НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

**Лабораторна робота №6**

з дисципліни «Аналіз зображень та візуалізація в діагностиці та лікуванні»

на тему: «Фільтрація зображень»

Варіант 6

Виконав:

студент гр. БС-41м

Григоревський Є.С.

Перевірила:

Алхімова С.М.

Київ – 2014

**Ціль:** вивчити теоретичні відомості фільтрації, ознайомитись з основними видами фільтрації медичних зображень, реалізувати лінійну та нелінійну фільтрацію медичних зображень, вивчити особливості фільтрації граничних пікселів.

**Завдання**

1. Ознайомитися з теоретичними відомостями реалізації фільтрації зображень.
2. Відповідно до свого варіанту розробити програмний додаток, що виконує реалізацію фільтрації до завантаженого медичного зображення в результаті обробки події від клавіатури:

|  |  |  |
| --- | --- | --- |
| **Номер варіанту** | **Від фільтрації** | **Обчислення граничних пікселів** |
| 6 | Фільтр Робертса | Довизначення за межами відзеркаленням |

**Лістинг програми**

**public class Main** {

public static void main(String[] args) {

//Deploying BufferedImage from DICOM to managers.DataManager...

DataManager.getInstance().setImage(ImageHelper.readDicom(DataManager.PATH));

launchSwingWithOpenGL2(DataManager.getInstance().getImage().getWidth(), DataManager.getInstance().getImage().getHeight());

}

public static void launchSwingWithOpenGL2(int width, int height) {

GLCanvas canvas = DataManager.getInstance().getCanvas();

canvas.setPreferredSize(new Dimension(width, height));

canvas.addGLEventListener(new Rendering());

canvas.addKeyListener(new KeyEventListener());

canvas.addMouseMotionListener(new MouseEventListener());

final JFrame frame = DataManager.getInstance().getFrame();

//Set the GUI content

frame.getContentPane().add(canvas, BorderLayout.CENTER);

//Set default options ...

frame.setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

frame.pack();

frame.setLocationRelativeTo(null);

SwingUtilities.invokeLater( () -> frame.setVisible(true) );

}

}

**public class Pixel** {

public int color;

public Pixel(int grayScale) {

this.color = grayScale;

}

@Override

public String toString() {

return " GrayScale (" + color + ") ";

}

public void setColor(int grayScale) {

this.color = grayScale;

}

public int getGrayScale(){

return color;

}

public int getARGB(){

return new Color(color,color,color).getRGB();

}

}

**public class Mouse** {

public int x;

public int y;

public void setPosition(int x, int y){

this.x = x;

this.y =y;

}

@Override

public String toString(){

return "Mouse ("+x+"|"+y+")";

}

}

**public class DataManager** {

private DataManager(){}

private static DataManager instance = null;

public static final String PATH = "lab6.dcm";

public static int NEW\_MAX\_GREYSCALE = 255;

private GLCanvas canvas = new GLCanvas();

private final JFrame frame = new JFrame();

private DICOM image=null;

private int currentTexturesAndTitle =0;

private Texture[] textures = new Texture[2];

private String[] titles = {"Original","Roberts Filter"};

private Mouse mouse = new Mouse();

public Mouse getMouse() {

return mouse;

}

public JFrame getFrame() {

return frame;

}

public int getCurrentTexturesAndTitle() {

return currentTexturesAndTitle;

}

public void setCurrentTexturesAndTitle(int currentTexturesAndTitle) {

this.currentTexturesAndTitle = currentTexturesAndTitle;

}

public Texture[] getTextures() {

return textures;

}

public String[] getTitles() {

return titles;

}

public GLCanvas getCanvas() {

return canvas;

}

public DICOM getImage() {

return image;

}

public void setImage(DICOM image) {

this.image = image;

}

public static DataManager getInstance(){

if(instance == null)

instance = new DataManager();

return instance;

}

}

**public class MouseEventListener implements MouseMotionListener** {

@Override

public void mouseDragged(MouseEvent e) {

}

@Override

public void mouseMoved(MouseEvent e) {

GLCanvas canvas = DataManager.getInstance().getCanvas();

DataManager.getInstance().getMouse().setPosition(e.getX(), canvas.getWidth() - 1 - e.getY());

canvas.display();

