TEACHER TRAINING MANUAL

MULTIMEDIA APPLICATIONS FOR EDUCATION

Part 1B: INTRODUCTION

Chapter 4: Introduction to Programming for education
University of Fine Arts Brera (IT)

Objectives
Methodology
2D Drawing
- Translations
- Changes in the Scale
- Rotations

3D Drawing
- Perspective Projection
- Field Angle
- Rotations

Animation
Interaction with the User

PART ONE / B

INTRODUCTION

Chapter Four
Introduction to Programming for education

University of Fine Arts of Brera (IT)

1. Programming

Programming, defined as "logical sequence of instructions formalised through specific languages", represents the essential and mostly characterising activity of the Information Technology.
In this way, since the birth of electronic computers less than sixty years ago, a lot of programming languages have been studied ; they have been developed and applied all over the world thanks to the information technological development, to its commercial diffusion and to the always new applicative necessities of a lot of users, who have grown in number and come from all the fields of human activities , not only mathematics, physics and engineers but also lawyers, doctors, artists and so on.

The development and the selling of more and more precise applicative software available for users who are not interested to study programming strategies by themselves - together with the chance of a world communication through Internet- have been the mainsprings that have brought computers not only in every place of work, but nearly in every home.

But when a user wants to carry out a certain operation of his/her creation to obtain a specific result which the ready-made can't offer, he/she must develop his/her own specific language.

We advise all the users of Information Technology to do, at least once, an experience of programming; this for various reasons that go from the learning of something new, like a programming language, to the logical planning of a project, the understanding of the enormous studies and works at the basis of their activity and the final satisfaction of realizing that the computer faithfully executes the instructions we give.
To the non-professional of programming all this represents not only a personal technical conquest but also men's re-appropriation of the control over the world of machines which are more and more invading our world and space.
As a simple example of programming applied in any field here we can consider the Hyper Text Mark-up Language (HTML), the well-known language used to build up the web pages.
This topic can be treated in a very simple way, focusing on the inner basic structure of the language itself and by-passing the complex and obviously more complete authoring software that allow using it in an automatic -although more powerful- way.
In the field of education, a teacher that posseses the basic of programming (also in a very simple way), has the opportunity to create his/her own text-book, exercises texts and also the "text-games" that are our real objective.
From Html authoring software to multimedia authoring tools from Power Point to NeoBook as we will see it is possible to create highly-customizable didactic application.

Java is a flexible and very powerful programming language which has been recently developed; the "open code source"of this language (that is the fact that its programming structure is left free by its first developers, like in the Linux operative system) made it an outmost exemplification of what Internet represents: it is a language which can be continuously enriched and improved (as actually it is) by the whole Internet community. In this case we give indications about how it is possible to draw by programming.

2. Objectives

The main aim of this didactic unit is to assert the creative value of programming, the only means which allows us to interact consciously in the Digital Era.

3. Methodology

To achieve this aim, we'll show how to build up a simple program of graphic three-dimensional elaboration in *Java* language; a series of steps, together with subsequent additions, will guide to the achievement of an *applet Java* to be seen on a *web page*. This programme will bring to the animation of a prospective drawing and will allow the user to move it according to his/her desire.

Note

Being aware of the fact that programming is not simple, we have decided to explain *practically* the different steps to achieve a prospective drawing through a software programme, willingly neglecting the mathematic and informatics aspects underneath.

Therefore, at least at the beginning, don't worry about deeply understanding and modifying the programmes themselves, but try to change only a few parameters, also because, even if simple, these *applet* are not so evident as they could appear[1].

- 0 -

There it is, our first applet which doesn't move, and following on the grey ground its source code (**`Step0.java`**).

