Praktische Softwaretechnologie

Károly Bósa
(Karoly.Bosa@jku.at)

Research Institute for Symbolic Computation
(RISC)

Exercise 12: RotatedMap – 1st Part: Fields

Karoly.Bosa@jku.at

Modify the class ***Map*** such that it displays the map rotated around x axis with the given value.

```
import java.io.FileInputStream;
import java.io.IOException;
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class RotatedMap extends Canvas  {
        private final static int SQUARE_SIDE = 201;
        private final static int SQUARE_SIZE = SQUARE_SIDE * SQUARE_SIDE;

        private final static int oX = SQUARE_SIDE/2;  //X coordinate of the origin
        private final static int oY = SQUARE_SIDE/2;  //Y coordinate of the origin

        private int[] map;
        private int[] rotatedYCoordinates;
        private String filename;
        private double[][] rotMatrix;
        private double angle;
        private Frame frm;
```

Exercise 12: RotatedMap – 1st Part: Methods

Karoly.Bosa@jku.at

Modify the class ***Map*** such that it displays the map rotated around x axis with given value.

```
public RotatedMap(String filename, int angle,) throws IOException {… }

public void readMapFromFile() throws IOException {… }

private void computeRotationMatrix() {… }

private  int computeYCoordinate(int x, int y, int z) {… }
        //Since we rotate the map only around X axis, we should notice that we need
        //to compute only the rotated Y coordinates!!!

public void paint(Graphics g) {… }

public static void main(String[] args) throws IOException {… }
}
```

Exercise 12: RotatedMap – 1st Part: Methods

Karoly.Bosa@jku.at

Modify the method ***main*** such that it expect a command line parameter.

```
public static void main(String[] args) throws IOException {
    int angle = 40;
    if (args.length > 0) angle = Integer.parseInt(args[0]);
    RotatedMap map = new RotatedMap("mountains.map", angle);
}

public RotatedMap(String filename, int angle) throws IOException {
    this.filename = filename;
    this.angle = angle * Math.PI/180;
    map = new int[SQUARE_SIZE];
    rotatedYCoordinates = new int[SQUARE_SIZE];
    rotMatrix = new double[3][3];
    readMapFromFile();
    frm = new Frame(filename);
    frm.add(this);
    frm.setSize(zoom*SQUARE_SIDE+50, zoom*SQUARE_SIDE+175);
    frm.setVisible(true);
    frm.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent we){
                System.exit(0);
        }
    });
}
```

Exercise 12: RotatedMap – 1st part: Methods

Karoly.Bosa@jku.at

```
private void computeRotationMatrix() {
        double co = Math.cos(angle);
        double si = Math.sin(angle);
        rotMatrix[0][0] = 1;
        rotMatrix[1][0] = 0;
        rotMatrix[2][0] = 0;
        rotMatrix[0][1] = 0;
        rotMatrix[1][1] = co;
        rotMatrix[2][1] = -si;
        rotMatrix[0][2] = 0;
        rotMatrix[1][2] = si;
        rotMatrix[2][2] = co;
}

//Since we rotate the map only around X axis, we should notice that we need
//to compute only the rotated Y coordinates!!!
private  int computeYCoordinate(int x, int y, int z) {
        int rotatedY;
        rotatedY = (int)Math.round(rotMatrix[0][1]*x+rotMatrix[1][1]*y+rotMatrix[2][1]*z);
        return rotatedY;
}
```

Exercise 12: RotatedMap – 1st Part: Methods

Karoly.Bosa@jku.at

-Use the method

java.awt.Graphics.fillPolygon(int[4] xPoints, int[4] yPoints, 4)

instead of

java.awt.Graphics.drawLine(int x1, int y1, int x2, int y2)

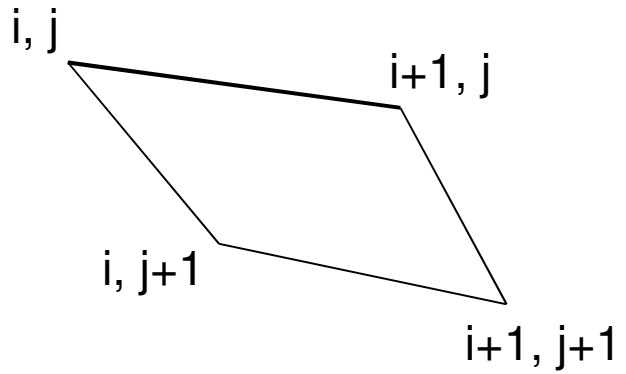
for displaying the map.

```
public void paint(Graphics g) {
    computeRotationMatrix();

    //compute rotated Y coordinates
    for (int j=0; j< SQUARE_SIDE; j++) {
        for (int i=0; i< SQUARE_SIDE; i++) {
            int index = i*SQUARE_SIDE+j;
            rotatedYCoordinates[index] = oY+computeYCoordinate(i-oX, j-oY, map[index]/2);
            // altitude divided by 2 --> nicer view
        }
    }

    // …method Paint() is continued on the next slide…
}
```

