

Tema 3. Image processing

Responsabil: Călin Cruceru

Deadline: 23:59:59 20.01.2015


Scopul temei:

- utilizarea structurilor de date
- lucrul cu fișiere, atât text, cât și binare
- abordarea cu pași mici a unei probleme mai complexe

Actualizări:

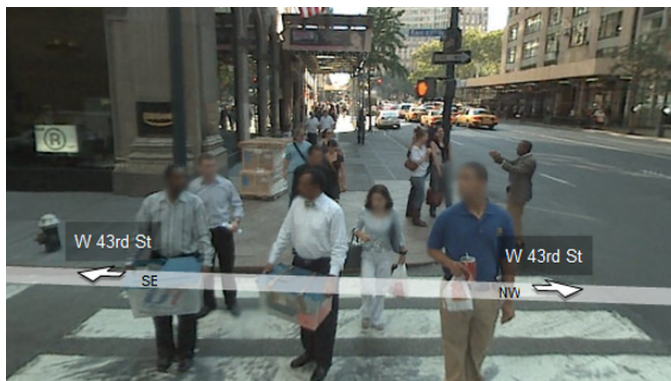
- **[21/12/2014 - 16:30]:** adăugat checker și 2 teste, modificat formatul datelor de intrare, modificat restricțiile (**2000×2000** ⇒ **1500×1500**), adăgat observații.
- **[21/12/2014 - 19:20]:** uploadat tema pe vmchecker; modificat modul în care se calculează punctajul în checker - **bc** nu mergea pe vmchecker.
- **[21/12/2014 - 22:05]:** corectat typo în numele imaginii de input. Este **input.bmp** și nu **image.bmp**. Corectat fișierele **ref** de crop - era greșit câmpul **bfSize** și **biImageSize** din **File Header**.
- **[22/12/2014 - 18:50]:** corectat fișierele de input care aveau câmpul **bfSize** greșit. Acest lucru a fost cauzat de faptul că inițial pozele avuseseră dimensiunea 3000056, unde 2 octeți au fost puși automat de programul în care a fost creat poza, astfel încât dimensiunea totală să fie multiplu de 4. Pentru că nu am considerat relevant acest aspect, am trunchiat imaginea fără cei 2 bytes însă am uitat să modific câmpul **bfSize**. Cel mai probabil implementările de până acum nu vor fi afectate de acest aspect.
- **[25/12/2014 - 11:35]:** arhiva pentru vmchecker va trebui să conțină și declararea structurilor pentru headerele BMP, fie că folosiți headerul **bmp_header.h**, fie că l-ați redenumit, fie că le-ați pus în fișierele **.c**, vor trebui incluse în arhivă. Inițial puseseam eu pe vmchecker headerul însă apăreau probleme dacă cineva îl includea în arhivă și cred că e mai intuitiv așa.


Introducere

Procesarea de imagini este un domeniu foarte vast în computer science, aflat în strânsă legătură cu alte domenii precum  computer graphics și  computer vision.

Cu toții ați auzit de **face detection** și de device-urile sau aplicațiile care îl integrează printre feature-urile lor și pe acesta. Câteva exemple sunt: focalizarea, înainte de a face o poză, pe smartphone-uri/camere foto, facebook, care îți recomandă să îți tăgăiești prietenii din poză, marcându-le fața într-un chenar, etc.

Ne propunem să simulăm, la un nivel simplificat, algoritmul care stă în spatele **face detection**-ului, având ca use case **protejarea identității**, prin **blur**-area fețelor detectate (puteți lua ca exemplu Street View-ul de la Google Maps). De ce simplificat? Deoarece complexitatea algoritmilor care fac asta ar depăși scopul acestui curs și ar fi, de asemenea, și foarte greu de automatizat testarea unei astfel de procesări.



Imagine preluată de pe  cnet.com

Structura formatului BMP

Vom lucra cu fișiere  BMP, deci, cu **fișiere binare**.