Its *htlm file* (**`Step0.html`**) will be:

```
<html>
<head>
<title></title>
</head>
<body>
<applet code="Step0" width="100" height="100"></applet>
</body>
</html>
```

```
import java.applet.*;

public class Step0 extends Applet
{
}
```

- 1 -

2D DRAWING

[1] The author of this code is Claudio Destro from Brera Academy, and an interactive version of the tutorial can be found –complete with usable source code- at http://multimediart.pixel-online.net/

Let's start with the drawing of a line: we insert in the body of the applet the *method* (*function* or *sub-routine*) **paint**:
`public void paint( Graphics g )`
and then we insert in the body of this method the instruction of the drawing of a line **g.drawLine** by specifying the coordinates of the pixel of the beginning and the end of the same line; in our examples the programme draws a line from pixel (10, 10) to pixel (100, 200). Notice that, being the area visible only within 100 per 100 pixels, the line is automatically cut out at the borders of the applet.

~

Try now to ricompile the applet by modifying the pixels at the start and the end, for example:
`g.drawLine( 0, 0, 20, 49 );`
and to write the htlm file which must contain it.

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

public class Step1 extends Applet
{
    public void paint( Graphics g )
    {
        g.drawLine( 10, 10, 100, 200 );
    }
}
```

- 2 -
2D DRAWING

It is of course possible to draw more than one line:

1. Let's create a list of vertices (l'*array* **geometry**) containing the coordinates of the initial and final pixel referred to the lines we want to draw
2. let's insert in the method **paint** a cycle **for** which scans all the list of the points and draws the lines.

In our example we have inserted the vertices of a square of 10 pixel per side: the first line of the array **geometry** locates a bi dimensional segment which goes from pixel (-5,-5) to pixel (5,-5), the second line locates a segment which is comprised between (5,-5) and (5,5) and so on…

~

Try to modify the points, or add other ones, or leave some out.

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

public class Step2 extends Applet
{
    int[][] geometry = new int[][]
    {
        { -5, -5 }, {  5, -5 },
        {  5, -5 }, {  5,  5 },
        {  5,  5 }, { -5,  5 },
        { -5,  5 }, { -5, -5 },
```

```
    };

    public void paint( Graphics g )
    {
        for( int i = 0; i < geometry.length; i = i + 2 )
        {
            g.drawLine( geometry[i][0],
                        geometry[i][1],
                        geometry[i + 1][0],
                        geometry[i + 1][1] );
        }
    }
}
```

- 3 -
2D DRAWING
TRANSLATIONS

As you have seen in our previous example, the square was not centred in the applet. Normally the pixel (0,0) of an applet, or of any other programme, as in your screen, is positioned left high.
Therefore, to make the origin O(0, 0) of our drawing converge with the centre of our applet, we need to *translate* the coordinates X and Y of the points we want to draw of a quantity which must correspond to half the width and half the height of the applet:

```
int CX = getSize().width / 2;
int CY = getSize().height / 2;
```

`CX` and **`CY`** contain the values to be added, and a new **`for`** cycle has been inserted to translate the vertices:

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

public class Step3 extends Applet
{
    int[][] geometry = new int[][]
    {
        { -5, -5 }, {  5, -5 },
        {  5, -5 }, {  5,  5 },
        {  5,  5 }, { -5,  5 },
        { -5,  5 }, { -5, -5 },
    };

    int[][] vertex = new int[ geometry.length ][2];

    public void paint( Graphics g )
    {
        int CX = getSize().width / 2;
        int CY = getSize().height / 2;

        for( int i = 0; i < geometry.length; i = i + 1 )
        {
            vertex[i][0] = CX + geometry[i][0];
            vertex[i][1] = CY - geometry[i][1];
        }

