JGL

Juggle - A 3D Graphics Class Library for Java™

Reference Manual

Revision 1.0

Copyright

JGL – An OpenGL^{TM} -like 3D Graphics Class Library for Java^{TM}.
Copyright (C) 1997 Mark Matthews – Purdue University CADLAB

TRADEMARKS

JGL Overview

The JGL package is a Java™ class library that provides the Java™ developer with much of the functionality of OpenGL®, but with a pure Java™ solution. These features include, but are not limited to:

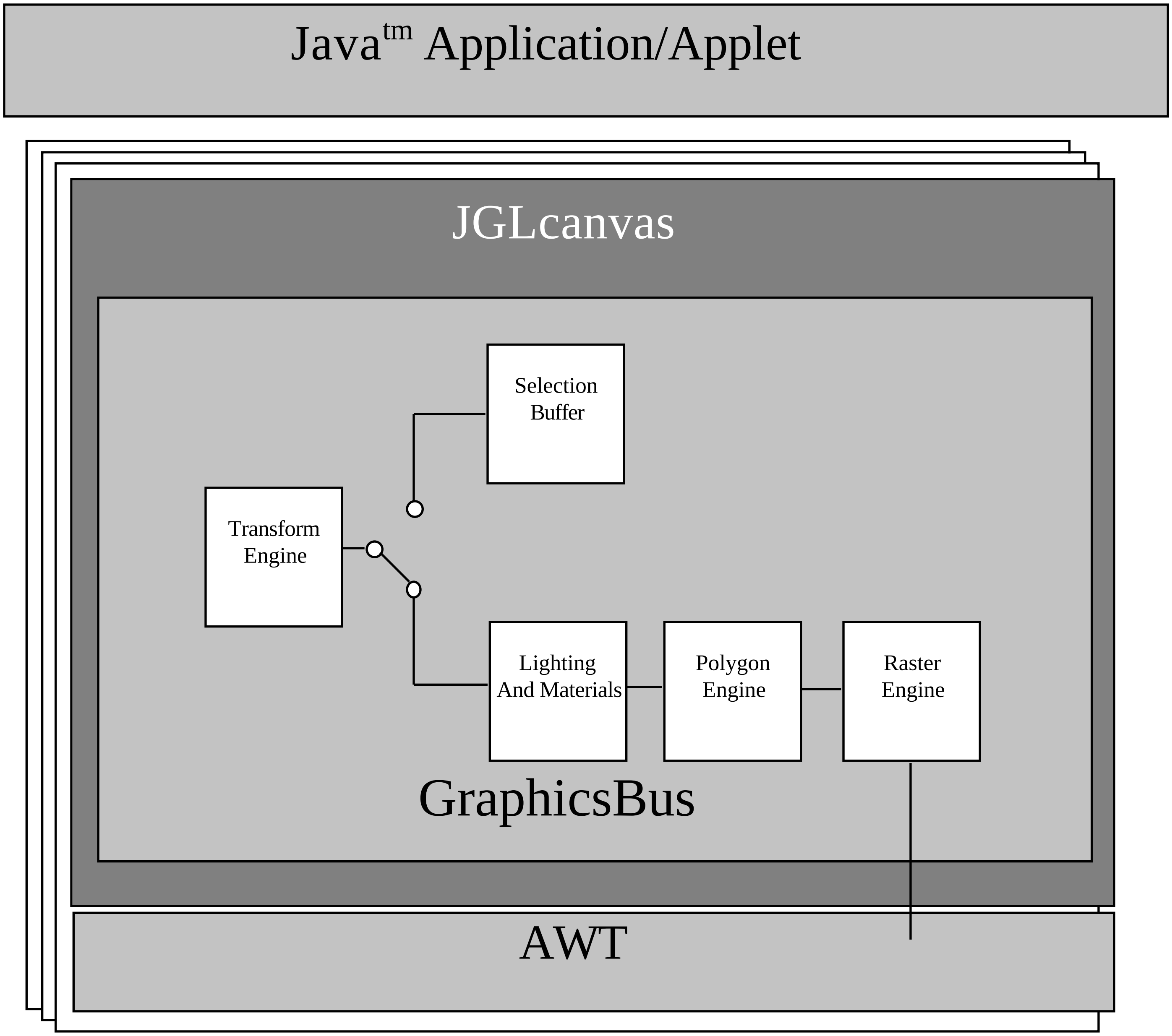
- ⦿ Geometric primitives (points, lines, and polygons)
- ⦿ RGB and RGBA color modes
- ⦿ Display list or immediate mode
- ⦿ Viewing and modeling transformations
- ⦿ Hidden Surface Removal (depth buffer and culling)
- ⦿ Alpha Blending (transparency)
- ⦿ Atmospheric Effects (fog, smoke, and haze)
- ⦿ Selection and Picking