O imagine BMP are următoarea structură:

- un **File Header** care are următoarele câmpuri:
 1. **signature** – 2 octeți - literele 'B' și 'M' în ASCII;
 2. **file size** – 4 octeți - dimensiunea întregului fișier;
 3. **reserved** – 4 octeți - nefolosit;
 4. **offset** – 4 octeți - offsetul de la începutul fișierului până la începutul bitmap-ului, adică al matricii de pixeli.
- un **Info Header** care poate avea structuri diferite, însă noi vom lucra cu cel care se numește **BITMAPINFOHEADER**. Are următoarele câmpuri:
 1. **size** – 4 octeți - dimensiunea Info Header-ului, care are o valoare fixă, 40;
 2. **width** – 4 octeți - lățimea matricii de pixeli (numărul de coloane);
 3. **height** – 4 octeți - înălțimea matricii de pixeli (numărul de rânduri);
 4. **planes** – 2 octeți - setat la valoarea fixă 1;
 5. **bit count** – 2 octeți - numărul de biți per pixel. În cazul nostru va avea mereu valoarea 24, adică reprezentăm fiecare pixel pe 3 octeți, adică cele 3 canale, RGB;
 6. **compression** – 4 octeți - tipul de compresie. Acest câmp va fi 0;
 7. **image size** – 4 octeți - se referă la dimensiunea matricii de pixeli, inclusiv padding-ul adăugat (vedeți mai jos);

Resurse generale

- Regulament: seria CA
- Regulament: seria CB/CD
- Calendar
- Catalog laborator
- Debugging
- Bune practici și erori frecvente în C
- Coding style - CA
- VM - CA
- Test practic - CA
- Checker laborator CB/CD

Cursuri

Continutul Tematic

Laboratoare

01. Unelte de programare
02. Tipuri de date. Operatori.
03. Instrucțiunile limbajului C
04. Funcții
05. Tablouri. Particularizare - vectori
06. Matrice. Operații cu matrice
07. Optimizarea programelor folosind operații pe biți
08. Pointeri. Abordarea lucrului cu tablouri folosind pointeri
09. Alocarea dinamică a memoriei. Aplicații folosind tablouri și matrice
10. Prelucrarea șirurilor de caractere. Funcții. Aplicații
11. Structuri. Uniuni. Aplicație: Matrice rare
12. Operații cu fișiere. Aplicații folosind fișiere.
13. Parametrii liniei de comandă. Preprocesorul. Funcții cu număr variabil de parametri
14. Recapitulare

Teme de casă (general)

- Indicații generale

Teme de casă: seria CA

Teme CB/CD (2019-2020)

- Tema 1

Teme CB/CD

- Tema 1
- Tema 2
- Tema 3
- Tema 4

Table of Contents

- Tema 3. Image processing
 - Introducere
 - Structura formatului BMP
 - Cerințe
 - Formatul datelor de intrare și de ieșire
 - Observații
 - Mențiuni
 - Testare locală

8. **x pixels per meter** – 4 octeți – se referă la rezoluția de printare. Pentru a simplifica puțin tema, veți seta acest câmp pe 0. Nu o să printăm imaginile :).
 9. **y pixels per meter** – la fel ca mai sus;
 10. **colors used** – numărul de culori din paleta de culori. Aceasta este o secțiune care va lipsi din imaginile noastre BMP, deoarece ea se află în general imediat după **Info Header** însă doar pentru imaginile care au câmpul **bit count** mai mic sau egal cu 8. Prin urmare, câmpul va fi setat pe 0;
 11. **colors important** – numărul de culori importante. Va fi, de asemenea, setat pe 0, ceea ce înseamnă că toate culorile sunt importante.
- **BitMap**-ul, care este matricea de pixeli și care ocupă cea mai mare parte din fișier. Trei lucruri trebuie menționate despre aceasta:
 1. pixelii propriu-ziși se află într-o matrice de dimensiune **height x width**, însă ea poate avea o dimensiune mai mare de atât din cauza **paddingului**. Acest padding este adăugat la sfârșitul fiecărei linii astfel încât fiecare linie să înceapă de la o adresă (offset față de începutul fișierului) multiplu de 4. Mare atenție la citire, pentru că acest padding trebuie **ignorat** (fseek). De asemenea, la scriere va trebui să puneți **explicit** valoarea 0 pe toți octeții de padding.
 2. este **răsturnată**, ceea ce înseamnă că prima linie în matrice conține de fapt pixelii din extremitatea de jos a imaginii. Vedeți exemplul de mai jos;
 3. canalele pentru fiecare pixel sunt în ordinea BGR (**B**lue **G**reen **R**ed).

Header-ele pe care le puteți folosi în implementare se află în scheletul de cod asociat temei.



ATENȚIE în special la cea de-a 3a cerință, pentru care va trebui să faceți mai multe modificări în header-ele asociate imaginii input: **file size, width, height, image size**. La acestea se adaugă și câmpurile **x pixels per meter** și **y pixels per meter** care se vor seta pe 0.