        for( int i = 0; i < geometry.length; i = i + 2 )
        {
```

```
            g.drawLine( vertex[i][0],
                        vertex[i][1],
                        vertex[i + 1][0],
                        vertex[i + 1][1] );
        }
    }
}
```

- 4 -
2D DRAWING
CHANGES IN THE SCALE

Leaving our geometry (the coordinates X, Y of the vertices) unchanged, let's try now to *zoom* our drawing by multiplying all the vertices per a factor of scale **RHO**, before translating them:
`double RHO = 5;`
Notice how the array **geometry** is now formed by numbers **double** instead of numbers **int**.
The numbers **double** correspond to real numbers, while the numbers **int** correspond to the pixels on the screen: a crucial difference.
A new step has therefore been added inside the cycle **for**:
```
int PX = (int)(RHO * X);
int PY = (int)(RHO * Y);
```

~

Try to change the value **RHO**, for example:
`double RHO = 1;`
or
`double RHO = 1.5;`

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

public class Step4 extends Applet
{
    double[][] geometry = new double[][]
    {
        { -5, -5 }, {  5, -5 },
        {  5, -5 }, {  5,  5 },
        {  5,  5 }, { -5,  5 },
        { -5,  5 }, { -5, -5 },
    };

    int[][] vertex = new int[ geometry.length ][2];

    double RHO   = 5;

    public void paint( Graphics g )
    {
        int CX = getSize().width / 2;
        int CY = getSize().height / 2;

        for( int i = 0; i < geometry.length; i = i + 1 )
        {
            double X = geometry[i][0];
            double Y = geometry[i][1];

            int PX = (int)(RHO * X);
```

```
            int PY = (int)(RHO * Y);

            vertex[i][0] = CX + PX;
            vertex[i][1] = CY - PY;
        }

        for( int i = 0; i < geometry.length; i = i + 2 )
        {
            g.drawLine( vertex[i][0],
                        vertex[i][1],
                        vertex[i + 1][0],
                        vertex[i + 1][1] );
        }
    }
}
```

- 5 -

2D DRAWING
ROTATIONS

Always leaving our geometry unaltered, let's now try to rotate our drawing of an angle **THETA**:
`double THETA = 0.5;`
This angle is expressed in radians not in degrees.
A further step has been added inside the cycle **for** to rotate the vertices:
```
double XE = -X * S1 + Y * C1;
double YE = -X * C1 - Y * S1;
```
S1 and **C1** are respectively the *sine* and *cosine* of the angle **THETA**.

~

Try to change the value of **THETA**, for example:
`double THETA = 0.6;`

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

public class Step5 extends Applet
{
    double[][] geometry = new double[][]
    {
        { -5, -5 }, {  5, -5 },
        {  5, -5 }, {  5,  5 },
        {  5,  5 }, { -5,  5 },
        { -5,  5 }, { -5, -5 },
    };

    int[][] vertex = new int[ geometry.length ][2];

    double RHO   = 5;
    double THETA = 0.5;

    public void paint( Graphics g )
    {

        double S1 = Math.sin( THETA );
        double C1 = Math.cos( THETA );

        int CX = getSize().width / 2;
```

```
        int CY = getSize().height / 2;

        for( int i = 0; i < geometry.length; i = i + 1 )
        {
            double X = geometry[i][0];
            double Y = geometry[i][1];

            double XE = -X * S1 + Y * C1;
            double YE = -X * C1 - Y * S1;

            int PX = (int)(RHO * XE);
            int PY = (int)(RHO * YE);

            vertex[i][0] = CX + PX;
            vertex[i][1] = CY - PY;
        }

        for( int i = 0; i < geometry.length; i = i + 2 )
        {
            g.drawLine( vertex[i][0],
                        vertex[i][1],
                        vertex[i + 1][0],
                        vertex[i + 1][1] );
        }
    }
}
```

- 6 -
3D DRAWING
PERSPECTIVE PROJECTION

Let's now add the third dimension..
To do this let's go back to point 3 (**Step3**) and modify it by adding to the coordinates X, Y of the vertices, the coordinate Z.
The vertices now specified in the array **geometry** define the initial and final points of segments in the tri-dimensional space, disposed to form a cube of side 10.
The procedure of ***perspective projection*** is based on the *homothetic* triangles (rectangular ones)