JGL uses an API very similar to OpenGL®. Many sub-systems in JGL are based upon Mesa, which is another 3D library for non-Java™ applications. JGL cannot be called an implementation of OpenGL as it is not an OpenGL licensed implementation. Also, JGL cannot claim to be OpenGL conformant since the conformance tests are only available to OpenGL licensees. At this point however, JGL is one of the few non-VRML based Java™ 3D libraries available.

Developers who have programmed with OpenGL® will find JGL's API very familiar. Many functions have the same or similar names. Some things have been simplified, or re-architected to conform to object-oriented ideals. The small differences will become obvious with a glance through Chapter 3, the JGL Reference Pages.

With the following disclaimers out of the way, the impatient developer can get started by looking at the "Hello World" of JGL in Appendix A. The curious may read ahead to find out about some of the internals of the JGL library.

Chapter 2
Architecture of the JGL library


Java^{tm} Application/Applet
JGLcanvas
Selection Buffer
Transform Engine
Lighting And Materials
Polygon Engine
Raster Engine
GraphicsBus
AWT

Chapter 3
JGL Reference Pages

Begin/End

Name

Begin/End – Delimit the vertices of a primitive or a group of primitives

Method Prototype

public void **Begin**(int *mode*) throws **JGLexception**

public void **End**(void) throws **JGLexception**

Parameters

mode - Specifies the primitive or primitives that will be created from vertices presented between **Begin** and **End**. Three constants are accepted: **jgl.LINES**, **jgl.LINE_LOOP**, and **jgl.POLYGON**.

Throws

JGLexception - If mode is set to an unacceptable value, or if a command other than **Vertex**, **Color**, or **Normal** is made, or if **End** is called before a corresponding **Begin**.

Description

Begin and End delimit the vertices that form primitives and groups of primitives in JGL. Begin accepts a single argument which specifies which type of primitive you wish to draw. With *I* as an integer index, and *N* representing the total number of vertices, the interpretations of the modes are as follows:

BlendFunc

Name

BlendFunc – Specify the blending functions used in pixel arithmetic

Method Prototype

public void **BlendFunc**(int *sfactor,* int *dfactor*) throws **JGLexception**

Parameters

sfactor – Specifies how the RGBA source blending factors are computed. Acceptable values are: **jgl.ZERO**, **jgl.ONE**, **jgl.DST_COLOR**, **jgl.ONE_MINUS_DST_COLOR**, **jgl.SRC_ALPHA**, **jgl.ONE_MINUS_SRC_ALPHA**, **jgl.DST_ALPHA**, **jgl.ONE_MINUS_DST_ALPHA** and **jgl.SRC_ALPHA_SATURATE**.

dfactor – Specifies how the RGBA source blending factors are computed. Acceptable values are: **jgl.ZERO**, **jgl.ONE**, **jgl.SRC_COLOR**, **jgl.ONE_MINUS_SRC_COLOR**, **jgl.SRC_ALPHA**, **jgl.ONE_MINUS_SRC_ALPHA**, **jgl.DST_ALPHA**, and **jgl.ONE_MINUS_DST_ALPHA**.

