# 3D Graphics in Java with Java Open GL - *JOGL 2*

**Introduction**

**Installing *JOGL* on *Windows***

***JOGL* connection to Java: *JOGL* event listener**

**Geometric transformations:**
- **viewport, projection, viewing transformations**
- **modeling transformation**

**Geometric modeling with GLU library**

**Examples with GLU objects and wireframe rendering**

**Geometric modeling with lines and polygons**

**Hidden surface removal**

**Lighting:**
- **lighting model**
- **optical properties of surfaces**
- **light sources**

**Examples with lighting and surface rendering**

**References:**

http://jogamp.org/ (for JOGL 2.0)

http://download.java.net/media/jogl/jogl-2.x-docs/

http://www.glprogramming.com/red/

# Introduction

- *3D graphics*:
  - *step 1:* define 3D objects in 3D space
  - *step 2:* rendering with a *virtual camera*
    → virtual photographs displayed on a panel of the GUI

- sequence of *geometric transformations* between the panel and the 3D objects:

  *viewport*, *projection*, *viewing*, *modeling* transformations

  → *projection* of the 3D objects over the panel

- *wireframe rendering*: surfaces of objects sketched out with a grid of lines
- *surface rendering*: surfaces of objects represented realistically:
  - lighting of the scene
  - reflection of light over the objects

- *OpenGL*

  *OpenGL* is a *state machine*

  → various state variables that keep their values until we change them:
  - current color
  - current normal
  - current projection matrix, modelview matrix
  - current drawing style
  - current shading model

  . . . . . .

  *OpenGL* has a rendering pipeline with several buffers

  → *flushing* required to force the execution of all drawing instructions

- *Java Open GL*

  *event driven system*: *OpenGL* modeling and rendering
  are specifed inside the event methods of **GLEventListener**

  *double-buffering*: one buffer is displayed while the other is being drawn
  then, the two buffers are swapped and vice-versa

  = allows smooth animation

- *GLU*: *OpenGL Utility Library*

  provides more elaborate functions than *OpenGL*:

  - easy specification of projection and viewing transformations: **gluPerspective**
    **gluLookAt**
  - creation of simple objects: quadric surfaces (cylinders, disks, spheres, ...)
    NURBS curves and surfaces

  . . . . . .

- *GLUT*: *OpenGL Utility Toolkit*

  toolkit to manage windows and events

→ we will use *Swing* and *AWT* instead.

# Installing *JOGL* on *Windows*

- *if not done already, install Java JDK and JRE*

  for example from files: **jdk-6u23-windows-x64.exe**
  **jre-6u23-windows-x64.exe**

- *install JOGL on Windows 64 bits:*

  download from **http://jogamp.org/deployment/autobuilds/master/**

  → open most recent folder **jogl-b******

  *for example:* **jogl-b269-2011-01-22_00-21-39/**

  → download **jogl-2.0-b269-20110122-windows-amd64.zip**
  ( **jogl-2.0-b269-20110122-windows-i586.zip** for *Windows 32 bits*)

  unzip this file
  → put the files inside it into the folder **C:\JOGL2** on your PC

  You now have in **C:\JOGL2** the files:

  **artifact.properties**
  **CHANGELOG.txt**
  **etc**
  **jar**
  **jnlp-files**
  **lib**
  **LICENSE.txt**
  **README.txt**
  **Userguide.html**
  **____VERSION.txt**

- *update system variables:*

  **Computer → Properties → Advanced system settings → Environment Variables**

  edit system variable **Path** and add at the end of it:

  **;C:\Program Files\Java\jdk1.6.0_23\bin;C:\JOGL2\lib**

  create new system variable **CLASSPATH** with:

  **.;C:\JOGL2\jar\jogl.all.jar**
  **;C:\JOGL2\jar\nativewindow.all.jar**
  **;C:\JOGL2\jar\gluegen-rt.jar**
  **;C:\JOGL2\jar\newt.all.jar**

- *compile the java program:*

  **javac P.java**

  *or*

  **javac -O P.java** *(optimization)*

- *run the java program:*

  **java -Dsun.java2d.noddraw=true P**

- *import statements:* (include them at the beginning of **P.java**)

```
import java.awt.*;    import java.awt.event.*;
import javax.swing.*;    import javax.swing.event.*;

import javax.media.opengl.*;    import javax.media.opengl.glu.*;
import javax.media.opengl.fixedfunc.*;    import javax.media.opengl.awt.*;
```

# *JOGL* connection to Java: *JOGL* event listener

*JOGL* is event driven:

- *drawing panel* defined as a subclass of **GLJPanel**

- *event listener* **GLEventListener** added to the drawing panel

  → 4 *event methods* defined inside **GLEventListener** :

  init : called only once when *JOGL* context is initialized
  = use it for one-time initialization tasks

  reshape : called at the beginning after **init** and each time the panel (i.e. the frame) is resized
  = use it to set the *viewport* and *projection transformations*

  display : called each time the drawing panel is (re)painted (especially after **p.repaint()** )
  = use it to:
  - set the *viewing* and *modeling transformations*
  - specify the 3D objects that must be displayed
  - flush the drawing buffer

  dispose : called at the end, just before *JOGL* context is destroyed
  = seldom used in practice → we will define it with an empty body **{}**

- *common argument* for these event methods:

  **GLAutoDrawable drawable** *(similar to* Graphics g *of* paintComponent *)*

  → **GL2 gl = drawable.getGL().getGL2();** *(similar to* Graphics2D g2 = (Graphics2D)g; *)*

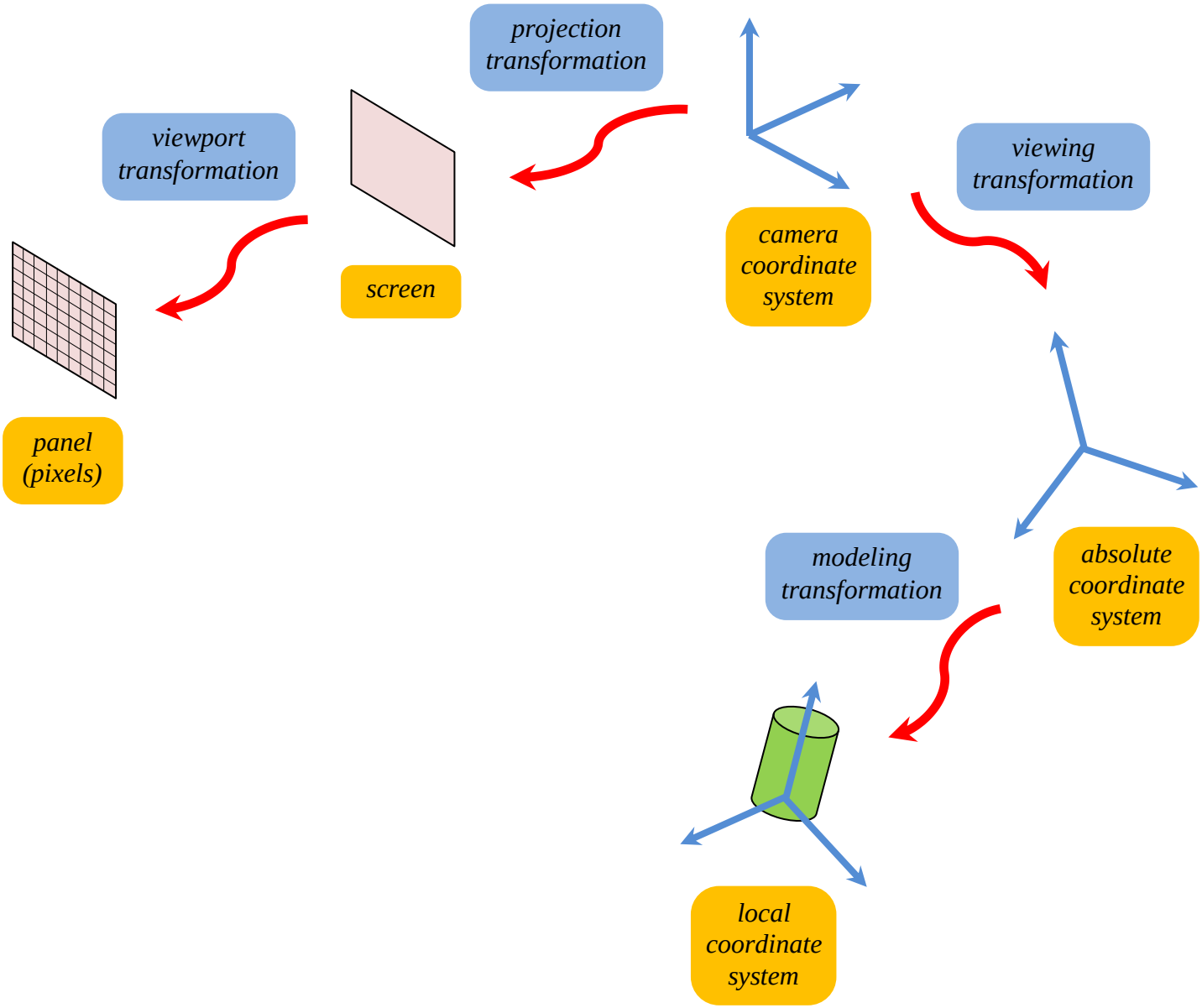
  we will apply most instructions over **gl** , the others will be applied over **glu**
  *(similar to 2D instructions applied over* g *or* g2 *)*

- *extra arguments* for **reshape** : **int x , int y , int w , int h** (**w**×**h** updated panel size)

template:

```
class Gui
{
   . . . . . .

   class DrawingPanel extends GLJPanel
   {
      GLU glu;    GLUquadric quad;          // to use the GLU library

      DrawingPanel()
      {  super(new GLCapabilities(GLProfile.getDefault()));

         this.addGLEventListener(new GLEventListener()
         {
            public void init(GLAutoDrawable drawable)
            {  GL2 gl = drawable.getGL().getGL2();
               gl.glClearColor(1.0f , 1.0f , 1.0f , 0.0f);   // define the clearing color
               . . . . . }

            public void reshape(GLAutoDrawable drawable , int x , int y , int w , int h)
            {  GL2 gl = drawable.getGL().getGL2();
               . . . . . }

            public void display(GLAutoDrawable drawable)
            {  GL2 gl = drawable.getGL().getGL2();
               gl.glClear(GL.GL_COLOR_BUFFER_BIT);   // clear the panel
               . . . . . .
               gl.glFlush(); }   // flush the drawing buffer

            public void dispose(GLAutoDrawable drawable)
            {}
         } );

      }
   }


   Gui()
   {
      p = new DrawingPanel();
      . . . . . .
   }
}
```

# Geometric transformations




- inside **reshape**:
  - *viewport transformation*
  - *projection transformation*
- inside **display**:
  - *viewing transformation*
  - *modeling transformation*

- *projection matrix*: *projection transformation*
- *modelview matrix*: *viewing transformation - modeling transformation*

viewport transformation:

```
gl.glViewport(0 , 0 , w , h);
```

*inside* **reshape**,
*specify the width* **w** *and height* **h** *of the panel (in pixels)*

projection transformation:

```
glu = new GLU();
```

*inside* **init**,
*create glu object*

*inside* **reshape**,
*the current matrix becomes the projection matrix*

```
gl.glMatrixMode(GLMatrixFunc.GL_PROJECTION);
gl.glLoadIdentity();
glu.gluPerspective(60.0f , (float) w / h , 1.0f , 10000.0f);
```

*the current matrix is set to identity matrix*

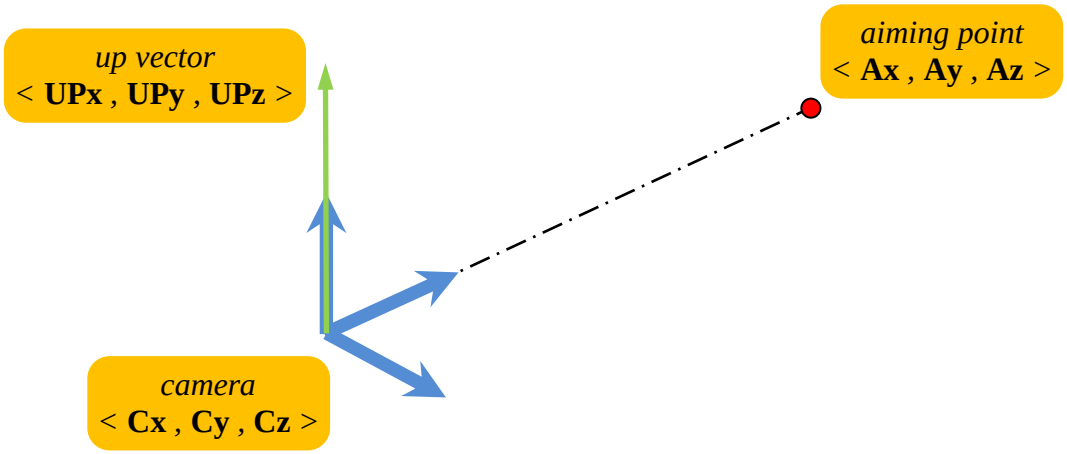
*the current matrix is multiplied by matrix expressing perspective:*
*field of view = angle in degrees= 60°*
*ratio width / height of the panel*
*minimal distance, maximal distance*
→ *objects not within these distances will be clipped*

all arguments of **gluPerspective** must be of type **float**

viewing transformation:

geometric transformation from *camera coordinate system* to *absolute coordinate system*

