Praktische Softwaretechnologie

Károly Bósa
(Karoly.Bosa@jku.at)

Research Institute for Symbolic Computation
(RISC)

A Useful Applet Example

Karoly.Bosa@jku.at

An active Menu for HTML Pages

Problems:

- **Parameterization of the applet**
- **Download the images**
- **Drawing the images**
- **Handling mouse events**
- **Download the destination pages of the links**

Active Menu Applet: index.html

Karoly.Bosa@jku.at

```
<HTML><BODY BACKGROUND="bg.jpg">
<HR>
<applet code="MappingImageWithText.class"
width=385 height=445>
<param name=crewImage value="crew.jpg">
<param name=numberOfCrew value="7">

<param name=name0 value="First Name">
<param name=name1 value="Second Name">
<param name=name2 value="Third Name">
<param name=name3 value="Fourth Name">
<param name=name4 value="Fifth Name">
<param name=name5 value="Sixth Name">
<param name=name6 value="Seventh Name">

<param name=url0 value="page.html">
<param name=url1 value="page.html">
<param name=url2 value="page.html">
<param name=url3 value="page.html">
<param name=url4 value="page.html">
<param name=url5 value="page.html">
<param name=url6 value="page.html">

<param name=x0 value="80">
<param name=y0 value="50">
<param name=x1 value="160">
<param name=y1 value="60">
<param name=x2 value="240">
<param name=y2 value="60">
<param name=x3 value="45">
<param name=y3 value="145">
<param name=x4 value="150">
<param name=y4 value="160">
<param name=x5 value="250">
<param name=y5 value="155">
<param name=x6 value="330">
<param name=y6 value="180">
</applet>
<HR></BODY></html>
```

Active Menu Applet: Program Structure

Karoly.Bosa@jku.at

```
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.net.URL;
import java.net.MalformedURLException;

final public class MappingImageWithText extends java.applet.Applet implements MouseListener,
MouseMotionListener {

        //Declarations of some Constants and Objects

        public void init() { … }
        public void paint(Graphics g) { … }
        public void update(Graphics g) { … }
        public boolean mouseMoved(MouseEvent e) { … }
        public boolean mouseClicked(MouseEvent e) { … }

        //empty mouse event handling methods is required because of the two interfaces
        public void mouseEntered(MouseEvent e) {}
        public void mouseExited(MouseEvent e) {}
        public void mousePressed(MouseEvent e) {}
        public void mouseReleased(MouseEvent e) {}
        public void mouseDragged(MouseEvent e) {}
}
```

Active Menu Applet: Declarations


```
private final static int textStartX = 240;
private final static int textStartY = 290;
private final static int fsize = 22;

private int chosen=0;
private int numberOfCrew;

private Image img;
private Image bg;
private Font chType;

private int x[];
private int y[];
private String names[];
private String crewUrls[];
```

Active Menu Applet: init()


```
public void init() {
        chType=new Font("TimesRoman",Font.BOLD, fsize);
        setBackground(Color.gray);
        bg=getImage(getCodeBase(),"bg.jpg");

        img=getImage(getCodeBase(),this.getParameter("crewImage"));

        numberOfCrew = Integer.parseInt(this.getParameter("numberOfCrew"));
        x = new int[numberOfCrew];
        y = new int[numberOfCrew];
        names = new String[numberOfCrew];
        crewUrls = new String[numberOfCrew];

        for (int j=0;j<numberOfCrew;j++) {
                names[j] = this.getParameter("name"+j);
                crewUrls[j] = this.getParameter("url"+j);
                x[j] = Integer.parseInt(this.getParameter("x"+j));
                y[j] = Integer.parseInt(this.getParameter("y"+j));
        }

        addMouseMotionListener(this);
        addMouseListener(this);
}
```

Active Menu Applet: paint(Graphics g)

Karoly.Bosa@jku.at

```
public void paint(Graphics g) {
    for (int k=0;k<2;k++) {                                  //drawing background images
        for (int l=0;l<3;l++) {
            g.drawImage(bg,l*128,268+k*128,this);
        }
    }
    g.drawImage(img,0,0,this);
    g.setFont(chType);
    for(int j=0;j<numberOfCrew;j++) {
        g.setColor(Color.gray);                              //drawing the shadow of menu texts
        g.drawString(names[j],textStartX+1,textStartY+2+j*(fsize+2));
        if (chosen == j) {                                   //drawing the active and inactive menu texts
            g.setColor(Color.red);
        }
        else {
            g.setColor(Color.blue);
        }
        g.drawString(names[j],textStartX,textStartY+j*(fsize+2));
    }
    g.setColor(Color.white);                    //drawing an oval around the head of chosen person
    g.drawOval(x[chosen]-30,y[chosen]-40,65,85);
}
```

Active Menu Applet: update(Graphics g)

Karoly.Bosa@jku.at

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

Active Menu Applet: mouseMoved(…)

Karoly.Bosa@jku.at

```
public void mouseMoved(MouseEvent e) {
    int mouseX = e.getX();
    int mouseY = e.getY();
    for(int j=0;j<numberOfCrew;j++) {
        if ((chosen!=j)
                && ((mouseX>x[j]-20
                        && mouseX<x[j]+20
                        && mouseY>y[j]-40
                        && mouseY<y[j]+40)
                    || (mouseX>=textStartX
                        && mouseX<=380
                        && mouseY>=textStartY-fsize/2+j*fsize
                        && mouseY<=textStartY+fsize/2+j*fsize))) {
            chosen=j;
            repaint();
            break;
        }
    }
}
```

Active Menu Applet: mouseClicked(…)

Karoly.Bosa@jku.at

```
public void mouseClicked(MouseEvent e) {
    int mouseX = e.getX();
    int mouseY = e.getY();
    for(int j=0;j<numberOfCrew;j++) {
        if ((mouseX>x[j]-20
                && mouseX<x[j]+20
                && mouseY>y[j]-40
                && mouseY<y[j]+40)
            ||  (mouseX>=textStartX
                && mouseX<=380
                && mouseY>=textStartY-fsize/2+j*fsize
                && mouseY<=textStartY+fsize/2+j*fsize)) {
            try {
                getAppletContext().showDocument(new URL(crewUrls[j]),"_self");
            } catch(MalformedURLException eURL) {
                break;
            }
        }
    }
}
```

Active Menu Applet: The Linked Page

Karoly.Bosa@jku.at

```
<HTML>
<HEAD>
<TITLE>New Page</TITLE>
</HEAD>
<BODY>
<HR>
<CENTER><B>UNDER CONSTRUCTION!<B></CENTER>
<HR>
</BODY>
</HTML>
```

The Spinning Cube in 3D

Karoly.Bosa@jku.at

Problems:

- **How to compute the 3D coordinates**
- **How to hide invisible surfaces**
- **Timing of the Animation ➔ Use threads**

Rotation of a Vector around Axis Z (I.)

