Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики і обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №1

З комп’ютерної графіки

*Виконав:*

Студент групи ІО-12

Нестерук Ю.О.

*Перевірив:*

Подрубайло О.О.

м. Київ

2012 р.

**1. Завдання:**

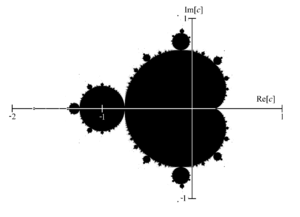
Побудувати параметризовану геометричну фігуру примітивами, доступними в стандартних класах Java роботи з графікою.

**2. Короткі теоретичні відомості:**

Обрана фігура – множина Мандельброта.

Мно́жество Мандельбро́та — это множество таких точек c на комплексной плоскости, для которых итеративная последовательность z0=0, zn=zn-12+c (n=1, 2, 3, …) не уходит на бесконечность. То есть, это множество таких c, для которых существует действительное R, что неравенство |zn|<R выполняется при всех натуральных n.

Множество Мандельброта является в некотором смысле фракталом. Его фрагменты не строго подобны исходному множеству, но при многократном увеличении определённые части всё больше похожи друг на друга.

Рисунок 1. Множина Мандельброта

**3. Лістинг програми:**

**package** mandelbrot;

**import** my.drawing.drawboard.\*;

**public** **class** MainClass {

**static** **double** *x0* = 0.002;

**static** **double** *y0* = 0.002;

**static** **double** *x*;

**static** **double** *y*;

**static** **int** *iteration*;

**static** **int** *maxIterations* = 1000;

**public** **static** **void** main(String[] args) {

DrawBoard board = **new** DrawBoard(*x0*, *y0*, *maxIterations*);}}

package my.drawing.drawboard;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.image.BufferedImage;

import java.util.Random;

import javax.swing.JComponent;

import javax.swing.JFrame;

import org.apache.commons.math3.geometry.euclidean.twod.Vector2D;

public class DrawBoard extends JFrame {

protected static final int WIDTH = 1024;

protected static final int HEIGHT = 768;

private static DrawPanel drPanel;

private static boolean paused;

public DrawBoard(double x0, double y0, int maxIter) {

this.addNotify();

this.setSize(this.getInsets().left + this.getInsets().right + WIDTH,

this.getInsets().top + this.getInsets().bottom + HEIGHT);

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

drPanel = new DrawPanel(x0, y0, maxIter);

this.add(drPanel);

this.setResizable(false);

this.setVisible(true);}

class DrawPanel extends JComponent {

private long time = System.currentTimeMillis();

private long deltaTime;

private int delay = 10;

private Random r = new Random();

private int width, height;

private BufferedImage dbImage;

private Graphics2D dbGraphics;

double sx, sy;

double cx, cy;

double x, xTemp, y;

int iteration;

int maxIterations = 1000;

double dx = -750;

double dy = -400;

int outRadius = 4;

public DrawPanel(double x0, double y0, int maxIter) {

super();

width = DrawBoard.WIDTH;

height = DrawBoard.HEIGHT;

this.setSize(width, height);

dbImage = new BufferedImage(width, height, BufferedImage.TYPE\_INT\_RGB);

dbGraphics = dbImage.createGraphics();

dbGraphics.setBackground(Color.black);

this.sx = x0;

this.sy = y0;

this.maxIterations = maxIter;

toPaint1();

}

public void toPaint1() {

dbGraphics.setColor(Color.BLACK);

dbGraphics.clearRect(0, 0, width, height);

for (double i = -width - 1000; i < width - 450; i += 0.5)

for (double j = -height - 100; j < height; j += 0.5) {

cx = i / width;

cy = j / height;

x = 0.0;

y = 0.0;

for (iteration = 0; (x \* x + y \* y < outRadius)

&& (iteration < maxIterations); iteration++) {

xTemp = x \* x - y \* y + cx;

y = 2 \* x \* y + cy;

x = xTemp;

}

dbGraphics.setColor(new Color((iteration \* 10) & 0x0000FF));

dbGraphics.drawOval((int) ((i / 2 + 1000) \* 0.8), (int) j / 2 + 400, 1, 1);

}

}

public void toPaint2() {

dbGraphics.setColor(Color.BLACK);

dbGraphics.clearRect(0, 0, width, height);

for (double i = 0; i < width; i += 0.5)

for (double j = 0; j < height; j += 0.5) {

cx = sx \* (i + dx);

cy = sy \* (j + dy);

x = cx;

y = cy;

for (iteration = 0; ((x \* x + y \* y < outRadius) && (iteration < maxIterations)); iteration++) {

xTemp = x \* x - y \* y + cx;

y = 2 \* x \* y + cy;

x = xTemp;

}

x = x / sx - dx;

y = y / sx - dy;

dbGraphics.setColor(new Color(iteration));

dbGraphics.drawOval((int) x, (int) y, 1, 1);

