Московский технический университет связи и информатики

Лабораторная работа 4

**Fractal Explorer**

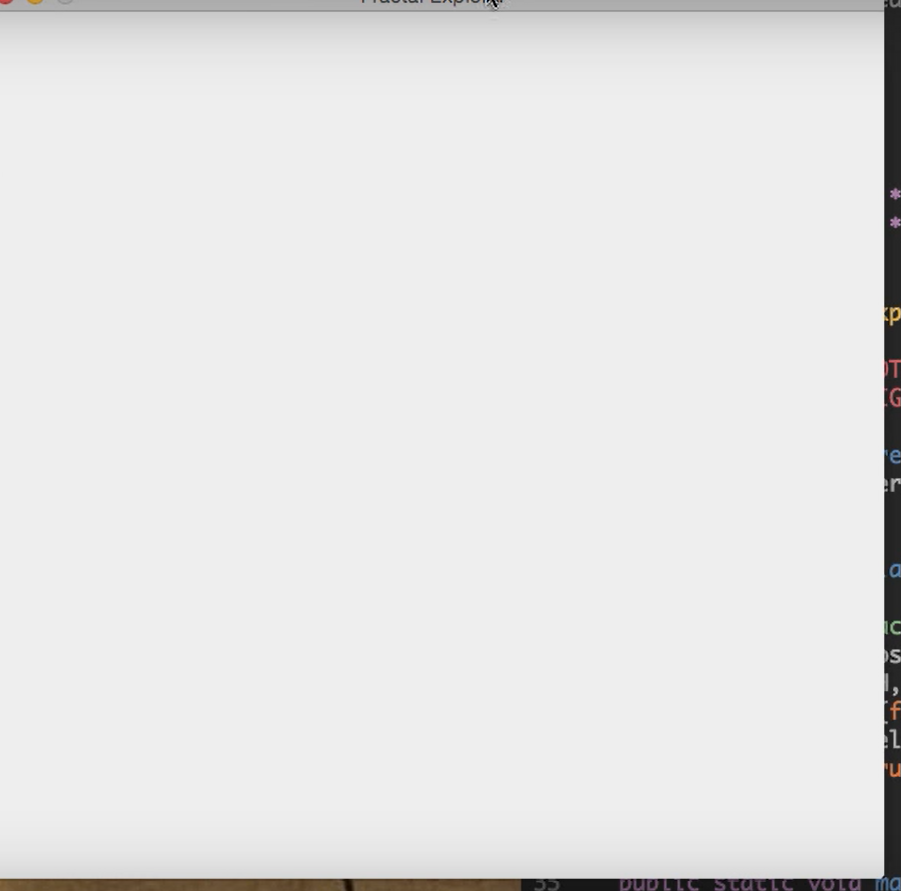
Выполнил студент группы БСТ1601

Кузнецов Александр

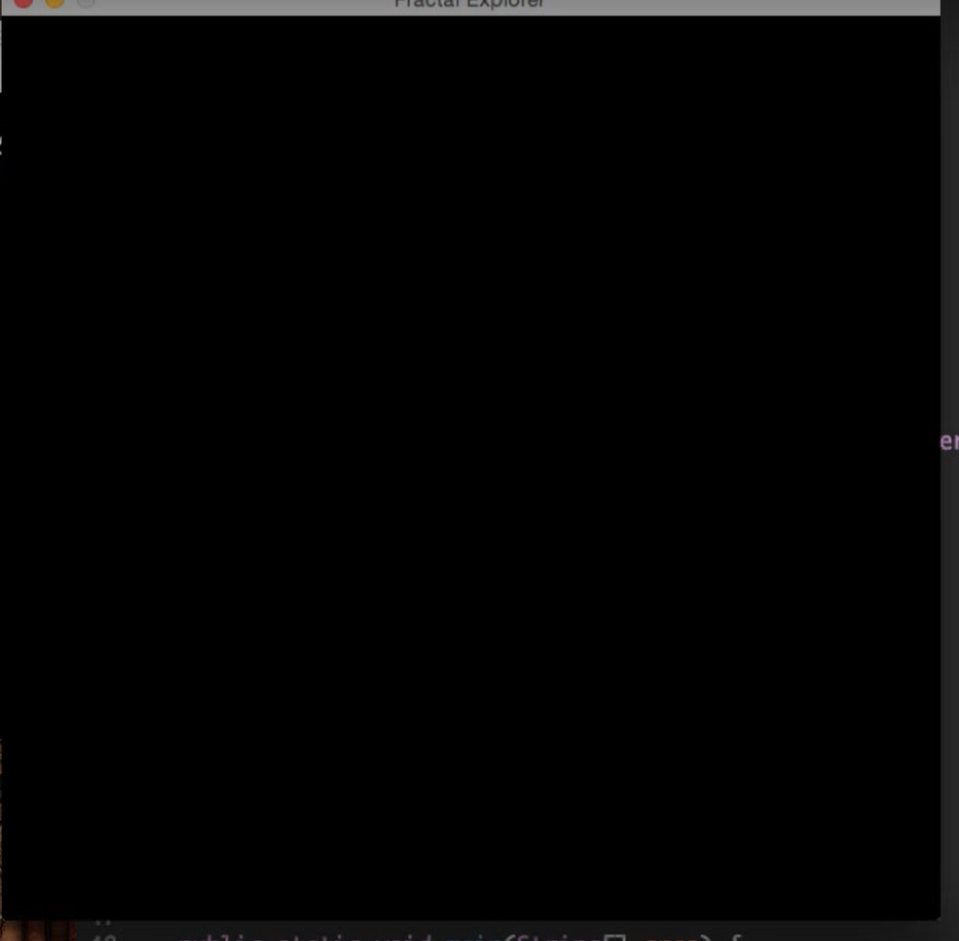
Москва 2018

В данной лабораторной мы будем создавать Java-приложение для рисования