Karoly.Bosa@jku.at

Let $\underline{u}$ be a location vector :

$$\underline{u} = \begin{vmatrix} u_x \\ u_y \\ u_z \end{vmatrix}$$

We assume u_z is fixed.
Let $\underline{r}$ be the projection of $\underline{u}$ on the plane determined by the axis x and y. Then:

(I.) $$\underline{r} = \begin{vmatrix} u_x \\ u_y \end{vmatrix} = \begin{vmatrix} r * \cos\gamma \\ r * \sin\gamma \end{vmatrix}$$

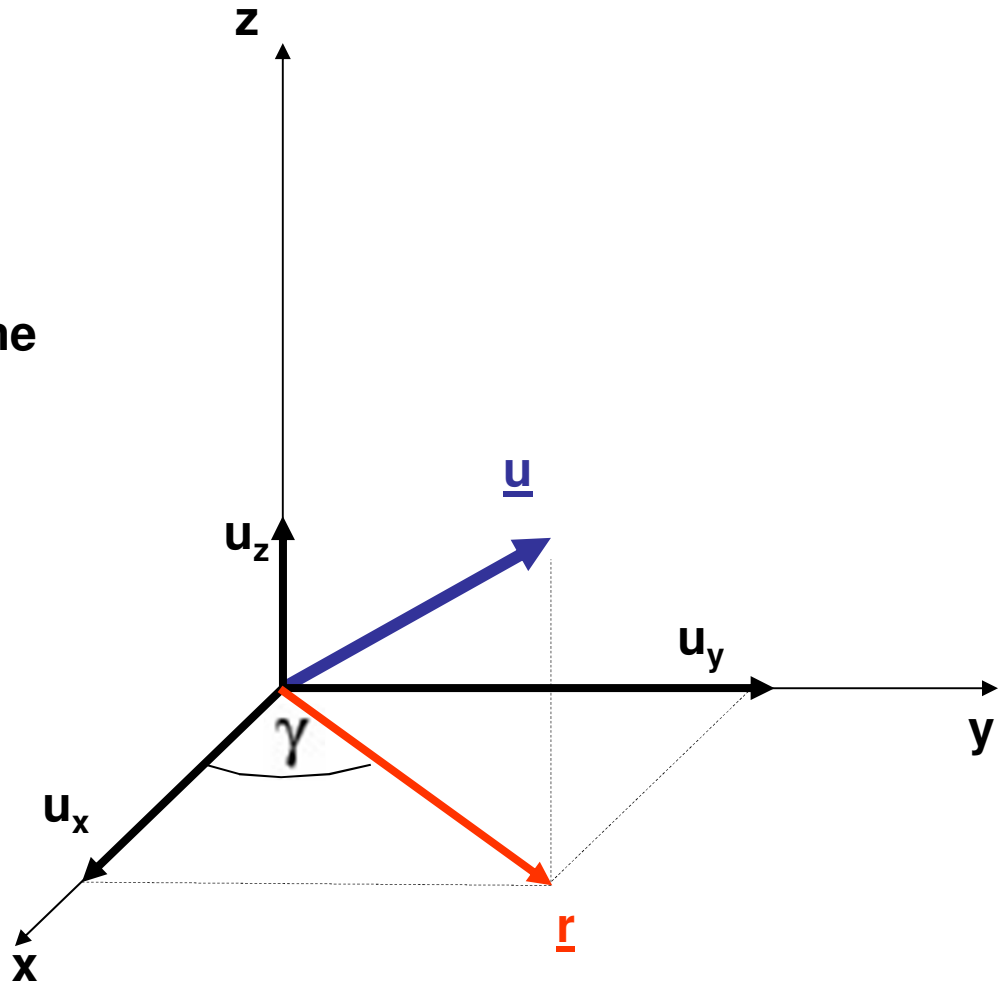
Hence,

$$\underline{u} = \begin{vmatrix} u_x \\ u_y \\ u_z \end{vmatrix} = \begin{vmatrix} r * \cos\gamma \\ r * \sin\gamma \\ u_z \end{vmatrix}$$

From (I.):

(II.) (III.)

$$\cos\gamma = \frac{u_x}{r} \quad \text{and} \quad \sin\gamma = \frac{u_y}{r}$$



Rotation of a Vector around Axis Z (II.)

Karoly.Bosa@jku.at

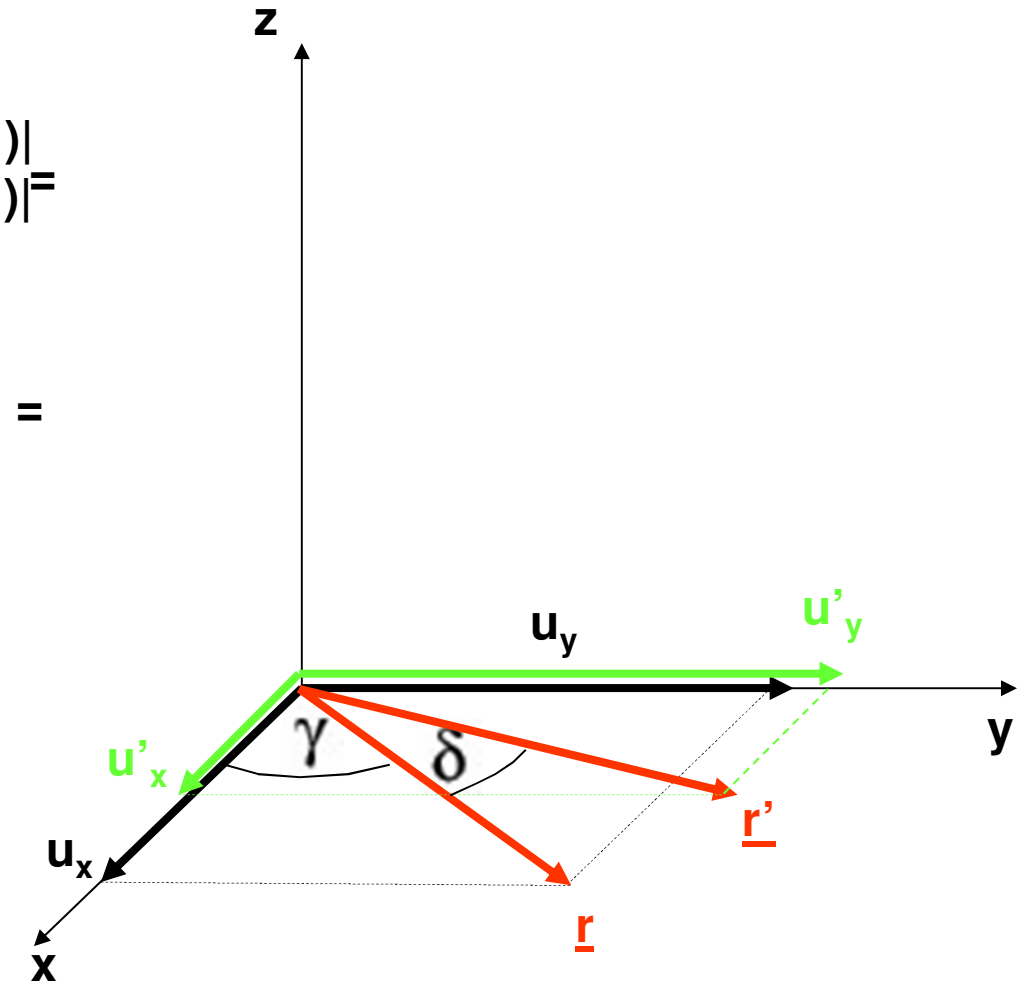
$$\underline{r'} = \begin{matrix} |\,u'_x\,| \\ |\,u'_y\,| \end{matrix} = \begin{matrix} |r*\cos(\gamma+\delta)| \\ |r*\sin(\gamma+\delta)| \end{matrix} =$$

$$= \begin{matrix} |r*\cos\gamma*\cos\delta - r*\sin\gamma*\sin\delta\,| \\ |r*\sin\gamma*\cos\delta + r*\cos\gamma*\sin\delta\,| \end{matrix} =$$

From (II.) and (III.):

(IV.)

$$= \begin{matrix} |u_x*\cos\delta - u_y*\sin\delta\,| \\ |u_y*\cos\delta + u_x*\sin\delta\,| \end{matrix}$$



Rotation of a Vector around Axis Z (III.)