Exercise 12: RotatedMap – 1st Part: Methods

Karoly.Bosa@jku.at

```
// … continuation of the method Paint() from the previous slide

//display rotated map
for (int j=0; j< SQUARE_SIDE-1; j++) {
    for (int i=0; i< SQUARE_SIDE-1; i++) {
        int index = i*SQUARE_SIDE+j;
        if (map[index] < 60) g.setColor(new Color(0+map[index]*3, 0+map[index]*3, 128));
        else if (map[index] < 150) g.setColor(new Color(0+map[index]-60, 128, 0+map[index]-60));
        else g.setColor(new Color(128-map[index]/3, 203-map[index]/2, 150-map[index]/2));
        int[] xPoints = new int[4];          // g.drawLine(20+i,120+rotatedYCoordinates[index], …);
        int[] yPoints = new int[4];
        xPoints[0] = 20+i;
        yPoints[0] = 120+rotatedYCoordinates[index];
        index = (i+1)*SQUARE_SIDE+j;
        xPoints[1] = 20+i+1;
        yPoints[1] = 120+rotatedYCoordinates[index];
        index = (i+1)*SQUARE_SIDE+j+1;
        xPoints[2] = 20+i+1;
        yPoints[2] = 120+rotatedYCoordinates[index];
        index = i*SQUARE_SIDE+j+1;
        xPoints[3] = 20+i;
        yPoints[3] = 120+rotatedYCoordinates[index];
        g.fillPolygon(xPoints, yPoints, 4);
} } }
```



Exercise 12: RotatedMap – 2nd Part: Fields

Karoly.Bosa@jku.at

Implement interface ***mouseMotionListener*** in the class ***Map***, such that if you drag and move the mouse up or down over the window, the rotation angle of the displayed map will be increased or decreased correspondingly (so, you should be able to rotate the map around the X axis with the help of the mouse).

```
…
import java.awt.event.MouseMotionListener;
import java.awt.event.MouseEvent;

public class RotatedMap extends Canvas implements MouseMotionListener {
    …
    private final static double ANGLE_STEP = 10 * Math.PI/180;
    …
    private int mouseY = 0;
    public RotatedMap(String filename, int angle, int zoom) throws IOException {
            …
            addMouseMotionListener(this);
    }
    …
    public void mouseMoved(MouseEvent e) {}  //empty method

    public void mouseDragged(MouseEvent e) {… }
```

Exercise 12: RotatedMap – 2nd Part: Methods

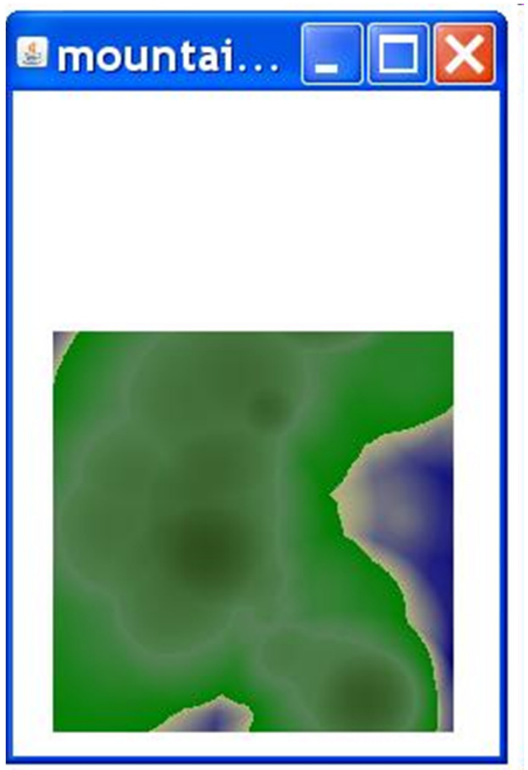
Karoly.Bosa@jku.at

```
public void mouseMoved(MouseEvent e) {
}

public void mouseDragged(MouseEvent e) {
        if (e.getY() > mouseY && angle > 0) {
                angle-=ANGLE_STEP;
                repaint();
        }
        if (e.getY() < mouseY && angle < Math.PI/2) {  // 90*Math.PI/180
                angle+=ANGLE_STEP;
                repaint();
        }
        mouseY = e.getY();
}
```