}

}

**public class KeyEventListener implements KeyListener** {

@Override

public void keyTyped(KeyEvent e) {

}

@Override

public void keyPressed(KeyEvent e) {

if (e.getKeyCode() == KeyEvent.VK\_SPACE) {

int currentIndex = DataManager.getInstance().getCurrentTexturesAndTitle();

int elementsLength = DataManager.getInstance().getTextures().length;

currentIndex = ++currentIndex % elementsLength;

DataManager.getInstance().setCurrentTexturesAndTitle(currentIndex);

DataManager.getInstance().getFrame().setTitle(DataManager.getInstance().getTitles()[currentIndex]);

DataManager.getInstance().getCanvas().display();

}

if(e.getKeyCode() == KeyEvent.VK\_ESCAPE)

DataManager.getInstance().getFrame().dispatchEvent(new WindowEvent(DataManager.getInstance().getFrame(),WindowEvent.WINDOW\_CLOSING));

}

@Override

public void keyReleased(KeyEvent e) {

}

}

**public class Rendering implements GLEventListener** {

@Override

public void init(GLAutoDrawable glAutoDrawable) {

GL2 gl2 = glAutoDrawable.getGL().getGL2();

gl2.glEnable(GL.GL\_TEXTURE\_2D);

//initialize all textures

Texture[] textures = DataManager.getInstance().getTextures();

//initialize original image:

textures[0] = AWTTextureIO.newTexture(glAutoDrawable.getGLProfile(), DataManager.getInstance().getImage().getBufferedImage(), false);

//initialize filtered image using Robert's filter

textures[1] = AWTTextureIO.newTexture(glAutoDrawable.getGLProfile(),

ImageHelper.getRobertsFiltrationImage(DataManager.getInstance().getImage()),

false);

for(int i=0;i<textures.length;i++)

textures[i].bind(gl2);

}

@Override

public void reshape(GLAutoDrawable glAutoDrawable, int x, int y, int width, int height) {

GL2 gl2 = glAutoDrawable.getGL().getGL2();

gl2.glOrtho(0, width, 0, height, -1, 1);

}

@Override

public void display(GLAutoDrawable glAutoDrawable) {

GL2 gl2 = glAutoDrawable.getGL().getGL2();

gl2.glClear(GL.GL\_COLOR\_BUFFER\_BIT);

int currentTextureIndex = DataManager.getInstance().getCurrentTexturesAndTitle();

Texture currentTexture = DataManager.getInstance().getTextures()[currentTextureIndex];

currentTexture.bind(gl2);

BufferedImage image = DataManager.getInstance().getImage().getBufferedImage();

gl2.glBegin(GL2.GL\_POLYGON);

gl2.glTexCoord2i(0, 1);

gl2.glVertex2i(0, 0);

gl2.glTexCoord2i(0, 0);

gl2.glVertex2i(0, image.getHeight());

gl2.glTexCoord2i(1, 0);

gl2.glVertex2i(image.getWidth(), image.getHeight());

gl2.glTexCoord2i(1, 1);

gl2.glVertex2i(image.getWidth(), 0);

gl2.glEnd();

//--------------Pixel color in text renderer----------------

gl2.glReadBuffer(GL.GL\_FRONT);

FloatBuffer pixelColorBuffer = FloatBuffer.allocate(4);

Mouse mouse = DataManager.getInstance().getMouse();

gl2.glReadPixels(mouse.x, mouse.y, 1, 1, GL.GL\_RGBA, GL.GL\_FLOAT, pixelColorBuffer);

Pixel hoveredPixel = new Pixel((int) (pixelColorBuffer.array()[0]\*255));

TextRenderer renderer = new TextRenderer(new Font("Arial", Font.BOLD, 14));

renderer.setColor(1.0f, 0f, 0f, 1.0f);

renderer.beginRendering(DataManager.getInstance().getCanvas().getWidth(), DataManager.getInstance().getCanvas().getHeight());

renderer.draw(mouse.toString() + hoveredPixel.toString(), 1, 1);

renderer.endRendering();

renderer.setColor(1.0f, 1.0f, 1.0f, 1.0f);

}

@Override

public void dispose(GLAutoDrawable glAutoDrawable) {

}

}

**public class ImageHelper**

{

public static DICOM readDicom(String path) {

DICOM image = new DICOM();

image.open(path);

return image;

}

public static int getMax(List<Pixel> imageData){

return Collections.max(imageData,new Comparator<Pixel>() {

@Override

public int compare(Pixel o1, Pixel o2) {

return Integer.compare(o1.getGrayScale(),o2.getGrayScale());

}

}).getGrayScale();

