

Kevin R Minn
CIS 457 Computer Graphics
Spring 2023
Assignment 1
(Due date: 02/20/2023)

1. Describe the difference between anisotropic and isotropic mapping modes. Why we need to introduce the pixelWidth/pixelHeight variables? [2 points]

In computer graphics, textures are mapped onto 3D objects using anisotropic or isotropic mapping modes. Isotropic mapping uniformly maps the texture across the object's surface, while anisotropic mapping takes into account the object's shape and orientation. PixelWidth and pixelHeight variables are used in anisotropic mapping to control the distortion of the texture and improve the quality of the mapping. Choosing the appropriate mapping mode and adjusting these variables can help achieve better texture mapping results for different types of objects.

2. Write a method that checks whether projection of point p lies on the closed segment AB. [5 points]

```
import java.awt.geom.Point2D;

public class PointOnSegmentChecker {

    public static boolean isPointOnSegment(Point2D p, Point2D a, Point2D b) {
        double dotProduct = (p.getX() - a.getX()) * (b.getX() - a.getX()) + (p.getY() - a.getY()) *
(b.getY() - a.getY());
        double segmentLength = a.distance(b);
        double projection = dotProduct / (segmentLength * segmentLength);

        if (projection < 0 || projection > 1) {
            // The projection of p is outside the segment AB.
            return false;
        } else {
            // The projection of p is on the segment AB.
            return true;
        }
    }

    public static void main(String[] args) {
        Point2D p = new Point2D.Double(2.5, 3.5);
        Point2D a = new Point2D.Double(1.0, 1.0);
        Point2D b = new Point2D.Double(4.0, 4.0);

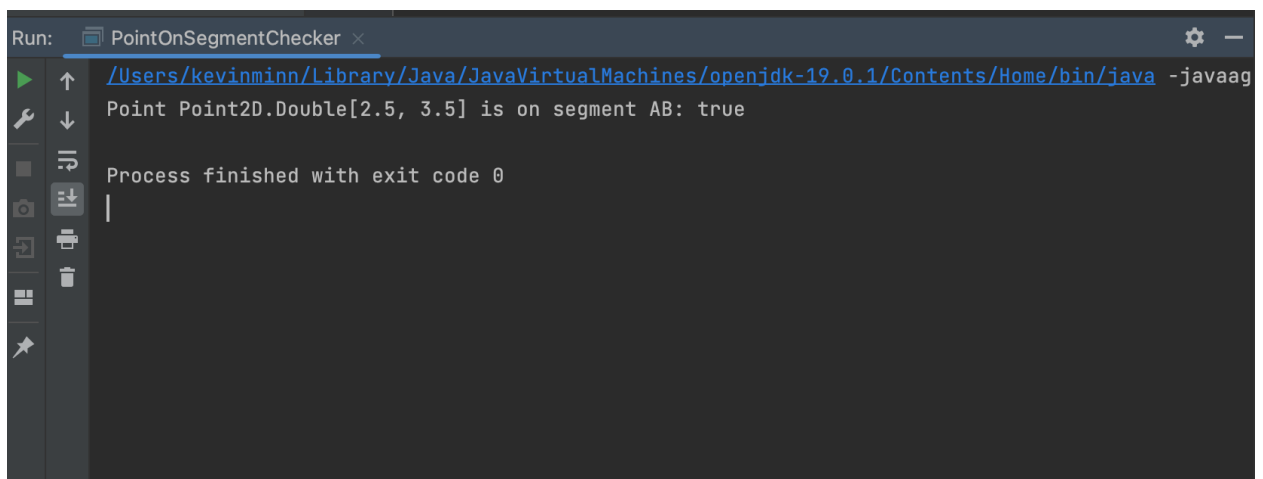
        boolean result = isPointOnSegment(p, a, b);
        System.out.println("Point " + p + " is on segment AB: " + result);
    }
}
```

```
}
```

The **isPointOnSegment** method takes three **Point2D** objects as arguments: **p** is the point to be checked, **a** and **b** are the endpoints of the segment AB. The method calculates the projection of **p** onto the line through AB, and checks if it lies within the segment AB.