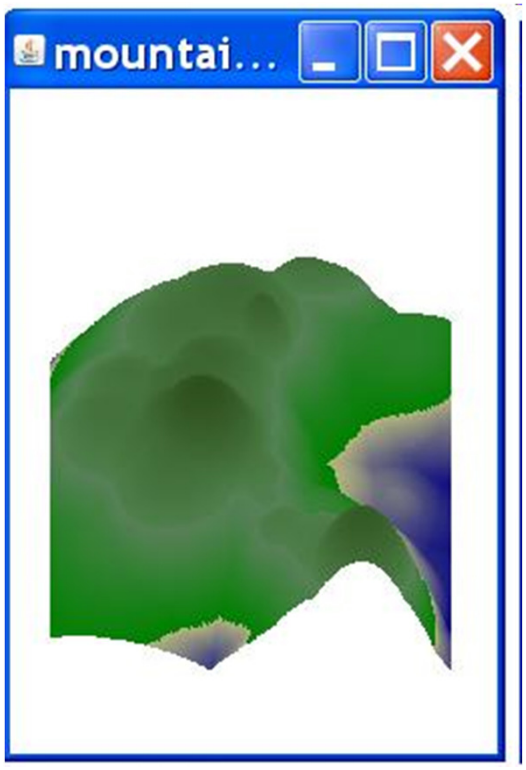
Exercise 12: RotatedMap – Output

Karoly.Bosa@jku.at

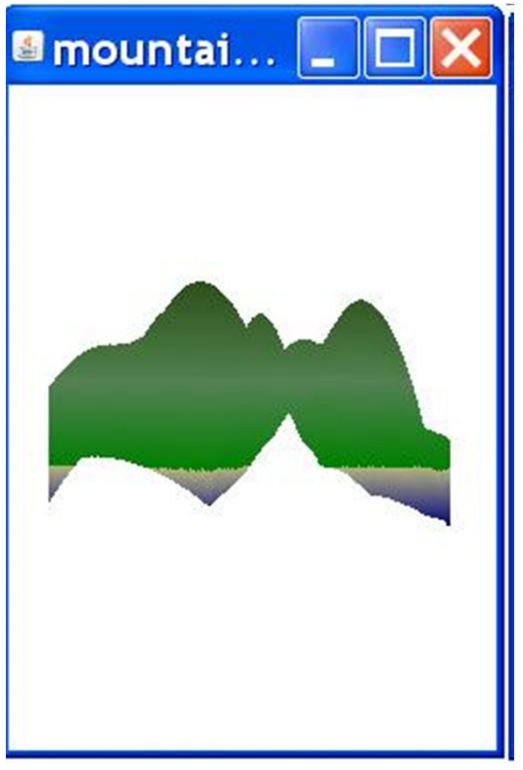
java RotatedMap 0



java RotatedMap 40



java RotatedMap 90



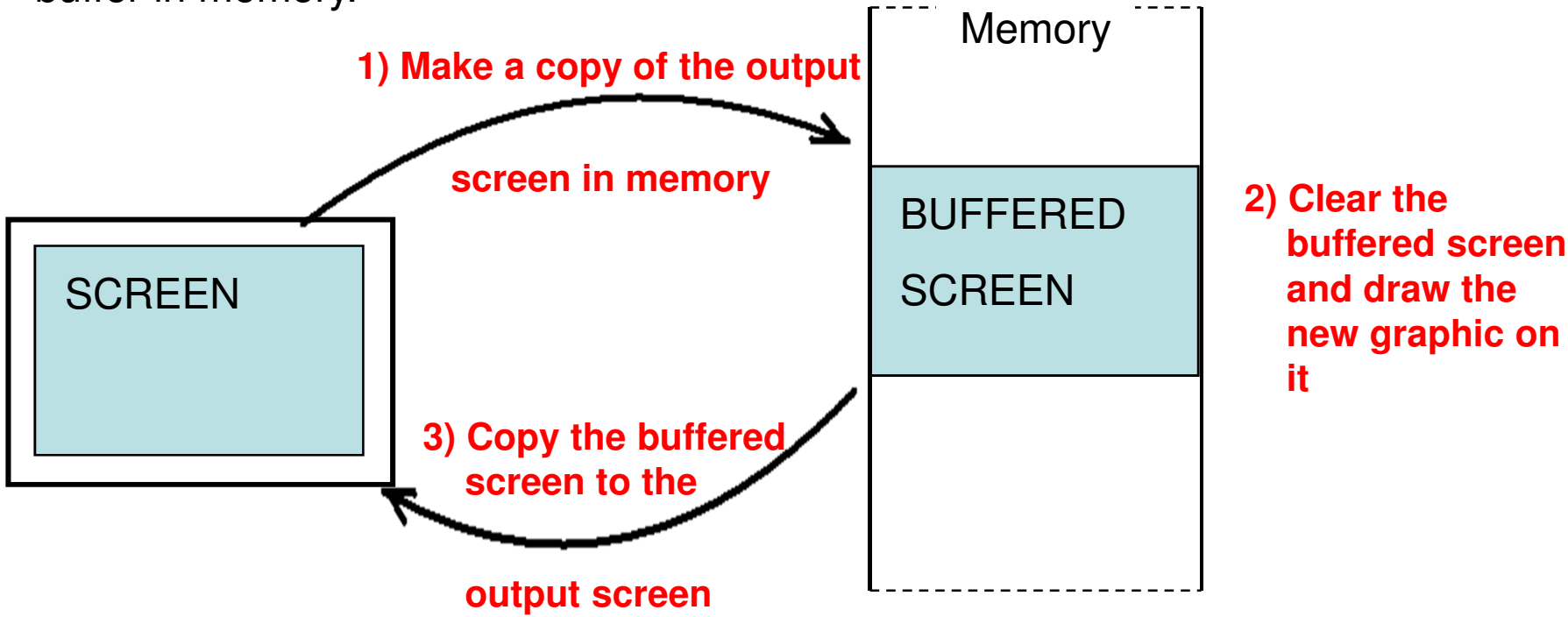
Addressing the Flickering Issue

Karoly.Bosa@jku.at

Now we can rotate a map with the help of the mouse, but the animation suffers some kind of flickering. How could we avoid this?

1) First we should override the screen update method to simply call the paint method and NOT clear the screen.

2) Then we will have to use a technique that holds a copy of the Java screen in a buffer in memory.



Buffering Screen in Practice

Karoly.Bosa@jku.at

```
import java.awt.*;
…
public class RotatedMap extends Canvas implements MouseMotionListener {
…

Image scrnBuf;
Graphics scrnG;

public void update(Graphics g) {
        paint(g);
}

public void paint(Graphics g) {… }
```

Buffering Screen in Practice – Paint()

Karoly.Bosa@jku.at

```
public void paint(Graphics g) {
    if (scrnG == null) {
        scrnBuf = createImage(SQUARE_SIDE+50, SQUARE_SIDE+175);
        scrnG = scrnBuf.getGraphics();
    }
    scrnG.clearRect(0, 0, SQUARE_SIDE+50, SQUARE_SIDE+175);
    computeRotationMatrix();

    //compute rotated Y coordinates
    for (int j=0; j< SQUARE_SIDE; j++) {
        for (int i=0; i< SQUARE_SIDE; i++) {
            int index = i*SQUARE_SIDE+j;
            rotatedYCoordinates[index] = oY+computeYCoordinate(i-oX, j-oY, map[index]/2);
