Mitran Ana-Theodora
331ABb

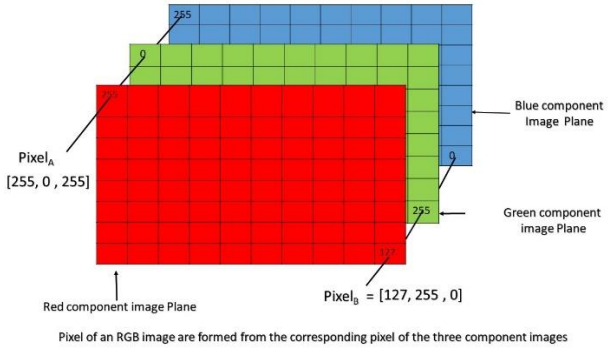
Edge Extraction (Detection)

Cuprins

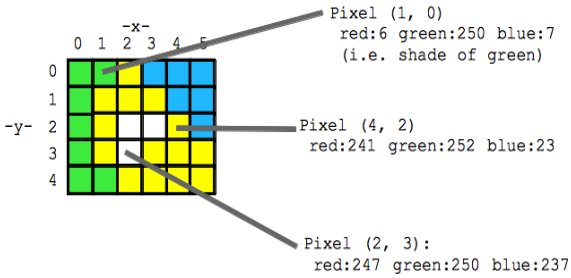
Introducere si fundament teoretic

Imaginile sunt formate din pixeli dispusi sub forma de matrice. Pixelii pot avea intensitati cuprinse in intervalul 0-255 (0 = intensitate minima; 255 – intensitate maxima).

In cazul imaginilor color(RGB), exista cateun canal pentru fiecare culoare elementara.

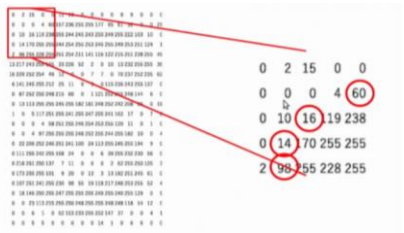


Pixel of an RGB image are formed from the corresponding pixel of the three component images



Fisierele .bmp ofera in header informatii multiple despre dimensiunile, tipul, alcatuirea imaginii, fiecare informatie despre un pixel fiind formata dintr-o triplet de valori corespunzatoare R, G, B.

Detectarea marginilor (Edge Detection/Extraction) este o tehnică de procesare a imaginii (Image Processing) utilizată pentru a identifica punctele dintr-o imagine digitală cu discontinuități = schimbări bruște ale luminozității imaginii. Aceste puncte în care luminozitatea imaginii variază brusc sunt numite marginile (sau limitele) imaginii. Este unul dintre pașii de bază în procesarea imaginilor, recunoașterea modelelor în imagini și viziunea computerizată (Computer Vision).



Mitran Ana-Theodora
331ABb

Există diverse metode de detectare a marginilor, iar următoarele sunt unele dintre cele mai frecvent utilizate metode:

- Prewitt
- Sobel
- Scharr
- Canny

Atunci când procesăm imagini digitale de foarte înaltă rezoluție, se folosesc tehnici de convoluție, care folosesc filtre(kernel) pentru transformarea imaginii, pe portiuni, in rezultatul dorit. Aceste filtre se aplica pe axele Ox si Oy. Voi folosi filtrul Prewitt, metoda foarte utilizata, ce detecteaza marginile verticale si orizontale:

Orizontal:

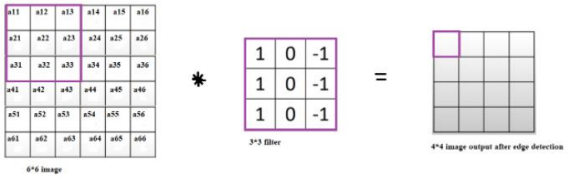
1	1	1
0	0	0
-1	-1	-1

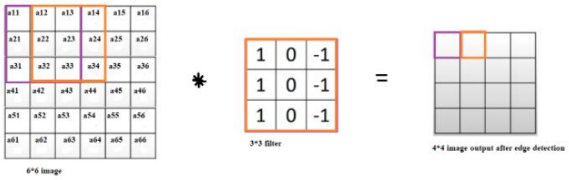
Vertical:

1	0	-1
1	0	-1
1	0	-1

Pe baza kernel-urilor de 3*3, prin convolutie, se obtine gradientul imaginii, pe ceel doua axe:

$$\mathbf{G_x} = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G_y} = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * \mathbf{A}$$





Folosind ambele rezultate ale convolutiilor, se obtine magnitudinea gradientului:

$$\mathbf{G} = \sqrt{\mathbf{G}_x^{\,2} + \mathbf{G}_y^{\,2}}$$

Mitran Ana-Theodora
331ABb

Implementare

Utilizatorul aplicatiei ii este cerut sa introduca de la tastatura calea catre imaginea ce se doreste a fi convertita si apoi sa introduca si pentru fisierul rezultat calea dorita.

Se incepe procesul de citire. ProducerThread depune in BufferClass pe rand cateun sfert din imagine si ConsumerThread preia sferturile pe rand (sincronizare). Intre pasii de tip put si get din buffer, se adauga timp de asteptare pentru a se evidentia pasii facuti, ceea ce face tot procesul mai lent.

Tot ConsumerThread realizeaza si unirea celor 4 sferturi de informatie si creeaza la final un obiect de tip EdgeExtraction, care se va ocupa de prelucracrea imaginii obtinute si va salva fisierul la final cu calea precizata de utilizator.

Prelucrarea propriu-zisa pentru a obtine marginile imaginii poate fi impartita in etape astfel:

-se converteste prima data imaginea la grayscale(tonuri de gri), prin convertirea fiecarui pixel in parte, luat ca triplet r, g, b
-se calculeaza Gx si Gy conform teoriei prezentate, luand pe rand cateun fragment de 3x3 din imaginea grayscale
-se calculeaza magnitudinea gradientului si se construieste pixelul nou

Utilizatorul poate introduce si date in linie de comanda, de exemplu, path-urile pentru fiesierele sursa/destinatie. Aceste argumente vor fi afisate in consola.

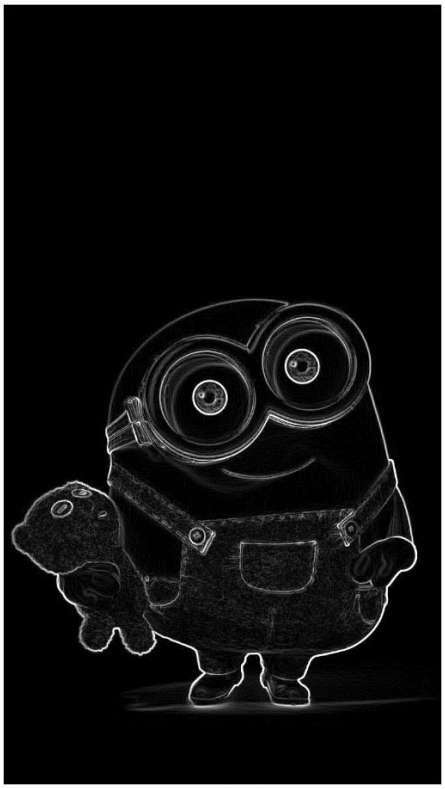
Lucrand cu fisiere, inevitabil pot aparea erori ce trebuie gestionate si semnalate, atunci cand apar la scriere fisiere, citire din fisiere.