фракталов. Для начала мы создаем окно, в котором будет рисоваться наш фрактал. В коде можно задать его размерность. В моем случае это 800x800.

В результате получим примерно следующий результат

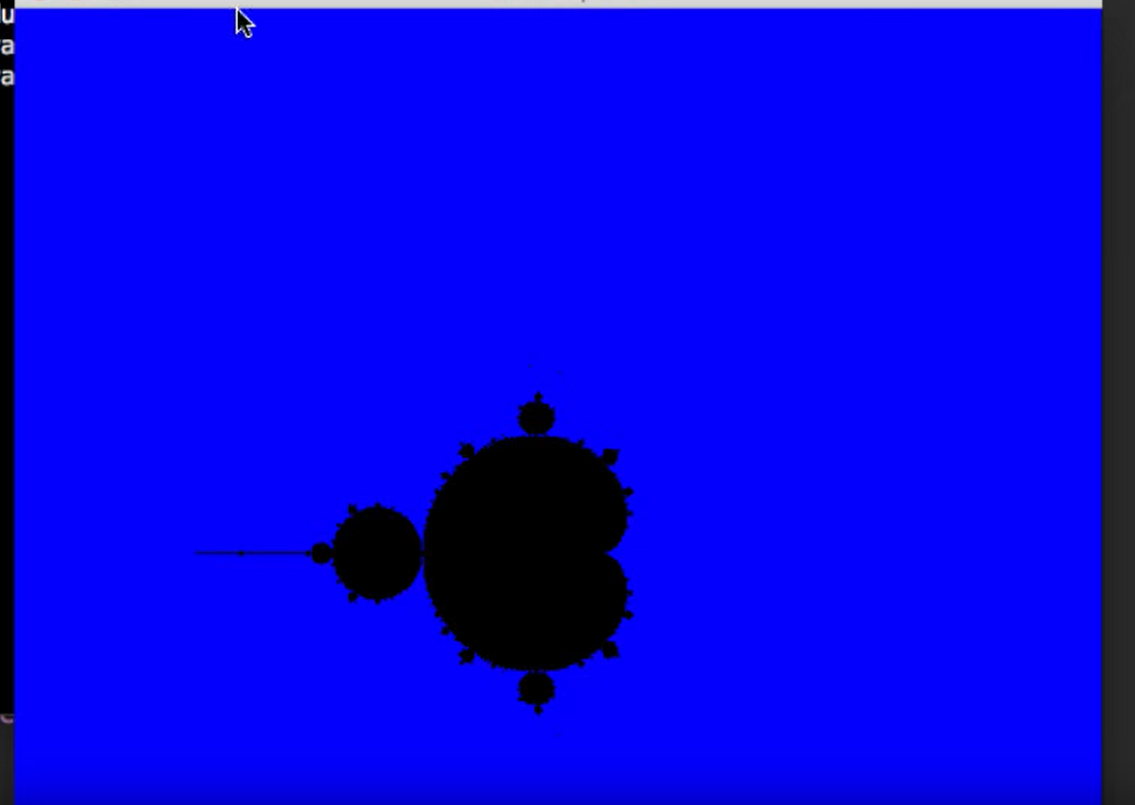


Далее в коде мы прописываем фоновый цвет окна, в данном случае – черный.