            }
    }

    // …method Paint() is continued on the next slide…
}
```

Buffering Screen in Practice – Paint()

Karoly.Bosa@jku.at

```
    // … continuation of the method Paint() from the previous slide
    for (int j=0; j< SQUARE_SIDE-1; j++) {
        for (int i=0; i< SQUARE_SIDE-1; i++) {
            int index = i*SQUARE_SIDE+j;
            if (map[index] < 60) scrnG.setColor(new Color(0+map[index]*3, 0+map[index]*3, 128));
            else if (map[index] < 150) scrnG.setColor(new Color(0+map[index]-60, 128, 0+map[index]-60));
            else scrnG.setColor(new Color(128-map[index]/3, 203-map[index]/2, 150-map[index]/2));

                int[] xPoints = new int[4];
                int[] yPoints = new int[4];
                xPoints[0] = 20+i;
                yPoints[0] = 120+rotatedYCoordinates[index];
                index = (i+1)*SQUARE_SIDE+j;
                xPoints[1] = 20+i+1;
                yPoints[1] = 120+rotatedYCoordinates[index];
                index = (i+1)*SQUARE_SIDE+j+1;
                xPoints[2] = 20+i+1;
                yPoints[2] = 120+rotatedYCoordinates[index];
                index = i*SQUARE_SIDE+j+1;
                xPoints[3] = 20+i;
                yPoints[3] = 120+rotatedYCoordinates[index];
                scrnG.fillPolygon(xPoints, yPoints, 4);
            }
        }
        //object g is assigned to the output screen, object scrnG is assigned to the buffered screen
        g.drawImage(scrnBuf, 0, 0, this);
}
```