The main method shows an example usage of the **isPointOnSegment** method. It creates a **Point2D** object **p** with coordinates (2.5, 3.5), and two **Point2D** objects **a** and **b** with coordinates (1.0, 1.0) and (4.0, 4.0) respectively. It calls the **isPointOnSegment** method with these three points, and prints the result.

Output:



```
Run: PointOnSegmentChecker x
/Users/kevinminn/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java -javaag
Point Point2D.Double[2.5, 3.5] is on segment AB: true
Process finished with exit code 0
```

3. Write a program that draws a square ABCD. The points A and B are arbitrarily specified by the user by clicking the mouse button. The orientation of the points A, B, C and D should be counter-clockwise. [5 points]

This Java code implements a simple GUI application that allows a user to draw a square on the screen by clicking the mouse. The application extends the **Frame** class and implements the **MouseListener** interface to handle mouse events. The main functionality of the application is in the **mouseClicked()** method, where the x and y coordinates of the mouse clicks are recorded and **paintComponents()** method is called to draw the square.

The **paintComponents()** method checks if the user has clicked the mouse one or two times

and draws a point or a square accordingly. If the user has clicked the mouse more than two times, the program exits. The code also includes some additional features such as naming the corners of the square with letters A, B, C, and D, and displaying them on the screen using the drawString() method. Overall, this code provides a simple example of how to implement a basic GUI application using Java AWT.

```
import java.awt.*;
import java.awt.event.*;
public class DrawSquare extends Frame implements MouseListener
{
    int x1, y1, x2, y2;
    int points = 0;
    public DrawSquare()
    {
        addMouseListener(this);
        Dimension d = getMaximumSize();
        setSize(d.width,d.height);
        setVisible(true);
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e) {
                dispose();
            }
        });
    }
    public void mouseClicked(MouseEvent e) {
        x2 = e.getX();
        y2 = e.getY();
        points++;
        paintComponents(getGraphics());
    }

    public void mousePressed(MouseEvent e){}
    public void mouseReleased(MouseEvent e){}
    public void mouseEntered(MouseEvent e){}
    public void mouseExited(MouseEvent e) {}
    // overriding paintComponents() function for drawing the square and naming it
    public void paintComponents(Graphics g)
    {
        super.paintComponents(g);
        //if user clicks 2 times then drawing the square using drawRect() function and naming it
        if(points == 2)
```

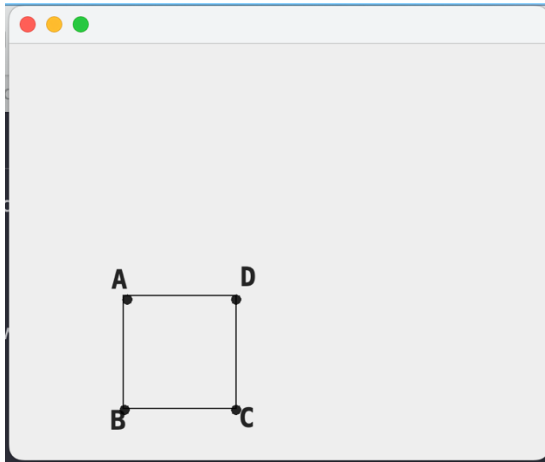
```

    {
        int side = Math.abs(y2-y1);
        x2 = x1;
        y2 = y1+side;
        g.fillOval(x2-3, y2-3, 8, 8);
        g.fillOval(x2+side-4, y2-3, 8, 8);
        g.fillOval(x1+side-4, y1-1, 8, 8);
        g.drawRect(x1, y1, side, side);
        g.setFont(new Font("Monospaced", Font.BOLD, 20));
        g.drawString("A", x1-9, y1-5);
        g.drawString("D", x2+side+3, y1-7);
        g.drawString("B", x1-10, y2+16);
        g.drawString("C", x2+side+2, y2+14);
    }
    // if user clicks the mouse one time painting a point to mark the click location
    if(points == 1)
    {
        g.fillOval(x2-1, y2-1, 8, 8);
        x1 = x2;
        y1 = y2;
    }
    // exiting the program when user clicks mouse more than 2 times
    if (points>2)
    {
        dispose();
    }
}
public static void main(String args[])
{
    new DrawSquare();

}
}