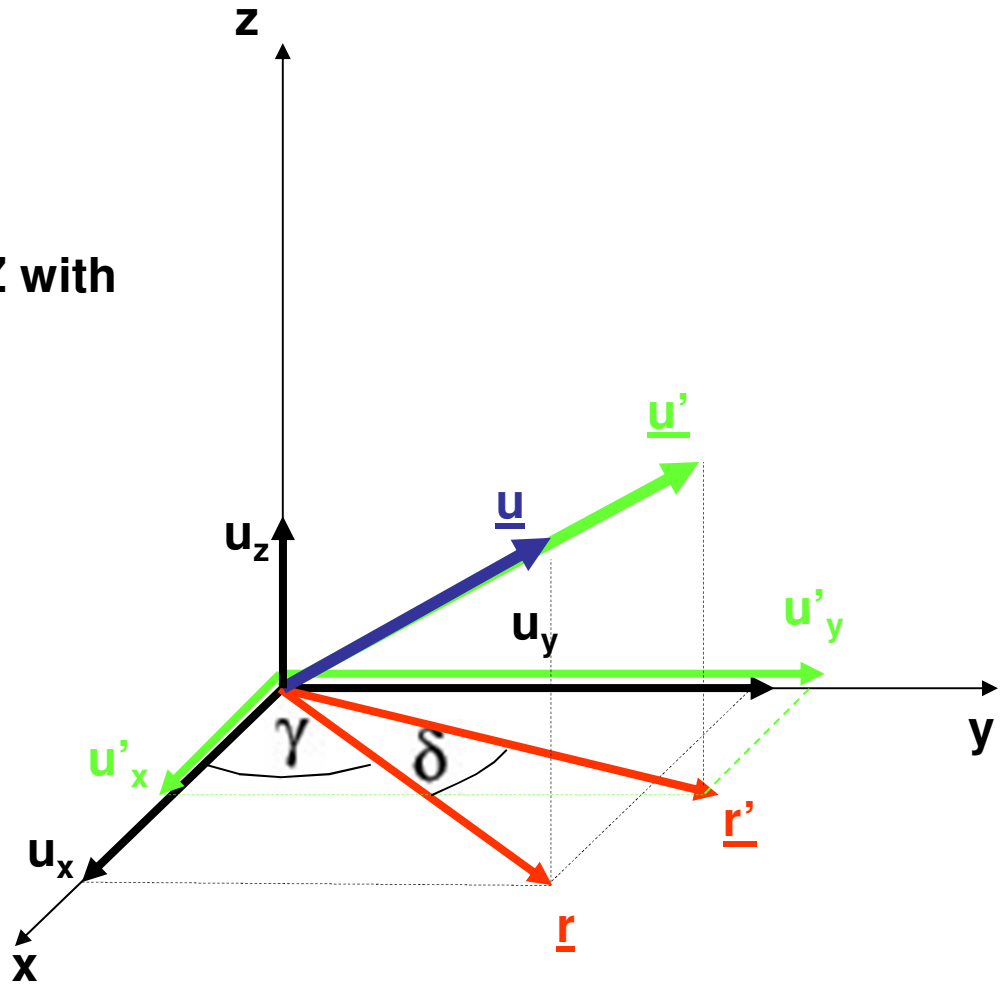
Karoly.Bosa@jku.at

From (IV.):

$$\underline{u'} = \begin{vmatrix} u'_x \\ u'_y \\ u_z \end{vmatrix} = R * \begin{vmatrix} u_x \\ u_y \\ u_z \end{vmatrix}$$

Where R is the rotation matrix for axis Z with the following content:

$$R = \begin{vmatrix} \cos\delta & -\sin\delta & 0 \\ \sin\delta & \cos\delta & 0 \\ 0 & 0 & 1 \end{vmatrix}$$



Rotation Matrices for X, Y and Z Axis

Karoly.Bosa@jku.at

$$R_x = \begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{vmatrix}$$

$$R_y = \begin{vmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{vmatrix}$$

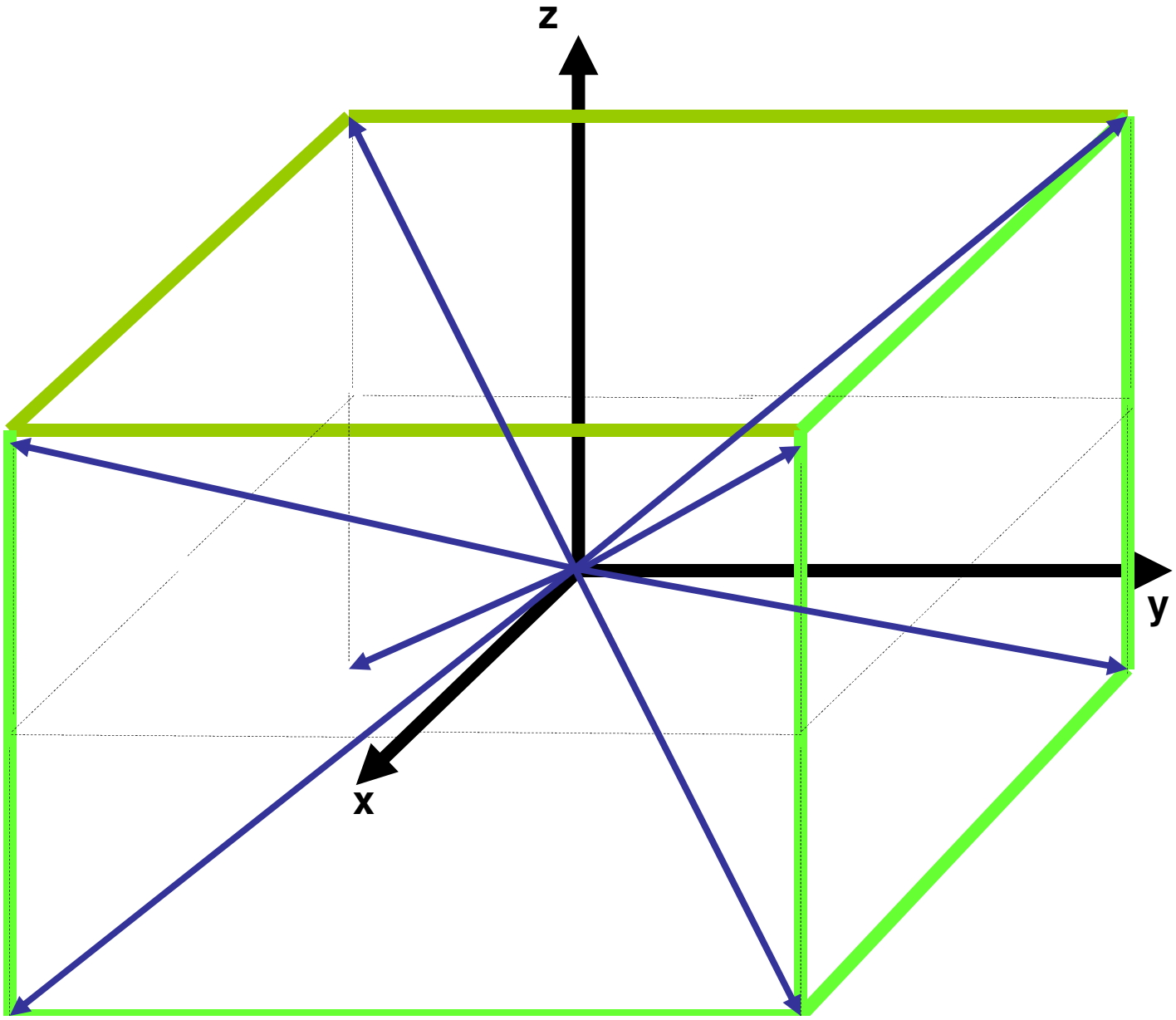
$$R_z = \begin{vmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$R_{xyz} = R_x * R_y * R_z$$

$$R_{xyz} = \begin{vmatrix} \cos\beta\cos\gamma & -\cos\beta\sin\gamma & \sin\beta \\ \cos\alpha\sin\gamma+\sin\alpha\sin\beta\cos\gamma & \cos\alpha\cos\gamma-\sin\alpha\sin\beta\sin\gamma & -\sin\alpha\cos\beta \\ \sin\alpha\sin\gamma-\cos\alpha\sin\beta\cos\gamma & \sin\alpha\cos\gamma+\cos\alpha\sin\beta\sin\gamma & \cos\alpha\cos\beta \end{vmatrix}$$

Spinning Cube in 3D

Karoly.Bosa@jku.at



Spinning Cube: Hiding Invisible Planes

Karoly.Bosa@jku.at

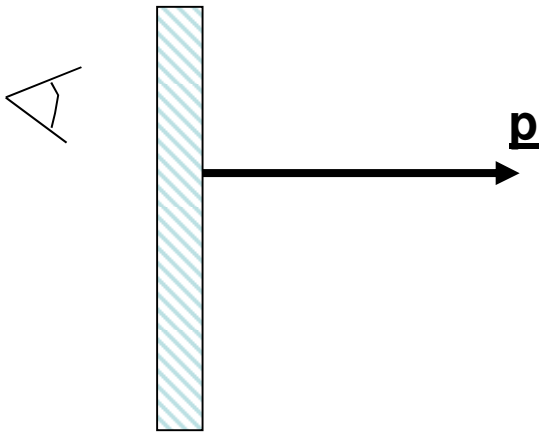
The following method is applicable only in case of convex solids bordered by planes (a solid is convex if it contains completely all lines which connect any two points of the solid).