}

}

@Override

protected void paintComponent(Graphics g) {

Graphics2D g2 = (Graphics2D) g;

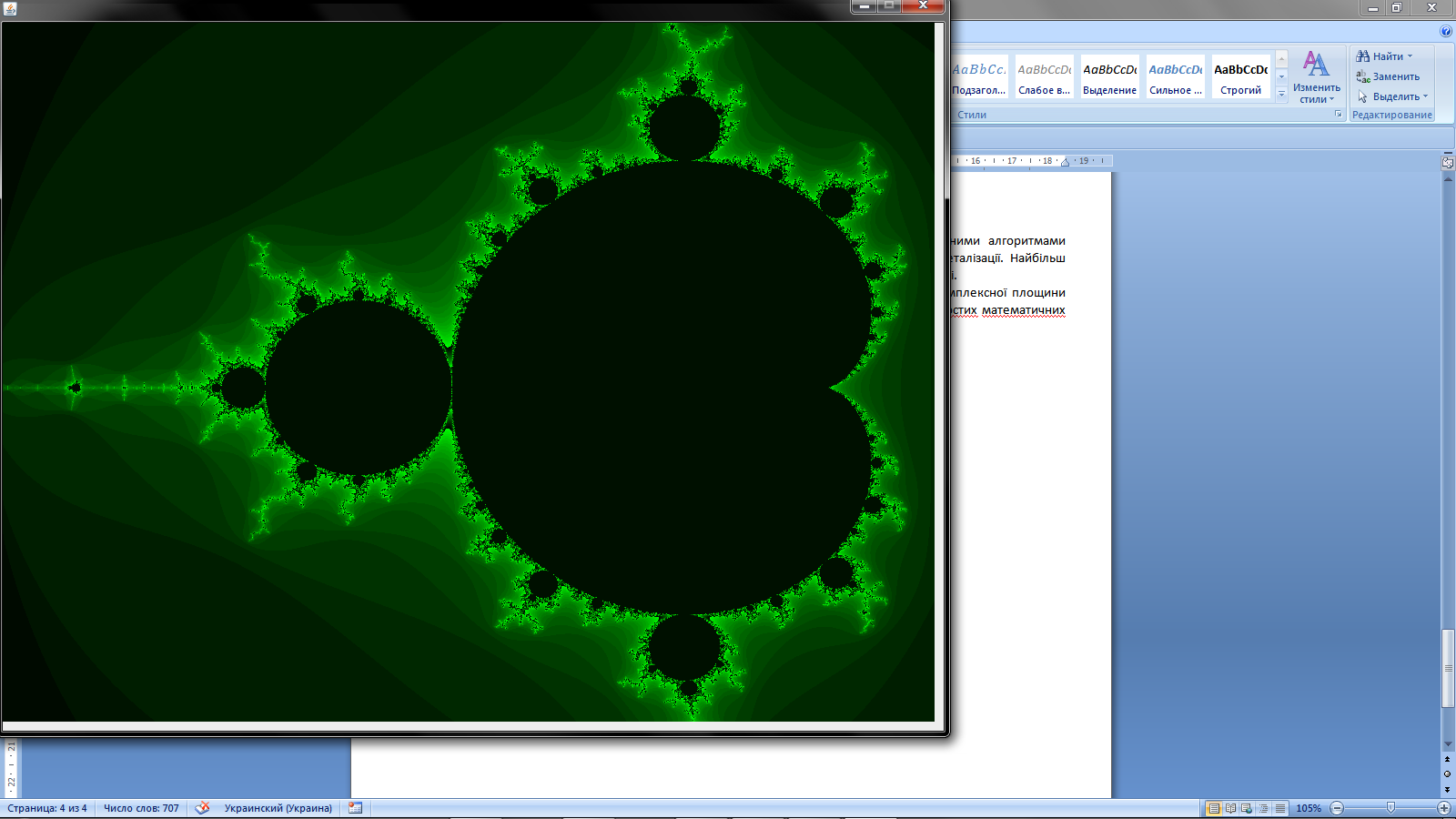
g2.drawImage(dbImage, 0, 0, this);

}

}

}

**4. Результати виконання програми:**



**5. Аналіз результатів (висновки):**

Були проведені запуски програми з різними коефіцієнтами (параметрами) та з різними алгоритмами побудову множини. Результати запусків відрізняються формою множини та степенем деталізації. Найбільш оптимальні параметри, з якими множина виходить найбільш ефектною, залишені в програмі.

При побудові даної множини дуже важливо правильно перетворити координати з комплексної площини діапазону х є [-2; 0.5] і y є [-1; 1] у зображення, де х є [0; 1023], y є [0; 768]. Шляхом простих математичних розрахунків мною були підібрані правильні нормуючі коефіцієнти.

Колір кожної точки площини залежить від швидкості (кількості ітерацій), з якою Zn для даної точки прямує до нескінченості.

Ітераційний крок відповідає за деталізацію зображення.