- *camera oriented toward aiming point:*




*inside* **display**, *the current matrix becomes the modelview matrix*

```
gl.glMatrixMode(GLMatrixFunc.GL_MODELVIEW);
gl.glLoadIdentity();
glu.gluLookAt(Cx , Cy , Cz , Ax , Ay , Az , UPx , UPy , UPz);
```

*the current matrix is set to identity matrix*

*the current matrix is multiplied by matrix expressing position and orientation of camera*

all arguments of **gluLookAt** must be of type **float**

- *camera oriented according to roll, pitch, yaw angles:*

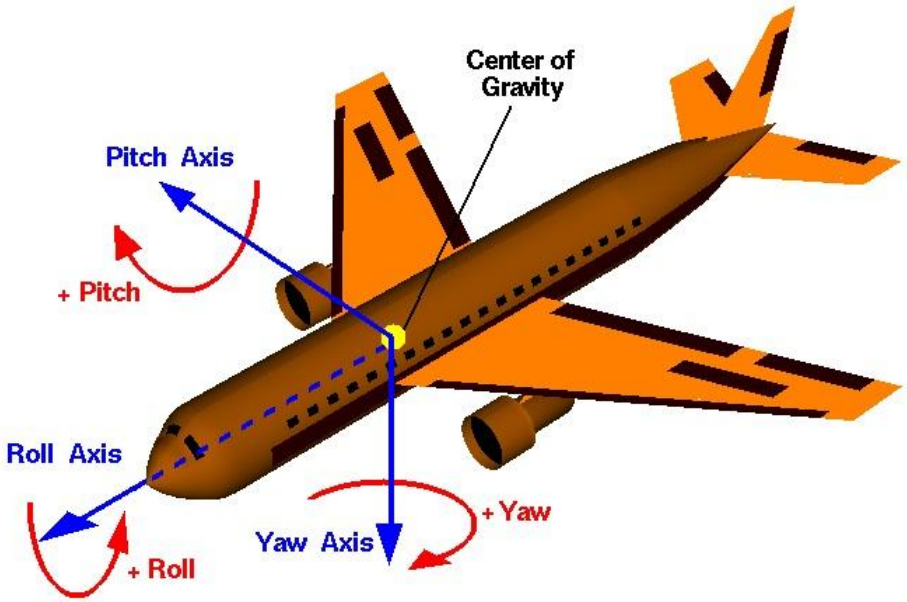



*Figure courtesy of NASA, Wikipedia*

*inside* **display**, *the current matrix becomes the modelview matrix*

```
gl.glMatrixMode(GLMatrixFunc.GL_MODELVIEW);
gl.glLoadIdentity();
gl.glRotatef(Croll , 0.0f , 0.0f , 1.0f);
gl.glRotatef(Cpitch , 1.0f , 0.0f , 0.0f);
gl.glRotatef(Cyaw , 0.0f , 1.0f , 0.0f);
gl.glTranslatef( - Cx,  - Cy,  - Cz);
```

*the current matrix is set to identity matrix*

*the current matrix is multiplied by matrix of rotation around local z axis*

*the current matrix is multiplied by matrix of rotation around local x axis*

*the current matrix is multiplied by matrix of rotation around local y axis*

*the current matrix is multiplied by matrix of translation from camera to origin of absolute coordinate system*

all arguments of **glRotatef** and **glTranslatef** must be of type **float**

modeling transformation:

geometric transformation from *absolute coordinate system* to each *local coordinate system* where some 3D objects are defined

→ sequence of *translations*, *rotations* and *scalings*

*inside* **display**,
*the current matrix becomes*
*the modelview matrix*

**gl.glMatrixMode(GL_MODELVIEW);**

***first, specify the viewing transformation***

***then, succession of translations, rotations and scalings:***

**gl.glTranslatef(delta_x , delta_y , delta_z);**

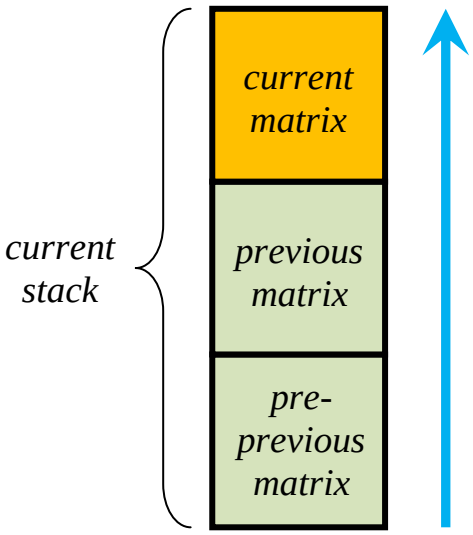
**gl.glRotatef(angle in degrees , vx , vy , vz);**

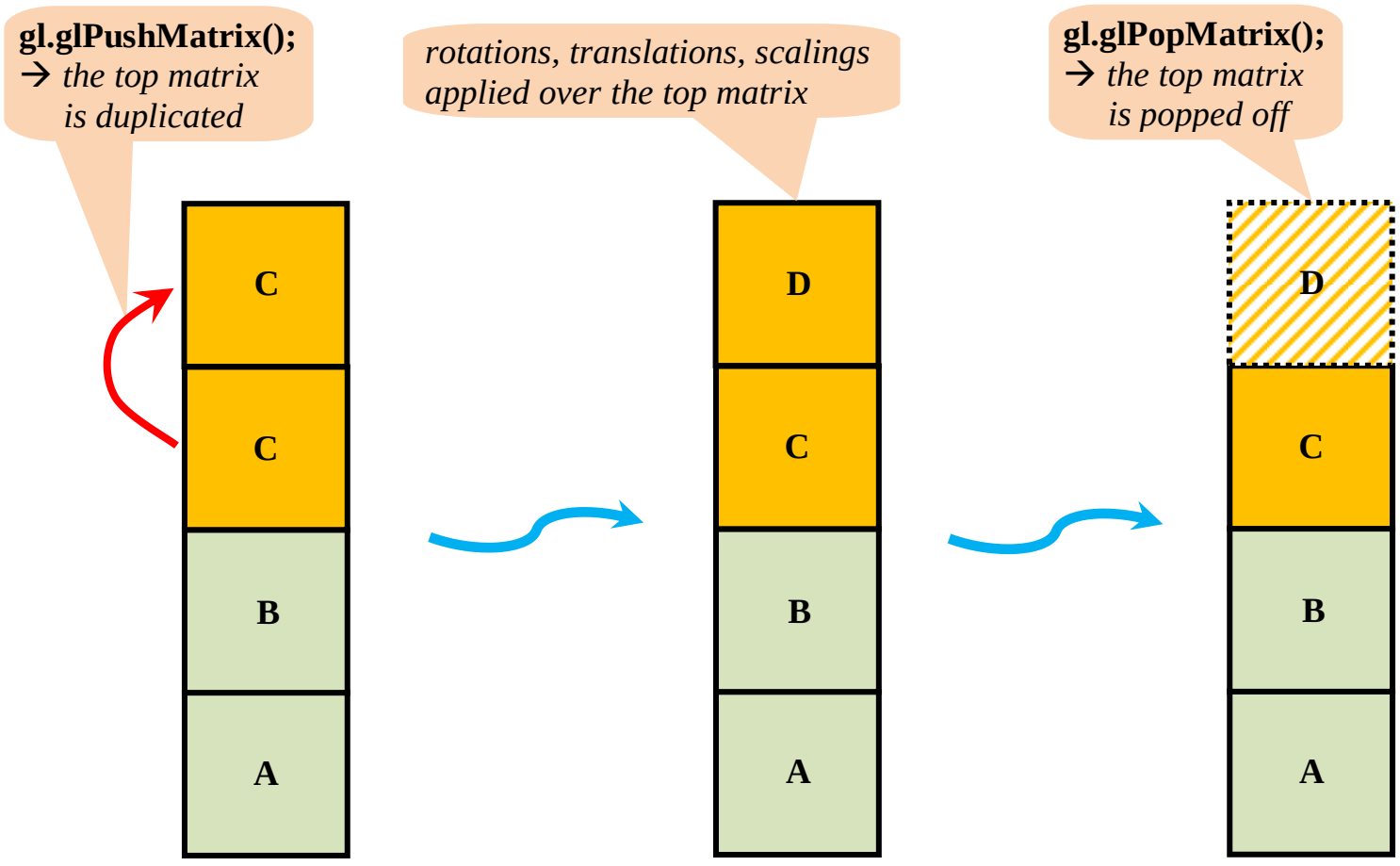
**gl.glScalef(scale_x , scale_y , scale_z);**

rotation around vector following the "*corkscrew rule*"

modelview matrix stack ; projection matrix stack:

*current matrix* = top matrix of the *current matrix stack* { *modelview matrix stack* or *projection matrix stack* (chosen with **glMatrixMode**)



# Geometric modeling with GLU library

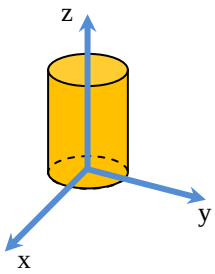
- *inside* **init** *:*

```
glu = new GLU();

quad = glu.gluNewQuadric();
```

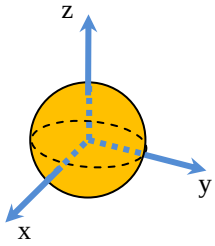
some ready-made objects in GLU library:

- *inside* **display** *:*

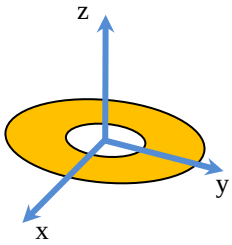
```
glu.gluCylinder(quad , base_radius , top_radius , height
                                , n_slices , n_loops);
```



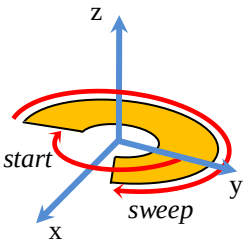

```
glu.gluSphere(quad , radius
                                , n_slices , n_loops);
```




```
glu.gluDisk(quad , inner_radius , outer_radius
                                , n_slices , n_loops);
```




```
glu.gluPartialDisk(quad , inner , outer
                                , n_slices , n_loops
                                , start_angle , sweep_angle);
```




**n_slices**, **n_loops** define the { wireframe / the grid of facets } that represents the surface

**n_slices**, **n_loops** are of type **int** , other arguments (except **quad**) are of type **float**

drawing style:

**glu.gluQuadricDrawStyle(quad , *style*);**

***style*** =
- **GLU.GLU_POINT**
- *or*
- **GLU.GLU_LINE** (*wireframe rendering*)
- *or*
- **GLU.GLU_SILHOUETTE**
- *or*
- **GLU.GLU_FILL** (*surface rendering*)

shading model:

**glu.gluQuadricNormals(quad , *normal*);**

***normal*** =
- **GLU.GLU_NONE** (default)
- *or*
- **GLU.GLU_FLAT** (color constant over each facet) — *only for surface rendering*
- *or*
- **GLU.GLU_SMOOTH** (color interpolated over each facet) — *only for surface rendering*

color:

**glColor4f(red , green , blue , alpha);**

*0.0f → transparent*
*1.0f → opaque*

*4* **float** *arguments from 0.0f to 1.0f*

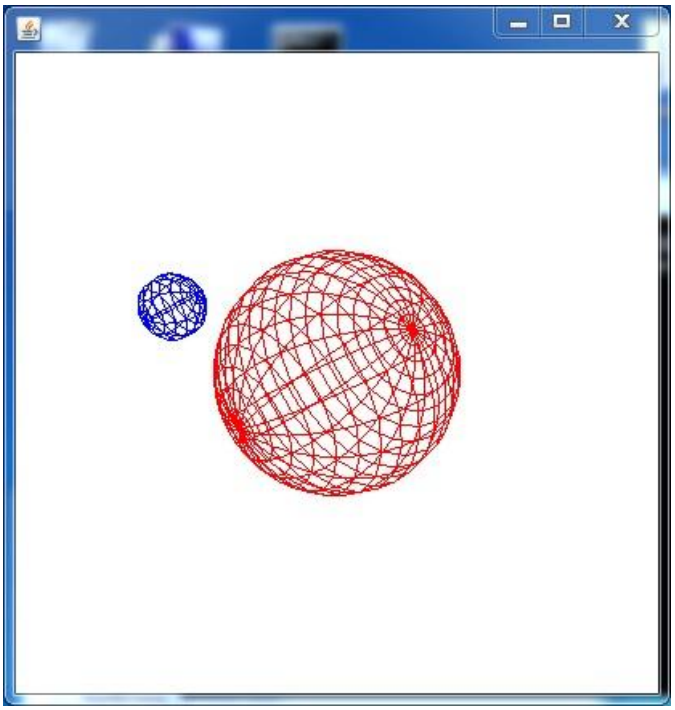
template:

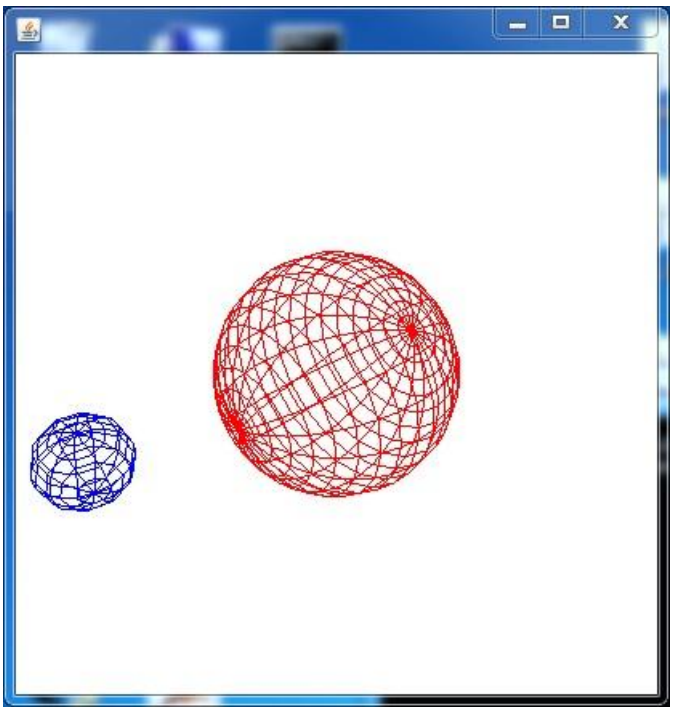
```
class DrawingPanel extends GLJPanel
{
   GLU glu;    GLUquadric quad;
   . . . . . .

        public void init(GLAutoDrawable drawable)
        {
           . . . . . .

           glu = new GLU();
           quad = glu.gluNewQuadric();
           glu.gluQuadricDrawStyle(quad , GLU.GLU_LINE);
           glu.gluQadricNormals(quad , GLU.GLU_SMOOTH);
           . . . . . .

        public void display(GLAutoDrawable drawable)
        {
           . . . . . .
           glColor4f(1.0f , 0.0f , 0.0f , 1.0f);
           glu.gluSphere(quad , 10.0f , 20 , 20);

           glColor4f(0.0f , 0.0f , 1.0f , 1.0f);
           glu.gluCylinder(quad , 4.0f , 2.0f , 20.0f , 20 , 20);
           . . . . . .
```

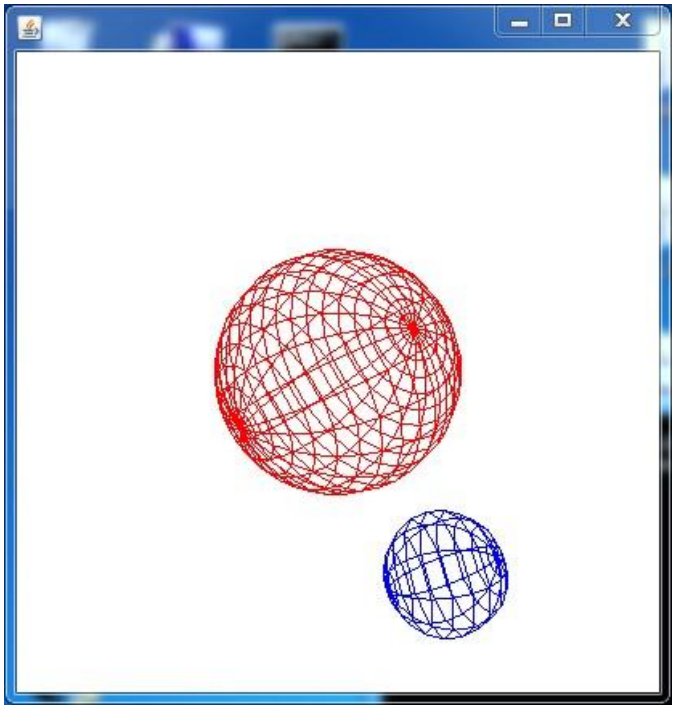
*only for surface rendering*

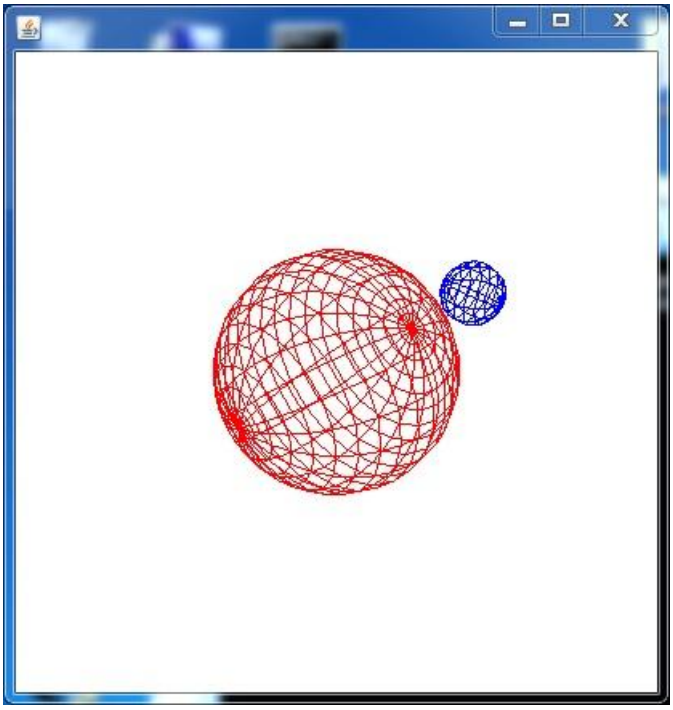
# Examples with GLU objects and wireframe rendering