Buffering Screen in Practice

Karoly.Bosa@jku.at

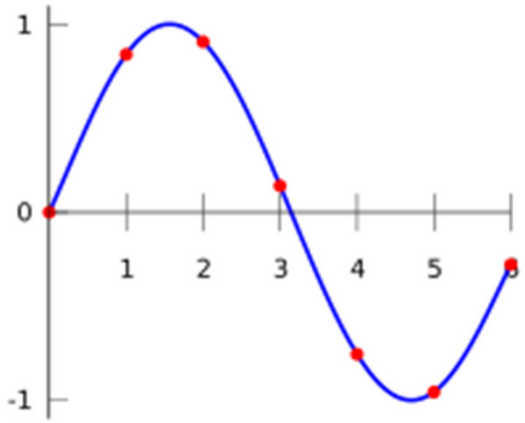
After these modification we can rotate the map in the window smoothly (no flickering anymore).

Lagrange Interpolation

Karoly.Bosa@jku.at

Interpolation is a method of constructing new data points within the range of a discrete set of known data points.

- **Linear interpolation**
- **Polynomial interpolation**
  - **Lagrange Form**
- **Spline interpolation**
- **Gaussian processes**
- **…**



Lagrange Interpolation

Karoly.Bosa@jku.at

Polynomial interpolation: if we have *n* data points, there is exactly one polynomial of degree at most *n*–1 going through all the data points.

Lagrange form:

Given a set of *k* + 1 data points

$$(x_0, y_0), \ldots, (x_k, y_k)$$

where no two *xj* are the same, the **interpolation polynomial in the Lagrange form** is

$$L(x) := \sum_{j=0}^{k} y_j \ell_j(x),$$

where

$$\ell_j(x) := \prod_{i=0,\, i \neq j}^{k} \frac{x - x_i}{x_j - x_i} = \frac{(x - x_0)}{(x_j - x_0)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_k)}{(x_j - x_k)}.$$

Lagrange Interpolation

Karoly.Bosa@jku.at

```
private int lagrangePolynomial (Vector<Integer> x, Vector<Integer> y, int desiredX)  {
    float desiredY = 0;

    for (int j = 0; j < x.size(); ++j) {
        float weight = 1;

        for (int i = 0; i < x.size(); ++i) {
            // The j-th turn has to be skipped
            if (i != j) {
                float xi = x.elementAt(i).floatValue();
                float xj = x.elementAt(j).floatValue();
                weight *= (desiredX - xi) / (xj - xi);
            }
        }
        float yj = y.elementAt(j).floatValue();
        desiredY += weight * yj;
    }
    return Math.round(desiredY);
}
```

$$L(x) := \sum_{j=0}^{k} y_j \ell_j(x) \qquad \ell_j(x) := \prod_{i=0, i\neq j}^{k} \frac{x - x_i}{x_j - x_i} = \frac{(x - x_0)}{(x_j - x_0)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_k)}{(x_j - x_k)}$$

Lagrange Interpolation – Data fields

Karoly.Bosa@jku.at

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;

public class LagrangeInterpolation extends Applet implements ActionListener {
        private static final int FRAMEX = 30;
        private static final int FRAMEY = 30;
        private int WINDOW_XSIZE = 800;
        private int WINDOW_YSIZE = 500;
        private int oX;  //X coordinate of the origin
        private int oY;  //Y coordinate of the origin
        private Image scrnBuf;
        private Graphics scrnG;
        private Button incPoint;
        private Button decPoint;
        private int nrOfPoints = 5;
        private Vector <Integer>listOfPointsX;   //x coordinates of the given points
        private Vector <Integer> listOfPointsY;  //y coordinates of the given points

        private int draggedPoint = -1;  // stores the index of the point dragged by the mouse
        private boolean mustDraw = true; // if the curve changed it should be redraw in the buffer
        private Mouse mouse; //See later…
```

Lagrange Interpolation - Methods

Karoly.Bosa@jku.at

```
public void init() {…}

private void starter() {…}

private int lagrangePolynomial (Vector<Integer> x, Vector<Integer> y, int desiredX)  {
…}

public void paint(Graphics g) {… }

public void update(Graphics g) { paint(g); }

public void actionPerformed(ActionEvent e) {… }

/*****************Embedded Class*******************************/
private class Mouse extends MouseAdapter {
    public void mousePressed(MouseEvent e) {… }

    public void mouseReleased(MouseEvent e) {… }

    public void mouseDragged(MouseEvent e) {… }
}
```

Lagrange Interpolation – Init method

Karoly.Bosa@jku.at