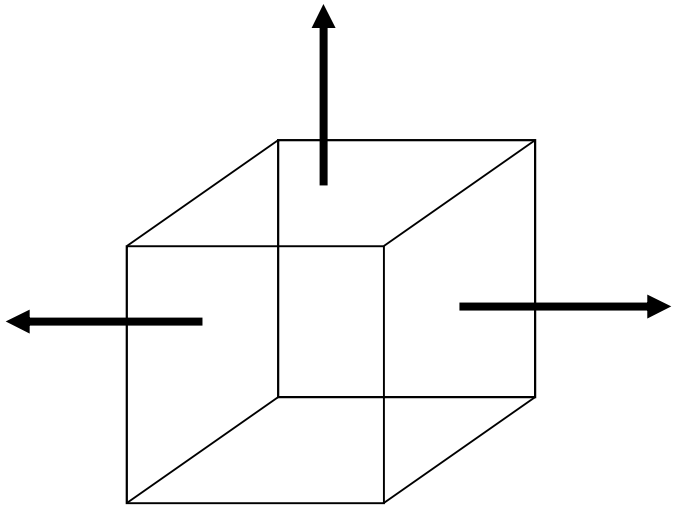
Condition of visibility: Let us assume, that

-we have a vector called p which is orthogonal to the plane of the screen and points to the directions of the observed solid

-there is a vector for each plane of the observed solid which orthogonal to the plane and points away from the solid,

then the plane is visible if the angle between its vector and vector p greater than 90 degree.





The surface of the screen

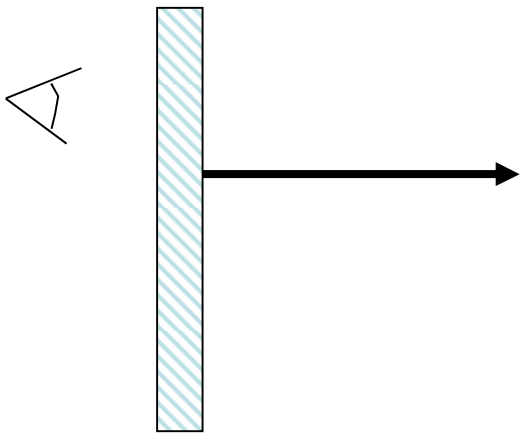
Spinning Cube: Hiding Invisible Planes

Karoly.Bosa@jku.at

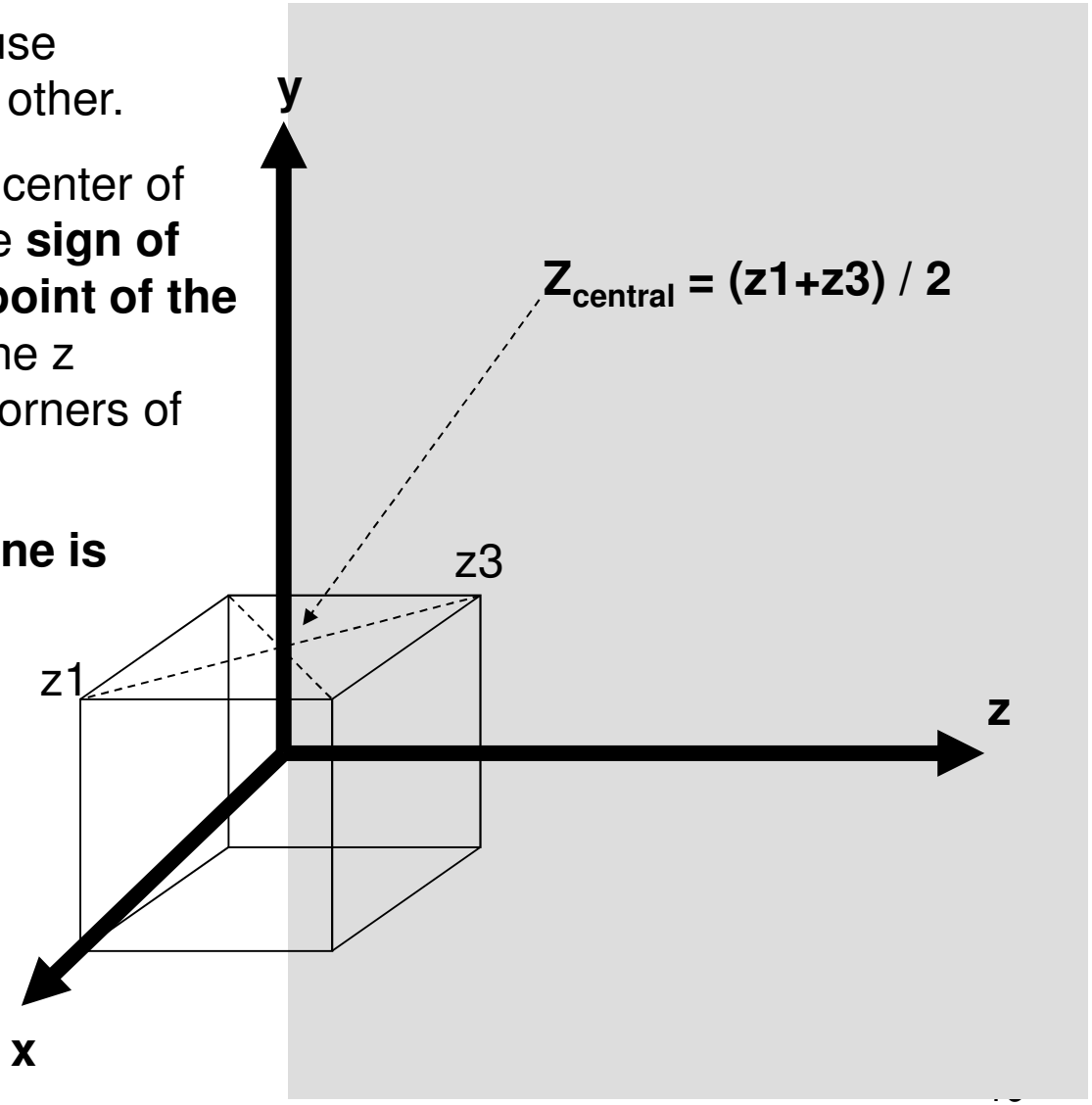
The cube is a special case, because its planes are orthogonal to each other.

Since we place the origin into the center of the cube, it is enough to check the **sign of the z coordinate of the central point of the plane** (it can be calculated from the z coordinates of any two diagonal corners of the plane).

If $z_{central}$ is negative then the plane is visible.