Rulare

```
Introduceti calea catre fisierul sursa ["X" pentru a iesi din aplicatie]:
C:/Random/minion.bmp
Introduceti calea catre fisierul destinatie:
C:/Random/rezultat.bmp
Producatorul a pus sfertul      1       din imagine
Consumatorul a primit sfertul   1       din imagine
Producatorul a pus sfertul      2       din imagine
Consumatorul a primit sfertul   2       din imagine
Producatorul a pus sfertul      3       din imagine
Consumatorul a primit sfertul   3       din imagine
Producatorul a pus sfertul      4       din imagine
Consumatorul a primit sfertul   4       din imagine
Citirea din fisier cu multithreading a durat 4.175 secunde
Transformarea imaginii a durat 0.154 secunde
Scrirea in fisier a durat 0.324 secunde
```

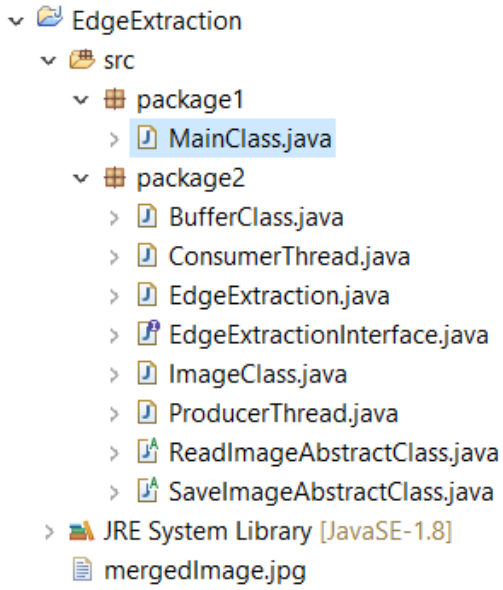
Mitran Ana-Theodora
331ABb

Rezultat:



Descriere structurala

Alcatuirea proiectului:



Mitran Ana-Theodora
331ABb

Clasele implementate sunt structurate in doua pachete, primul pachet cu singurul fisier.java contine main-ul, pentru a testa aplicatia.

Pentru a realiza proiectul, am folosit concepte de programare orientata pe obiecte:

Abstractizare

Clase abstracte: ReadImageAbstractClass, SaveImageAbstractClass

Interfata: EdgeExtractionInterface

Mostenire

Ierarhie:

ReadImageAbstractClass -> SaveImageAbstractClass -> ImageClass -> EdgeExtraction (prin „extends”)

EdgeExtractionInterface -> EdgeExtraction (prin „implements”)

Thread->ProducerThread, ConsumerThread

Incapsulare

Datele si metodele din clase sunt protejate si au acees doar clasele care le mostenesc sau daca se folosesc de catre alte clase din exterior datele/metodele de tip public.

Polimorfism

In SaveImageAbstractClass am creat doua metode abstracte cu acelasi nume, una fara parametru pentru salvare imagine in workspace si alta cu parametru calea fisierului destinatie dorit.

Mitran Ana-Theodora
331ABb

Cod sursa

MainClass:

```java
public class MainClass {
    public static void main(String[] args) throws IOException {

        //daca se introduc parametrii in linie de comanda
        for(String s: args){
            System.out.println("Parametru din linie de comanda:");
            System.out.println(s);
        }
        String file1 = args[0];
        String file2 = args[1];

         //citire de la tastatura/consola
        Scanner scanner = new Scanner(System.in);
        try {
            System.out.println("Introduceti calea catre fisierul sursa [\"X\" pentru a iesi din aplicatie]:");
            String path1 = scanner.nextLine(); //citirea informatiei introduse
            //intentie de iesire
            if ("X".equals(path1)) {
                System.out.println("---Iesire efectuata cu succes---");
                return; //iesire fortata
            }
            //a fost introdusa o cale pentru fisierul de intrare
            //solicit si o cale de iesire pentru rezultat
            System.out.println("Introduceti calea catre fisierul destinatie:");
            String path2 = scanner.nextLine();

            //multithreading pentru citire, dupa care se trece la prelucrarea imaginii
            BufferClass b = new BufferClass(); //buffer cu sincronizare Producer-Consumer
            ProducerThread p = new ProducerThread(b, path1);
            ConsumerThread c = new ConsumerThread(b, path2);
            p.start();
            c.start();
        } finally {
            scanner.close(); //se opreste citirea de la tastatura
        }
    }
}
```

