

# Programmation 3D en java avec JOGL

## Exemples avec Swing

Rémy Malgouyres

LIMOS UMR 6158, IUT, département info

Université Clermont 1

B.P. 86

63172 AUBIERE cedex

[http ://www.malgouyres.fr/](http://www.malgouyres.fr/)

# Table des matières

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>L'API JOGL 2</b>                                   | <b>2</b>  |
| 1.1      | Les Canvas . . . . .                                  | 2         |
| 1.2      | L'interface <code>GLEventListener</code> . . . . .    | 4         |
| <b>2</b> | <b>Pattern Model View Contrôler (MVC)</b>             | <b>7</b>  |
| 2.1      | Le pattern MVC . . . . .                              | 7         |
| 2.2      | La vue . . . . .                                      | 8         |
| 2.3      | Le modèle . . . . .                                   | 13        |
| 2.4      | Le Contrôleur . . . . .                               | 18        |
| <b>3</b> | <b>Développement d'un Framework Métier pour la 3D</b> | <b>23</b> |
| 3.1      | Transformations géométriques . . . . .                | 23        |
| 3.2      | Hierarchie d'objets 3D . . . . .                      | 32        |
| 3.3      | Gestion de la vue et des caméras . . . . .            | 42        |
| 3.4      | Gestion de l'éclairage . . . . .                      | 50        |
| 3.5      | Exemple d'application . . . . .                       | 57        |
| <b>4</b> | <b>Graphe de scène</b>                                | <b>74</b> |
| 4.1      | Définition du graphe de scène . . . . .               | 74        |
| 4.2      | L'exemple d'un robot . . . . .                        | 75        |

# Chapitre 1

## L'API JOGL 2

Ce cours suppose comme prérequis une connaissance de base en *OpenGL*. l'API *JOGL 2* permet d'accéder aux fonctionnalités d'*OpenGL* dans des applications *Java*. A cet effet, *JOGL* définit un ensemble d'interfaces et de classes les implémentant telles que *GL2* (qui définit les méthodes de rendu basées sur le zbuffer d'*OpenGL* préfixées par *gl* comme *glBegin()*), *GLUgl2* (qui définit les méthodes de la *GLU* telles que *gluLookAt*) ou encore *GLUT* (gui définit les méthodes de la *GLUT* comme *glutWireTeapot*).

### 1.1 Les Canvas

l'API *JOGL 2* définit aussi des composants comme *GLCanvas*, qui hérite du composant *AWT Canvas*, que l'on peut insérer dans des composants *AWT* ou *Swing* comme *JPanel* ou *JFrame*, et dans lequel on peut afficher des buffers *OpenGL* dans une méthode d'affichage.

**Exemple 1.** Voici un exemple qui dessine une théière avec une vue en perspective et un point de vue spécifié par *gluLookAt*.

exemplesLatex/CoursJOGL1/src/app/AppFrm1.java

```
1 /**
2  *
3  */
4 package app;
5
6 import javax.swing.JFrame;
7 import javax.media.opengl.awt.GLCanvas;
8
9 import swingView.GLDisplayEx1;
10
11
12 /** Fenêtre principale de l'application
13  * Créé toutes les données et instances de classes de l'application et la vue.
14  */
15 public class AppFrm1 extends JFrame {
16
17     /**
18     * serialVersionUID (sert pour la serialisation , obligatoire pour une JFrame
19     */
20     private static final long serialVersionUID = 9167791876718956063L;
21
22     /**
```

```

23     * @param args non utilisé
24     */
25     public static void main(String [] args) {
26         // TODO Auto-generated method stub
27
28         AppFrm1 frame = new AppFrm1();
29         frame.setVisible(true);
30     }
31
32     /*_____*/
33     /** Constructeur de la vue. Crée les instances du modèle, du contrôleur,
34         * démarre le timer, etc...
35         */
36     public AppFrm1() {
37         // TODO Auto-generated constructor stub
38         GLCanvas canvas = new GLCanvas();
39         canvas.addGLEventListener(new GLDisplayEx1());
40         setSize(500, 500);
41         MyWindowAdapter winAdapt = new MyWindowAdapter();
42         addWindowListener(winAdapt);
43         add(canvas);
44         JFrame.setDefaultLookAndFeelDecorated(true);
45     }
46
47 }

```

exemplesLatex/CoursJOGL1/src/app/MyWindowAdapter.java

```

1  /**
2  *
3  */
4  package app;
5
6  import java.awt.event.WindowAdapter;
7  import java.awt.event.WindowEvent;
8
9  /**
10     * @author remy
11     */
12     /**
13     * Classe définissant la réaction de l'application aux événements de fenêtre (
14         * comme la fermeture de la fenêtre)
15     */
16     public class MyWindowAdapter extends WindowAdapter {
17
18         /** Constructeur initialisant l'animateur
19         * */
20         public MyWindowAdapter() {
21         }
22
23         /** Termine l'application après avoir stoppé l'animateur (si gestion avec un
24             * animateur) */
25         @Override
26         public void windowClosing(WindowEvent e) {
27             // TODO Auto-generated method stub