Throws

JGLexception - If *sfactor* and/or *dfactor* is set to an invalid value.

Description

If blending is enabled (see **Enable/Disable**), then the color values for all pixels are written stored into a color buffer as well as being drawn to the screen. **BlendFunc** specifies how the color for new pixels are computed based upon two blending functions, *sfactor* and *dfactor.* By default, blending is disabled.

When JGL blends pixel colors, it does so in a two-stage process. First, the source and destination scaling factors are specified. The *sfactor* argument specifies which of nine scaling functions are applied to the source color components. *dfactor* specifies which of eight scaling functions are applied to the destination color components. The eleven possible combinations are described in the table below. In the table, the RGBA values of the source and destination are indicated with the subscripts s and d, respectively. The relevant factor column indicates which constant can be used to specifiy the source or destination blending factor.

With the source and destination factors computed, the corresponding components in the two sets of RGBA are added together:

Let the source and destination blending factors be (S_r, S_g, S_b, S_a) and (D_r, D_g, D_b, D_a), and the RGBA components of the source and destination color values be indicated with a subscript of s or d. The final RGBA values are then given by:

$$(R_sS_r + R_dD_r,\ G_sS_g + G_dD_g,\ B_sS_b + B_dD_b,\ A_sS_a + A_dD_a)$$

Constant	Relevant Factor	Blend Factor
jgl.ZERO	source or destination	(0, 0, 0, 0)
jgl.ONE	source or destination	(1, 1, 1, 1)
jgl.DST_COLOR	source	(R_d, G_d, B_d, A_d)
jgl.SRC_COLOR	destination	(R_s, G_s, B_s, A_s)
jgl.ONE_MINUS_DST_COLOR)	source	(1, 1, 1, 1) - (R_d, G_d, B_d, A_d
jgl.ONE_MINUS_SRC_COLOR	destination	(1, 1, 1, 1) - (R_s, G_s, B_s, A_s)
jgl.SRC_ALPHA	source or destination	(A_s, A_s, A_s, A_s)
jgl.ONE_MINUS_SRC_ALPHA	source or destination	(1, 1, 1, 1) - (A_s, A_s, A_s, A_s)
jgl.DST_ALPHA	source or destination	(A_d, A_d, A_d, A_d)
jgl.ONE_MINUS_DST_ALPHA	source or destination	(1, 1, 1, 1)-(A_d, A_d, A_d, A_d)
jgl.SRC_ALPHA_SATURATE	source	(f, f, f, 1); f=min(A_s , 1- A_d)

Color3

Name

Color3 – Set the current drawing color

Method Prototype

```
public void Color3( double red,
                    double green,
                    double blue )
```

Parameters

red, green, blue – Specify new values for the current color

Description

JGL stores a current 3-valued color. **Color3** specifies new red, green and blue values. The current color is assigned to each vertex as it is created. Color3 values are clamped to (0.0 , 1.0). You may call **Color3** at any time.

DrawBuffer

Name

DrawBuffer – Specify which image buffers are drawn into

Method Prototype

public void **DrawBuffer**(int *mode*) throws **JGLexception**

Parameters

mode – Either **jgl.FRONT** or **jgl.BACK** .

Throws

JGLexception – If *mode* is set to an unacceptable value, or if **DrawBuffer** is called between **Begin** and **End**.

Description

When JGL draws on the canvas, it can draw to either one of two buffers. The visible buffer, specified with **gl.FRONT**, or a “back” buffer, specified with **gl.BACK** which is not visible, but can be drawn on, and it’s contents later copied to the front, visible buffer.

Enable/Disable

Name

Enable/Disable – Enable or disable JGL canvas capabilities

Method Prototype

public void **Enable/Disable**(int *capability*) throws JGLexception

Parameters

capability – A symbolic constant specifying a JGL capability.

Throws

JGLexception - If *capability* is set to an unacceptable value, or if **Enable** or **Disable** is called between **Begin** and **End**.