The surface of the screen



Spinning Cube: index.html

Karoly.Bosa@jku.at

```
<HTML>
<HR>
<CENTER>
<applet code=SpinningCube.class width=300 height=300>
</applet>
</CENTER>
<HR>
</html>
```

Spinning Cube: Program Structure

Karoly.Bosa@jku.at

```
import java.util.Arrays;
import java.awt.*;
import java.applet.*;

public final class SpinningCube extends Applet implements Runnable {
//Declarations of some Constants and Objects

public void init() { …}

public void start() { … }

public void stop() { … }

private void computeRotationMatrix() { … }

private  int[] computeCoordinates(int[] xyz) { … }

public void run() { … }

public void paint(Graphics g) { … }

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

Spinning Cube: Declarations


```
private final static int oX = 150;
//X coordinate of the origin
private final static int oY = 150;
//Y coordinate of the origin

private final static int step = 2;  //2 degrees is the unit of the rotation steps
private final static int numberOfParts = (360 / step);
private final static int edgeSize = 100;
private final static int[][] cube = {{
-edgeSize/2, -edgeSize/2, -edgeSize/2},
    {edgeSize/2, -edgeSize/2, -edgeSize/2},
    {edgeSize/2, edgeSize/2, -edgeSize/2},
    {-edgeSize/2, edgeSize/2, -edgeSize/2},

    {-edgeSize/2, -edgeSize/2, edgeSize/2},
    {edgeSize/2, -edgeSize/2, edgeSize/2},
    {edgeSize/2, edgeSize/2, edgeSize/2},
    {-edgeSize/2, edgeSize/2, edgeSize/2}};

private int[][] rotated, previous;
private double[][] rotMatrix;
private double[] si;    //stores the precalculated sinus values
private double[] co;    //stores the precalculated cosinus values

private int a = 0; //rotation angle around axis X
private int b = 0; //rotation angle around axis Y
private int g = 0; //rotation angle around axis Z

Thread anim;
```

Spinning Cube: init()


```
public void init() {
   rotated = new int[8][];
   rotMatrix = new double[3][3];
   si = new double[numberOfParts];
   co = new double[numberOfParts];

   // We compute the sinus and cosine values in advance
   // because these values will be used numerous times
   for (int i = 0; i < numberOfParts; i++) {
      si[i] = Math.sin(i*step*Math.PI/180); //the argument is in terms of in radian
      co[i] = Math.cos(i*step*Math.PI/180); //the argument is in terms of in radian
   }
}
```

Spinning Cube: start() and stop()


```
public void start() {
   anim =new Thread(this);
   anim.start();
}

public void stop() {
   if (anim != null) {
      if (anim.isAlive())
         anim.stop();
      anim = null;
   }
}
```

Spinning Cube: computeRotationMatrix(…)


```
private void computeRotationMatrix() {
  rotMatrix[0][0] = co[b]*co[g];
  rotMatrix[1][0] = -co[b]*si[g];
  rotMatrix[2][0] = si[b];
  rotMatrix[0][1] = co[a]*si[g]+si[a]*si[b]*co[g];
  rotMatrix[1][1] = co[a]*co[g]-si[a]*si[b]*si[g];
  rotMatrix[2][1] = -si[a]*co[b];
  rotMatrix[0][2] = si[a]*si[g]-co[a]*si[b]*co[g];
  rotMatrix[1][2] = si[a]*co[g]+co[a]*si[b]*si[g];
  rotMatrix[2][2] = co[a]*co[b];
}
```

Spinning Cube: computeCoordinates(…)

Karoly.Bosa@jku.at

```
private  int[] computeCoordinates(int[] xyz) {
   int[] rotatedXYZ = new int[3];

   //it follows a computation of the product of a 1x3 and a 3x3 matrices
   rotatedXYZ[0] =
(int)Math.round(rotMatrix[0][0]*xyz[0]+rotMatrix[1][0]*xyz[1]+rotMatrix[2][0]*xyz[2]);
   rotatedXYZ[1] =
(int)Math.round(rotMatrix[0][1]*xyz[0]+rotMatrix[1][1]*xyz[1]+rotMatrix[2][1]*xyz[2]);
   rotatedXYZ[2] =
(int)Math.round(rotMatrix[0][2]*xyz[0]+rotMatrix[1][2]*xyz[1]+rotMatrix[2][2]*xyz[2]);

   return rotatedXYZ;
}
```

Spinning Cube: run()


```
public void run() {
  while (anim != null) {
            repaint();
            try {
              Thread.sleep(30);
            } catch (InterruptedException e) {}

            a = (a+1) % numberOfParts; // we could apply some randomization here
            b = (b+1) % numberOfParts; // we could apply some randomization here
            g = (g+1) % numberOfParts; // we could apply some randomization here

            computeRotationMatrix();

            rotated[0] = computeCoordinates(cube[0]);
            rotated[1] = computeCoordinates(cube[1]);
            rotated[2] = computeCoordinates(cube[2]);
            rotated[3] = computeCoordinates(cube[3]);
            rotated[4] = computeCoordinates(cube[4]);
            rotated[5] = computeCoordinates(cube[5]);
            rotated[6] = computeCoordinates(cube[6]);
            rotated[7] = computeCoordinates(cube[7]);
  }
}
```

Spinning Cube: update()


```
public void update(Graphics g) {
  g.setColor(Color.white);
  if (previous!=null) {
    paint(g);
  }
  previous = Arrays.copyOf(rotated, rotated.length);
  g.setColor(Color.black);
  paint(g);
}
```

Spinning Cube: paint(Graphics g) 2/1

Karoly.Bosa@jku.at

```
public void paint(Graphics g) {
   //Redundant edges must be drawn because of visiblity conditions

   if (previous[0][2]+previous[2][2] >0) { //visibility condition
      g.drawLine(oX+previous[0][0], oY+previous[0][1], oX+previous[1][0], oY+previous[1][1]);
      g.drawLine(oX+previous[1][0], oY+previous[1][1], oX+previous[2][0], oY+previous[2][1]);
      g.drawLine(oX+previous[2][0], oY+previous[2][1], oX+previous[3][0], oY+previous[3][1]);
      g.drawLine(oX+previous[3][0], oY+previous[3][1], oX+previous[0][0], oY+previous[0][1]);
   }

   if (previous[0][2]+previous[5][2] >0) { //visibility condition
      g.drawLine(oX+previous[0][0], oY+previous[0][1], oX+previous[1][0], oY+previous[1][1]);
      g.drawLine(oX+previous[1][0], oY+previous[1][1], oX+previous[5][0], oY+previous[5][1]);
      g.drawLine(oX+previous[5][0], oY+previous[5][1], oX+previous[4][0], oY+previous[4][1]);
      g.drawLine(oX+previous[4][0], oY+previous[4][1], oX+previous[0][0], oY+previous[0][1]);
   }

   if (previous[1][2]+previous[6][2] >0) { //visibility condition
      g.drawLine(oX+previous[1][0], oY+previous[1][1], oX+previous[2][0], oY+previous[2][1]);
      g.drawLine(oX+previous[2][0], oY+previous[2][1], oX+previous[6][0], oY+previous[6][1]);
      g.drawLine(oX+previous[6][0], oY+previous[6][1], oX+previous[5][0], oY+previous[5][1]);
      g.drawLine(oX+previous[5][0], oY+previous[5][1], oX+previous[1][0], oY+previous[1][1]);
   }
```

Spinning Cube: paint(Graphics g) 2/2

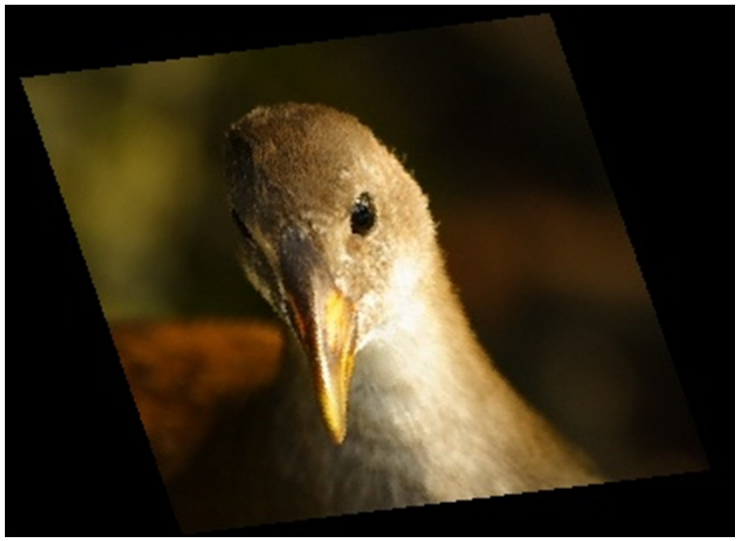
Karoly.Bosa@jku.at