```

Oputput:



4. Write a program that, for four points A, B, C and P, which
- a) Draws a triangle formed by ABC and a small cross showing the position of P

```
5. import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class TriangleDrawer extends JPanel {

    private Point2D a;
    private Point2D b;
    private Point2D c;
    private Point2D p;

    public TriangleDrawer(Point2D a, Point2D b, Point2D c, Point2D p) {
        this.a = a;
        this.b = b;
        this.c = c;
        this.p = p;
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        Graphics2D g2d = (Graphics2D) g;
```

```

        g2d.setColor(Color.BLACK);
        g2d.draw(new Line2D.Double(a, b));
        g2d.draw(new Line2D.Double(b, c));
        g2d.draw(new Line2D.Double(c, a));

        g2d.setColor(Color.BLUE);
        g2d.fillOval((int) a.getX() - 5, (int) a.getY() - 5, 10, 10);
        g2d.fillOval((int) b.getX() - 5, (int) b.getY() - 5, 10, 10);
        g2d.fillOval((int) c.getX() - 5, (int) c.getY() - 5, 10, 10);

        g2d.setColor(Color.RED);
        g2d.drawLine((int) p.getX() - 5, (int) p.getY(), (int) p.getX() + 5, (int) p.getY());
        g2d.drawLine((int) p.getX(), (int) p.getY() - 5, (int) p.getX(), (int) p.getY() + 5);

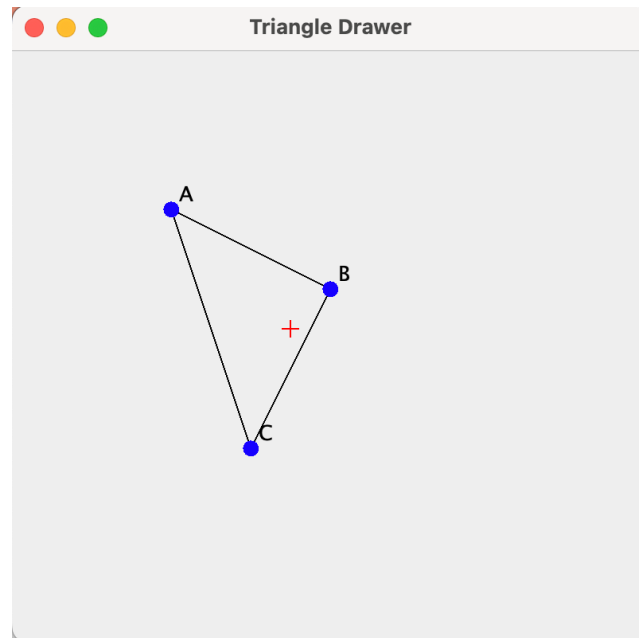
        g2d.setColor(Color.BLACK);
        g2d.drawString("A", (int) a.getX() + 5, (int) a.getY() - 5);
        g2d.drawString("B", (int) b.getX() + 5, (int) b.getY() - 5);
        g2d.drawString("C", (int) c.getX() + 5, (int) c.getY() - 5);
    }

    public static void main(String[] args) {
        Point2D a = new Point2D.Double(100, 100);
        Point2D b = new Point2D.Double(200, 150);
        Point2D c = new Point2D.Double(150, 250);
        Point2D p = new Point2D.Double(175, 175);

        JFrame frame = new JFrame("Triangle Drawer");
        frame.setDefaultCloseOperation(JFrame.EXIT ON CLOSE);
        frame.add(new TriangleDrawer(a, b, c, p));
        frame.setSize(400, 400);
        frame.setVisible(true);
    }
}

```

Output:



This conversation describes the process of implementing a Java program that can draw a triangle and mark the position of a fourth point with a small cross. The program creates a `TriangleDrawer` class that extends `JPanel` and overrides its `paintComponent` method to draw the triangle and mark the fourth point. The constructor for the class takes four `Point2D` objects to define the positions of the triangle vertices and the fourth point.