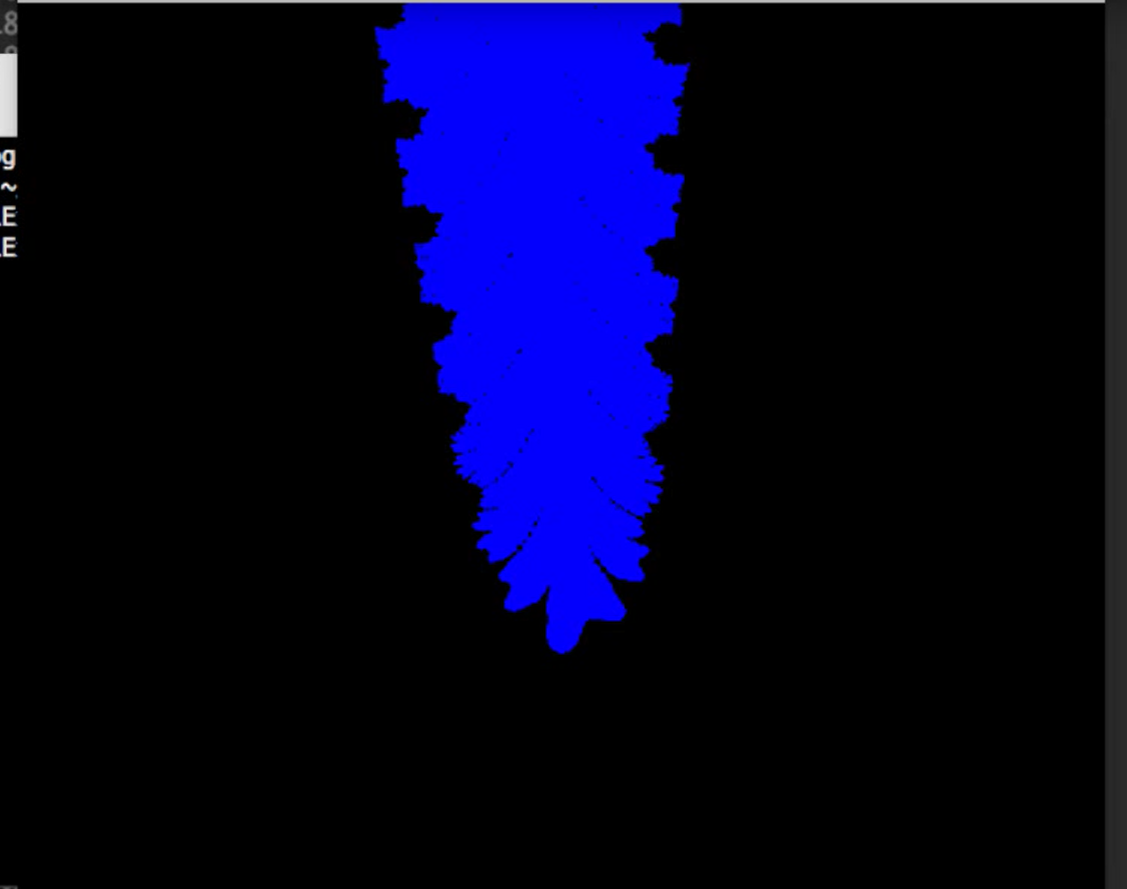


Далее мы начинаем писать код для уже рисования фрактала. Сам код в полном объеме будет приведен в конце отчета.

В результате у нас получится следующий результат.



Далее мы можем изменять масштаб, к примеру увеличить определенную деталь фрактала.



Код программы

import java.util.\*;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class FractalExplorer extends JFrame

{

static final int WIDTH = 800;

static final int HEIGHT = 800;

Canvas canvas;

BufferedImage fractalImage;

static final int MAX\_ITER =200;

static final double DEFAULT\_ZOOM =100.0;

static final double DEFAULT\_TOP\_LEFT\_Y=-3.0;

static final double DEFAULT\_TOP\_LEFT\_X= -3.0;

double zoomFactor = DEFAULT\_ZOOM;

double topLeftY = DEFAULT\_TOP\_LEFT\_Y;

double topLeftX = DEFAULT\_TOP\_LEFT\_X;

public FractalExplorer()

{

setInitialGUIProperties();

addCanvas();

canvas.addKeyStrokeEvents();

updateFractal();

}

private double getYPos(double y)

{

return y/zoomFactor-topLeftY;

}

private double getXPos(double x)

{

return x/zoomFactor+topLeftX;

}

public void updateFractal()

{

for (int x =0 ;x<WIDTH; x++)

{

for (int y=0; y<WIDTH; y++)

{

double c\_r= getXPosX(x);

double c\_i= getYPosY(y);

int iterCount = computeIterations(c\_r, c\_i);

int pixelColor= maleColor(iterCount);

fractalImage.setRGB(x,y,pixelColor);

}

}

canvas.repaint();