```
    if (previous[2][2]+previous[7][2] >0) { //visibility condition
      g.drawLine(oX+previous[2][0], oY+previous[2][1], oX+previous[3][0], oY+previous[3][1]);
      g.drawLine(oX+previous[3][0], oY+previous[3][1], oX+previous[7][0], oY+previous[7][1]);
      g.drawLine(oX+previous[7][0], oY+previous[7][1], oX+previous[6][0], oY+previous[6][1]);
      g.drawLine(oX+previous[6][0], oY+previous[6][1], oX+previous[2][0], oY+previous[2][1]);
    }

    if (previous[3][2]+previous[4][2] >0) { //visibility condition
      g.drawLine(oX+previous[3][0], oY+previous[3][1], oX+previous[0][0], oY+previous[0][1]);
      g.drawLine(oX+previous[0][0], oY+previous[0][1], oX+previous[4][0], oY+previous[4][1]);
      g.drawLine(oX+previous[4][0], oY+previous[4][1], oX+previous[7][0], oY+previous[7][1]);
      g.drawLine(oX+previous[7][0], oY+previous[7][1], oX+previous[3][0], oY+previous[3][1]);
    }

    if (previous[4][2]+previous[6][2] >0) { //visibility condition
      g.drawLine(oX+previous[4][0], oY+previous[4][1], oX+previous[5][0], oY+previous[5][1]);
      g.drawLine(oX+previous[5][0], oY+previous[5][1], oX+previous[6][0], oY+previous[6][1]);
      g.drawLine(oX+previous[6][0], oY+previous[6][1], oX+previous[7][0], oY+previous[7][1]);
      g.drawLine(oX+previous[7][0], oY+previous[7][1], oX+previous[4][0], oY+previous[4][1]);
    }
}
```

Rotating Bitmaps in 3D

Karoly.Bosa@jku.at

- algorithm is similar to the cube rotation, but much more points (ten thousands) need to be calculated -> special solutions is required for optimal performance.



- Java can read and write the following image types: jpg, gif, png (since Java 1.3), bmp, wbmp (since Java 1.4), etc.

- Accessing to the pixel information: via the class java.awt.image.BufferedImage with the following methods:
  - int getRGB(int x, int y)
  - int setRGB(int x, int y, int RGBInfo)

Rotating Bitmaps in 3D 3/1

Karoly.Bosa@jku.at

denotes a point calculated by the rotation matrix

method 1: (slow) Pass each pixel through the rotation matrix.

Rotating Bitmaps in 3D 3/2

Karoly.Bosa@jku.at

denotes a point calculated by the rotation matrix

method 2: (faster) Pass the endpoints of each horizontal line in your image through the rotation matrix. Use line drawing (e.g.: Bresenham's) algorithm to plot the points in between but instead of plotting all of the pixels the same color, iterate through your source line in the bitmap and use the values you find there.

Description of some line drawing algorithm:

http://www.cs.unc.edu/~mcmillan/comp136/Lecture6/Lines.html

Rotating Bitmaps in 3D 3/3

Karoly.Bosa@jku.at

denotes a point calculated by the rotation matrix

method 3: (fastest) Rotate the corner coordinates of your rectangle (bitmap), use a line drawing algorithm (e.g.:Bresenham's) to generate the coordinates of the endpoints of your rotated (previously horizontal) lines, and apply method 2 using these endpoints.

NOTE: There may be problems with line drawing algorithm because they may not generate the same number of points for the rotated line as the horizontal source line. You need to study their behavior before you use them for bitmap rotation.

Description of some line drawing algorithm:
http://www.cs.unc.edu/~mcmillan/comp136/Lecture6/Lines.html

Spinning Image: index.html


```
<HTML>
<HR>
<CENTER>
<applet code=SpinningImage.class width=800 height=600>
<param name=image value="butterfly.jpg">
</applet>
</CENTER>
<HR>
</html>
```

Spinning Image: Program Structure

Karoly.Bosa@jku.at

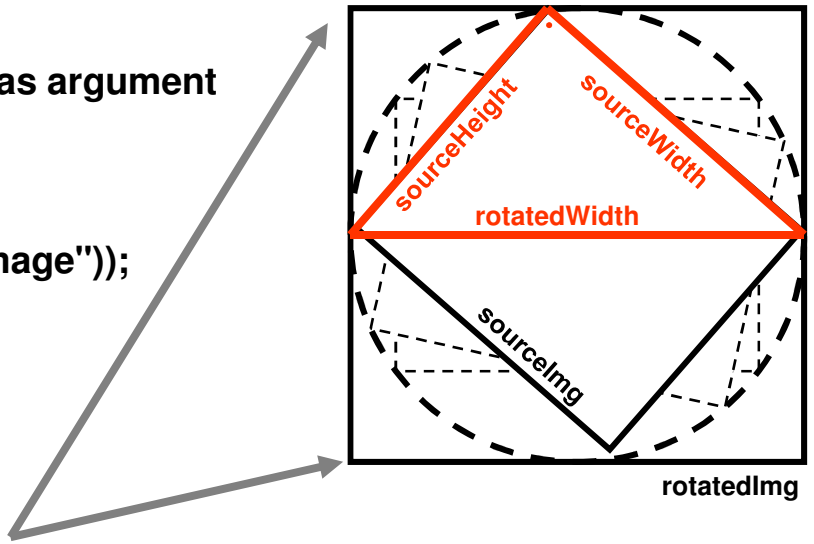
```
import java.awt.*;
import java.io.IOException;
import java.awt.image.*;
import java.applet.*;
import javax.imageio.*;
import java.net.URL;

public final class SpinningImage extends Applet implements Runnable {

//Declarations of some Constants and Objects

public void init() { … }

public void start() { … } //Same as before

public void stop() { … } //Same as before

private void computeRotationMatrix() { … } //Same as before

private  int[] computeCoordinates(int[] xyz) { … } //Same as before

public void run() { … }

public void paint(Graphics g) { … }

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

Spinning Image: Declarations


```
private int oX;  //X coordinate of the origin
private int oY;  //Y coordinate of the origin

private final static int step = 2;  //2 degrees is the unit of the rotation steps
private final static int numberOfParts = (360 / step);

private double[][] rotMatrix;
private double[] si;    //stores the pre-calculated sinus values
private double[] co;    //stores the pre-calculated cosine values

private int a = 0; //rotation angle around axis X
private int b = 0; //rotation angle around axis Y
private int g = 0; //rotation angle around axis Z

BufferedImage sourceImg = null;
BufferedImage rotatedImg = null;

int sourceHeight;  //height of the original image
int sourceWidth;   //width of the original image
int rotatedHeight;  //height of the rotated image.
int rotatedWidth;   //width of the rotated image.
int rotatedType;    //color type of the source and rotated images

Thread anim;
```

Spinning Image: init()