BufferClass:

```java
public class BufferClass { //cu sincronizare
    private BufferedImage image= null;
    private boolean available = false;

    public synchronized BufferedImage get() {
        while (!available) {
            try {
                wait();// Asteapta producatorul sa puna o valoare
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        available = false;
        notifyAll ();
        return image;
    }

    public synchronized void put (BufferedImage image) throws IOException {
        while (available) {
            try {
                wait();// Asteapta consumatorul sa preia valoarea
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        this.image = image;
        available = true;
        notifyAll ();
    }
}
```

Mitran Ana-Theodora
331ABb

ProducerThread:

```
11 public class ProducerThread extends Thread{
12     private BufferClass buffer;
13     private String path;
14 
15     public ProducerThread(BufferClass buffer, String path) {
16         this.buffer = buffer;
17         this.path = path;
18     }
19 
20     public void run () {
21         //incep masurarea timpului de citire
22         long time = System.currentTimeMillis();
23 
24         //folosesc un vector pentru a obtine sferturile de imagine
25         BufferedImage images[]= new BufferedImage[4];
26 
27         try {
28             BufferedImage image = null;
29             try (FileInputStream stream = new FileInputStream(path)) {
30                 image = ImageIO.read(new File(path));
31             } catch (FileNotFoundException e) {
32                 System.out.println("Fisierul nu a fost gasit!");
33             }
34             //numarul de fragmente in care se imparte imaginea pe linii si coloane
35             int row = 2;
36             int col = 2;
37 
38             //dimensiunile unui fragment de imagine
39             int width = image.getWidth()/ col;
40             int height = image.getHeight() / row;
41 
42             int x = 0;
43             int y = 0;
44             int count = 0;
46             for (int i = 0; i < row; i++) {
47                 y = 0;
48                 for (int j = 0; j < col; j++) {
49                     try {
50                         images[count++] = image.getSubimage(y, x, width, height);
51                         y += width;
52                     } catch (Exception e1) {
53                         e1.printStackTrace();
54                     }
55                 }
56                 x += height;
57             }
58         } catch (IOException e) {
59             e.printStackTrace();
60         }
61 
62         for (int i = 0; i < 4; i++) {
63             try {
64                 buffer.put(images[i]);
65             } catch (IOException e1) {
66                 e1.printStackTrace();
67             }
68             System.out.println("Producatorul a pus sfertul\t" + (i+1) + "\tdin imagine");
69             try {
70                 sleep (1000);
71             } catch (InterruptedException e) {}
72         }
73         //prin diferenta de timp, aflu durata citirii
74         time = System.currentTimeMillis() - time;
75         System.out.println("Citirea din fisier cu multithreading a durat " + time / 1000.0f + " secunde");
76     }
77 }
```

Mitran Ana-Theodora
331ABb

ConsumerThread:

```
 9 public class ConsumerThread  extends Thread {
10     private BufferClass buffer;
11     private String path;
12 
13     public ConsumerThread(BufferClass buffer, String path) {
14         this.buffer = buffer;
15         this.path = path;
16     }
17 
18     public void run () {
19         BufferedImage images[] = new BufferedImage[4]; //preiau sferturile de imagine din buffer
20 
21         for (int i = 0; i < 4; i++) {
22           images[i] = buffer.get();
23           System.out.println("Consumatorul a primit sfertul\t" + (i+1) + "\tdin imagine");
24 
25           try {
26             sleep (1000);
27           } catch (InterruptedException e) {}
28         }
29 
30         int type = images[0].getType();
31         int width = images[0].getWidth();
32         int height = images[0].getHeight();
33 
34         BufferedImage mergedImage = new BufferedImage(2*width, 2*height, type);
35 
36         int k = 0;
37         for ( int i = 0; i < 2; i++) {
38             for (int j = 0; j < 2; j++) {
39                 mergedImage.createGraphics().drawImage(images[k++], width * j, height * i, null);
40             }
41         }
```

```
41         }
42         try {
43             ImageIO.write(mergedImage, "bmp", new File("mergedImage.jpg"));
44             //se salveaza local imaginea obtinuta din sferturi
45         } catch (IOException e) {
46             e.printStackTrace();
47         }
48 
49         //ETAPA DE PRELUCRARE DUPA CITIRE
50         EdgeExtraction object = new EdgeExtraction(mergedImage);
51         object.extractEdges();
52 
53         //SALVARE REZULTAT
54         try {
55             object.saveImage(path);
56         } catch (IOException e) {
57             e.printStackTrace();
58         }
59     }
60 
```