*example 1: planet and satellite - animation loop - wireframe rendering*









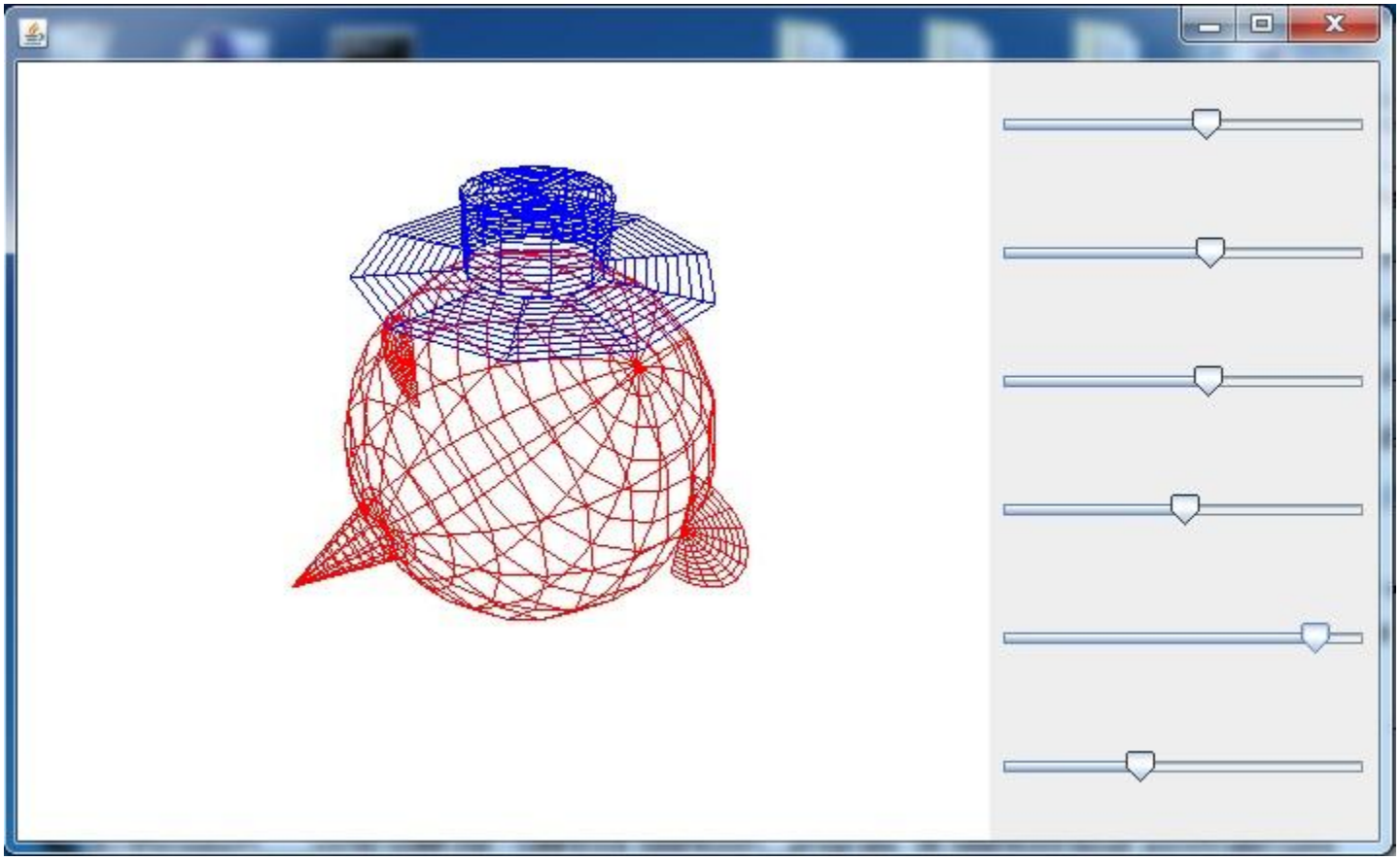
*example 1: (continued)*

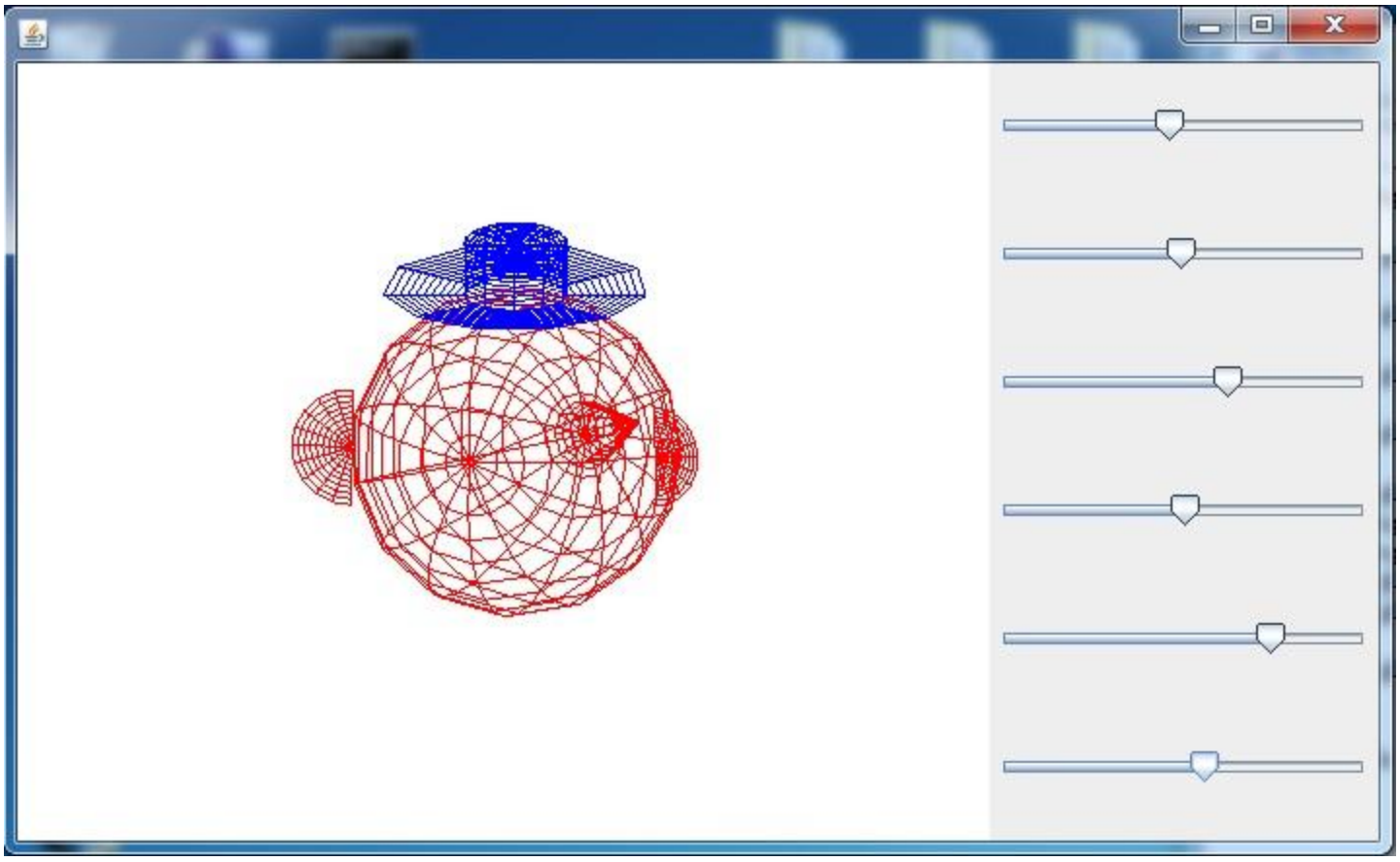
```
import java.awt.*;   import java.awt.event.*;
import javax.swing.*;   import javax.swing.event.*;
import javax.media.opengl.*;   import javax.media.opengl.glu.*;
import javax.media.opengl.fixedfunc.*;   import javax.media.opengl.awt.*;


public class P
{  public static void main(String[] arg)
   {  Gui gui = new Gui();

      long dt = 100;  // 0.1s
      while (true)
      {
         long t_start = System.currentTimeMillis();

         gui.alpha += 2.0;   gui.f.repaint();

         long dt_real = System.currentTimeMillis() - t_start;
         if (dt_real < dt)  try {Thread.sleep(dt  - dt_real);} catch(InterruptedException e){}
                  else  System.out.println("PC too slow; please increase dt");
      }
   }
}


class Gui
{
   JFrame f;   DrawingPanel p;
   float dx = 30 , alpha = 0;

   class DrawingPanel extends GLJPanel
   {
      GLU glu;   GLUquadric quad;

      DrawingPanel()
      {
         super(new GLCapabilities(GLProfile.getDefault()));

         this.addGLEventListener(new GLEventListener()
         {
            public void init(GLAutoDrawable drawable)   //***  INIT
            {
               GL2 gl = drawable.getGL().getGL2();

               glu = new GLU();
               quad = glu.gluNewQuadric();   glu.gluQuadricDrawStyle(quad , GLU.GLU_LINE);

               gl.glClearColor(1.0f , 1.0f , 1.0f , 0.0f);
            }
```

*example 1: (continued)*

```
        public void reshape(GLAutoDrawable drawable , int x , int y , int w , int h)    //***  RESHAPE
        {
           GL2 gl = drawable.getGL().getGL2();

           gl.glViewport(0 , 0 , w , h);

           gl.glMatrixMode(GLMatrixFunc.GL_PROJECTION);
           gl.glLoadIdentity();    glu.gluPerspective(60.0f , (float) w / h , 1.0f , 10000.0f);
        }

        public void display(GLAutoDrawable drawable)    //***  DISPLAY
        {
           GL2 gl = drawable.getGL().getGL2();

           gl.glClear(GL.GL_COLOR_BUFFER_BIT);
           gl.glMatrixMode(GLMatrixFunc.GL_MODELVIEW);
           gl.glLoadIdentity();    glu.gluLookAt(40.0f , 40.0f , 40.0f , 0.0f , 0.0f , 0.0f , 0.0f , 1.0f , 0.0f);

           gl.glColor4f(1.0f, 0.0f, 0.0f , 1.0f);    glu.gluSphere(quad , 15.0f , 20 , 20);

           gl.glRotatef(alpha , 0.0f , 1.0f  , 0.0f);    gl.glTranslatef(dx , 0f , 0f);

           gl.glColor4f(0.0f, 0.0f, 1.0f , 1.0f);    glu.gluSphere(quad , 5.0f , 10 , 10);

           gl.glFlush();
        }

        public void dispose(GLAutoDrawable drawable)    //***  DISPOSE
        {}
      } );

    }
  }


  Gui()
  {
    f = new JFrame();    f.setFocusable(true);    f.setVisible(true);
    p = new DrawingPanel();    f.getContentPane().add(p , BorderLayout.CENTER);

    f.setSize(new Dimension(400 + 16 , 400 + 38));
  }
}
```

*example 2: head with hat - roll, pitch, yaw camera orientation*





*example 2: (continued)*

```
class Gui
{
   JFrame f;    DrawingPanel p;

   float Cx =80, Cy = 80, Cz = 80, Croll = 0 , Cpitch = 45 , Cyaw  = -45;
   JPanel psC;    JSlider sCx , sCy , sCz , sCroll , sCpitch , sCyaw;

   class DrawingPanel extends GLJPanel
   {
      GLU glu;    GLUquadric quad;

      DrawingPanel()
      {
         super(new GLCapabilities(GLProfile.getDefault()));

         this.addGLEventListener(new GLEventListener()
         {
            public void init(GLAutoDrawable drawable)    //***  INIT
            {
               GL2 gl = drawable.getGL().getGL2();

               glu = new GLU();    quad = glu.gluNewQuadric();
               glu.gluQuadricDrawStyle(quad , GLU.GLU_LINE);

               gl.glClearColor(1.0f , 1.0f , 1.0f , 0.0f);
            }

            public void reshape(GLAutoDrawable drawable , int x , int y , int w , int h)    //***  RESHAPE
            {
               GL2 gl = drawable.getGL().getGL2();

               gl.glViewport(0 , 0 , w , h);

               gl.glMatrixMode(GLMatrixFunc.GL_PROJECTION);
               gl.glLoadIdentity();    glu.gluPerspective(60.0f , (float) w / h , 1.0f , 10000.0f);
            }
```