Urmățiți cu foarte mare atenție exemplul de [aici](#) și încercați să înțelegeți cum e reprezentată o imagine BMP **înainte** de a începe implementarea. Dacă e ceva neclar, puteți întreba oricând pe forum.

Cerințe

Cum simulăm de fapt procedeu de **face detection**? Știm că în format BMP, o imagine nu este altceva decât o matrice de pixeli. Astfel, a identifica fața unei persoane într-o imagine, la un nivel simplificat, ar însemna identificarea zonei din imagine care are toți pixelii într-un interval dat de un pixel referință și un pixel offset (dați ca input în această temă). O **zonă**, în acest caz, e definită ca o mulțime de pixeli adiacenți din imagine (sau poziții vecine în matrice). Această zonă nu este necesar să aibă o formă regulată.

Observați cum simplificăm modul în care funcționează **face detection**-ul în aplicațiile reale, prin faptul că oferim ca input valorile RGB ale unui pixel. Acesta este și motivul pentru care nu vom putea folosi ca input "poze reale", deoarece culoarea pielii diferă, în realitate, de luminozitatea externă, calitatea camerei foto, etc. Prin urmare, nu vă așteptați la efecte vizuale impresionante.

O să îi spunem **cluster** zonei din imagine despre care vorbeam mai sus și o să numim un pixel **valid** dacă se află în intervalul dat de pixelul referință și cel offset. Prin urmare, un pixel dat prin valorile sale RGB (**Rv, Gv, Bv**), este **valid**, dacă și numai dacă, dându-se 2 pixeli numiți **referință (Rr, Gr, Br)** și **offset (Ro, Go, Bo)**, următoarele inegalități sunt adevărate.

```
Rr - Ro <= Rv <= Rr + Ro
Gr - Go <= Gv <= Gr + Go
Br - Bo <= Bv <= Br + Bo
```

Luăm ca exemplu imaginea dată prin matricea de mai jos, care are 30 pixeli (am folosit parantezele pentru a delimita pixelii).

```
(255 0 0) (225 0 0) (240 0 0) (0 0 0) (0 0 0) (0 0 0) 2 bytes padding
(240 0 0) (225 0 0) (250 0 0) (245 0 0) (0 0 0) (0 0 0) 2 bytes padding
(240 0 0) (0 0 0) (0 0 0) (0 0 0) (0 0 0) (255 0 0) (0 0 0) 2 bytes padding
(0 0 0) (0 0 0) (0 0 0) (240 0 0) (240 0 0) (240 0 0) 2 bytes padding
(0 0 0) (0 0 0) (0 0 0) (0 0 0) (0 0 0) (0 0 0) 2 bytes padding
```

Dacă pixelul referință ar avea valoarea **(240, 0, 0)**, iar cel offset **(15, 0, 0)**, atunci numărul de cluster ar fi **2** (vezi exemplul de mai jos). Dacă, în schimb, pixelul referință ar avea valoarea **(100, 15, 1)**, iar cel offset ar rămâne la fel, numărul de cluster ar fi **0**. În cele din urmă, dacă pixelul referință ar fi **(240, 0, 0)**, iar cel offset **(0, 0, 0)**, numărul de cluster ar fi **3**.

Cerința 1 – Cluster (30 puncte)

Să se afișeze în ordine crescătoare dimensiunile clusterelor de pixeli care se află în intervalul determinat de pixelii referință și offset dați ca input. Se vor considera doar clusterelor în care se găsesc **cel puțin P*Height*Width** pixeli, unde **P** este un procent dat ca input. Cu alte cuvinte, nu ne interesează acele cluster care au *prea puțini* pixeli.

Exemplul 1: În cazul imaginii date prin matricea de mai sus, cu cei 2 pixeli având valorile **(240, 0, 0)** și **(15, 0, 0)** și cu **P = 0.1**, outputul ar trebui să fie:

```
4 8
```

Cele 2 cluster evidențiate, în matricea de mai sus, sunt (o să ignorăm paddingul în exemplele următoare):

```
(255 0 0) (225 0 0) (240 0 0) (0 0 0) (0 0 0) (0 0 0)
(240 0 0) (225 0 0) (250 0 0) (245 0 0) (0 0 0) (0 0 0)
(240 0 0) (0 0 0) (0 0 0) (0 0 0) (0 0 0) (255 0 0) (0 0 0)
(0 0 0) (0 0 0) (0 0 0) (240 0 0) (240 0 0) (240 0 0)
(0 0 0) (0 0 0) (0 0 0) (0 0 0) (0 0 0) (0 0 0)
```



Observați cum 2 pixeli nu fac parte din același cluster dacă se învecinează pe diagonală.