```
public void init() {
    WINDOW_XSIZE = Integer.parseInt(this.getParameter("Width"));
    WINDOW_YSIZE = Integer.parseInt(this.getParameter("Height"));
    oX = WINDOW_XSIZE/2;
    oY = WINDOW_YSIZE/2;
    listOfPointsX = new Vector<Integer>(8,5);
    listOfPointsY = new Vector<Integer>(8,5);
    starter();

    scrnBuf = createImage(WINDOW_XSIZE, WINDOW_YSIZE);
    scrnG = scrnBuf.getGraphics();

    incPoint = new Button("Add Point");
    decPoint = new Button("Subtract Point");
    add(incPoint);
    add(decPoint);
    incPoint.addActionListener(this);
    decPoint.addActionListener(this);

    mouse = new Mouse();
    addMouseListener(mouse);
    addMouseMotionListener(mouse);
}
```

Lagrange Interpolation – Starter method

Karoly.Bosa@jku.at

```
private void starter() {
   int initLength = WINDOW_XSIZE-2*FRAMEX;
   int stepSize = initLength/(nrOfPoints-1);

   listOfPointsX.removeAllElements();
   listOfPointsY.removeAllElements();

   for (int i = 0; i < nrOfPoints; i++) {
      listOfPointsX.add(new Integer(i*stepSize-oX+FRAMEX));
      listOfPointsY.add(new Integer(0));
   }
}
```

- This method initializes the coordinates of the given points (It places the given points on X axis of the coordinate system).
- The method is called when the applet is started and when the number of the given points are changed (incremented/decremented)

Lagrange Interpolation – Paint method

Karoly.Bosa@jku.at

```
public void paint(Graphics g) {
    if  (mustDraw) {
       scrnG.clearRect(0, 0, WINDOW_XSIZE, WINDOW_YSIZE);
       scrnG.setColor(Color.black);
       scrnG.setFont(new Font("TimesRoman",Font.BOLD, 18));
       scrnG.drawString("Drag a point with the mouse!", 20, 20);
       scrnG.fillRect(oX-1, FRAMEY, 2, WINDOW_YSIZE-2*FRAMEY);  //Y axis
       scrnG.fillRect(FRAMEX, oY-1,WINDOW_XSIZE-2*FRAMEX, 2);   //X axis
       for (int i = 0; i < listOfPointsX.size(); i++) {
          scrnG.fillRect(oX+listOfPointsX.elementAt(i).intValue()-3,
          oY+listOfPointsY.elementAt(i).intValue()-3, 6, 6);
       }
       int start_point_x = FRAMEX-oX;
       int end_point_x = WINDOW_XSIZE-FRAMEX-oX;
       int y1 = lagrangePolynomial(listOfPointsX, listOfPointsY, start_point_x);
       for (int j = start_point_x+1; j <= end_point_x; j++) {
          int y2 = lagrangePolynomial(listOfPointsX, listOfPointsY, j);
          scrnG.drawLine(oX+j-1, oY+y1, oX+j, oY+y2);
          y1 = y2;
       }
       mustDraw = false;
    }
    g.drawImage(scrnBuf, 0, 0, this);
}
```

Lagrange Interpolation – Paint method

Karoly.Bosa@jku.at

```
public void paint(Graphics g) {
    if  (mustDraw) {
       scrnG.clearRect(0, 0, WINDOW_XSIZE, WINDOW_YSIZE);
       scrnG.setColor(Color.black);
       scrnG.setFont(new Font("TimesRoman",Font.BOLD, 18));
       scrnG.drawString("Drag a point with the mouse!", 20, 20);
       scrnG.fillRect(oX-1, FRAMEY, 2, WINDOW_YSIZE-2*FRAMEY);  //Y axis
       scrnG.fillRect(FRAMEX, oY-1,WINDOW_XSIZE-2*FRAMEX, 2);   //X axis
       for (int i = 0; i < listOfPointsX.size(); i++) {
          scrnG.fillRect(oX+listOfPointsX.elementAt(i).intValue()-3,
          oY+listOfPointsY.elementAt(i).intValue()-3, 6, 6);
       }
       int start_point_x = FRAMEX-oX;
       int end_point_x = WINDOW_XSIZE-FRAMEX-oX;
       int y1 = lagrangePolynomial(listOfPointsX, listOfPointsY, start_point_x);
       for (int j = start_point_x+1; j <= end_point_x; j++) {
          int y2 = lagrangePolynomial(listOfPointsX, listOfPointsY, j);
          scrnG.drawLine(oX+j-1, oY+y1, oX+j, oY+y2);
          y1 = y2;
       }
       mustDraw = false;
    }
    g.drawImage(scrnBuf, 0, 0, this);
}
```

Lagrange Interpolation - actionPerformed

Karoly.Bosa@jku.at