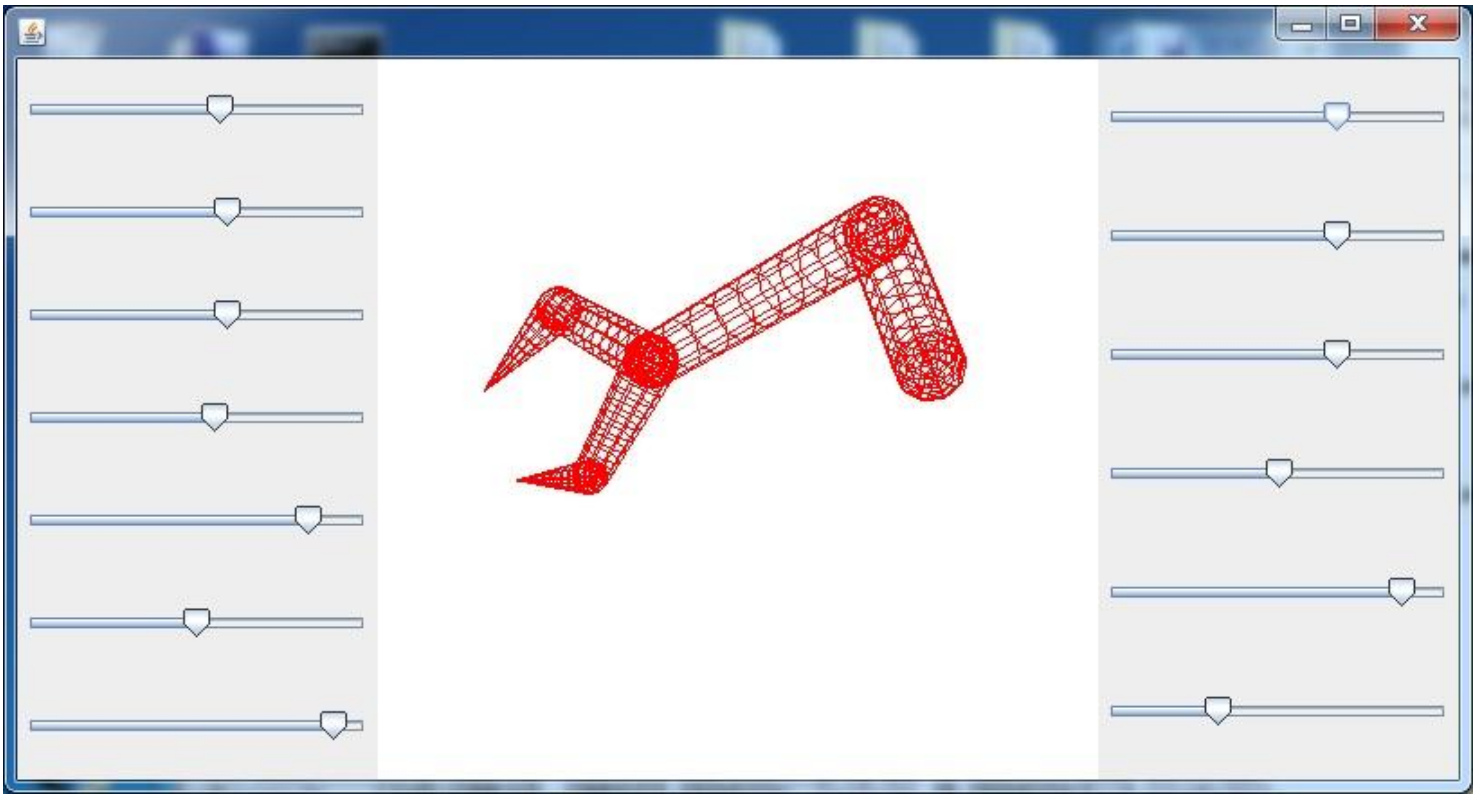
*example 2: (continued)*

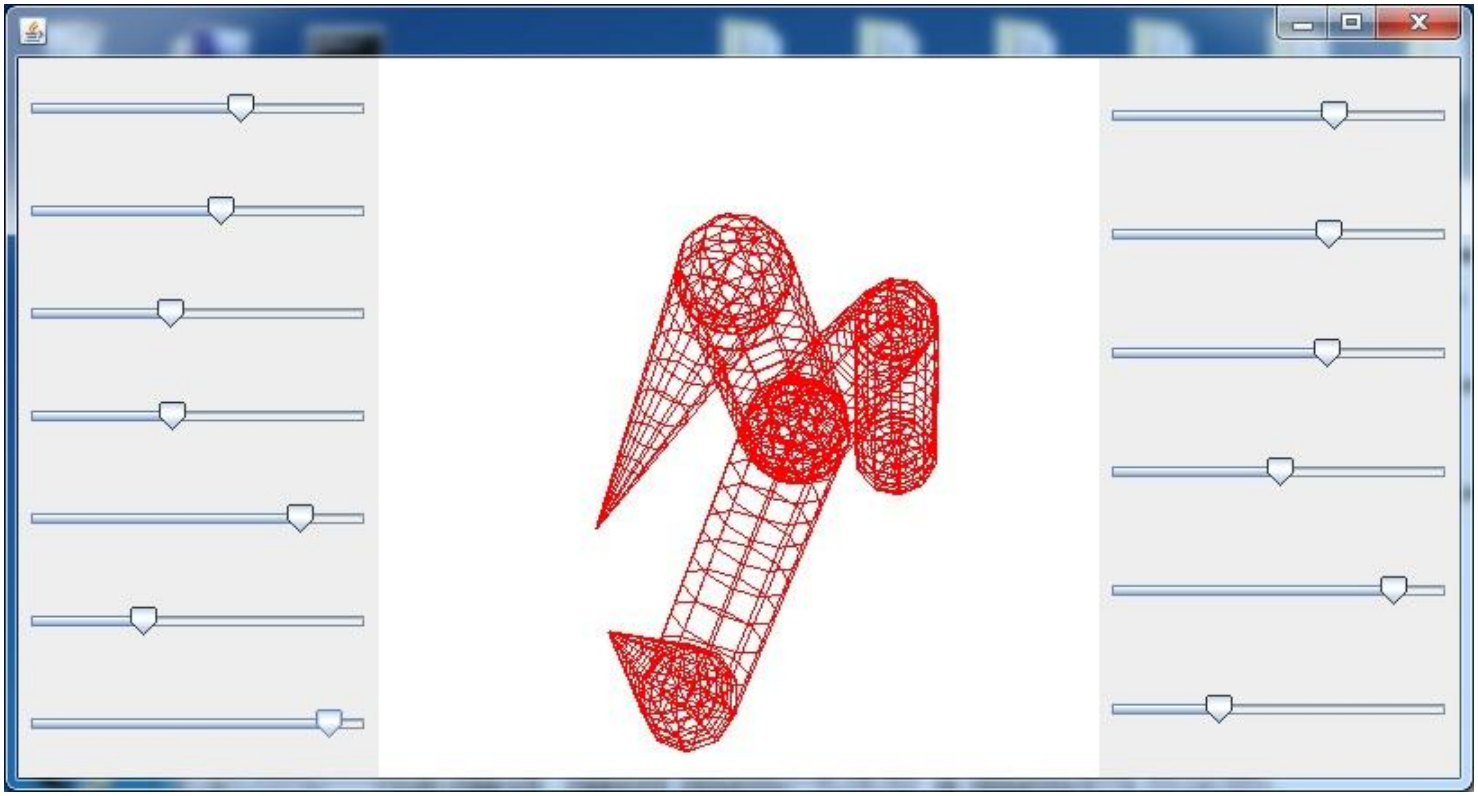
```
        public void display(GLAutoDrawable drawable)    //***  DISPLAY
        {
           GL2 gl = drawable.getGL().getGL2();

           gl.glClear(GL.GL_COLOR_BUFFER_BIT);

           gl.glMatrixMode(GLMatrixFunc.GL_MODELVIEW);       //// CAMERA
           gl.glLoadIdentity();
           gl.glRotatef(Croll , 0.0f , 0.0f , 1.0f);
           gl.glRotatef(Cpitch , 1.0f , 0.0f , 0.0f);
           gl.glRotatef(Cyaw , 0.0f , 1.0f , 0.0f);
           gl.glTranslatef( - Cx,  - Cy,  - Cz);


           gl.glColor4f(1.0f, 0.0f, 0.0f , 1.0f);       //// HEAD

           glu.gluSphere(quad , 30.0f , 15 , 15);

           gl.glPushMatrix();    gl.glTranslatef(0.0f , 0.0f , 30.0f);
           glu.gluCylinder(quad , 5.0f , 0.0f , 15.0f , 10 , 10);    gl.glPopMatrix();

           gl.glPushMatrix();    gl.glTranslatef(30.0f , 0.0f , 0.0f);
           glu.gluPartialDisk(quad , 0.0f , 10.0f , 10 , 10 , 0.0f , +180.0f);    gl.glPopMatrix();

           gl.glPushMatrix();    gl.glTranslatef(-30.0f , 0.0f , 0.0f);
           glu.gluPartialDisk(quad , 0.0f , 10.0f , 10 , 10 , 0.0f , -180.0f);    gl.glPopMatrix();


           gl.glColor4f(0.0f, 0.0f, 1.0f , 1.0f);       //// HAT

           gl.glPushMatrix();    gl.glTranslatef(0.0f , 30.0f , 0.0f);    gl.glRotatef(-90.0f , 1.0f , 0.0f , 0.0f);
           glu.gluDisk(quad , 10.0f , 25.0f , 10 , 10);
           glu.gluCylinder(quad , 10.0f , 10.0f , 10.0f , 10 , 10);

           gl.glPushMatrix();    gl.glTranslatef(0.0f , 0.0f , 10.0f);
           glu.gluDisk(quad , 0.0f , 10.0f , 10 , 10);    gl.glPopMatrix();

           gl.glPopMatrix();

           gl.glFlush();
        }

        public void dispose(GLAutoDrawable drawable)    //***  DISPOSE
        {}
      } );

   }
}
```

*example 2: (continued)*

```
   Gui()
   {
      f = new JFrame();    f.setFocusable(true);    f.setVisible(true);
      p = new DrawingPanel();    f.getContentPane().add(p , BorderLayout.CENTER);

      ////---------------------- CAMERA
      psC = new JPanel();    psC.setLayout(new GridLayout(0 , 1));
      f.getContentPane().add(psC , BorderLayout.EAST);

      sCx = new JSlider(JSlider.HORIZONTAL , -200 , +200 , 80);    psC.add(sCx);
      sCy = new JSlider(JSlider.HORIZONTAL , -200 , +200 , 80);    psC.add(sCy);
      sCz = new JSlider(JSlider.HORIZONTAL , -200 , +200 , 80);    psC.add(sCz);

      sCroll  = new JSlider(JSlider.HORIZONTAL , -180 , +180 , 0);    psC.add(sCroll);
      sCpitch = new JSlider(JSlider.HORIZONTAL , -270 , +90 , 45);    psC.add(sCpitch);
      sCyaw   = new JSlider(JSlider.HORIZONTAL , -180 , +180 , -45);    psC.add(sCyaw);

      sCx.addChangeListener(    . . . . . .    {  Cx = sCx.getValue();    f.repaint();  } } );
      sCy.addChangeListener(    . . . . . .    {  Cy = sCy.getValue();    f.repaint();  } } );
      sCz.addChangeListener(    . . . . . .    {  Cz = sCz.getValue();    f.repaint();  } } );

      sCroll.addChangeListener(    . . . . . .    {  Croll  = sCroll.getValue();     f.repaint();  } } );
      sCpitch.addChangeListener(   . . . . . .    {  Cpitch = sCpitch.getValue();    f.repaint();  } } );
      sCyaw.addChangeListener(    . . . . . .    {  Cyaw   = sCyaw.getValue();     f.repaint();  } } );

      f.setSize(new Dimension(800 + 16 , 400 + 38));
   }
}
```

*example 3: robotic arm - roll, pitch, yaw camera orientation*





*example 3: (continued)*

```
class Gui
{
   JFrame f;    DrawingPanel p;

   float Cx =260, Cy = 260, Cz = 260, Croll = 0 , Cpitch = 45 , Cyaw  = -45;
   JPanel psC;    JSlider sCx , sCy , sCz , sCroll , sCpitch , sCyaw;

   float a_ = 0 , a0 = 0 , a1 = 0 , a2 = 0 , a3 = 0 , a4 = 0 , a5 = 0;
   float b0 = 110 , b1 = 90 , b2 = 30 , b3 = 20 , b4 = 30 , b5 = 20;
   float r0 =   16 ,  r1 = 10 ,  r2 =   6 , r3 =   4 ,  r4 =   6 ,  r5 =   4;
   JPanel ps;    JSlider sa_ , sa0 , sa1 , sa2 , sa3 , sa4 , sa5;

   class DrawingPanel extends GLJPanel
   {
      GLU glu;    GLUquadric quad;

      DrawingPanel()
      {
         super(new GLCapabilities(GLProfile.getDefault()));

         this.addGLEventListener(new GLEventListener()
         {
            public void init(GLAutoDrawable drawable)    //***  INIT
            {
               GL2 gl = drawable.getGL().getGL2();

               glu = new GLU();    quad = glu.gluNewQuadric();
               glu.gluQuadricDrawStyle(quad , GLU.GLU_LINE);

               gl.glClearColor(1.0f , 1.0f , 1.0f , 0.0f);
            }

            public void reshape(GLAutoDrawable drawable , int x , int y , int w , int h)    //***  RESHAPE
            {
               GL2 gl = drawable.getGL().getGL2();

               gl.glViewport(0 , 0 , w , h);

               gl.glMatrixMode(GLMatrixFunc.GL_PROJECTION);
               gl.glLoadIdentity();    glu.gluPerspective(60.0f , (float) w / h , 1.0f , 10000.0f);
            }
```

*example 3: (continued)*

```
        public void display(GLAutoDrawable drawable)    //***  DISPLAY
        {
           GL2 gl = drawable.getGL().getGL2();

           gl.glClear(GL.GL_COLOR_BUFFER_BIT);

           gl.glMatrixMode(GLMatrixFunc.GL_MODELVIEW);       //// CAMERA
           gl.glLoadIdentity();
           gl.glRotatef(Croll , 0.0f , 0.0f , 1.0f);
           gl.glRotatef(Cpitch , 1.0f , 0.0f , 0.0f);
           gl.glRotatef(Cyaw , 0.0f , 1.0f , 0.0f);
           gl.glTranslatef( - Cx,  - Cy,  - Cz);

           gl.glColor4f(1.0f, 0.0f, 0.0f , 1.0f);                    //// ROBOTIC ARM

           gl.glRotatef(a_ , 0.0f , 1.0f , 0.0f);    gl.glRotatef( - a0 , 1.0f , 0.0f  , 0.0f);
           glu.gluSphere(quad , r0 , 10 , 10);    glu.gluCylinder(quad , r0 , r1 , b0 , 10 , 10);

           gl.glTranslatef(0f , 0f , b0);    gl.glRotatef(a1 , 1.0f , 0.0f  , 0.0f);
           glu.gluSphere(quad , r1 , 10 , 10);    glu.gluCylinder(quad , r1 , r2 , b1 , 10 , 10);

           gl.glTranslatef(0f , 0f , b1);

           gl.glPushMatrix();

           gl.glRotatef( - a2 , 1.0f , 0.0f  , 0.0f);
           glu.gluSphere(quad , r2 , 10 , 10);    glu.gluCylinder(quad , r2 , r3 , b2 , 10 , 10);

           gl.glTranslatef(0f , 0f , b2);    gl.glRotatef(a3 , 1.0f , 0.0f  , 0.0f);
           glu.gluSphere(quad , r3 , 10 , 10);    glu.gluCylinder(quad , r3 , 0.0f , b3 , 10 , 10);

           gl.glPopMatrix();

           gl.glRotatef(a4 , 1.0f , 0.0f  , 0.0f);
           glu.gluSphere(quad , r4 , 10 , 10);    glu.gluCylinder(quad , r4 , r5 , b4 , 10 , 10);

           gl.glTranslatef(0f , 0f , b4);    gl.glRotatef( - a5 , 1.0f , 0.0f  , 0.0f);
           glu.gluSphere(quad , r5 , 10 , 10);    glu.gluCylinder(quad , r5 , 0.0f , b5 , 10 , 10);

           gl.glFlush();
        }

        public void dispose(GLAutoDrawable drawable)    //***  DISPOSE
        {}
      } );

   }
}
```