```
int PX = (int)(X / Z);
int PY = (int)(Y / Z);
```

PX and **PY** contain the coordinates of a point (X,Y,Z) projected on the *fore ground*.
As you can notice, no cube can be seen...

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

public class Step6 extends Applet
{
    double[][] geometry = new double[][]
    {
        { -5, -5, -5 }, {  6, -5, -5 },
        {  5, -5, -5 }, {  5,  5, -5 },
        {  5,  5, -5 }, { -5,  5, -5 },
        { -5,  6, -5 }, { -5, -5, -5 },
            { -5, -5, -5 }, { -5, -5,  6 },
            {  5, -5, -5 }, {  5, -5,  5 },
```

```
            {  5,  5, -5 }, {  5,  5,  5 },
            { -5,  5, -5 }, { -5,  5,  5 },
        { -5, -5,  5 }, {  5, -5,  5 },
        {  5, -5,  5 }, {  5,  5,  5 },
        {  5,  5,  5 }, { -5,  5,  5 },
        { -5,  5,  5 }, { -5, -5,  5 },
    };

    int[][] vertex = new int[ geometry.length ][2];

    public void paint( Graphics g )
    {
        int CX = getSize().width / 2;
        int CY = getSize().height / 2;

        for( int i = 0; i < geometry.length; i = i + 1 )
        {
            double X = geometry[i][0];
            double Y = geometry[i][1];
            double Z = geometry[i][2];

            int PX = (int)(X / Z);
            int PY = (int)(Y / Z);

            vertex[i][0] = CX + PX;
            vertex[i][1] = CY - PY;
        }

        for( int i = 0; i < geometry.length; i = i + 2 )
        {
            g.drawLine( vertex[i][0],
                        vertex[i][1],
                        vertex[i + 1][0],
                        vertex[i + 1][1] );
        }
    }
}
```

- 7 -
3D DRAWING
PERSPECTIVE PROJECTION

In our formulas the *geometric centre* of the cube converges with the origin of the Cartesian axes O (0, 0, 0), and with the point of view itself.
We must therefore distance our cube of a certain amount to make it visible:
`double D = 15;`
`D` defines the distance of the cube from the centre of the perspective projection.
Nonetheless we can't see any cube...

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

public class Step7 extends Applet
{
    double[][] geometry = new double[][]
    {
        { -5, -5, -5 }, {  6, -5, -5 },
```

```
        {  5, -5, -5 }, {  5,  5, -5 },
        {  5,  5, -5 }, { -5,  5, -5 },
        { -5,  6, -5 }, { -5, -5, -5 },
            { -5, -5, -5 }, { -5, -5,  6 },
            {  5, -5, -5 }, {  5, -5,  5 },
            {  5,  5, -5 }, {  5,  5,  5 },
            { -5,  5, -5 }, { -5,  5,  5 },
        { -5, -5,  5 }, {  5, -5,  5 },
        {  5, -5,  5 }, {  5,  5,  5 },
        {  5,  5,  5 }, { -5,  5,  5 },
        { -5,  5,  5 }, { -5, -5,  5 },
    };

    int[][] vertex = new int[ geometry.length ][2];

    double D     = 15;

    public void paint( Graphics g )
    {
        int CX = getSize().width / 2;
        int CY = getSize().height / 2;

        for( int i = 0; i < geometry.length; i = i + 1 )
        {
            double X = geometry[i][0];
            double Y = geometry[i][1];
            double Z = geometry[i][2];

            Z = Z + D;

            int PX = (int)(X / Z);
            int PY = (int)(Y / Z);