}

public static int getMin(List<Pixel> imageData){

return Collections.min(imageData,new Comparator<Pixel>() {

@Override

public int compare(Pixel o1, Pixel o2) {

return Integer.compare(o1.getGrayScale(),o2.getGrayScale());

}

}).getGrayScale();

}

public static BufferedImage createImageFromPixelsList(List<Pixel> imageData, int width, int height) {

BufferedImage result = new BufferedImage(width,height,BufferedImage.TYPE\_INT\_ARGB);

for(int i=0;i<height;i++)

for(int j=0;j<width;j++)

result.setRGB(i,j, imageData.get(i\*width+j).getARGB());

return result;

}

public static List<Pixel> createPixelsListFromImage(DICOM image){

List<Pixel> imageData = new ArrayList<>(image.getWidth()\*image.getHeight());

for(int i=0;i<image.getHeight();i++)

for(int j=0;j<image.getWidth();j++)

imageData.add(new Pixel(image.getPixel(i,j)[0]));

int minGrayScale = getMin(imageData);

List<Pixel> resultSet = new ArrayList<>(image.getWidth()\*image.getHeight());

for(Pixel pixel:imageData)

resultSet.add(new Pixel(pixel.getGrayScale() - minGrayScale));

return resultSet.isEmpty() ? null : resultSet;

}

public static BufferedImage getRobertsFiltrationImage(DICOM original){

List<Pixel> pixels = createPixelsListFromImage(original);

for(int i=0;i<pixels.size();i++){

int[][] pixelSet = getPixelsSetWithMirror(pixels,i, original.getWidth());

//For Roberts Filter newI= sqrt( P^2 + Q^2);

//Rx = [0 1; -1 0] Ry = [1 0; 0 -1] => P(x,y) = I(x,y+1) - I(x+1,y) & Q(x,y) = I(x,y) - I(x+1,y+1)

//newI = sqrt( (I(x,y+1) - I(x+1,y))^2 + (I(x,y) - I(x+1,y+1))^2 );

int newGrayScale = (int) Math.sqrt(Math.pow((pixelSet[0][1] - pixelSet[1][0]),2) + Math.pow((pixelSet[0][0] - pixelSet[1][1]),2));

pixels.get(i).setColor(newGrayScale);

}

double min = getMin(pixels);

double max = getMax(pixels);

pixels.forEach((pixel) -> pixel.setColor((int) ((pixel.getGrayScale() - min) / max \* DataManager.NEW\_MAX\_GREYSCALE)) );

return createImageFromPixelsList(pixels,original.getWidth(),original.getHeight());

}

public static int [][] getPixelsSetWithMirror(List<Pixel> pixels, int currentElementIndex, int imageWidth){

int x= currentElementIndex / imageWidth;

int y= currentElementIndex % imageWidth;

int [][] pixelSet = new int[2][2];

if (x == imageWidth - 1 && y == imageWidth - 1) {

pixelSet[0][0] = pixels.get(x \* imageWidth + y).getGrayScale();

pixelSet[0][1] = pixels.get(x \* imageWidth + y - 1).getGrayScale();

pixelSet[1][0] = pixels.get((x - 1) \* imageWidth + y).getGrayScale();

pixelSet[1][1] = pixels.get((x - 1) \* imageWidth + y - 1).getGrayScale();

return pixelSet;

}

if (x == imageWidth - 1) {

pixelSet[0][0] = pixels.get(x \* imageWidth + y).getGrayScale();

pixelSet[0][1] = pixels.get(x \* imageWidth + y + 1).getGrayScale();

pixelSet[1][0] = pixels.get((x - 1) \* imageWidth + y).getGrayScale();

pixelSet[1][1] = pixels.get((x - 1) \* imageWidth + y - 1).getGrayScale();

return pixelSet;

}

if (y == imageWidth - 1) {

pixelSet[0][0] = pixels.get(x \* imageWidth + y).getGrayScale();

pixelSet[0][1] = pixels.get(x \* imageWidth + y - 1).getGrayScale();

pixelSet[1][0] = pixels.get((x + 1) \* imageWidth + y).getGrayScale();

pixelSet[1][1] = pixels.get((x + 1) \* imageWidth + y - 1).getGrayScale();

return pixelSet;

}

pixelSet[0][0] = pixels.get(x \* imageWidth + y).getGrayScale();

pixelSet[0][1] = pixels.get(x \* imageWidth + y + 1).getGrayScale();

pixelSet[1][0] = pixels.get((x + 1) \* imageWidth + y).getGrayScale();