*example 3: (continued)*

```
   Gui()
   {
      f = new JFrame();   f.setFocusable(true);   f.setVisible(true);
      p = new DrawingPanel();   f.getContentPane().add(p , BorderLayout.CENTER);

      ////---------------------- CAMERA
      psC = new JPanel();   psC.setLayout(new GridLayout(0 , 1));
      f.getContentPane().add(psC , BorderLayout.EAST);

      sCx = new JSlider(JSlider.HORIZONTAL , -500 , +500 , 260);   psC.add(sCx);
      sCy = new JSlider(JSlider.HORIZONTAL , -500 , +500 , 260);   psC.add(sCy);
      sCz = new JSlider(JSlider.HORIZONTAL , -500 , +500 , 260);   psC.add(sCz);

      sCroll  = new JSlider(JSlider.HORIZONTAL , -180 , +180 , 0);   psC.add(sCroll);
      sCpitch = new JSlider(JSlider.HORIZONTAL , -270 , +90 , 45);   psC.add(sCpitch);
      sCyaw   = new JSlider(JSlider.HORIZONTAL , -180 , +180 , -45);   psC.add(sCyaw);

      sCx.addChangeListener(   . . . . . { Cx = sCx.getValue();   f.repaint(); } } );
      sCy.addChangeListener(   . . . . . { Cy = sCy.getValue();   f.repaint(); } } );
      sCz.addChangeListener(   . . . . . { Cz = sCz.getValue();   f.repaint(); } } );

      sCroll.addChangeListener(   . . . . . { Croll  = sCroll.getValue();   f.repaint(); } } );
      sCpitch.addChangeListener(   . . . . . { Cpitch = sCpitch.getValue();   f.repaint(); } } );
      sCyaw.addChangeListener(   . . . . . { Cyaw  = sCyaw.getValue();   f.repaint(); } } );

      ////---------------------- ROBOTIC ARM
      ps = new JPanel();   ps.setLayout(new GridLayout(0 , 1));
      f.getContentPane().add(ps , BorderLayout.WEST);

      sa_ = new JSlider(JSlider.HORIZONTAL , -180, +180 , 0);   ps.add(sa_);
      sa0 = new JSlider(JSlider.HORIZONTAL ,   0 , +90 , 0);   ps.add(sa0);
      sa1 = new JSlider(JSlider.HORIZONTAL ,   0 , +90 , 0);   ps.add(sa1);
      sa2 = new JSlider(JSlider.HORIZONTAL ,   0 , +90 , 0);   ps.add(sa2);
      sa3 = new JSlider(JSlider.HORIZONTAL ,   0 , +90 , 0);   ps.add(sa3);
      sa4 = new JSlider(JSlider.HORIZONTAL ,   0 , +90 , 0);   ps.add(sa4);
      sa5 = new JSlider(JSlider.HORIZONTAL ,   0 , +90 , 0);   ps.add(sa5);

      sa_.addChangeListener(   . . . . . { a_ = sa_.getValue();   f.repaint(); } } );
      sa0.addChangeListener(   . . . . . { a0 = sa0.getValue();   f.repaint(); } } );
      sa1.addChangeListener(   . . . . . { a1 = sa1.getValue();   f.repaint(); } } );
      sa2.addChangeListener(   . . . . . { a2 = sa2.getValue();   f.repaint(); } } );
      sa3.addChangeListener(   . . . . . { a3 = sa3.getValue();   f.repaint(); } } );
      sa4.addChangeListener(   . . . . . { a4 = sa4.getValue();   f.repaint(); } } );
      sa5.addChangeListener(   . . . . . { a5 = sa5.getValue();   f.repaint(); } } );

      f.setSize(new Dimension(800 + 16 , 400 + 38));
   }
}
```

# Geometric modeling with lines and polygons

we may model a complex surface as an assemblage of *graphic primitives*

*graphic primitive* = basic surface → we will use polygons: triangles, quadrilaterals, ...

*OpenGL* polygons must be flat and convex → use preferently triangles...

definition of some *graphic primitives:*

- *defining a vertex (point):*

```
gl.glVertex3f(x , y , z);
```

*3* **float** *arguments*

*or*

```
float[] v = { x , y , z };
gl.glVertex3fv(v);
```

*argument: vector of 3* **float**

- *defining a graphic primitive with a succession of vertices:*

```
gl.glBegin( graphic primitive );
   gl.glVertex3fv(v0);
   gl.glVertex3fv(v1);
   . . . . . .
   gl.glVertex3fv(vk);
gl.glEnd();
```

| *graphic primitives* described by a sequence of vertices | |
|---|---|
| **GL2.GL_POINTS** | individual points |
| **GL2.GL_LINES** | pairs of vertices interpreted as individual line segments |
| **GL2.GL_LINE_STRIP** | series of connected line segments |
| **GL2.GL_LINE_LOOP** | series of connected line segments with a segment added between last and first vertices |
| **GL2.GL_TRIANGLES** | triples of vertices interpreted as triangles |
| **GL2.GL_TRIANGLE_STRIP** | linked strip of triangles |
| **GL2.GL_TRIANGLE_FAN** | linked fan of triangles |
| **GL2.GL_QUADS** | quadruples of vertices interpreted as four-sided polygons |
| **GL2.GL_QUAD_STRIP** | linked strip of quadrilaterals |
| **GL2.GL_POLYGON** | boundary of a simple, convex polygon |

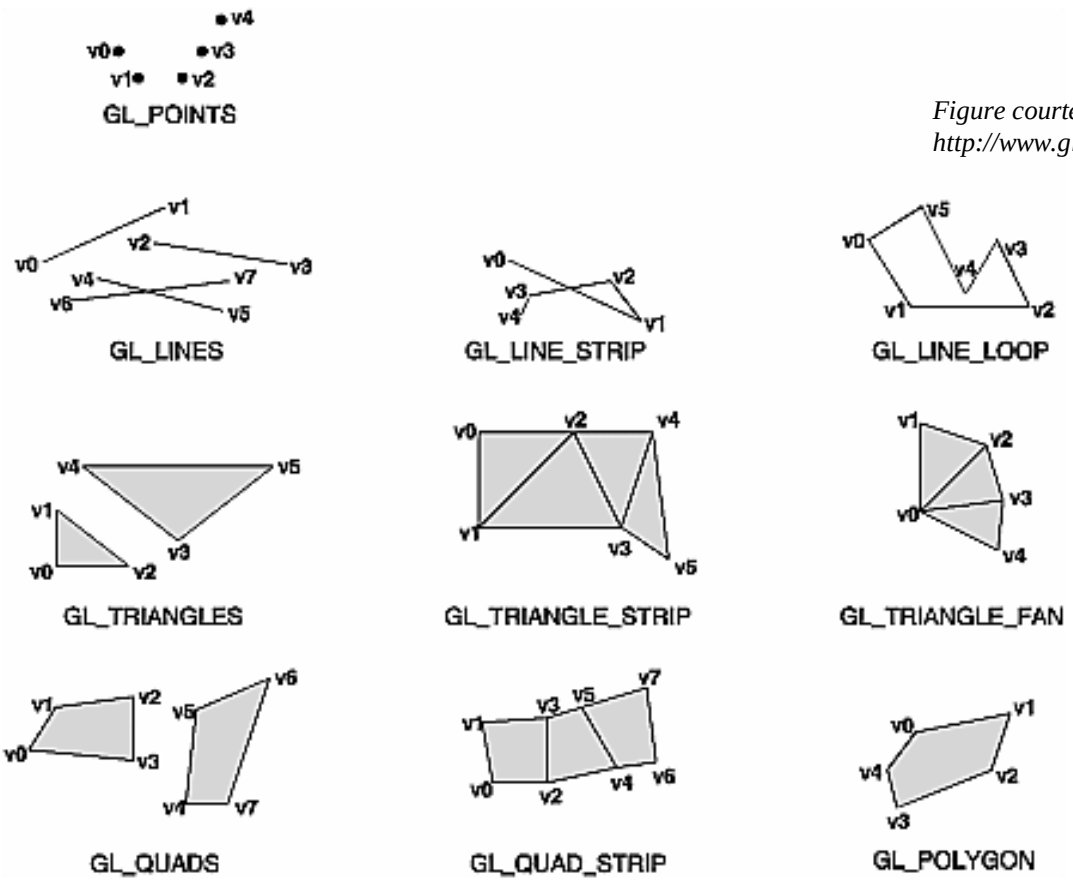



*Figure courtesy of http://www.glprogramming.com/red/*

normal vectors:

*normal vector* of a surface: { vector *orthogonal* to the surface, at each point of this surface *and* *unit* vector (length 1)

- *calculating the coordinates of a normal vector:*

assume **u** and **v** are vectors tangent to the surface at a certain point

{ *step 1:* calculate the vector product **u × v**

*step 2:* normalize the resulting vector

```
n[0] = u[1] * v[2] - u[2] * v[1];
n[1] = u[2] * v[0] - u[0] * v[2];
n[2] = u[0] * v[1] - u[1] * v[0];

float d = Math.sqrt(n[0] * n[0] + n[1] * n[1] + n[2] * n[2]);
if (d < 1.0e-10) { System.out.println("zero length vector"); System.exit(0); }
n[0] /= d; n[1] /= d; n[2] /= d;
```

- *setting the current normal vector:*

```
gl.glNormal3f(nx , ny , nz);
```

*3* **float** *arguments*

*or*

```
float[] n = { nx , ny , nz };

gl.glNormal3fv(n);
```

*argument: vector of 3* **float**

- *assigning normal vectors to the vertices of a graphic primitive:*

the normal vector can remain the same for all vertices
or can change for each vertex or group of vertices

```
gl.glBegin( graphic primitive );
   gl.glNormal3fv(n0);   gl.glVertex3fv(v0);
   gl.glNormal3fv(n1);   gl.glVertex3fv(v1);
   . . . . . .
   gl.glNormal3fv(nk);   gl.glVertex3fv(vk);
gl.glEnd();
```

*or*

```
gl.glBegin( graphic primitive );
   gl.glNormal3fv(n0);   gl.glVertex3fv(v0);   gl.glVertex3fv(v1);   gl.glVertex3fv(v2);
   gl.glNormal3fv(n1);   gl.glVertex3fv(v3);
   gl.glNormal3fv(n2);   gl.glVertex3fv(v4);   . . . . . .   gl.glVertex3fv(vk);
gl.glEnd();
```

shading model:

```
gl.glShadeModel( normal );
```

***normal*** = { *or* **GL2.GL_FLAT** (color constant over each facet) / **GL2.GL_SMOOTH** (color interpolated over each facet) }

# Hidden surface removal

OpenGL uses a depth buffer to discard the surfaces or parts of surfaces that are hidden from the camera by other surfaces

- *enable depth buffering inside* **init** *:*

```
gl.glEnable(GL.GL_DEPTH_TEST);
```

- *clear up the depth buffer at the beginning of* **display** *:*

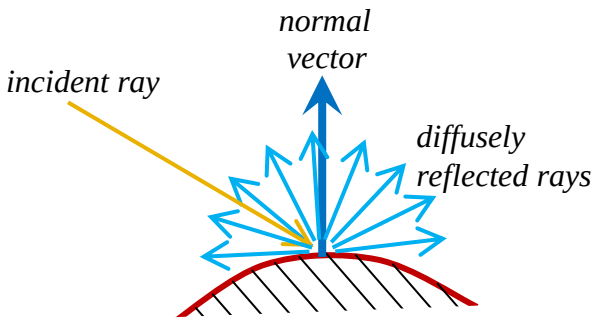
```
gl.glClear(GL.GL_DEPTH_BUFFER_BIT);
```

# Lighting and surface rendering

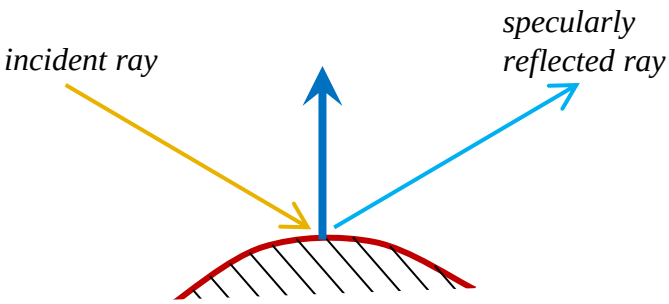
lighting model:

*ambient light*: uniform all over the scene

*diffuse light*: due to the diffuse reflections over the surface (*mat surface*)




*specular light*: due the specular reflections over the surface (*mirror*)




- *enable lighting inside* **init** *:*

```
glEnable(GL_LIGHTING);
```

- *specify global ambient light inside* **init** *:*

```
gl.glLightModelfv( GL2ES1.GL_LIGHT_MODEL_AMBIENT
                 , new float[] { 0.2f , 0.2f , 0.2f , 1.0f }
                 , 0);
```

material colors:

for a given surface, we must specify which proportion (0.0f to 1.0f)
of the red, green, blue of the incoming light is reflected diffusely and specularly

→ the coefficients for diffuse reflection actually determine the color of the surface

*note:* a green surface is seen as green because it reflects only the green incoming light

if not specified, the specular reflection is absent by default

- *inside* **display** , *before the geometric description of each surface:*

*for the front and back of the surface*

```
gl.glMaterialfv( GL.GL_FRONT_AND_BACK
               , GLLightingFunc.GL_AMBIENT_AND_DIFFUSE
               , new float[] { 1.0f , 1.0f , 0.0f , 1.0f }
               , 0);
```

*for the front and back of the surface*

```
gl.glMaterialfv( GL.GL_FRONT_AND_BACK
               , GLLightingFunc.GL_SPECULAR
               , new float[] { 1.0f , 1.0f , 1.0f , 1.0f }
               , 0);
```

*for the front and back of the surface*

```
gl.glMaterialfv( GL.GL_FRONT_AND_BACK
               , GLLightingFunc.GL_SHININESS
               , new float[] { 128.0f }
               , 0);
```