            vertex[i][0] = CX + PX;
            vertex[i][1] = CY - PY;
        }

        for( int i = 0; i < geometry.length; i = i + 2 )
        {
            g.drawLine( vertex[i][0],
                        vertex[i][1],
                        vertex[i + 1][0],
                        vertex[i + 1][1] );
        }
    }
}
```

- 8 -
3D DRAWING
FIELD ANGLE

It seems that the cube has been drawn too little to be seen.
Let's add, therefore, a scale factor **RHO**:
`double RHO = 90;`
RHO is our scale factor, but it is also our field angle, it defines the quantity of space framed by the applet.
And here is our cube:

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

public class Step8 extends Applet
{
    double[][] geometry = new double[][]
    {
        { -5, -5, -5 }, {  6, -5, -5 },
        {  5, -5, -5 }, {  5,  5, -5 },
        {  5,  5, -5 }, { -5,  5, -5 },
        { -5,  6, -5 }, { -5, -5, -5 },
            { -5, -5, -5 }, { -5, -5,  6 },
            {  5, -5, -5 }, {  5, -5,  5 },
            {  5,  5, -5 }, {  5,  5,  5 },
            { -5,  5, -5 }, { -5,  5,  5 },
        { -5, -5,  5 }, {  5, -5,  5 },
        {  5, -5,  5 }, {  5,  5,  5 },
        {  5,  5,  5 }, { -5,  5,  5 },
        { -5,  5,  5 }, { -5, -5,  5 },
    };

    int[][] vertex = new int[ geometry.length ][2];

    double RHO   = 90;
    double D     = 15;

    public void paint( Graphics g )
    {
        int CX = getSize().width / 2;
        int CY = getSize().height / 2;

        for( int i = 0; i < geometry.length; i = i + 1 )
        {
            double X = geometry[i][0];
            double Y = geometry[i][1];
            double Z = geometry[i][2];

            Z = Z + D;

            double PX = X / Z;
            double PY = Y / Z;

            int SX = (int)(RHO * PX);
            int SY = (int)(RHO * PY);

            vertex[i][0] = CX + SX;
            vertex[i][1] = CY - SY;
        }

        for( int i = 0; i < geometry.length; i = i + 2 )
        {
            g.drawLine( vertex[i][0],
                        vertex[i][1],
                        vertex[i + 1][0],
                        vertex[i + 1][1] );
        }
    }
}
```

- 9 -
3D DRAWING
ROTATIONS

Now you must simply add the rotation formulas, extended to the three dimensions, to rotate the cube and observe it from every angle.

```
double XE = -X * S1 + Y * C1;
double YE = -X * C1 * C2 - Y * S1 * C2 + Z * S2;
double ZE = -X * S2 * C1 - Y * S2 * S1 - Z * C2;
```

Let's not forget about the two angles of horizontal and vertical rotation: **THETA** e **PHI**.

```
double THETA = 0.5;
double PHI = 0.9;
```

Here is our cube, now rotated.

~

Now you have the chance to modify four parameters:: **THETA**, **PHI**, **RHO**, **D**.
Keeping in mind the geometry of our example, try to draw and rotate a pyramid instead of a cube:

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

public class Step9 extends Applet
{
    double[][] geometry = new double[][]
    {
        { -5, -5, -5 }, {  6, -5, -5 },
        {  5, -5, -5 }, {  5,  5, -5 },
        {  5,  5, -5 }, { -5,  5, -5 },
        { -5,  6, -5 }, { -5, -5, -5 },
            { -5, -5, -5 }, { -5, -5,  6 },
            {  5, -5, -5 }, {  5, -5,  5 },
            {  5,  5, -5 }, {  5,  5,  5 },
            { -5,  5, -5 }, { -5,  5,  5 },
        { -5, -5,  5 }, {  5, -5,  5 },
        {  5, -5,  5 }, {  5,  5,  5 },
        {  5,  5,  5 }, { -5,  5,  5 },
        { -5,  5,  5 }, { -5, -5,  5 },
    };

    int[][] vertex = new int[ geometry.length ][2];

    double RHO   = 90;
    double THETA = 0.5;
    double PHI   = 0.9;
    double D     = 20;

    public void paint( Graphics g )
    {
        double S1 = Math.sin( THETA );
        double C1 = Math.cos( THETA );
        double S2 = Math.sin( PHI );
        double C2 = Math.cos( PHI );

        int CX = getSize().width / 2;
        int CY = getSize().height / 2;
```

```
        for( int i = 0; i < geometry.length; i = i + 1 )
        {
            double X = geometry[i][0];
            double Y = geometry[i][1];
            double Z = geometry[i][2];

            double XE = -X * S1 + Y * C1;
            double YE = -X * C1 * C2 - Y * S1 * C2 + Z * S2;
            double ZE = -X * S2 * C1 - Y * S2 * S1 - Z * C2;