ReadImageAbstractClass -> Am implementat si o metoda cu varargs, pentru a putea citi mai multe fisiere si a le retine intr-un vector de tip BufferedImage

```
 6 //clasa abstracta, poate reprezenta primul nivel de mostenire
 7 public abstract class ReadImageAbstractClass {
 8     abstract public BufferedImage readFile(String path) throws IOException;
 9     abstract public BufferedImage[] readMultipleFiles(int...x) throws IOException;
10 
11     public void description(){
12         System.out.println("Clasa abstracta, mostenire nivel 1");
13     }
14 }
```

Mitran Ana-Theodora
331ABb

SaveImageAbstractClass-> apare polimorfismul

```java
//clasa abstracta care mosteneste o alta clasa abstracta
public abstract class SaveImageAbstractClass extends ReadImageAbstractClass{
    abstract public void saveImage(String path) throws IOException;
    abstract public void saveImage(); //polimorfism

    public void description(){
        System.out.println("Clasa abstracta, mostenire nivel 2");
    }
}
```

ImageClass:

```java
public class ImageClass extends SaveImageAbstractClass{

    private BufferedImage image = null;

    public ImageClass(){ //constructor fara parametru

    }

    public ImageClass(BufferedImage image){//constructor cu parametru
        this.image= image;
    }

    @Override
    public void saveImage(String path) throws IOException {
        try (FileOutputStream stream = new FileOutputStream(path)) {
            ImageIO.write(image, "BMP", stream);//salvare tot in format .bmp
        } catch (FileNotFoundException e) {
            System.out.println("Calea invalida!");
            return;
        } catch (IOException e) {
            System.out.println("Eroare in timpul salvarii");
            return;
        }
    }

    @Override
    public void saveImage() {
        try {
            ImageIO.write(image, "bmp", new File("test_salvare.jpg"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public BufferedImage readFile(String path) throws IOException {
        //returneaza valoare pentru a putea folosi la citire multipla
        try (FileInputStream stream = new FileInputStream(path)) {
            image = ImageIO.read(new File(path));
        } catch (FileNotFoundException e) {
            System.out.println("Fisierul nu a fost gasit!");
        }
        return image;
    }

    @Override
    public BufferedImage[] readMultipleFiles(int... x) throws IOException {
        BufferedImage [] images = new BufferedImage[x.length];
        Scanner scanner = new Scanner(System.in); //citire date introduse de la tastatura
        for(int i: x){

            System.out.println("Introduceti calea catre fisierul sursa:");
            String path = scanner.nextLine();
            images[i] = readFile(path);
        }
        scanner.close(); //se finalizeaza citirea de la consola
        return images;
    }

}
```

Mitran Ana-Theodora
331ABb

EdgeExtractionInterface :

```java
//exemplu de interfata ce va fi implementata pentru aplicarea algoritmului
public interface EdgeExtractionInterface {
    public int[][] convertToGrayscale();
    public void extractEdges();
}
```

EdgeExtraction - implementeaza metodele abstracte din interfata si mosteneste proprietatile si metodele din ImageClass (nivel 4)

```java
public class EdgeExtraction extends ImageClass implements EdgeExtractionInterface { //mostenire

    private BufferedImage image = null;

    public EdgeExtraction(BufferedImage img){
        this.image=img;
    }

    //implementarea metodelor abstracte din interfata

    @Override
    public int[][] convertToGrayscale() {
        //dimensiunile imaginii
        int width = image.getWidth();
        int height = image.getHeight();

        int[][] result;
        result = new int[width][height];

        for (int j = 0; j < height; j++) {
            for (int i = 0; i < width; i++) {
                //citesc RGB-ul pixelilor pe rand
                int pixel = image.getRGB(i, j);
                //impart informatia pe canale de culoare elementara: R, G, B
                int r = (pixel >> 16) & 0xff;
                int g = (pixel >> 8) & 0xff;
                int b = pixel & 0xff;
                //tonul de gri se obtine prin media valorilor canalelor de culoare
                int grayscale = (r+g+b)/3;
                //retin informatia in matricea creata anterior
                result[i][j] = grayscale;
                // RGB-ul se inlocuieste cu grayscale
                pixel = (grayscale << 24) | (grayscale << 16) | (grayscale << 8) | grayscale;
                image.setRGB(i, j, pixel);
            }
        }
        return result;
    }
```