*specular exponent :*
*from 0.0f (as scattered as diffuse reflection)*
*to 128.0f (highly concentrated)*

light sources:

up to 8 light sources → index 0 to 7

- *inside* **init**:

  - enable a light source:

```
gl.glEnable(GLLightingFunc.GL_LIGHT0 );
```

*0 to 7*

- *inside* **display**:

  - define the position:

```
gl.glLightfv( GLLightingFunc.GL_LIGHT0
            , GLLightingFunc.GL_POSITION
            , new float[]{ x , y , z , w }
            , 0);
```

*0.0f → directional*
*1.0f → positional*

  - define the ambient, diffuse and specular lights:

```
gl.glLightfv( GLLightingFunc.GL_LIGHT0
            , GLLightingFunc.GL_AMBIENT
            , new float[]{ 0.1f , 0.1f , 0.1f , 1.0f }
            , 0);
```

```
gl.glLightfv( GLLightingFunc.GL_LIGHT0
            , GLLightingFunc.GL_DIFFUSE
            , new float[]{ 1.0f , 1.0f , 1.0f , 1.0f }
            , 0);
```

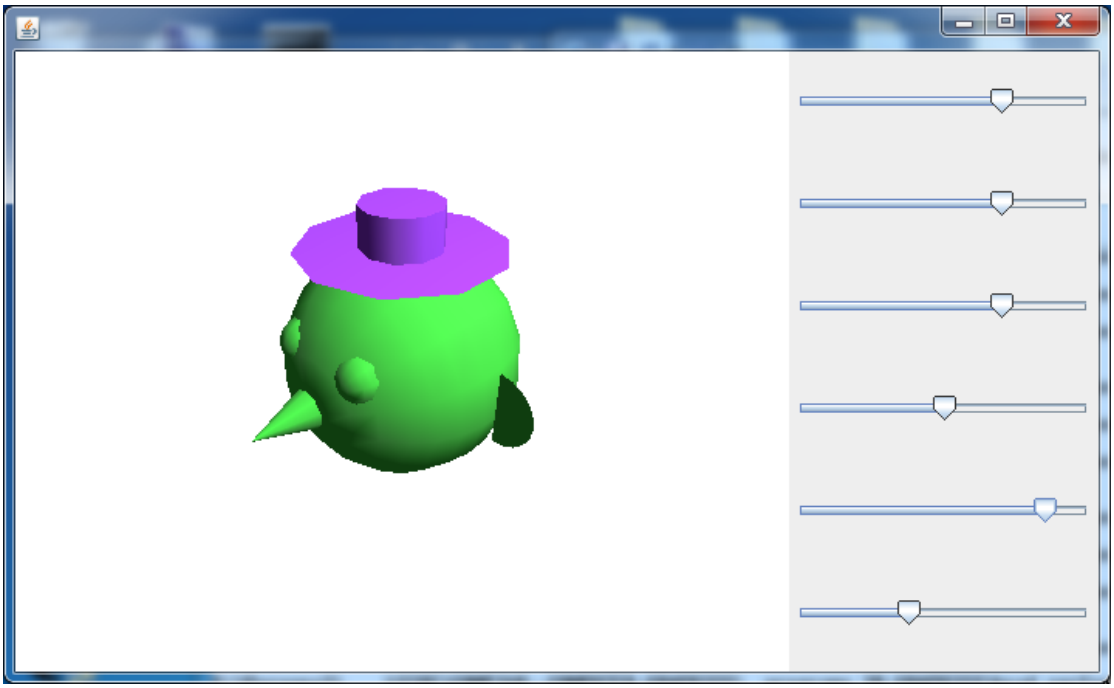
```
gl.glLightfv( GLLightingFunc.GL_LIGHT0
            , GLLightingFunc.GL_SPECULAR
            , new float[]{ 1.0f , 1.0f , 1.0f , 1.0f }
            , 0);
```

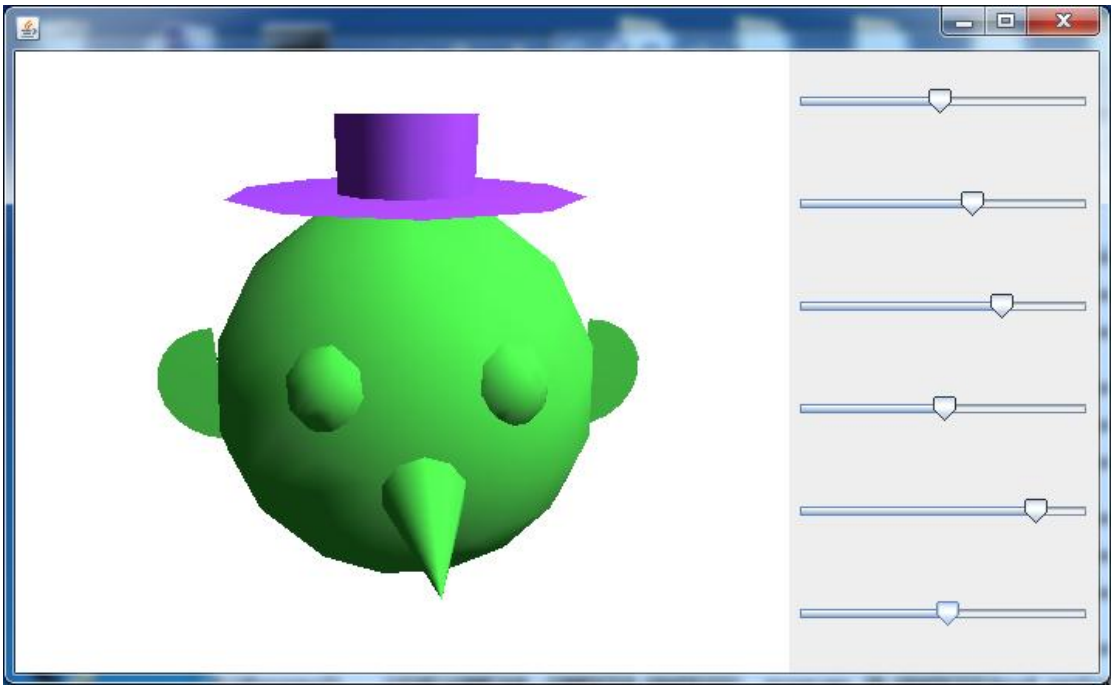
template:

```
class DrawingPanel extends GLJPanel
{
   GLU glu;    GLUquadric quad;
   . . . . . .

   public void init(GLAutoDrawable drawable)
   {
      . . . . . .

      glEnable(GL_LIGHTING);

      gl.glLightModelfv(  GL2ES1.GL_LIGHT_MODEL_AMBIENT
                        , new float[] { 0.2 , 0.2 , 0.2 , 1.0 } , 0);

      gl.glEnable(GLLightingFunc.GL_LIGHT0);
      . . . . . .

   public void display(GLAutoDrawable drawable)
   {
      . . . . . .

      // light source 0
      gl.glLightfv(  GLLightingFunc.GL_LIGHT0 , GLLightingFunc.GL_POSITION
                   , new float[]{ 0.0f , 30.0f , 30.0f , 1.0f } , 0);

      gl.glLightfv(  GLLightingFunc.GL_LIGHT0 , GLLightingFunc.GL_AMBIENT
                   , new float[]{ 0.1f , 0.1f , 0.1f , 1.0f } , 0);

      gl.glLightfv(  GLLightingFunc.GL_LIGHT0 , GLLightingFunc.GL_DIFFUSE
                   , new float[]{1.0f , 1.0f , 1.0f , 1.0f } , 0);

      gl.glLightfv(  GLLightingFunc.GL_LIGHT0 , GLLightingFunc.GL_SPECULAR
                   , new float[]{ 1.0f , 1.0f , 1.0f , 1.0f } , 0);

      // for each object:
      gl.glMaterialfv(  GL.GL_FRONT_AND_BACK
                      , GLLightingFunc.GL_AMBIENT_AND_DIFFUSE
                      , new float[] { 0.1 , 0.5 , 0.8 , 1.0 } , 0);

      gl.glMaterialfv(  GL.GL_FRONT_AND_BACK , GLLightingFunc.GL_SPECULAR
                      , new float[] { 1.0 , 1.0 , 1.0 , 1.0 } , 0);

      gl.glMaterialfv(  GL.GL_FRONT_AND_BACK , GLLightingFunc.GL_SHININESS
                      , new float[]{ 128.0f } , 0);
      . . . . . .
```

# Examples with lighting and surface rendering

*example 1: head with hat - previous example modified for surface rendering*





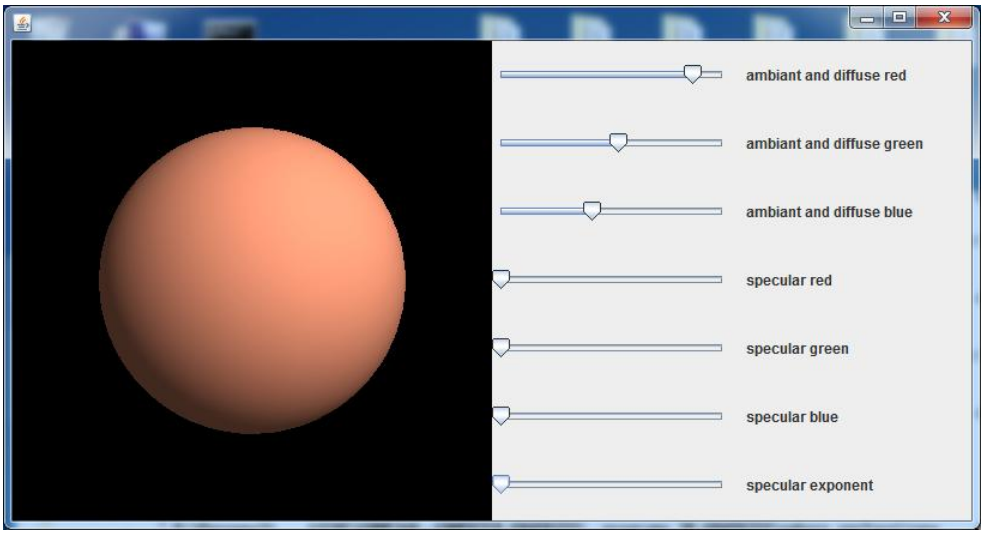
*example 1: (continued)*

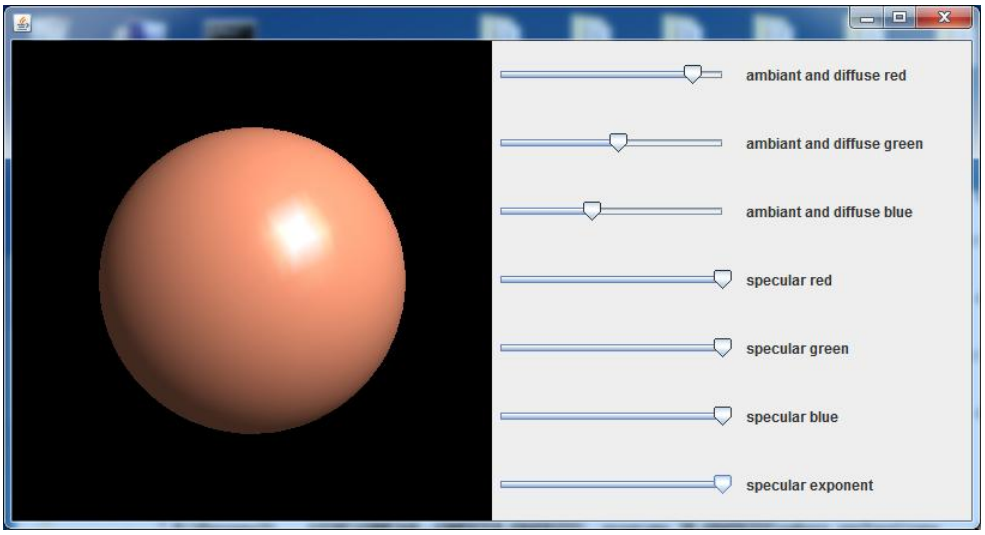
```
class Gui
{
   . . . . . .

   class DrawingPanel extends GLJPanel
   {
      GLU glu;    GLUquadric quad;

      DrawingPanel()
      {
         super(new GLCapabilities(GLProfile.getDefault()));

         this.addGLEventListener(new GLEventListener()
         {
            public void init(GLAutoDrawable drawable)    //***  INIT
            {
               GL2 gl = drawable.getGL().getGL2();

               glu = new GLU();    quad = glu.gluNewQuadric();
               glu.gluQuadricDrawStyle(quad , GLU.GLU_FILL);
               glu.gluQuadricNormals(quad , GLU.GLU_SMOOTH);

               gl.glClearColor(1.0f , 1.0f , 1.0f , 0.0f);

               gl.glEnable(GL.GL_DEPTH_TEST);
               gl.glEnable(GLLightingFunc.GL_LIGHTING);
               gl.glLightModelfv(  GL2ES1.GL_LIGHT_MODEL_AMBIENT
                                 , new float[] { 0.2f , 0.2f , 0.2f , 1.0f } , 0);

               gl.glEnable(GLLightingFunc.GL_LIGHT0);
            }

            public void reshape(GLAutoDrawable drawable , int x , int y , int w , int h)    //***  RESHAPE
            {
                 . . . . . .
            }
```