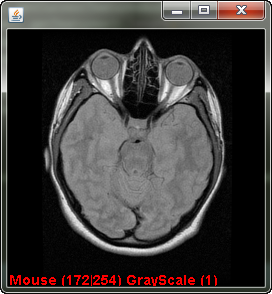
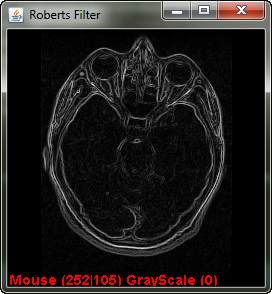
pixelSet[1][1] = pixels.get((x + 1) \* imageWidth + y + 1).getGrayScale();

return pixelSet;

}

}

**Результати роботи програми**

|  |  |
| --- | --- |
|  |  |
|  |  |

**Відповіді на контрольні запитання**

1. **Низькочастотна фільтрація, їх властивості, недоліки.**

Низькочастотна фільтрація знижує локальну контрастність зображення. Недоліком цієї фільтрації є розмиття границь об’єктів зображення.

1. **Як розрахувати коефіцієнти маски фільтра Гауса?**

За формулою , де (i, j) – окіл точки.

1. **Маски високочастотної фільтрації, їх особливості.**

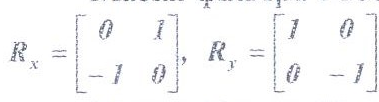
Високочастотна фільтрація збільшує різкість зображення. Маски високочастотних фільтрів можна отримати відніманням згладженого низькочастотним фільтром зображення від оригінального подання.

1. **Диференціальні оператори визначення границь для безперервних функцій та для дискретних зображень.**

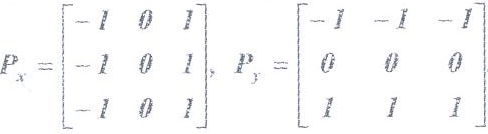
Для безперервних функцій:

До дискретних диференціальних операторів відносяться фільтри Робертса, Превіта, Собеля, ізотропний фільтр.

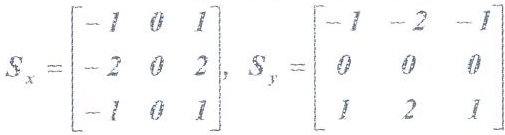
Фільтр Робертса:



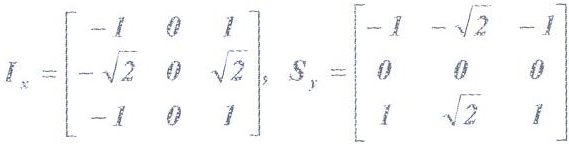
Фільтр Превіта



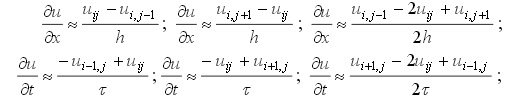
Фільтр Собеля



Ізотропний фільтр



1. **Апроксимація перших частинних похідних методом скінченних різниць.**



1. **Апроксимація других частинних похідних методом скінченних різниць.**



1. **Нелінійна фільтрація, її переваги над лінійною. Нельнійні фільтри.**

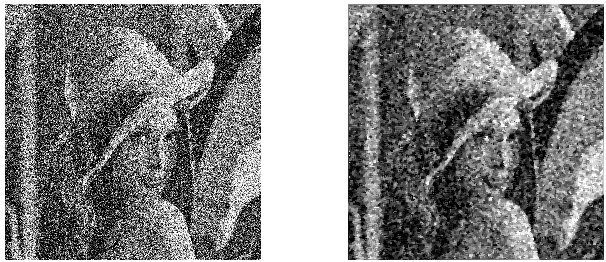
Лінійні алгоритми фільтрації призводять до згладжування різких перепадів яскравості зображень, які пройшли обробку.

Нелінійними фільтрами є медіанна фільтрація, мінімаксна фільтрація.

1. **Медіанна фільтрація, її маски. Пояснити на прикладі, коли маски медіанної фільтрації усувають не точкові імпульсні перешкоди, а коли ні.**

Медіанна фільтрація може виявитись неефективною, коли більше половини пікселів є викидами однієї полярності. 

Зображення до медіанної фільтрації Зображення після проведення медіанної фільтрації



Зображення до медіанної фільтрації Зображення після проведення медіанної фільтрації