Description

Enable and **Disable** control certain JGL capabilities. Both **Enable** and **Disable** take a single argument, *capability*, which can be one of the following values:

jgl.BLENDING If enabled, blend incoming vertex color with the value in the color buffer. See **BlendFunc**.

jgl.CULL_FACE If enabled, cull polygons based on whether they are front or rear-facing. See **CullFace**.

jgl.DEPTH_TEST If enabled, do z-comparisons, and update the z-buffer for any pixels drawn. See **DepthFunc**.

jgl.LIGHTING If enabled, use the current lighting parameters to compute per-vertex colors. Otherwise, use the current color for each vertex. See **Color3**, **Material**, **Light** and **LightModel**.

jgl.NORMALIZE If enabled, normal vectors created by **Normal3** are scaled to unit length after transformation. See **BlendFunc**.

Enable/DisableLight

Note:
Unlike OpenGL^{â}, JGL has no fixed number of lights. Therefore the enabling and disabling of Individual lights has been accomodated with this new funcition call.

Name

Enable/DisableLight – Enable or disable individual light sources.

Method Prototype

public void **Enable/Disable**(int *light*) throws **JGLexception**

Parameters

light – An integer specifying an individual light.

Throws

JGLexception - ?????

Description

EnableLight and **DisableLight** determine if an individual light's parameters will be evaluated during lighting calculations. If you enable a light that does not yet exist, it will be created for you. Disabling a light that does not exist will throw an exception. While the integer number corresponding to a light must be unique, it is completely arbitrary. However, light number 0 is the JGL default light, as in OpenGL^{â}, and will be present, but disabled when a lighting model is in place. It is recommended that light numbers start at 0, and be numbered consecutively, as in OpenGL^{â} for backwards compatibility.

Frustum

Name

Frustum - Multiply the current matrix by a perspective projection matrix

Method Prototype

```
public void Frustum( double left,
                     double right,
                     double bottom,
                     double top,
                     double near,
                     double far ) throws JGLexception
```

Parameters

left - Coordinate for the left vertical clipping plane
right - Coordinate for the right vertical clipping plane
bottom - Coordinate for the bottom horizontal clipping plane
top - Coordinate for the top horizontal clipping plane
near - Distance to near depth clipping plane
far - Distance to far depth clipping plane

Throws

JGLexception - If *near* or *far* are not positive, or if **Frustum** is called between **Begin** and **End**.

Description

Frustum creates a perspective projection matrix. (*left*, *bottom*, -*near*) and (*right*, *top*, -*near*) specify the points on the near clipping plane that are mapped to the lower left and upper right corners of the window, assuming that the eye is located at (0, 0, 0). -*far* specifies the location of the far clipping plane. The values for *near* and *far* must be positive.

Light

Name

Light – Set parameters for individual light sources

Method Prototype

```
public void Light(  int light,
                    int parameterName,
                    double parameter ) throws JGLexception

public void Light(  int light,
                    int parameterName,
                    Vector4 parameter ) throws JGLexception
```

Parameters

light - Specifies a light.

parameterName - Specifies a light source parameter for a light. One of: **jgl.SPOT_EXPONENT, jgl.SPOT_CUTOFF, jgl.CONSTANT_ATTENUATION, jgl.LINEAR_ATTENUATION, jgl.QUADRATIC_ATTENUATION, gl.AMBIENT, jgl.DIFFUSE, jgl.SPECULAR, jgl.POSITION**, or **jgl.SPOT_DIRECTION**.

Description

To be done.

LoadIdentity

Name

LoadIdentity – Replace the current matrix with the identity matrix.

Method Prototype

public void **LoadIdentity**(void) throws **JGLexception**

Throws

JGLexception – If **LoadIdentity** is called between **Begin** and **End**.

Description

LoadIdentity replaces the current matrix with the identity matrix. It is functionally equivalent to calling **LoadMatrix** with the following matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

but is written to be more efficient.

