OPERATORUL PREWITT

ARDELEANU STEFAN

331AB

1. PREZENTARE

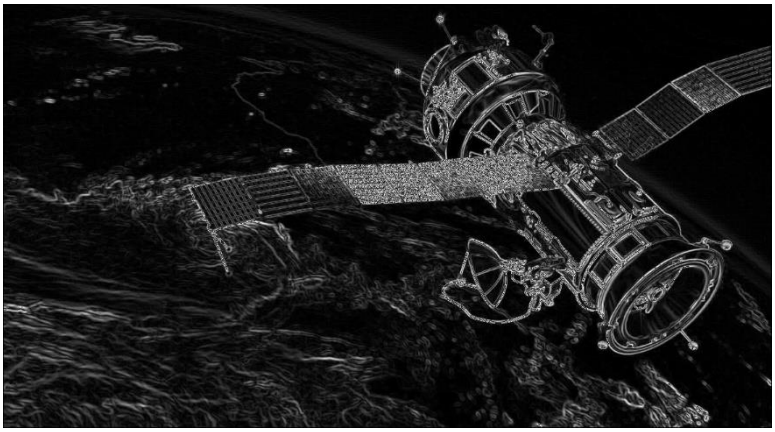
Operatorul Prewitt este utilizat pentru detectarea marginilor într-o imagine. Detectează două tipuri de margini:

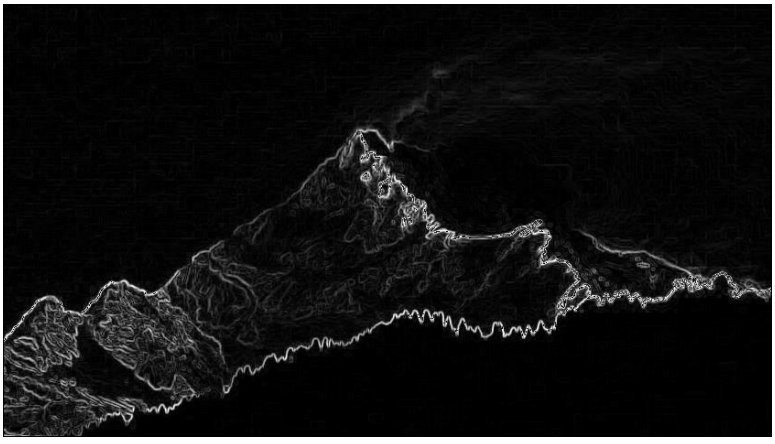
- Margini orizontale
- Margini verticale

Marginile sunt calculate folosind diferența dintre intensitățile pixelilor corespunzătoare unei imagini. Toate măștile care sunt utilizate pentru detectarea marginilor sunt, de asemenea, cunoscute ca măști derivate.

INAINTE DUPA





2. ALGORTIM

Din punct de vedere matematic, această tehnica folosește blocuri matriceale de dimensiune 3x3 care sunt convolutate cu matricea originala formată din datele pixelilor imaginii.

A – matricea imaginii

Gx – matricea derivativă orizontală

Gy – matricea derivativă verticală

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

Unde "*" este operația de convoluție.

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$ -formula de calculare a gradientului

Astfel, se poate calcula direcția gradientului:

$$\theta = atan2(G_y, G_x)$$

3. DESCRIEREA APLICATIEI

Aplicatia este formata din urmatoarele clase:

Abstract – o clasa abstracta de baza ce contine metode abstracte

Prelucrare – mosteneste clasa **Abstract**

Prewitt – mosteneste clasa **Prelucrare**, se aplica operatorul pewitt

Gradient – interfata

Calcul_Gradient – mosteneste interfata **Gradient**, se calculeaza gradientul imaginii

Buffer – clasa in care se memoreaza matricea care se transmite de la **Producer** la **Consumer**

Producer – clasa in care se transmite cate un sfert in buffer

Consumer – clasa in care se preia din buffer si se construieste matricea finala

Clasa **Prewitt**

Citirea imaginii:

```
File inputFile = new File( args[0] ); //memorarea caii imaginii

long timpCitire1 = System.currentTimeMillis();
inImage = ImageIO.read(inputFile); // memorarea imaginii
long timpCitire2 = System.currentTimeMillis();
//Se calculeaza timpul necesar citirii imaginii
System.out.print("Timpul necesar citirii imaginii este de:  " + (double)(timpCitire2 - timpCitire1)/ 1000 + " secunde\n");
```

Punerea in matrice si calcularea operatorului prewitt:

```
//parcurgem imaginea
for( int i = 1; i < inImgWidth - 1; i++ ) {
    for( int j = 1; j < inImgHeight - 1; j++ ) {
        //punem in matricea pixelMatrix valorile RGB pentru fiecare pixel

        pixelMatrix[0][0]=new Color(afterImage.getRGB(i-1, j-1)).getRed();
        pixelMatrix[0][1]=new Color(afterImage.getRGB(i-1,j)).getRed();
        pixelMatrix[0][2]=new Color(afterImage.getRGB(i-1,j+1)).getRed();
        pixelMatrix[1][0]=new Color(afterImage.getRGB(i,j-1)).getRed();
        pixelMatrix[1][2]=new Color(afterImage.getRGB(i,j+1)).getRed();
        pixelMatrix[2][0]=new Color(afterImage.getRGB(i+1,j-1)).getRed();
        pixelMatrix[2][1]=new Color(afterImage.getRGB(i+1,j)).getRed();
        pixelMatrix[2][2]=new Color(afterImage.getRGB(i+1,j+1)).getRed();

        int margine = (int) aux.gradient(pixelMatrix); //calculam marginea
        outImage.setRGB(i, j, ( margine << 16 | margine << 8 | margine ));
    }
}
```

