARMv7 arhitekture, imenovana NEON, ki podpira do izvajanje 16 ukazov hkrati. Takšna

optimizacija bi zahtevala specifično vektorizacijo parametrov, zato presega okvirje tega članka, daje pa možnost za nove izzive. V izvorni kodi verzije 2.0 OpenCV knjižnice je več funkcij že optimiziranih za Intelove SIMD razširitve SSE1 do SSE4, zato bi lahko za navdih in primer lahko uporabili kar to implementacijo. NEON razširitev pa žal ni prisotna na ARMv6 arhitekturi, kar v praksi pomeni le optimizacijo na iPhone 3GS. iPhone 2G in 3G sicer vsebujeta Vector Floating Point (VFP) enoto, za katero obstaja knjižnica vfpmathlibrary [7], ki bi jo bilo mogoče uporabiti za paralelizacijo izračunov in dodatne pohitritve.

LITERATURA

1. B. Batagelj, F. Solina, P. Peer (2004), 15 seconds of fame - an interactive, computer-vision based art installation, ACM International Conference on Multimedia, pp. 764-765, New York, NY, USA

2. http://www.morethantechnical.com/2009/08/09/near-realtime-face-detection-on-the-iphone-w-opencv-port-wcodevideo
   Near realtime face detection on the iPhone w/ OpenCV port

3. http://www.computer-vision-software.com/blog/2009/06/fastfurious-face-detection-with-opencv
   Fast & Furious face detection with OpenCV

4. http://niw.at/articles/2009/03/14/using-opencv-on-iphone/en
   Using OpenCV on iPhone

5. http://opencv.willowgarage.com/wiki/FullOpenCVWiki
   FullOpenCVWiki

6. G. Bradski, A. Kaehler (2008), Learning OpenCV: Computer Vision with the OpenCV Library, O'Reilly

7. http://code.google.com/p/vfpmathlibrary
   VFP based math library for the iPhone / iPod touch

8. http://www.anandtech.com/printarticle.aspx?i=3579
   iPhone 3GS Hardware Exposed & Analyzed

9. http://www.arm.com/products/CPUs/architecture.html
   ARM Processor Instruction Set Architecture

10. http://www.codecodex.com/wiki/index.php?title=Calculate_an_integer_square_root
    Calculate an integer square root