```
public void actionPerformed(ActionEvent e) {
      if (e.getSource().equals(incPoint) && nrOfPoints< 20) {
         nrOfPoints++;
         starter();
         mustDraw = true;
         repaint();
      }
      else if (e.getSource().equals(decPoint) && nrOfPoints > 3) {
         nrOfPoints--;
         starter();
         mustDraw = true;
         repaint();
      }
   }
```

Lagrange Interpolation – Embeded Class

Karoly.Bosa@jku.at

```
private class Mouse extends MouseAdapter {
   public void mousePressed(MouseEvent e) {
      int mouseX = e.getX();
      int mouseY = e.getY();
      for (int i = 0; i < listOfPointsX.size(); i++) {
         int pointX = oX+listOfPointsX.elementAt(i).intValue();
         int pointY = oY+listOfPointsY.elementAt(i).intValue();
         if ((mouseX >= pointX-4 && mouseX <= pointX+4)
             &&
             (mouseY >= pointY-4 && mouseY <= pointY+4)) {
            draggedPoint = i;
            break;
         }
      }
   }

   public void mouseReleased(MouseEvent e) {
      draggedPoint = -1;
   }
```

Lagrange Interpolation

Karoly.Bosa@jku.at

```
public void mouseDragged(MouseEvent e) {
    if (draggedPoint >= 0) {
    if ((draggedPoint == 0
        ||
       listOfPointsX.elementAt(draggedPoint-1).intValue()+oX < e.getX())
    &&
       (draggedPoint== nrOfPoints-1
        ||
       listOfPointsX.elementAt(draggedPoint+1).intValue()+oX > e.getX())) {
              listOfPointsX.setElementAt(new Integer(e.getX()-oX), draggedPoint);
              listOfPointsY.setElementAt(new Integer(e.getY()-oY), draggedPoint);
              mustDraw = true;
              repaint();
    }
    }
}
```

Displaying the Polynomial of the Depicted Curve

Karoly.Bosa@jku.at

Lagrange form:

$$L(x) := \sum_{j=0}^{k} y_j \ell_j(x).$$

$$\ell_j(x) := \prod_{i=0, i\neq j}^{k} \frac{x - x_i}{x_j - x_i} = \frac{(x - x_0)}{(x_j - x_0)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_k)}{(x_j - x_k)}.$$

For getting the polynomial of the depicted curve, we should perform the polynomial multiplications and additions defined by the Lagrange form above.

Interpolation Combined with our Polynomial Library

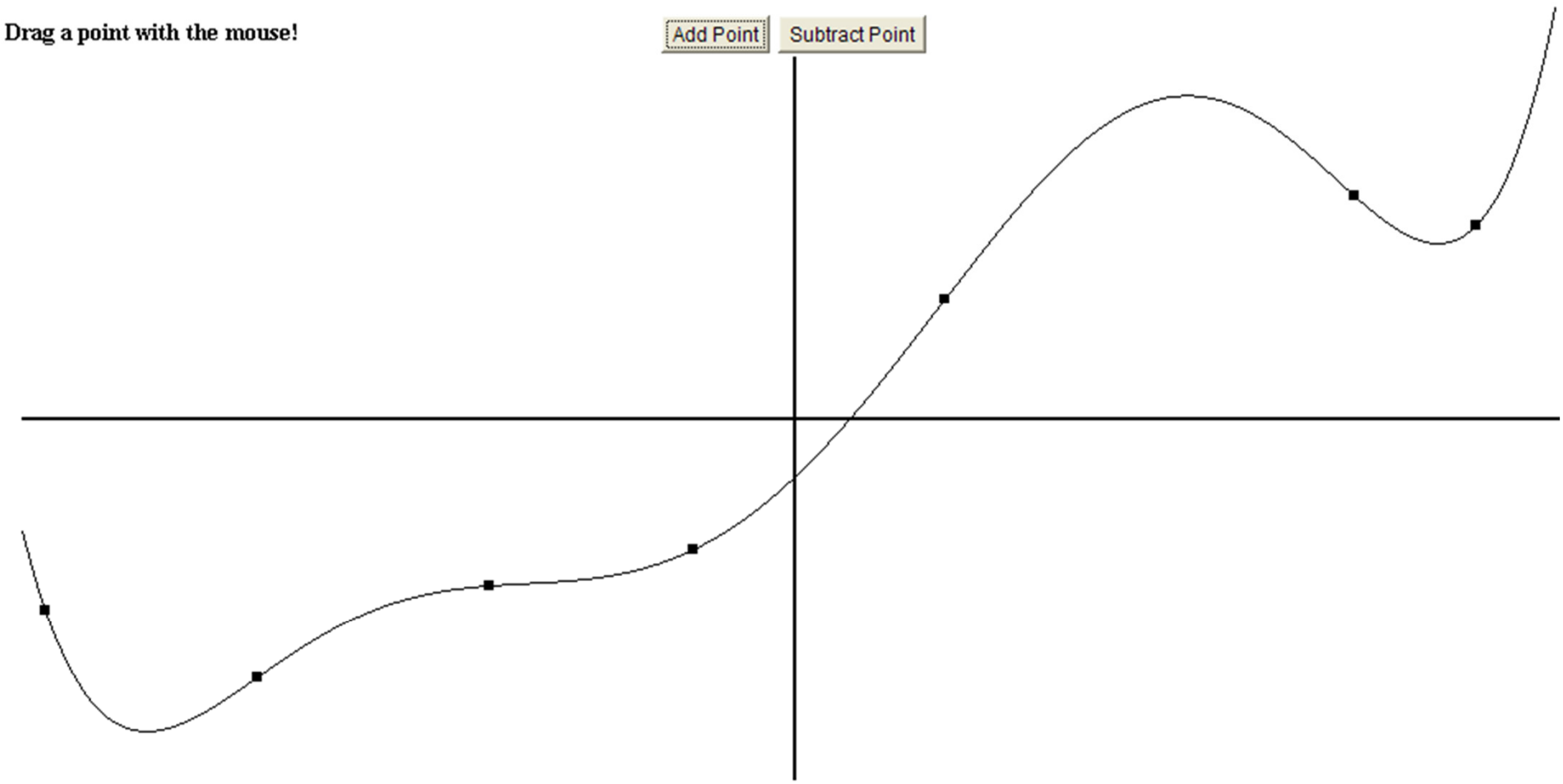
Karoly.Bosa@jku.at