Scrierea imaginii:

```
File outFile = new File( args[1] );

long start = System.currentTimeMillis();
ImageIO.write(outImage, "bmp", outFile); //scriem imaginea
long stop = System.currentTimeMillis();
//Se calculeaza timpul necesar scrierii imaginii
System.out.print("Timpul necesar scrierii imaginii este de:  " + (double)(stop - start)/ 1000 + " secunde\n");
```

Clasa **Calcul_Gradient**

```
public class Calcul_Gradient implements Gradient {

    @Override
    public double gradient(int[][] pixelMatrix) {

        int gx=(pixelMatrix[0][0]*(-1))+(pixelMatrix[0][1]*(-1))+(pixelMatrix[0][2]*(-1))+(pixelMatrix[2][0]*1)+(pixelMatrix[2][1]*1)+(pixelMatrix[2][2]*1);//m
        int gy=(pixelMatrix[0][0]*(-1))+(pixelMatrix[0][2]*1)+(pixelMatrix[1][0]*(-1))+(pixelMatrix[1][2]*1)+(pixelMatrix[2][0]*(-1))+(pixelMatrix[2][2]*1);//m

        double gradient = Math.sqrt(Math.pow(gy, 2)+Math.pow(gx, 2)); //calcularea gradientului

        return gradient;
    }

}
```

In aceasta clasa calculam gradientul imaginii.

Clasa **Buffer**

```
public class Buffer {

    private int[][] pixelMatrix; //declaram o matrice de tip int
    private boolean busy = false; //declaram un flag pentru a vedea daca buffer-ul este sau nu ocupat

    public synchronized int[][] get() {
        while( !busy ) { //cat timp bufferul nu este ocupat asteapta
            try {
                wait();
            } catch ( InterruptedException e ) {
                e.printStackTrace();
            }
        }
        busy = false;
        notifyAll();
        return this.pixelMatrix;
    }

    public synchronized void put( int[][] pixelMatrix ) {
        while( busy ) { //cat timp bufferul este ocupat bufferul asteapta
            try {
                wait();
            } catch ( InterruptedException e ) {
                e.printStackTrace();
            }
        }
        this.pixelMatrix = pixelMatrix;
        busy = true;
        notifyAll();
    }

}
```

Clasa **Producer**

```
public class Producer extends Thread {

    private Buffer buffer; //variabila de tip buffer
    private BufferedImage Image; //variabila de tip image

    Producer( Buffer buffer, BufferedImage Image ) {
        this.buffer = buffer;
        this.Image = Image;
    }

    public void run() {
        int width = Image.getWidth();
        int height = Image.getHeight();

        for( int i = 0; i < 4; i++ ) {
            //imparitm imaginea in 4
            int[][] pixelMatrix = new int[width][height]; //declaram o matrice pentru a o trimite la consumer

            for( int k = 0; k < width; k++ )
                for( int j = height/4 * i; j < height/4 * (i+1); j++ )
                    pixelMatrix[k][j] = Image.getRGB(k,j); //adaugam in matrice cate un sfert
            buffer.put(pixelMatrix); //adaugam matricea in buffer pentru a o prelucra consumerul

            System.out.println("Producatorul a pus sfertul cu numarul " + (i + 1) + " al imaginii.");
            try {
                sleep(1000);
            } catch( InterruptedException e ) {
                System.out.println(e);
            }
        }

    }

}
```

Clasa **Consumer**

```
public class Consumer extends Thread {
    private Buffer buffer;

    Consumer( Buffer buffer ) {
        this.buffer = buffer;
    }

    public void run() {

        int width = Prewitt.inImage.getWidth();
        int height = Prewitt.inImage.getHeight();

        for (int i = 0; i < 4; i++) {
            // se imparte in 4 imaginea

            int[][] pixelMatrix = new int[width][height]; // se declara o matrice in care vom stoca sfertul de imagine primit

            pixelMatrix = buffer.get();// se ia sfertul curent din Buffer

            for( int k = 0; k < width; k++ )
                for( int j = height/4 * i; j < height/4 * (i+1); j++ )
                    Prewitt.afterImage.setRGB(k, j, pixelMatrix[k][j]);
                    // se seteaza pixelul curent in imaginea ce va fi trimisa spre procesare

            System.out.println("Consumatorul a preluat sfertul cu numarul " + (i + 1) + " al imaginii.");
            try {
                sleep(1000);// se executa o secventa de sleep inainte de a trimite urmatorul sfert
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }

    }

}
```