}

private int makeColor(int iterCount)

{

int color = 0b011011100001100101101000;

int color = 0b000000000000010101110111;

int shiftMag = iterCount/13;

if(iterCount == MAX\_ITER)

{

return Color.BLACK.getRGB();

}

return Color | (mask <<shiftMag);

}

private void addcanvas(){

canvas= new Canvas();

fractalImage= new BufferedImage(WIDTH, HEIGHT,BufferedImage.TYPE\_INT\_RGB);

canvas.setVisible(true);

this.add(canvas, BorderLayout.CENTER);

}

public void setInitialGUIProperties()

{

this.setTitle("My Fractal Explorer");

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.setSize(WIDTH,HEIGHT);

this.setResizable(false);

this.setLocationRelativeTo(null);

this.setVisible(true);

}

public static void main(String[] args)

{

new FractalExplorer();

}

private int computeIterations(double c\_r, double c\_i)

{

double z\_r=0.0;

double z\_i=0.0;

int iterCount = 0;

while (z\_r\*z\_r+z\_i\*z\_i<=4.0)

{

double z\_r\_tmp=z\_r;

z\_r=z\_r\*z\_r-z\_i\*z\_i+c\_r;

z\_i=2\*z\_i\*z\_r\_tmp+c\_i;

if(iterCount>=MAX\_ITER)

{

return MAX\_ITER;

}

iterCount++;

}

return iterCount;

}

private void moveUp()

{

double curHeight = HEIGHT/zoomFactor;

topLeftY += curHeight/6;

updateFractal();

}

private void moveDown()

{

double curHeight = HEIGHT/zoomFactor;

topLeftY -= curHeight/6;

updateFractal();

}

private void moveLeft()

{

double curWidth = WIDTH/zoomFactor;

topLeftX -= curWidth/6;

updateFractal();

}

private void moveRight()

{

double curWidth = WIDTH/zoomFactor;

topLeftX += curWidth/6;

updateFractal();

}

private void adjustZoom(double newX, double newY, double ZoomFactor)

{

topLeftX +=newX/zoomFactor;

topLeftY -= newY/zoomFactor;

zoomFactor = newZoomFactor;

topLeftX -= (WIDTH/2)/zoomFactor;

topLeftY += (HEIGHT/2)/zoomFactor;

updateFractal();

}

private class Canvas extends JPanel implements MouseListener

{

public Canvas()

{

addMouseListener(this);

}

@Override public Dimension getPreferredSize()

{

return new Dimension(WIDTH,HEIGHT);

}

@Override public void paintComponent(Graphics drawingObj)

{

drawingObj.drawImage( fractalImage, 0, 0, null);

}

@Override public void mousePressed (MouseEvent mouse)

{

double x = (double) mouse.getX();

double y = (double) mouse.getY();

switch(mouse.getButton() )

{ //left

case MouseEvent.BUTTON1:

adjustZoom(x,y,zoomFactor\*2);

break;

//right

case MouseEvent.BUTTON3:

adjustZoom(x,y,zoomFactor/2)

break;

}

}

public void addKeyStrokeEvents()

{

KeyStroke wKey=KeyStroke.getKetStroke(KeyEvent.VK\_W,0);

KeyStroke aKey=KeyStroke.getKetStroke(KeyEvent.VK\_A,0);

KeyStroke sKey=KeyStroke.getKetStroke(KeyEvent.VK\_S,0);

KeyStroke dKey=KeyStroke.getKetStroke(KeyEvent.VK\_D,0);

Action wPressed = new AbstractAction()

{

@Override public void actionPerformed(ActionEvent e)

{

moveUp();

}

};

Action aPressed = new AbstractAction()

{

@Override public void actionPerformed(ActionEvent e)

{

moveLeft();

}

};

Action sPressed = new AbstractAction()

{

@Override public void actionPerformed(ActionEvent e)

{

moveDown();

}

};

Action dPressed = new AbstractAction()

{

@Override public void actionPerformed(ActionEvent e)

{

moveRight();

}

};

this.getInputMap().put(wKey,"w\_key");

this.getInputMap().put(aKey,"a\_key");

this.getInputMap().put(sKey,"s\_key");

this.getInputMap().put(dKey,"d\_key");

this.getActionMap().put("w\_key", wPressed);

this.getActionMap().put("a\_key", aPressed);

this.getActionMap().put("s\_key", sPressed);

this.getActionMap().put("d\_key", dPressed);

}

@Override public void mouseReleased (MouseEvent mouse){}

@Override public void mouseClicked (MouseEvent mouse){}

@Override public void mouseEntered (MouseEvent mouse){}

@Override public void mouseExited (MouseEvent mouse){}

}

}