Exemplul 2: În schimb, dacă am schimba pixelul offset în **(10, 0, 0)**, atunci outputul ar trebui să fie:

3 3

Cele 2 clustere evidențiate cu roșu și albastru, din matricea de mai sus, sunt:

(255 0 0)	(225 0 0)	(240 0 0)	(0 0 0)	(0 0 0)	(0 0 0)
(240 0 0)	(225 0 0)	(250 0 0)	(245 0 0)	(0 0 0)	(0 0 0)
(240 0 0)	(0 0 0)	(0 0 0)	(0 0 0)	(255 0 0)	(0 0 0)
(0 0 0)	(0 0 0)	(0 0 0)	(240 0 0)	(240 0 0)	(240 0 0)
(0 0 0)	(0 0 0)	(0 0 0)	(0 0 0)	(0 0 0)	(0 0 0)



Observați cum nu ne interesează clusterul care are 2 pixeli (marcat cu verde), deoarece $P * Height * Width = 0.1 * 5 * 6 = 3 > 2$.

Cerința 2 - Blur (30 puncte)

Se cere **blur**-area clusterelor identificate la cerința anterioară. Efectul de blur presupune înlocuirea fiecărui pixel care face parte dintr-un cluster cu media aritmetică a vecinilor săi (stânga, dreapta, jos, sus).



Atenție! Aceeași regulă se aplică și pentru pixelii care se află la "marginea" clusterului, chiar dacă asta presupune însumarea a unuia, până la 3, pixeli care se află în afara clusterului.

Pentru ca efectul să fie vizibil, acest proces se va repeta de **100** ori.

Exemplu: Considerăm inputul de la exemplul 2, cerința 1. După o iterație, matricea de pixeli ar trebui să fie următoarea:

(255 0 0)	(225 0 0)	(158 0 0)	(0 0 0)	(0 0 0)	(0 0 0)
(240 0 0)	(225 0 0)	(177 0 0)	(62 0 0)	(0 0 0)	(0 0 0)
(240 0 0)	(0 0 0)	(0 0 0)	(0 0 0)	(255 0 0)	(0 0 0)
(0 0 0)	(0 0 0)	(0 0 0)	(60 0 0)	(183 0 0)	(80 0 0)
(0 0 0)	(0 0 0)	(0 0 0)	(0 0 0)	(0 0 0)	(0 0 0)

Procedeul se continuă pentru încă 99 iterații.



Observați că aplicăm efectul de blur doar pe clusterele identificate la cerința 1, exemplul 2.

Cerința 3 – Crop (40 puncte)

Să se decupeze clusterele identificate la prima cerință. **Neblurate!**

Ne dorim, însă, ca imaginile rezultate să aibă o formă regulată, de dreptunghi. Pentru asta, ceea ce va trebui să "tăiați" pentru fiecare cluster este **cel mai mic dreptunghi** care îl cuprinde.

Exemplu: Luând ca input matricea de pixeli de la exemplul 1, prima cerință, noile imagini care conțin clusterele identificate vor avea următoarele matrici de pixeli:

Imaginea 1:

(255 0 0)	(225 0 0)	(240 0 0)	(0 0 0)
(240 0 0)	(225 0 0)	(250 0 0)	(245 0 0)
(240 0 0)	(0 0 0)	(0 0 0)	(0 0 0)

Imaginea 2:

(0 0 0)	(255 0 0)	(0 0 0)
(240 0 0)	(240 0 0)	(240 0 0)

Formatul datelor de intrare și de ieșire

Imaginea procesată se va numi **input.bmp**. Aceasta nu se va mai da de la tastatură și nu se va citi nici din fișier! Celelalte date de intrare se vor citi din fișierul **input.txt**, care va avea următoarea structură:

```
240 0 0
15 0 0
0.1
```

- **240 0 0** este pixelul **referință**;
- **15 0 0** este pixelul **offset**;
- **0.1** este procentul **P**.