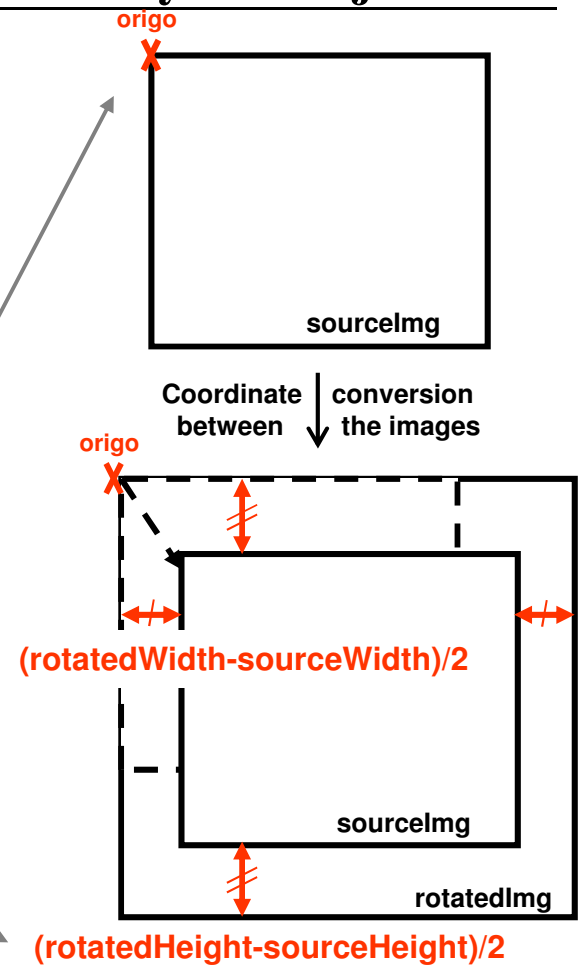

```
public void init() {
  rotMatrix = new double[3][3];
  si = new double[numberOfParts];
  co = new double[numberOfParts];

  // We compute the sinus and cosine values in advance
  // because these values will be used up numerous times
  for (int i = 0; i < numberOfParts; i++) {
     si[i] = Math.sin(i*step*Math.PI/180); //radian is expected
     co[i] = Math.cos(i*step*Math.PI/180); //radian is expected as argument
  }

  try {
     URL url = new URL(getCodeBase(), this.getParameter("image"));
     sourceImg = ImageIO.read(url);
  } catch (IOException e) {System.exit(0);}

  sourceWidth = sourceImg.getWidth();
  sourceHeight = sourceImg.getHeight();

  rotatedWidth = (int)Math.round(Math.sqrt(sourceWidth * sourceWidth + sourceHeight * sourceHeight))+2;
  rotatedHeight = rotatedWidth;

  oX = rotatedWidth / 2;
  oY = rotatedHeight/ 2;
  rotatedType = sourceImg.getType();
}
```



Spinning Image: run()

Karoly.Bosa@jku.at

```
public void run() {
  while (anim != null) {
      try {
         Thread.sleep(10);
      } catch (InterruptedException e) {}
      a = (a+1) % numberOfParts;
      b = (b+1) % numberOfParts;
      g = (g+1) % numberOfParts;
      computeRotationMatrix();
      rotatedImg = new BufferedImage(rotatedWidth,
                                     rotatedHeight, rotatedType);
      for (int j = 0; j < sourceHeight; j++)
      {
         for (int i = 0; i < sourceWidth; i++)
         {
               int[] xyz = new int[3];
               xyz[0] = i - oX + (rotatedWidth - sourceWidth) / 2;
               xyz[1] = j - oY + (rotatedHeight - sourceHeight) / 2;
               xyz[2] = 0;
               int[] rotated = computeCoordinates(xyz);

               rotatedImg.setRGB(rotated[0]+oX, rotated[1]+oY, sourceImg.getRGB(i,j));
         }
      }
     repaint();
  }
}
```



Spinning Image: paint(…) and update(…)

Karoly.Bosa@jku.at

```
public void paint(Graphics g) {
   g.drawImage(rotatedImg,0,0,this);
}


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

}
```

Graphics in Programs

The easiest way to use graphics in a java program:

- Extend the class ***java.awt.Canvas*** and rewrite its method **paint(Graphics g)** like we did in the applets

- This time we need to create a window where our graphic will take place

Recall our map exercise.

Graphics in Programs – In General

Karoly.Bosa@jku.at

```
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
…
private Frame frm;

public class Map extends Canvas {
    public Map(String filename) throws IOException {
                this.filename = filename;
                map = new int[SQUARE_SIZE];
                readMapFromFile();
                frm = new Frame(filename);
                frm.add(this);
                frm.setSize(SQUARE_SIDE+50, SQUARE_SIDE+75);
                frm.setVisible(true);
                frm.addWindowListener(new WindowAdapter(){
                 public void windowClosing(WindowEvent we){  System.exit(0); }
                });
  }

  public void paint(Graphics g) {… }

  public static void main(String[] args) throws IOException {
                Map map = new Map(filename);
  }
}
```

Closing a Window


```
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
…
private Frame frm;

public class Map extends Canvas {
    public Map(String filename) throws IOException {
                this.filename = filename;
                map = new int[SQUARE_SIZE];
                readMapFromFile();
                frm = new Frame(filename);
                frm.add(this);
                frm.setSize(SQUARE_SIDE+50, SQUARE_SIDE+75);
                frm.setVisible(true);
                frm.addWindowListener(new WindowAdapter(){
                 public void windowClosing(WindowEvent we){  System.exit(0); }
                });
  }

  public void paint(Graphics g) {… }

  public static void main(String[] args) throws IOException {
                 Map map = new Map(filename);
  }
}
```

Closing a Window


```
frm.addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent we){  System.exit(0); }
});
```

We have to possibilties:

- Implement interface ***WindowListener*** with **all(!)** the following methods : ***windowActivated, windowClosed, windowClosing, windowDeactivated, windowDeiconified, windowIconified, windowOpened*** (However we need only the windowClosing method).

- Or we use the abstract class ***WindowAdapter*** which implements empty methods for the interface ***WindowListener*** (and some other window related event interfaces) so we have to rewrite only those methods which we really need.

Remark: We can also simplify our mouse events for instance ; instead of implementing all the methods of the interfaces ***MouseListener*** and ***MouseMotionListener*** use the abstract class ***MouseInputAdapter.***

Exercise 12: RotatedMap Deadline: *

Karoly.Bosa@jku.at

Recall the map program from Exercise 9. and reuse its source files (or download them from the course web page).

1st part:

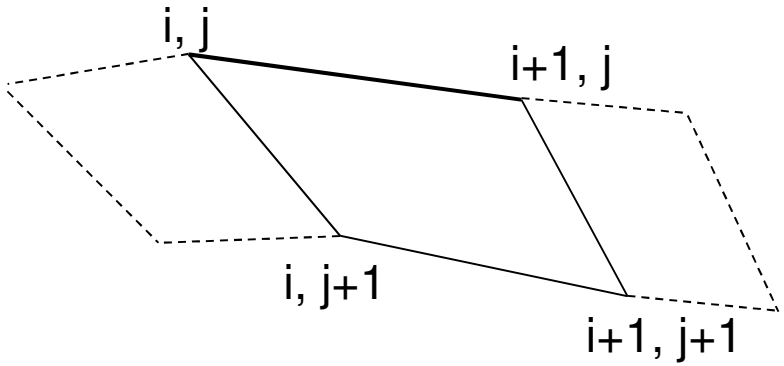
-Modify the method ***main*** such that it expect a command line parameter (between 0 and 90).

-Modify the class ***Map*** such that it displays the map rotated around x axis with the given value. So the rotation matrix:

$$Rx = \begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos & -\sin \\ 0 & \sin & \cos \end{vmatrix}$$

Extension of the exercise (not obligatory, but you can get extra +15% in the evaluation):

-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 (this means you should always use the X and Y coordinates of 4 rotated points for the method fillPolygon).