            ZE = ZE + D;

            double PX = XE / ZE;
            double PY = YE / ZE;

            int SX = (int)(RHO * PX);
            int SY = (int)(RHO * PY);

            vertex[i][0] = CX + SX;
            vertex[i][1] = CY - SY;
        }

        for( int i = 0; i < geometry.length; i = i + 2 )
        {
            g.drawLine( vertex[i][0],
                        vertex[i][1],
                        vertex[i + 1][0],
                        vertex[i + 1][1] );
        }
    }
}
```

- 10 -
ANIMATION

At this point, we can at last come to the movement of the cube…
To do this, we must as first thing declare our applet as "executable":
`public class Step10 extends Applet` **`implements Runnable`**
And afterwards we launch a *thread* on this executable. In few words, a *thread* is a programme within a programme, in a *contemporary* execution.
In the method **`start`** we launch the *thread*:
```
self = new Thread( this );
self.start();
```
the method **`stop`** , on the other hand, stops it:
```
self = null;
```
The method **`run`** is the body of our *thread*: it modifies the values of the rotation angles **`THETA`** e **`PHI`**, and therefore it redraws the applet, continually, at intervals of ten milliseconds:

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

public class Step10 extends Applet implements Runnable
{
    private Thread self;

    public synchronized void start()
    {
```

```
        if( self == null )
        {
            self = new Thread( this );
            self.start();
        }
    }

    public synchronized void stop()
    {
        self = null;
    }

    public void run()
    {
        while( self == Thread.currentThread() )
        {
            try
            {
                self.sleep( 10L );
            }
            catch(InterruptedException ie)
            {
                break;
            }
            update( getGraphics() );
            THETA += .01;
            PHI += .001;
        }
    }

    double[][] geometry = new double[][]
    {
        { -5, -5, -5 }, {  6, -5, -5 },
        {  5, -5, -5 }, {  5,  5, -5 },
        {  5,  5, -5 }, { -5,  5, -5 },
        { -5,  6, -5 }, { -5, -5, -5 },
            { -5, -5, -5 }, { -5, -5,  6 },
            {  5, -5, -5 }, {  5, -5,  5 },
            {  5,  5, -5 }, {  5,  5,  5 },
            { -5,  5, -5 }, { -5,  5,  5 },
        { -5, -5,  5 }, {  5, -5,  5 },
        {  5, -5,  5 }, {  5,  5,  5 },
        {  5,  5,  5 }, { -5,  5,  5 },
        { -5,  5,  5 }, { -5, -5,  5 },
    };

    int[][] vertex = new int[ geometry.length ][2];

    double RHO   = 90;
    double THETA = 0.5;
    double PHI   = 0.9;
    double D     = 20;

    public void paint( Graphics g )
    {
        double S1 = Math.sin( THETA );
        double C1 = Math.cos( THETA );
        double S2 = Math.sin( PHI );
        double C2 = Math.cos( PHI );

        int CX = getSize().width / 2;
```

```
        int CY = getSize().height / 2;

        for( int i = 0; i < geometry.length; i = i + 1 )
        {
            double X = geometry[i][0];
            double Y = geometry[i][1];
            double Z = geometry[i][2];

            double XE = -X * S1 + Y * C1;
            double YE = -X * C1 * C2 - Y * S1 * C2 + Z * S2;
            double ZE = -X * S2 * C1 - Y * S2 * S1 - Z * C2;

            ZE = ZE + D;

            double PX = XE / ZE;
            double PY = YE / ZE;

            int SX = (int)(RHO * PX);
            int SY = (int)(RHO * PY);

            vertex[i][0] = CX + SX;
            vertex[i][1] = CY - SY;
        }

        for( int i = 0; i < geometry.length; i = i + 2 )
        {
            g.drawLine( vertex[i][0],
                        vertex[i][1],
                        vertex[i + 1][0],
                        vertex[i + 1][1] );
        }
    }
}
```