În ceea ce privește datele de ieșire:

- outputul **cerinței 1** se va afișa în fișierul **output.txt**;
- pentru **cerința 2**, imaginea output se va numi **output_blur.bmp**;
- pentru **cerința 3**, numele imaginilor rezultate vor fi **output_crop1.bmp**, **output_crop2.bmp**, ..., **output_crop[n].bmp**, unde **[n]** este numărul de clustere identificate la cerința 1.

Observații

- Nu folosiți variabile globale;
- Fiți consistenti în ceea ce privește **coding style-ul**;
- Modularizați pe cât de mult posibil codul – folosiți **funcții** și **struct-uri**;

- Aveți mare atenție când lucrați cu formatul BMP. Citiți bine descrierea formatului și a header-elor;
- Aveți mare atenție în special la cerința 3, când scrieți header-ele specifice BMP în imaginile output, deoarece câmpurile care se referă la dimensiunea fișierului/matricii de pixeli se vor modifica.
- Încercați să aveți o abordare sistematică și cu pași mici dar siguri. Testați fiecare pas. Spre exemplu, înainte de a implementa cerințele propriu-zise, **testați** dacă ați citit corect matricea de pixeli. Apoi, pentru cerința 1 gândiți-vă la algoritmul prin care determinați **mulțimile** de pixeli adjuncți și asigurați-vă că funcționează corect înainte de a trece mai departe, etc.
- Această temă este și un bun exercițiu pentru a vă dezvolta skillurile de debugging. Pentru această temă, pe lângă binele cunoscute metode de debug **printf** și **gdb**, veți fi nevoiți să comparați fișiere binare (imagini în format BMP). Pentru asta aveți la dispoziție atât programul **diff_bmp** din scheletul de cod cât și utilitare precum **hexdump**, **diff**, **:%!xxd** în vim, etc. Vă sfătuim să le folosiți pe cât de mult posibil. Nu uploadați fișierul vostru de output pe forum de îndată ce checkerul vă spune că ați greșit. Cel mai probabil veți primi un răspuns care vă va trimite la descrierea formatului BMP sau la anumite atenționări din enunțul temei.

Mențiuni

- Dimensiunile imaginilor nu vor depăși **1500x1500**;
- Se garantează faptul că inputul este corect, adică imaginea există și are formatul BMP, valorile RGB pentru **pixelul referință** și pentru **pixelul offset** se încadrează într-un unsigned char, iar procentul **P** este un real în intervalul **(0, 1]**;
- Se garantează faptul că $P * \text{Height} * \text{Width}$ este un număr întreg. Aveți totuși grijă la casturile implicite;
- Tema va fi uploadată pe vmchecker și va fi corectată cu testerul local pus la dispoziție **în curând** la resurse;
- Arhiva pentru vmchecker va conține direct în rădăcina arhivei:
 1. **Makefile**, cu cel puțin 2 targeturi, **build** și **clean**;
 2. **sursele** voastre, adică fișierele .c și .h. **Inclusiv headerul bmp_header.h sau sub orice altă denumire îl folosiți**;
 3. **README**, în care trebuie să dați detalii despre implementare, de ce ați ales să rezolvați într-un anumit fel, etc.



Toate soluțiile vor fi verificate folosind o unealtă de detectare a plagiatului. În cazul detectării unui astfel de caz, atât plagiatorul cât și autorul original vor primi punctaj 0 pe temă și nu li se va permite intrarea în examen.

Testare locală

Puteti descărca de [aici](#) scheletul de cod, precum și scriptul de testare și testele. Arhiva conține:

- **bmp_header.h**, headerul care conține declarațiile struct-urilor pe care le veți folosi în citirea unui fișier BMP.
- **diff_bmp.c**, fișier **C** care face compararea a 2 imagini BMP cu o toleranță **tol** dată ca argument. Nu veți avea nevoie de el explicit decât dacă vreți să faceți debug. El este apelat de scriptul test.sh.
- **Makefile**, un exemplu de Makefile, în care va trebui să adăugați doar sursele voastre.
- **test.sh**, scriptul de testare, care poate fi rulat în următoarele moduri:
 1. ./test.sh sau ./test.sh all- va rula toate testele;
 2. ./test.sh **n** - unde n este un număr între 1 și *numărul de teste* (2 în prezent, însă se vor adăuga teste noi în zilele următoare) - va rula doar testul **n**.
- **input**, director care conține fișierele de intrare.
- **ref**, director care conține fișierele de ieșire referință.

programare/teme_2014b/tema3.txt · Last modified: 2014/12/25 11:34 by calin.cruceru

Old revisions

Media Manager Back to top