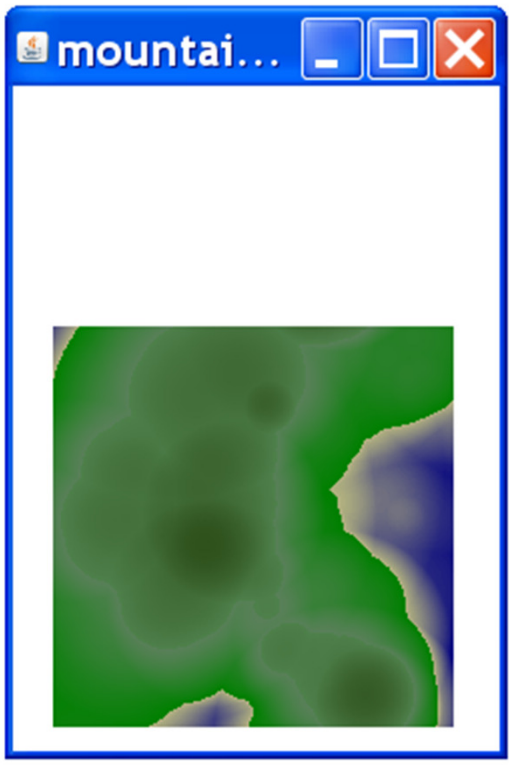


Exercise 12: RotatedMap Deadline: *

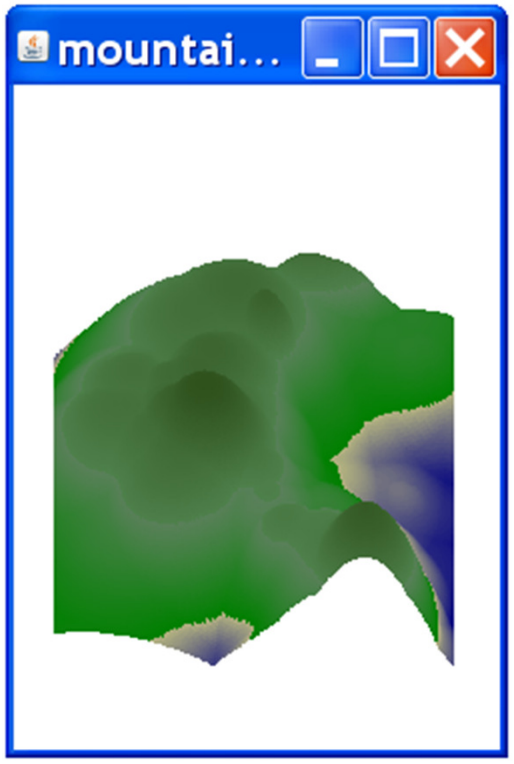
Karoly.Bosa@jku.at

If you do the right job you should see some similar windows on the screen if you run class ***Map*** the following command line values:

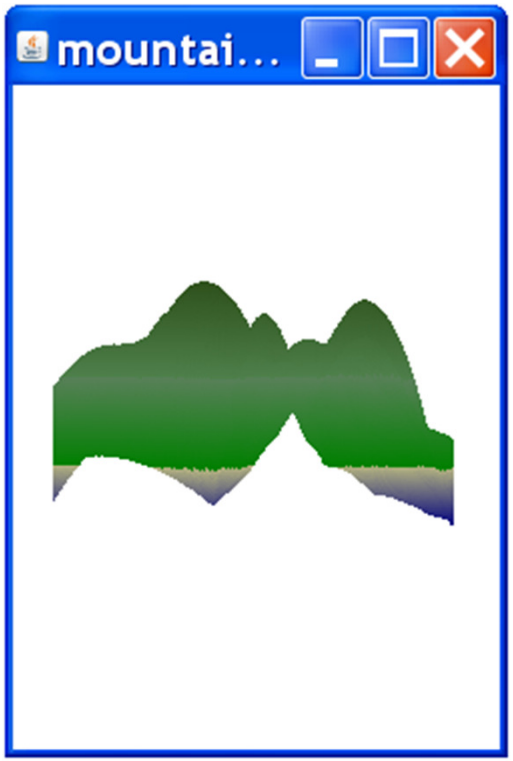
RotatedMap 0



RotatedMap 40



RotatedMap 90



Exercise 12: RotatedMap Deadline: *

Karoly.Bosa@jku.at

2nd part(obligatory):

Implement the interface ***mouseMotionListener (***or the class ***MouseInputAdapter)*** in the class ***Map***, such that if you drag (press a mouse button and move the mouse) 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).

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)

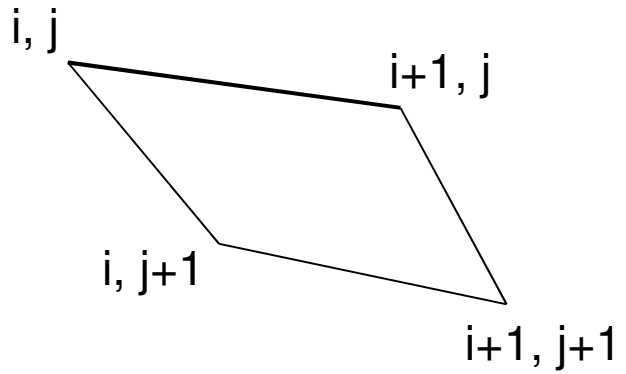
for displaying the map.

```
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

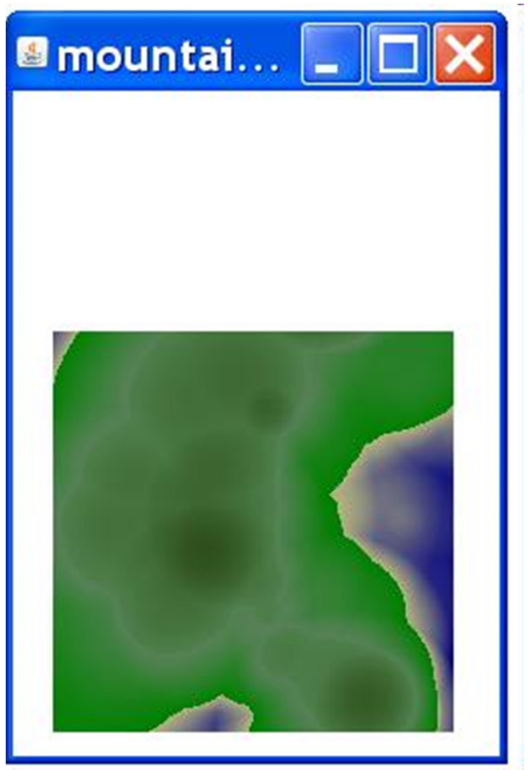
Karoly.Bosa@jku.at

```
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();
}
```

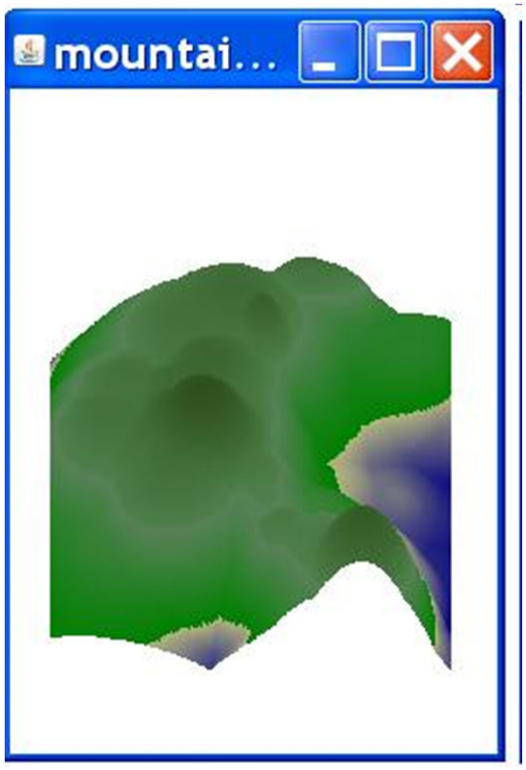
Exercise 12: RotatedMap – Output

Karoly.Bosa@jku.at

java RotatedMap 0



java RotatedMap 40



java RotatedMap 90