Ernest Ramazani

IUPUI Summer 2023

CSCI 43700

Homework 2 : 2d Graphic Demo

Project Description

This project, a simple Java-based fractal drawer, is an interactive and graphical demonstration of two popular fractal structures: the Sierpinski Triangle and the Koch Snowflake. Users can manipulate the depth of recursion, choose between the two fractals, change the background color, and save their creations as images. The software provides an intuitive and hands-on approach to understanding and visualizing recursive structures and the concept of fractals.

The application uses the Java Swing library for the creation of its graphical user interface, and leverages the Graphics and Polygon classes for the actual rendering of the fractals. Additionally, the application uses Timer, JComboBox, JSlider, JButton and other Swing components to manage user interaction and automate the rendering process.

Learning Outcomes

This project allowed for the exploration and application of various important programming concepts:

1. Recursion: The crux of generating fractals is recursion. Through this project, we demonstrated the power of recursion in rendering complex structures with simple, self-repeating rules.

2. Graphics programming: This project involved working with Java's Graphics2D API. We learned how to draw, fill, and color different shapes, and manage the rendering process

3. GUI programming with Java Swing: Creating the user interface involved the use of Swing components, layouts, and event handlers. The project thus served as an excellent practical exploration of GUI programming in Java.

4. Timers and animation: Implementing an automated depth-increasing feature required the use of Timers and understanding of basic animation principles.

5. File I/O: Implementing the image save feature necessitated learning how to write images to disk using Java's ImageIO API.

Libraries Used

The project uses Java's built-in libraries, primarily the Swing library for GUI elements, the AWT library for graphics and color manipulation, and the ImageIO library for writing images.

Steps to Run

To run this project, you will need a system with a Java Runtime Environment (JRE) installed

1. Compile the `myTriangles.java` file using the Java compiler (`javac`): `javac myTriangles.java`

2. Run the compiled class using the Java interpreter (`java`): `java myTriangles`

Upon running, a window will appear with options to select the fractal type, depth, and background color. The window will also contain a Pause/Resume button for the automatic depth increment feature, and a Save button to save the current fractal image.

Conclusion

In conclusion, this project serves as an engaging graphical demonstration of recursion and fractals, offering a hands-on experience with Java's graphics capabilities and Swing library. As with any project, there is potential for further enhancement, such as adding more fractal types or more customization options.