LoadMatrix

Name

LoadIdentity – Replace the current matrix with an arbitrary matrix.

Method Prototype

public void **LoadIdentity**(**GMatrix** Matrix) throws **JGLexception**

Throws

JGLexception – If **LoadMatrix** is called between **Begin** and **End**.

Description

LoadMatrix replaces the current matrix with an arbitrary matrix. Using JGL methods for transformations will be more efficient in most cases.

MatrixMode

Name

MatrixMode – Select the matrix to work with.

Method Prototype

public void **MatrixMode**(int *mode*) throws **JGLexception**

Parameters

mode – Specifies which stack of matrices is the target for subsequent matrix operations. These values are accepted: **jgl.MODELVIEW**, **jgl.PROJECTION**, or **jgl.TEXTURE.**

Throws

JGLexception - If *mode* is set to an unknown value, or if **MatrixMode** is called between **Begin** and **End**.

Description

MatrixMode sets the current matrix mode. *mode* can be one of the following values:

jgl.MODELVIEW	Matrix operations applied after the call to **MatrixMode** will be applied to the modelview matrix stack.
jgl.PROJECTION	Matrix operations applied after the call to **MatrixMode** will be applied to the projection matrix stack.
jgl.TEXTURE	Matrix operations applied after the call to **MatrixMode** will be applied to the texture matrix stack.

Note: Texture mapping is not implemented at this time in the JGL library.

Normal3

Name

Normal3 – Set the current normal vector.

Method Prototype

```
public void Normal3(     double x,
                    double y,
                    double z )
```

Parameters

x, y, and z - Specify the *x*, *y*, and *z* components of the current normal. The initial value of the current normal is (0, 0, 1).

Description

The current normal is set to the given normal whenever **Normal3** is called. Normals specified with **Normal3** need not be unit length. The current normal can be updated at anytime. **Normal3** may be called between a call to **Begin** and the corresponding call to **End**.

Ortho

Name

Ortho - Multiply the current matrix by an orthographic projection matrix

Method Prototype

```
public void Ortho( double left,
                   double right,
                   double bottom,
                   double top,
                   double near,
                   double far ) throws JGLexception
```

Parameters

left -	Coordinate for the left vertical clipping plane
right -	Coordinate for the right vertical clipping plane
bottom -	Coordinate for the bottom horizontal clipping plane
top -	Coordinate for the top horizontal clipping plane
near -	Distance to near depth clipping plane
far -	Distance to far depth clipping plane

Throws

JGLexception – If **Ortho** is called between **Begin** and **End**.

Description

Ortho creates a perspective projection matrix producing a parallel projection. (*left*, *bottom*, -*near*) and (*right*, *top*, -*near*) specify the points on the near clipping plane that are mapped to the lower left and upper right corners of the window, assuming that the eye is located at (0, 0, 0). -*far* specifies the location of the far clipping plane. The values for *near* and *far* can be positive or negative.

Push/PopMatrix

Name

Push/PopMatrix – Store/Restore state of current matrix.

Method Prototype

public void **Push/PopMatrix**(void) throws **JGLexception**

Throws

JGLexception – If **PushMatrix** or **PopMatrix** is called between **Begin** and **End**.

Description

PushMatrix and **PopMatrix** allow transformation states to be saved and restored by placing or removing the contents of the current matrix (see **MatrixMode**) onto a stack of respective matrices. You are allowed to store as many matrices at one time as memory will allow.

Rotate

Name

Rotate - Multiply the current matrix by a rotation matrix.

Method Prototype

```
public void Rotate ( double angle,
                     double x,
                     double y,
                     double z ) throws JGLexception
```

Parameters

angle – Specifies the angle in degrees.
x, *y, and z* – Define the axis to rotate about.

Throws

JGLexception – If **Rotate** is called between **Begin** and **End**.

Description

Rotate computes a rotation matrix to perform a counterclockwise rotation of *angle* degrees about the vector from the origin through the point (*x*, *y*, *z*).