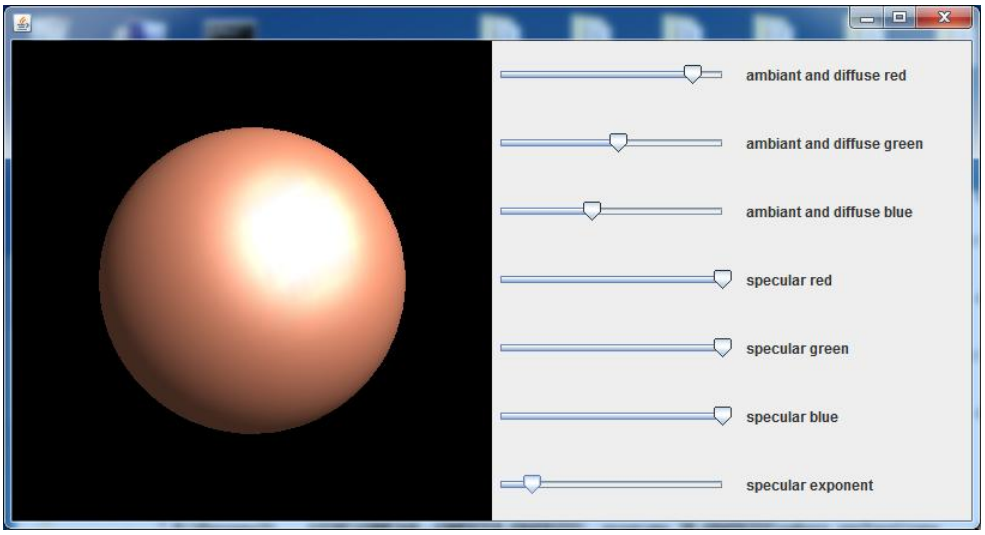
*example 1: (continued)*

```
        public void display(GLAutoDrawable drawable)    //***  DISPLAY
        {
           GL2 gl = drawable.getGL().getGL2();
           gl.glClear(GL.GL_COLOR_BUFFER_BIT);    gl.glClear(GL.GL_DEPTH_BUFFER_BIT);

           //// description of the viewing transformation
           . . . . . .

           ////------- LIGHT
           gl.glLightfv(  GLLightingFunc.GL_LIGHT0 , GLLightingFunc.GL_POSITION
                        , new float[]{ 100.0f , 100.0f , 30.0f , 1.0f } , 0);
           gl.glLightfv(  GLLightingFunc.GL_LIGHT0 , GLLightingFunc.GL_AMBIENT
                        , new float[]{ 0.1f , 0.1f , 0.1f , 1.0f } , 0);
           gl.glLightfv(  GLLightingFunc.GL_LIGHT0 , GLLightingFunc.GL_DIFFUSE
                        , new float[]{ 1.0f , 1.0f , 1.0f , 1.0f } , 0);
           gl.glLightfv(  GLLightingFunc.GL_LIGHT0 , GLLightingFunc.GL_SPECULAR
                        , new float[]{ 1.0f , 1.0f , 1.0f , 1.0f } , 0);

           ////------- HEAD
           gl.glMaterialfv(GL.GL_FRONT_AND_BACK , GLLightingFunc.GL_AMBIENT_AND_DIFFUSE
                         , new float[] { 0.2f , 0.8f , 0.2f , 1.0f } , 0);

           //// eyes added to the previous example:
           gl.glPushMatrix();    gl.glTranslatef(-10.0f , 10.0f , 22.0f);
                                 glu.gluSphere(quad , 6.0f , 10 , 10);    gl.glPopMatrix();

           gl.glPushMatrix();    gl.glTranslatef(10.0f , 10.0f , 22.0f);
                                 glu.gluSphere(quad , 6.0f , 10 , 10);    gl.glPopMatrix();
           //// rest of the geometric description of the head
           . . . . . .

           ////------- HAT
           gl.glMaterialfv(GL.GL_FRONT_AND_BACK , GLLightingFunc.GL_AMBIENT_AND_DIFFUSE
                         , new float[] { 0.6f , 0.2f , 1.0f , 1.0f } , 0);
           //// geometric description of the hat
           . . . . . .
           gl.glFlush();
        }

        public void dispose(GLAutoDrawable drawable)    //***  DISPOSE
        {}
      } );
    }
  }

  Gui()
  {
    . . . . . .
  }
}
```

*example 2: sphere with diffuse and specular coefficients adjusted interactively*



*example 2: (continued)*

```
class Gui
{
   JFrame f;    DrawingPanel p;

   float diff_r , diff_g , diff_b , spec_r , spec_g , spec_b , spec_e;
   JPanel ps;    JSlider s_diff_r , s_diff_g , s_diff_b , s_spec_r , s_spec_g , s_spec_b , s_spec_e;
                 JLabel  l_diff_r , l_diff_g , l_diff_b  , l_spec_r , l_spec_g  , l_spec_b ,  l_spec_e;

   class DrawingPanel extends GLJPanel
   {
      GLU glu;    GLUquadric quad;

      DrawingPanel()
      {
         super(new GLCapabilities(GLProfile.getDefault()));

         this.addGLEventListener(new GLEventListener()
         {
            public void init(GLAutoDrawable drawable)    //***  INIT
            {
               GL2 gl = drawable.getGL().getGL2();

               glu = new GLU();    quad = glu.gluNewQuadric();
               glu.gluQuadricDrawStyle(quad , GLU.GLU_FILL);
               glu.gluQuadricNormals(quad , GLU.GLU_SMOOTH);

               gl.glClearColor(0.0f , 0.0f , 0.0f , 0.0f);

               gl.glEnable(GL.GL_DEPTH_TEST);
               gl.glEnable(GLLightingFunc.GL_LIGHTING);
               gl.glLightModelfv(  GL2ES1.GL_LIGHT_MODEL_AMBIENT
                                 , new float[] { 0.2f , 0.2f , 0.2f , 1.0f } , 0);

               gl.glEnable(GLLightingFunc.GL_LIGHT0);
            }
```

*example 2: (continued)*

```
        public void reshape(GLAutoDrawable drawable , int x , int y , int w , int h)    //***  RESHAPE
        {
           GL2 gl = drawable.getGL().getGL2();

           gl.glViewport(0 , 0 , w , h);

           gl.glMatrixMode(GLMatrixFunc.GL_PROJECTION);
           gl.glLoadIdentity();    glu.gluPerspective(60.0f , (float) w / h , 1.0f , 10000.0f);
        }

        public void display(GLAutoDrawable drawable)    //***  DISPLAY
        {
           GL2 gl = drawable.getGL().getGL2();

           gl.glClear(GL.GL_COLOR_BUFFER_BIT);    gl.glClear(GL.GL_DEPTH_BUFFER_BIT);

           gl.glMatrixMode(GLMatrixFunc.GL_MODELVIEW);
           gl.glLoadIdentity();
           glu.gluLookAt(100.0f , 100.0f , 100.0f , 0.0f , 0.0f , 0.0f , 0.0f , 1.0f , 0.0f);

           ////------- LIGHT
           gl.glLightfv(  GLLightingFunc.GL_LIGHT0 , GLLightingFunc.GL_POSITION
                        , new float[]{ 100.0f , 100.0f , 50.0f , 1.0f } , 0);
           gl.glLightfv(  GLLightingFunc.GL_LIGHT0 , GLLightingFunc.GL_AMBIENT
                        , new float[]{ 0.1f , 0.1f , 0.1f , 1.0f } , 0);
           gl.glLightfv(  GLLightingFunc.GL_LIGHT0 , GLLightingFunc.GL_DIFFUSE
                        , new float[]{ 1.0f , 1.0f , 1.0f , 1.0f } , 0);
           gl.glLightfv(  GLLightingFunc.GL_LIGHT0 , GLLightingFunc.GL_SPECULAR
                        , new float[]{ 1.0f , 1.0f , 1.0f , 1.0f } , 0);

           ////------- SPHERE
           gl.glMaterialfv(GL.GL_FRONT_AND_BACK , GLLightingFunc.GL_AMBIENT_AND_DIFFUSE
                         , new float[] { diff_r , diff_g , diff_b, 1.0f } , 0);

           gl.glMaterialfv(GL.GL_FRONT_AND_BACK , GLLightingFunc.GL_SPECULAR
                         , new float[] { spec_r , spec_g , spec_b , 1.0f } , 0);

           gl.glMaterialfv(GL.GL_FRONT_AND_BACK , GLLightingFunc.GL_SHININESS
                         , new float[] { spec_e } , 0);

           glu.gluSphere(quad , 60.0f , 50 , 50);

           gl.glFlush();
        }

        public void dispose(GLAutoDrawable drawable)    //***  DISPOSE
        {}
      } );
   }
}
```

*example 2: (continued)*

```
  Gui()
  {
    f = new JFrame();   f.setFocusable(true);   f.setVisible(true);
    p = new DrawingPanel();   f.getContentPane().add(p , BorderLayout.CENTER);

    ////---------------------- SLIDERS
    ps = new JPanel();   ps.setLayout(new GridLayout(0 , 2));
    f.getContentPane().add(ps , BorderLayout.EAST);

    s_diff_r   = new JSlider(JSlider.HORIZONTAL, 0 , 100 , 0);  ps.add(s_diff_r);
    l_diff_r   = new JLabel("    ambiant and diffuse red");          ps.add(l_diff_r);

    s_diff_g  = new JSlider(JSlider.HORIZONTAL, 0 , 100 , 0);  ps.add(s_diff_g);
    l_diff_g  = new JLabel("    ambiant and diffuse green");       ps.add(l_diff_g);

    s_diff_b  = new JSlider(JSlider.HORIZONTAL, 0 , 100 , 0);  ps.add(s_diff_b);
    l_diff_b  = new JLabel("    ambiant and diffuse blue");        ps.add(l_diff_b);

    s_spec_r  = new JSlider(JSlider.HORIZONTAL, 0 , 100 , 0);  ps.add(s_spec_r);
    l_spec_r  = new JLabel("    specular red");                    ps.add(l_spec_r);

    s_spec_g = new JSlider(JSlider.HORIZONTAL, 0 , 100 , 0);  ps.add(s_spec_g);
    l_spec_g = new JLabel("    specular green");                  ps.add(l_spec_g);

    s_spec_b = new JSlider(JSlider.HORIZONTAL, 0 , 100 , 0);  ps.add(s_spec_b);
    l_spec_b = new JLabel("    specular blue");                   ps.add(l_spec_b);

    s_spec_e = new JSlider(JSlider.HORIZONTAL, 0 , 128 , 0);  ps.add(s_spec_e);
    l_spec_e = new JLabel("    specular exponent");               ps.add(l_spec_e);

    s_diff_r.addChangeListener(    . . . . . { diff_r   = 0.01f * s_diff_r.getValue();     f.repaint(); } } );
    s_diff_g.addChangeListener(    . . . . . { diff_g  =  0.01f * s_diff_g.getValue();    f.repaint(); } } );
    s_diff_b.addChangeListener(    . . . . . { diff_b  =  0.01f * s_diff_b.getValue();    f.repaint(); } } );
    s_spec_r.addChangeListener(    . . . . . { spec_r  = 0.01f * s_spec_r.getValue();    f.repaint(); } } );
    s_spec_g.addChangeListener(    . . . . . { spec_g = 0.01f * s_spec_g.getValue();    f.repaint(); } } );
    s_spec_b.addChangeListener(    . . . . . { spec_b = 0.01f * s_spec_b.getValue();    f.repaint(); } } );
    s_spec_e.addChangeListener(    . . . . . { spec_e =          s_spec_e.getValue();    f.repaint(); } } );

    f.setSize(new Dimension(800 + 16 , 400 + 38));
  }
}
```