Appendix – code

```
//Ernest Ramazani

//IUPUI Summer 23

//CSCI43700 Homework 2

//May 26, 2023

//2d Graphic Demo

import javax.swing.*;

import java.awt.*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.ItemEvent;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

//This class represents a JPanel that will be used to draw the triangles.

public class myTriangles extends JPanel {

// Depth is the current recursion level of the fractals.

private int depth = 0;

// A Swing Timer that will be used to automatically update the drawing every
second.

private Timer timer;

// CurrentShape is the index of the currently selected fractal shape (0 for
Sierpinski, 1 for Koch).

private int currentShape = 0;

private Color[] colors = {

new Color(255, 192, 203),

new Color(173, 216, 230),
```

```
new Color(152, 251, 152),
new Color(240, 230, 140),
new Color(221, 160, 221),
new Color(176, 224, 230),
new Color(255, 218, 185)
};
private String[] colorNames = {
"Pink",
"Light Blue",
"Light Green",
"Khaki",
"Plum",
"Powder Blue",
"Peach"
};
private String[] shapeNames = {
"Sierpinski Triangle",
"Koch Snowflake"
};
// The constructor sets up the GUI components and their event listeners.
public myTriangles(JFrame frame, boolean autoAnimate) {
// Create and configure the Pause/Resume button.
JButton pauseButton = new JButton("Pause");
pauseButton.addActionListener(new ActionListener() {
@Override
public void actionPerformed(ActionEvent e) {
if (timer.isRunning()) {
```

```
timer.stop();

pauseButton.setText("Resume");

} else {

timer.start();

pauseButton.setText("Pause");

}

}

});

frame.add(pauseButton, BorderLayout.SOUTH);

// Create the timer that will automatically increase the depth every second,
if autoAnimate is true.

// Only start the timer if autoAnimate is true

if (autoAnimate) {

timer = new Timer(1000, new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

depth++;

if (currentShape == 0)

frame.setTitle("Sierpinski Triangle - Depth: " + depth + ", Triangles: " +
(int) Math.pow(3, depth));

else

frame.setTitle("Koch Snowflake - Depth: " + depth);

repaint();

}

});

timer.start();

}

// Create and configure the shape selection combo box.
```

```
JComboBox<String> shapeBox = new JComboBox<>(shapeNames);

shapeBox.addItemListener(e -> {

if (e.getStateChange() == ItemEvent.SELECTED) {

currentShape = shapeBox.getSelectedIndex();

repaint();

}

});

// Create and configure the depth control slider.

JSlider slider = new JSlider(JSlider.HORIZONTAL, 0, 10, 0);

slider.addChangeListener(e -> {

depth = slider.getValue();

if (currentShape == 0)

frame.setTitle("Sierpinski Triangle - Depth: " + depth + ", Triangles: " +
(int) Math.pow(3, depth));

else

frame.setTitle("Koch Snowflake - Depth: " + depth);

repaint();

});

// Color control

JComboBox<String> colorBox = new JComboBox<>(colorNames);

colorBox.addItemListener(e -> {

if (e.getStateChange() == ItemEvent.SELECTED) {

Color color = colors[colorBox.getSelectedIndex()];

setBackground(color);

repaint();

}

});

// Save button
```

```
JButton saveButton = new JButton("Save Image");

saveButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

BufferedImage image = new BufferedImage(getWidth(), getHeight(),
BufferedImage.TYPE_INT_RGB);

Graphics2D g2 = image.createGraphics();

paintComponent(g2);

g2.dispose();

JFileChooser fileChooser = new JFileChooser();

int option = fileChooser.showSaveDialog(frame);

if (option == JFileChooser.APPROVE_OPTION) {

File file = fileChooser.getSelectedFile();

try {

ImageIO.write(image, "png", file);

JOptionPane.showMessageDialog(frame, "Image saved successfully!");

} catch (IOException ex) {

JOptionPane.showMessageDialog(frame, "Error saving image: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);

}

}

}

});

// Add the controls to a JPanel that will be placed at the top of the window.

JPanel controlPanel = new JPanel();

controlPanel.add(new JLabel("Shape:"));

controlPanel.add(shapeBox);

controlPanel.add(new JLabel("Depth:"));
```

```
controlPanel.add(slider);

controlPanel.add(new JLabel("Color:"));

controlPanel.add(colorBox);

controlPanel.add(saveButton);

frame.add(controlPanel, BorderLayout.NORTH);

}

// This method is called when the JPanel needs to be painted (for example,