Mitran Ana-Theodora
331ABb

```java
    @Override
    public void extractEdges() {

        long startTime = System.currentTimeMillis(); // masurarea timpului de executie pentru extragerea marginilor

        //dimensiunile imaginii
        int width = image.getWidth();
        int height = image.getHeight();

        //ETAPA 1: convertirea imaginii la grayscale(tonuri de gri)
        int [][] result = new int[width][height];
        result= convertToGrayscale();

        //ETAPA 2: aplicarea filtrelor(Gx, Gy) pe directiile x si y pentru a obtine gradientul

        //filtrele orizontal si vertical pentru convolutie
        int[][] Gx = {{-1,0,1},
                      {-1,0,1},
                      {-1,0,1}};

        int[][] Gy = {{-1,-1,-1},
                      {0,  0, 0},
                      {1,  1, 1}};

        for (int y = 1; y < height-1; y++) {
            for (int x = 1; x < width-1; x++) {
                //portiune 3x3 din imagine in convolutie cu Gx
                int gx = (Gx[0][0] * result[x-1][y-1]) + (Gx[0][1] * result[x][y-1]) + (Gx[0][2] * result[x+1][y-1]) +
                         (Gx[1][0] * result[x-1][y])   + (Gx[1][1] * result[x][y])   + (Gx[1][2] * result[x+1][y]) +
                         (Gx[2][0] * result[x-1][y+1]) + (Gx[2][1] * result[x][y+1]) + (Gx[2][2] * result[x+1][y+1]);
                //portiune 3x3 din imagine in convolutie cu Gy
                int gy = (Gy[0][0] * result[x-1][y-1]) + (Gy[0][1] * result[x][y-1]) + (Gy[0][2] * result[x+1][y-1]) +
                         (Gy[1][0] * result[x-1][y])   + (Gy[1][1] * result[x][y])   + (Gy[1][2] * result[x+1][y]) +
                         (Gy[2][0] * result[x-1][y+1]) + (Gy[2][1] * result[x][y+1]) + (Gy[2][2] * result[x+1][y+1]);

```

Suprascriere a metodei saveImage din ImageClass pentru a calcula si timpul de executie a salvarii imaginii conform caii indicate de utilizator:

```java
    public void saveImage(String path) throws IOException{
        long startTime = System.currentTimeMillis();

        try (FileOutputStream stream = new FileOutputStream(path)) {
            ImageIO.write(image, "BMP", stream);//salvare tot in format .bmp
        } catch (FileNotFoundException e) {
            System.out.println("Calea invalida!");
            return;
        } catch (IOException e) {
            System.out.println("Eroare in timpul salvarii");
            return;
        }
        // intrevalul de timp necesar scrierii in fisierul destinatie si se afiseaza
        long stopTime = System.currentTimeMillis() - startTime;

        System.out.println("Scrierea in fisier a durat " + stopTime / 1000.0f + " secunde");
    }
}
```

Performante

Pentru citire, prelucrare si salvare se calculeaza timpii necesari, pentru a evalua performantele procesului. Algoritmul are o performanta liniara de prelucrare a imaginii si timpii obtinuti depind de dimensiunea imaginii.

Mitran Ana-Theodora
331ABb

Bibliografie

https://www.mygreatlearning.com/blog/introduction-to-edge-detection/

https://en.wikipedia.org/wiki/Prewitt_operator

https://www.analyticsvidhya.com/blog/2021/03/edge-detection-extracting-the-edges-from-an-image/
https://medium.com/javarevisited/building-a-java-edge-detection-application-6147b68e5d79