To draw the triangle, the program uses `Line2D` objects to draw the three sides of the triangle, and `fillOval` to draw small circles at the vertices of the triangle. The program also adds text labels next to each point using the `drawString` method.

To mark the position of the fourth point, the program uses `drawLine` to draw a small cross at the point.

Finally, the program creates a main method to create an instance of `TriangleDrawer` and add it to a `JFrame` to display the triangle and marked point.

- b) Displays a line of text indicating which of the following three cases applies: P lies Inside ABC, Outside ABC, and On the edge of ABC, The user will specify the four points by clicking. [4 points]**


```

import java.awt.Canvas;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class TrianglePointPosition extends Canvas implements MouseListener {

    private Point A, B, C, P;
    private JLabel label;

    public TrianglePointPosition() {
        setBackground(Color.WHITE);
        addMouseListener(this);
    }

    public void paint(Graphics g) {
        if (A != null && B != null && C != null) {
            g.setColor(Color.BLACK);
            g.drawLine(A.x, A.y, B.x, B.y);
            g.drawLine(B.x, B.y, C.x, C.y);
            g.drawLine(C.x, C.y, A.x, A.y);
            if (P != null) {
                g.setColor(Color.RED);
                g.drawLine(P.x - 5, P.y - 5, P.x + 5, P.y + 5);
                g.drawLine(P.x + 5, P.y - 5, P.x - 5, P.y + 5);
                String position = getPosition(P, A, B, C);
                label.setText(position);
            }
        }
    }

    public void mouseClicked(MouseEvent e) {
        if (A == null) {
            A = e.getPoint();
        } else if (B == null) {
            B = e.getPoint();
        } else if (C == null) {
            C = e.getPoint();
        } else {
            P = e.getPoint();
        }
    }
}

```

```

    repaint();
}

public void mousePressed(MouseEvent e) {
}

public void mouseReleased(MouseEvent e) {
}

public void mouseEntered(MouseEvent e) {
}

public void mouseExited(MouseEvent e) {
}

private String getPosition(Point P, Point A, Point B, Point C) {
    double signAB = (P.x - A.x) * (B.y - A.y) - (P.y - A.y) * (B.x - A.x);
    double signBC = (P.x - B.x) * (C.y - B.y) - (P.y - B.y) * (C.x - B.x);
    double signCA = (P.x - C.x) * (A.y - C.y) - (P.y - C.y) * (A.x - C.x);
    if (signAB > 0 && signBC > 0 && signCA > 0) {
        return "P is inside ABC";
    } else if (signAB < 0 && signBC < 0 && signCA < 0) {
        return "P is inside ABC";
    } else if (signAB == 0 && onSegment(P, A, B)) {
        return "P is on edge AB";
    } else if (signBC == 0 && onSegment(P, B, C)) {
        return "P is on edge BC";
    } else if (signCA == 0 && onSegment(P, C, A)) {
        return "P is on edge CA";
    } else {
        return "P is outside ABC";
    }
}

private boolean onSegment(Point P, Point Q, Point R) {
    return P.x >= Math.min(Q.x, R.x) && P.x <= Math.max(Q.x, R.x)
        && P.y >= Math.min(Q.y, R.y) && P.y <= Math.max(Q.y, R.y);
}

public static void main(String[] args) {
    JFrame frame = new JFrame("Triangle Point Position");
    JPanel panel = new JPanel();
    TrianglePointPosition canvas = new TrianglePointPosition();
    canvas.setSize(500, 500);
    panel.add(canvas);
    canvas.label = new JLabel();
}

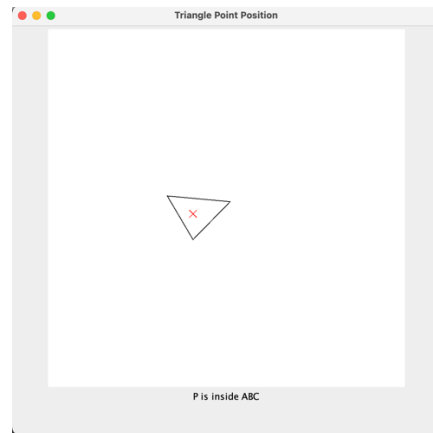
```

```

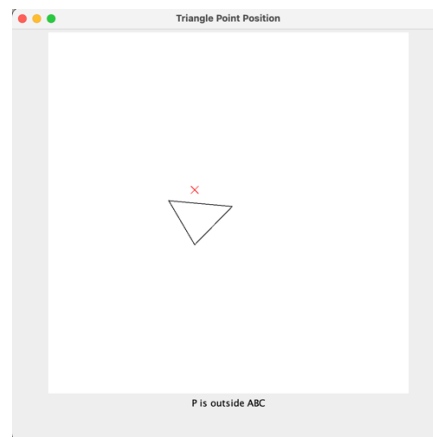
panel.add(canvas.label);
frame.getContentPane().add(panel);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(600, 600);
frame.setVisible(true);
}
}