when it is resized or when repaint() is called).

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

switch (currentShape) {

case 0:

drawSierpinski(g, depth, 10, getHeight() - 50, getWidth() - 20, getHeight() -
50, getWidth() / 2, 10);

break;

case 1:

drawKoch(g, depth, getWidth() / 4, getHeight() / 4, 3 * getWidth() / 4,
getHeight() / 4);

drawKoch(g, depth, 3 * getWidth() / 4, getHeight() / 4, getWidth() / 2, 3 *
getHeight() / 4);

drawKoch(g, depth, getWidth() / 2, 3 * getHeight() / 4, getWidth() / 4,
getHeight() / 4);

break;

}

g.setColor(Color.BLACK);

g.setFont(new Font("Arial", Font.BOLD, 20)); // setting the font here

if (currentShape == 0)

g.drawString("Triangles: " + (int) Math.pow(3, depth), getWidth() - 150, 30);

else
```

```
g.drawString("Koch lines: " + (int) Math.pow(4, depth), getWidth() - 150,
30);

}

// This method draws a Sierpinski triangle at a given depth.

// x1, y1, x2, y2, x3, y3 are the coordinates of the three vertices of the
triangle.

private void drawSierpinski(Graphics g, int depth, int x1, int y1, int x2,
int y2, int x3, int y3) {

// If the depth is 0, it means we have reached the base case and we simply
draw the triangle.

if (depth == 0) {

Polygon p = new Polygon();

p.addPoint(x1, y1);

p.addPoint(x2, y2);

p.addPoint(x3, y3);

g.drawPolygon(p);

// If the depth is not 0, we are at an intermediate step. We first fill the
triangle with a color

// depending on the depth, then call drawSierpinski recursively for the three
smaller triangles.

} else {

Color[] colors = {

new Color(255, 192, 203),

new Color(173, 216, 230),

new Color(152, 251, 152),

new Color(240, 230, 140),

new Color(221, 160, 221),

new Color(176, 224, 230),

new Color(255, 218, 185)

};

g.setColor(colors[depth % colors.length]);
```

```
Polygon p = new Polygon(); // A new Polygon object is created

p.addPoint(x1, y1); // Add the first vertex of the triangle to the polygon

p.addPoint(x2, y2); // Add the second vertex of the triangle to the polygon

p.addPoint(x3, y3); // Add the third vertex of the triangle to the polygon

g.fillPolygon(p); // Fill and draw the polygon (triangle) on the graphics
object with the currently set color

// Recursively draw the three subtriangles, each at a different corner of the
current triangle

drawSierpinski(g, depth - 1, x1, y1, (x1 + x2) / 2, (y1 + y2) / 2, (x1 + x3)
/ 2, (y1 + y3) / 2); // Draw top subtriangle

drawSierpinski(g, depth - 1, (x1 + x2) / 2, (y1 + y2) / 2, x2, y2, (x2 + x3)
/ 2, (y2 + y3) / 2); // Draw left subtriangle

drawSierpinski(g, depth - 1, (x1 + x3) / 2, (y1 + y3) / 2, (x2 + x3) / 2, (y2
+ y3) / 2, x3, y3); // Draw right subtriangle

}

}

void drawKoch(Graphics g, int n, int x1, int y1, int x2, int y2) {

// If the depth is 0, we are at the base case and we simply draw a line
segment.

if (n == 0) {

g.setColor(Color.BLACK);

g.drawLine(x1, y1, x2, y2);

} else {

// If the depth is not 0, we are at an intermediate step. We calculate the
coordinates

// of the points needed to break the line segment into four smaller ones,

// which will form a "spike" in the shape of an equilateral triangle.

int dx = x2 - x1, dy = y2 - y1;

int x3 = x1 + dx / 3, y3 = y1 + dy / 3; // Point 1/3 of the way from (x1, y1)
to (x2, y2)

int x4 = x1 + dx * 2 / 3, y4 = y1 + dy * 2 / 3; // Point 2/3 of the way from
(x1, y1) to (x2, y2)
```

```
int x5 = (int) ((x1 + x2) / 2.0 - Math.sqrt(3.0) * (y1 - y2) / 6.0); // Top
point of the triangle

int y5 = (int) ((y1 + y2) / 2.0 - Math.sqrt(3.0) * (x2 - x1) / 6.0);

// We then call drawKoch recursively for each of the four smaller line
segments.

drawKoch(g, n - 1, x1, y1, x3, y3);

drawKoch(g, n - 1, x3, y3, x5, y5);

drawKoch(g, n - 1, x5, y5, x4, y4);

drawKoch(g, n - 1, x4, y4, x2, y2);

}

}

//Main function

public static void main(String[] args) {

JFrame frame = new JFrame("Fractal Drawer");

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

frame.setSize(new Dimension(800, 800));

frame.setLayout(new BorderLayout());

int choice = JOptionPane.showConfirmDialog(null, "Do you want the animation
to start automatically?", "Choose an option", JOptionPane.YES_NO_OPTION);

boolean autoAnimate = choice == JOptionPane.YES_OPTION;

frame.add(new myTriangles(frame, autoAnimate), BorderLayout.CENTER);

frame.setVisible(true);

}

}
```