- 11 -
INTERACTION WITH THE USER

Finally, to make our applet interact (and therefore our cube) with the user it is first of all necessary to tell the *AWT* (the *graphic system* of Java) that we want to handle the events linked to the mouse, and you can do it within the method **`init`** as follows:

```
enableEvents( AWTEvent.MOUSE_EVENT_MASK | AWTEvent.MOUSE_MOTION_EVENT_MASK );
```

In this way the *AWT* will notify (in an *asynchronic* way) the following methods

```
protected void processMouseEvent( MouseEvent e )
```

and

```
protected void processMouseMotionEvent( MouseEvent e )
```

respectively of an event of the mouse as the pressing of a button and the movement of the mouse, on the applet.
Basically you must change the values of the rotation angles **`THETA`** and **`PHI`** coherently with the movements of the user; then you draw a new image with the new values. The value **`scale`** inside the method **`processMouseMotionEvent`**, serves to regulate the speed of rotation, which otherwise would result too fast.
Try to click and drag the mouse on the applet:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
```

```
public class Step11 extends Applet
{
    public void init()
    {
        enableEvents( AWTEvent.MOUSE_EVENT_MASK |
                      AWTEvent.MOUSE_MOTION_EVENT_MASK );
    }

    private Point click;

    protected void processMouseEvent( MouseEvent e )
    {
        if( e.getID() == e.MOUSE_PRESSED )
        {
            click = e.getPoint();
            e.consume();
        }
        super.processMouseEvent( e );
    }

    protected void processMouseMotionEvent( MouseEvent e )
    {
        if( e.getID() == e.MOUSE_DRAGGED )
        {
            Point p = e.getPoint();
            final double scale = .01;
            THETA -= (p.x - click.x) * scale;
            PHI -= (p.y - click.y) * scale;
            click = p;
            repaint();
            e.consume();
        }
        super.processMouseMotionEvent( e );
    }

    double[][] geometry = new double[][]
    {
        { -5, -5, -5 }, {  6, -5, -5 },
        {  5, -5, -5 }, {  5,  5, -5 },
        {  5,  5, -5 }, { -5,  5, -5 },
        { -5,  6, -5 }, { -5, -5, -5 },
            { -5, -5, -5 }, { -5, -5,  6 },
            {  5, -5, -5 }, {  5, -5,  5 },
            {  5,  5, -5 }, {  5,  5,  5 },
            { -5,  5, -5 }, { -5,  5,  5 },
        { -5, -5,  5 }, {  5, -5,  5 },
        {  5, -5,  5 }, {  5,  5,  5 },
        {  5,  5,  5 }, { -5,  5,  5 },
        { -5,  5,  5 }, { -5, -5,  5 },
    };

    int[][] vertex = new int[ geometry.length ][2];

    double RHO   = 90;
    double THETA = 1.0;
    double PHI   = 1.0;
    double D     = 20;

    public void paint( Graphics g )
    {
        double S1 = Math.sin( THETA );
```

```
        double C1 = Math.cos( THETA );
        double S2 = Math.sin( PHI );
        double C2 = Math.cos( PHI );

        int CX = getSize().width / 2;
        int CY = getSize().height / 2;

        for( int i = 0; i < geometry.length; i = i + 1 )
        {
            double X = geometry[i][0];
            double Y = geometry[i][1];
            double Z = geometry[i][2];

            double XE = -X * S1 + Y * C1;
            double YE = -X * C1 * C2 - Y * S1 * C2 + Z * S2;
            double ZE = -X * S2 * C1 - Y * S2 * S1 - Z * C2;

            ZE = ZE + D;

            double PX = XE / ZE;
            double PY = YE / ZE;

            int SX = (int)(RHO * PX);
            int SY = (int)(RHO * PY);

            vertex[i][0] = CX + SX;
            vertex[i][1] = CY - SY;
        }

        for( int i = 0; i < geometry.length; i = i + 2 )
        {
            g.drawLine( vertex[i][0],
                        vertex[i][1],
                        vertex[i + 1][0],
                        vertex[i + 1][1] );
        }
    }
}
```