```
private String polynomialCalc (Vector<Integer> x, Vector<Integer> y)  {
    List<Map<Integer, Double>> forAdd = new LinkedList<Map<Integer,Double>>();
     for (int j = 0; j < x.size(); ++j) {
            List<Map<Integer, Double>> forMultip = new LinkedList<Map<Integer,Double>>();
            for (int i = 0; i < x.size(); ++i) {
                // The i-th turn has to be skipped
                if (i != j) {
                        double xi = x.elementAt(i).floatValue();
                        double xj = x.elementAt(j).floatValue();
                         Map<Integer, Double> pol = new TreeMap<Integer, Double>();
                         pol.put(1, 1.0/(xj - xi));  pol.put(0, -xi/(xj - xi));  forMultip.add(pol);
                } //if
            } //for
            double yj = y.elementAt(j).floatValue();
            Map<Integer, Double> pol = new TreeMap<Integer, Double>();
            pol.put(0, yj); forMultip.add(pol);
            forAdd.add(Polynomial.multiplication(forMultip));
    } //for
    return Polynomial.toString(Polynomial.addtion(forAdd)); }
```

$$\sum_{j=0}^{k} y_j \left(\frac{1}{x_j - x_0} x - \frac{x_0}{x_j - x_0} \right) \left(\frac{1}{x_j - x_1} x - \frac{x_1}{x_j - x_1} \right) \ldots \left(\frac{1}{x_j - x_k} x - \frac{x_k}{x_j - x_k} \right)$$

Interpolation Combined with our Polinomial Library

Karoly.Bosa@jku.at



The Polynomial (the precision of the coefficients is max. 30 decimals):

+ 34.7642063367691560 - 0.9466755149154360x - 0.0035517431001901084x^2 +
0.0000063228670109093910x^3 + 0.0000000470214421440345200x^4 -
0.0000000000170911218907149300x^5 - 0.0000000000000153765236256419610x^6

Java3D

Karoly.Bosa@jku.at

- Java 3D basically is a high level 3D application programming interface for the Java platform. It functions on the OpenGL or Direct3D.
- The programmer works with high-level constructs for creating and manipulating 3D geometric objects...
- Java3D is an optional Java package (NOT PART of JDK 6) available free from Sun's web site.
- Everything needed can be obtained from: *https://java3d.dev.java.net/*
- Documentation and tutorials are also available for downloading.