The current matrix (see **MatrixMode**) is multiplied by this rotation matrix, with the product replacing the current matrix.

If the matrix mode is either **jgl.MODELVIEW** or **jgl.PROJECTION**, all objects drawn after the call to **Rotate** is made are rotated. Use **PushMatrix** and **PopMatrix** to save and restore the desired coordinate system.

Scale

Name

Scale - Multiply the current matrix by a scaling matrix.

Method Prototype

public void **Scale** (double *x*,
double *y*,
double *z*) throws **JGLexception**

Parameters

x, y, and z – Specify the scale factor along the x, y, or z axis respectively.

Throws

JGLexception – If **Scale** is called between **Begin** and **End**.

Description

Scale performs a scaling along the x,y, and z axes. The arguments specify the scale factors along each of the three axes. The matrix generated is:

$$\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The current matrix (see **MatrixMode**) is multiplied by this scaling matrix, with the product replacing the current matrix.

If the matrix mode is either **jgl.MODELVIEW** or **jgl.PROJECTION**, all objects drawn after the call to **Scale** is called are scaled. Use **PushMatrix** and **PopMatrix** to save and restore the desired coordinate system.

SetGraphics

Name

SetGraphics – Set the current graphics context.

Method Prototype

public void **SetGraphics** (Graphics *g*) throws **JGLexception**

Parameters

g – The **awt.Graphics** context that subsequent drawing methods will draw to.

Description

SetGraphics sets the current drawing context for the **JGLcanvas**. This also allocates memory for the back buffer and z-buffer as necessary. The Graphics *g* argument is usually the one that you are passed in your overridden **update()** or **paint()** method.

Translate

Name

Translate - Multiply the current matrix by a translation matrix.

Method Prototype

```
public void Translate ( double x
                        double y,
                        double z ) throws JGLexception
```

Parameters

x, y, and z - Values for a translation vector in the corresponding plane

Throws

JGLexception – If **Translate** is called between **Begin** and **End**.

Description

Translate moves the coordinate system origin to the point specified by (x, y, z). This vector is then used to compute a 4x4 translation matrix:

$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The current matrix (see **MatrixMode**) is multiplied by this translation matrix, with the product replacing the current matrix.

If the matrix mode is either **jgl.MODELVIEW** or **jgl.PROJECTION**, all objects drawn after the call to **Translate** is called are translated. Use **PushMatrix** and **PopMatrix** to save and restore the desired coordinate system.

Vertex3

Name

Vertex3 – Specify a vertex.

Method Prototype

public void **Vertex3** (double x,
double *y,*
double *z*) throws **JGLexception**

Parameters

x, y, and z – Specify *x, y,* and *z* coordinates of a vertex.

Throws

JGLexception – If **Vertex3** is called between **Begin** and **End**.

Description

Vertex3 is used within Begin and End to specify line and polygon vertices. Current color, normal, and texture coordinates are associated with the vertex upon calling **Vertex3**.

zclear

Name

zclear – Clears all values in the z-buffer to the point furthest from the front clipping plane.

Method Prototype

public void **zclear** (void) throws **JGLexception**

Throws

JGLexception – If **zclear** is called and depth testing is not enabled. (See **Enable**).

Description

To be done.