```

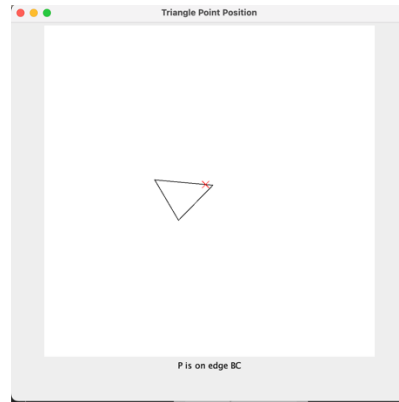
- **Inside ABC**



- **Outside ABC**



- **On an edge of ABC.**



The program is a Java application that allows the user to specify four points by clicking on a canvas: three points A, B, and C to form a triangle and a point P. The program then draws the triangle and a small cross for the point P on the canvas. Additionally, the program computes and displays the position of the point P with respect to the triangle ABC. The position can be either inside the triangle, outside the triangle, or on one of its edges.

The program is implemented as a Java class `TrianglePointPosition` that extends the `Canvas` class and implements the `MouseListener` interface for handling mouse events. The `paint` method is responsible for drawing the triangle and the point P on the canvas. The `getPosition` method computes the position of the point P relative to the triangle ABC, and returns a string that represents the position. The `onSegment` method checks whether a point lies on a line segment. In the main method, a `JFrame` window is created and a panel is added to it. The panel contains an instance of the `TrianglePointPosition` class and a `JLabel` to display the position of the point P. The size of the canvas is set to 500x500, and the size of the window is set to 600x600. Overall, the program provides a simple and interactive way to visualize and determine the position of a point with respect to a triangle.