Appendix A
Example Applet

Listing A – AppletTest.java

```
/* juggle/AppletTest.java - Test canvas for JGL package
 *
 * $Id$
 *
 * Written by Mark Matthews, 1996.
 *
 */



import java.util.*;
import java.awt.*;
import java.applet.*;
import cadlab.jgl.*;

public class AppletTest extends Applet {


  public void init()
  {
    try
      {
      ThreeDCanvas TestCanvas = new ThreeDCanvas();
      this.add( TestCanvas );
      }
    catch ( JGLexception e )
      {
      // do nothing
      }
  }

}

class ThreeDCanvas extends JGLcanvas {

  private double azimuth = 45, inclination = 70, dist=0;
  private int dx = 0, dy = 0, x1 = 0, x2 = 0, y1 = 0, y2 = 0;
  private double panx = 0, pany = 0;

  private Vector4 ambient      = new Vector4( 0.0, 0.0, 0.0, 1.0 );
  private Vector4 diffuse      = new Vector4( 1.0, 1.0, 1.0, 0.0 );
  private Vector4 specular     = new Vector4( 1.0, 1.0, 1.0, 1.0 );
  private Vector4 spot_pos     = new Vector4( 0.0, 3.0, 2.0, 0.0 );
  private Vector4 lmodel_ambient = new Vector4( 0.4, 0.4, 0.4, 1.0 );

   // For the wireframe navigation
  protected boolean old_mouseDown_stat = false;
  protected boolean mouseDown = false;

  public ThreeDCanvas() throws JGLexception
  {
    Enable( jgl.DEPTH_TEST );
    Enable( jgl.LIGHTING );
    Enable( jgl.CULL_FACE );

    // Enable a light source
    EnableLight( 0 );
    EnableLight( 1 );

    Vector4 spot_ambient = new Vector4( 1.0, 1.0, 1.0, 1.0 );
    Vector4 spot_color = new Vector4( 1.0, 1.0, 1.0, 0.0 );
    Vector4 spot_pos = new Vector4( 0.0, 0.0, 1.0, 1.0 );
    Vector4 spot_dir = new Vector4( 0.0, 0.0, -1.0, 0.0 );

    // Here's our lighting model
    Light( 1, jgl.AMBIENT, spot_ambient );
    Light( 1, jgl.DIFFUSE, spot_color );
    Light( 1, jgl.SPOT_DIRECTION, spot_dir );
    Light( 1, jgl.POSITION, spot_pos );
    Light( 1, jgl.SPOT_CUTOFF, 180.0F );

    // Materials
    Vector4 mat_ambient = new Vector4( 0.3, 0.3, 0.5, 1.0 );
    Vector4 mat_diffuse = new Vector4( 0.5, 0.5, 0.8, 1.0 );
    Vector4 mat_specular = new Vector4( 0.3, 0.3, 0.5, 1.0 );
    Vector4 mat_emission = new Vector4( 0.0, 0.0, 0.0, 0.0 );
    Material( jgl.FRONT, jgl.AMBIENT, mat_ambient );
    Material( jgl.FRONT, jgl.DIFFUSE, mat_diffuse );
    Material( jgl.FRONT, jgl.SPECULAR, mat_specular );
    Material( jgl.FRONT_AND_BACK, jgl.SHININESS, 20.0F );


  }

  public void Reset()
  {
    azimuth = 0;
    inclination =0;
    repaint();
  }

  public void set2D3D() throws JGLexception
  {
    if( old_mouseDown_stat != mouseDown )
      {
      if(mouseDown)
        {
          Disable( jgl.DEPTH_TEST );
          Disable( jgl.LIGHTING );
          Disable( jgl.CULL_FACE );
          Disable( jgl.NORMALIZE );
        }
      else
        {
          Enable( jgl.DEPTH_TEST );
          Enable( jgl.LIGHTING );
          Enable( jgl.CULL_FACE );
          Enable( jgl.NORMALIZE );
        }
      old_mouseDown_stat = mouseDown ;
      }
  }

  public void draw() throws JGLexception
```

```
{
  set2D3D();

  if( mouseDown )
    drawWire();
  else
    drawSmooth();
}

public boolean handleEvent( Event event )
{

  switch( event.id )
    {


    // look for MOUSE_DOWN events
    case Event.MOUSE_UP:
      mouseDown = false;
      repaint();
    break;

    case Event.MOUSE_DOWN:
      mouseDown = true;