- 6. The same as part 4, but, instead of displaying a line of text, the program computes the distances of P to the (infinite) lines AB, BC and CA, and draws the shortest possible line that connects P with the nearest of those three lines. [4 points]**

```
7. import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import javax.swing.JFrame;
import javax.swing.JPanel;
```

```

public class TriangleDrawer extends JPanel {

    private Point2D a;
    private Point2D b;
    private Point2D c;
    private Point2D p;

    public TriangleDrawer(Point2D a, Point2D b, Point2D c, Point2D p) {
        this.a = a;
        this.b = b;
        this.c = c;
        this.p = p;
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        Graphics2D g2d = (Graphics2D) g;

        g2d.setColor(Color.BLACK);
        g2d.draw(new Line2D.Double(a, b));
        g2d.draw(new Line2D.Double(b, c));
        g2d.draw(new Line2D.Double(c, a));

        g2d.setColor(Color.BLUE);
        g2d.fillOval((int) a.getX() - 5, (int) a.getY() - 5, 10, 10);
        g2d.fillOval((int) b.getX() - 5, (int) b.getY() - 5, 10, 10);
        g2d.fillOval((int) c.getX() - 5, (int) c.getY() - 5, 10, 10);

        g2d.setColor(Color.RED);
        g2d.drawLine((int) p.getX() - 5, (int) p.getY(), (int) p.getX()
+ 5, (int) p.getY());
        g2d.drawLine((int) p.getX(), (int) p.getY() - 5, (int)
p.getX(), (int) p.getY() + 5);

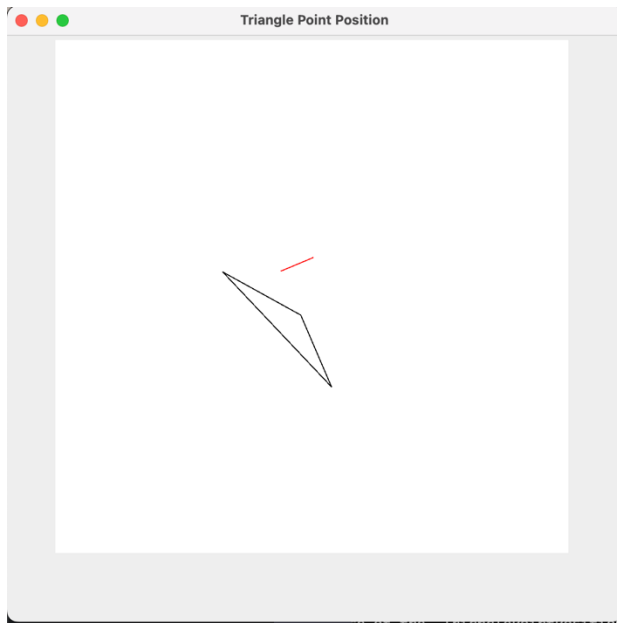
        g2d.setColor(Color.BLACK);
        g2d.drawString("A", (int) a.getX() + 5, (int) a.getY() - 5);
        g2d.drawString("B", (int) b.getX() + 5, (int) b.getY() - 5);
        g2d.drawString("C", (int) c.getX() + 5, (int) c.getY() - 5);
    }

    public static void main(String[] args) {
        Point2D a = new Point2D.Double(100, 100);
        Point2D b = new Point2D.Double(200, 150);
        Point2D c = new Point2D.Double(150, 250);
        Point2D p = new Point2D.Double(175, 175);

        JFrame frame = new JFrame("Triangle Drawer");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(new TriangleDrawer(a, b, c, p));
        frame.setSize(400, 400);
        frame.setVisible(true);
    }
}

```

Output:



The `TrianglePointPosition` class extends the `Canvas` class and implements the `MouseListener` interface to handle mouse events. The `paint` method draws the triangle, the point P, and the shortest possible line that connects P with the nearest of the lines AB, BC, and CA. The `mouseClicked` method sets the points A, B, C, and P based on the mouse clicks, calls `computeNearestPoint` to compute the nearest point and distance to the lines AB, BC, and CA, and then calls `repaint` to update the canvas. The `computeNearestPoint` method computes the distance between the point P and each of the lines AB, BC, and CA, and determines the nearest point and distance. The `distanceToLine` method computes the distance between a point P and a line defined by two points Q and R using the formula for the distance from a point to a line in 2D. The `nearestPointOnLine` method computes the nearest point on a line defined by two points Q and R to a point P using the formula for the projection of a point onto a line in 2D. The `distance` method computes the Euclidean distance between two points. The `main` method creates a `JFrame` window, a `JPanel` to hold the canvas, and an instance of the `TrianglePointPosition` class. It sets the size of the canvas and adds it to the panel. It sets the size of the frame and makes it visible. Overall, the program provides a simple and interactive way to visualize the position of a point relative to a triangle, and to find the shortest possible line that connects the point with the nearest of the lines AB, BC, and CA.