    x1 = event.x;
    y1 = event.y;
    break;

    case Event.MOUSE_DRAG:
    x2 = event.x;
    y2 = event.y;
    dx = x2 -x1;
    dy = y2 - y1;
    if ( dx != 0 || dy != 0 )
      {
        if ( (event.modifiers & Event.META_MASK) != 0 )
          {
          panx += dx / 50.0;
          pany -= dy / 50.0;
          }
        else if ( (event.modifiers & Event.ALT_MASK) != 0 )
          {
          dist += dy / 4;
          }
        else
          {
          azimuth += 1 * dx;
          inclination += 1 * dy;
          }
      }
    x1 = x2;
    y1 = y2;
    repaint();
    break;
    }
  return false;
}
```

```
  public void paint( Graphics g )
  {

    try
      {
      SetGraphics( g );
      draw();
      }
     catch ( JGLexception e )
      {
      // Do nothing
      }
  }

  public void drawSmooth() throws JGLexception
  {

    try
      {

      MatrixMode(jgl.MODELVIEW);
      LoadIdentity();
      zclear();

      // Transform the "Eye Coordinates"
      Translate( panx, pany, -20.0 + dist );
      Rotate( azimuth, 0, 1, 0 );
      Rotate( -inclination, 1, 0 ,0 );

      // Create a projection
      MatrixMode( jgl.PROJECTION );
      LoadIdentity();
      Perspective( 30.0, 1.0, 0.1, 100.0 );

      MatrixMode( jgl.MODELVIEW );

      // Draw to the back buffer
      DrawBuffer( jgl.BACK );


      Clear();


      smooth();

      // Copy the back buffer to the front
      SwapBuffers();

      }
    catch ( JGLexception e )
      {
      // Do nothing
      }
  }

  public void drawWire() throws JGLexception
  {
    try
      {
      Color3( 0.3F, 0.3F, 0.5F );
      MatrixMode(jgl.MODELVIEW);
      LoadIdentity();

      // Transform the "Eye Coordinates"
      Translate( panx, pany, -20.0 + dist );
      Rotate( azimuth, 0, 1, 0 );
      Rotate( -inclination, 1, 0 ,0 );

      // Create a projection
      MatrixMode( jgl.PROJECTION );
      LoadIdentity();
      Perspective( 30.0, 1.0, 0.1, 100.0 );

      MatrixMode( jgl.MODELVIEW );

      // Draw to the back buffer
      DrawBuffer( jgl.BACK );


      Clear();


      wire();

      // Copy the back buffer to the front
      SwapBuffers();

      }
    catch ( JGLexception e )
      {
      // Do nothing
      }
  }


  private void smooth() throws JGLexception
  {
    Begin(jgl.POLYGON);
    Normal3( 0.000000, -1.000000, 0.000000);
    Vertex3( 4.000000, 0.000000, -1.000000 );
    Vertex3( 4.000000, 0.000000, 0.000000 );
    Vertex3( 0.000000, 0.000000, 0.000000 );
    Vertex3( 0.000000, 0.000000, -1.000000 );
    End();

    Begin(jgl.POLYGON);
    Normal3( 0.195091, -0.980785, 0.000000);
    Vertex3( 3.107365, 2.471178, 0.000000 );
    Normal3( 0.195091, -0.980785, 0.000000);
    Vertex3( 3.107365, 2.471178, -1.000000 );
    Normal3( 0.098017, -0.995185, 0.000000);
    Vertex3( 3.400000, 2.500000, -1.000000 );
    Normal3( 0.098017, -0.995185, 0.000000);
    Vertex3( 3.400000, 2.500000, 0.000000 );
    End();

      .
      .
      .
  }



  public void wire() throws JGLexception
  {
    Begin(jgl.LINE);
    Vertex3( 4.000000, 0.000000, -1.000000 );
    Vertex3( 4.000000, 0.000000, 0.000000 );
    End();

    Begin(jgl.LINE);
    Vertex3( 0.000000, 0.000000, 0.000000 );
    Vertex3( 4.000000, 0.000000, 0.000000 );
    End();

      .
      .
      .
  }

}
```