```

```

26     new Thread(new Runnable() {
27         public void run() {
28             System.exit(0);
29         }
30     }).start();
31 }
32
33 }
```

## 1.2 L'interface GLEventListener

L'API *JOGL 2* d finit aussi l'interface `GLEventListener`, qui h rite de `EventListener`, et qui permet d'impl menter des  v nements tels que `reshape` (qui se produit lorsque les dimensions du canvas ont chang es) ou `display`, qui se produit lorsque l'affichage doit  tre rafraichi, soit par un appel explicite, soit par un `repaint` du `GLCanvas` (ou en g n ral du `GLAutoDrawable`) associ .

exemplesLatex/CoursJOGL1/src/swingView/GLDisplayEx1.java

```

1 package swingView;
2
3
4 import javax.media.opengl.*;
5 import javax.media.opengl.glu.gl2.GLUgl2;
6
7 import com.jogamp.opengl.util.gl2.GLUT;
8
9
10
11 /**
12  * Classe g rant les  v nements d'affichage OpenGL (mais pas la GLUT)
13  */
14 public class GLDisplayEx1 implements GLEventListener{
15
16
17     /** M thode d'affichage (appel e lors d'un appel explicite au repaint() du
18         canvas ou par l'Animator)
19     * pour rafraichir la vue. Contient tout le code d'affichage
20     */
21     @Override
22     public void display(GLAutoDrawable glDrawable) {
23         // TODO Auto-generated method stub
24         final GL2 gl = glDrawable.getGL().getGL2();
25         GLUgl2 glu = new GLUgl2();
26         GLUT glut = new GLUT();
27
28         gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
29         gl.glMatrixMode(GL2.GL_MODELVIEW);
30         gl.glLoadIdentity();
31         glu.gluLookAt(10,10,10,0,0,0,0,1,0);
32
33         glut.glutWireTeapot(5);
34
35         gl.glFlush();
36     }
37 }
```

```
35 }
36
37
38
39 /** Appelé lors de l'initialisation de la vue. Définir le ViewPoint et la
    projection en perspective */
40 @Override
41 public void init(GLAutoDrawable glDrawable) {
42     // TODO Auto-generated method stub
43     final GL2 gl = glDrawable.getGL().getGL2();
44
45     initViewPortProjectionandLight(glDrawable, 0, 0, 500, 500);
46
47     gl.glColor3f(0.0f, 0.0f, 0.0f);
48     gl.glClearColor(1, 1, 1, 1);
49     gl.glLineWidth(1);
50 }
51
52 /** Appelé lors d'un redimensionnement de la fenêtre. Définir le ViewPoint et
    la projection en perspective */
53 @Override
54 public void reshape(GLAutoDrawable glDrawable, int x, int y, int width,
55     int height) {
56     // TODO Auto-generated method stub
57     initViewPortProjectionandLight(glDrawable, x, y, width, height);
58 }
59
60 /** Méthode d'initialisation de la scène, création des objets, des sources
    lumineuses...
61 * @param glDrawable référence du GLCanvas
62 * @param x non utilisé
63 * @param y non utilisé
64 * @param width largeur de la fenêtre graphique
65 * @param height hauteur de la fenêtre graphique
66 */
67 private void initViewPortProjectionandLight(GLAutoDrawable glDrawable, int x,
68     int y, int width,
69     int height){
70     final GL2 gl = glDrawable.getGL().getGL2();
71     GLUgl2 glu = new GLUgl2();
72
73     gl.glMatrixMode(GL2.GL_PROJECTION);
74     gl.glLoadIdentity();
75     glu.gluPerspective(45.0f, (float)width/(float)height, 0.1, 200.0);
76     gl.glMatrixMode(GL2.GL_MODELVIEW);
77     gl.glLoadIdentity();
78 }
79
80 @Override
81 public void dispose(GLAutoDrawable arg0) {
82     // TODO Auto-generated method stub
83 }
84 }
```

En revanche, la gestion des événements tels que le click de souris, le clavier pour les animations doivent être gérés pour leurs implémentations *Java* classique (par exemple les interfaces *AWT* `MouseListener` et `MouseMotionListener`).

# Chapitre 2

## Pattern Model View Controller (MVC)

### 2.1 Le pattern MVC

Le pattern MVC (voir la figure 2.1) permet de gérer les mises à jour d'une interface lors d'événements divers (utilisateur, timer, etc...).

Le principe est de faciliter la maintenance du code en séparant les aspects visualisation (Vue), les aspects de gestion des événements (Contrôleur), et la gestion des données de l'application (Modèle). Ces trois aspects sont gérés dans des packages distincts, chacun par une ou plusieurs classes.

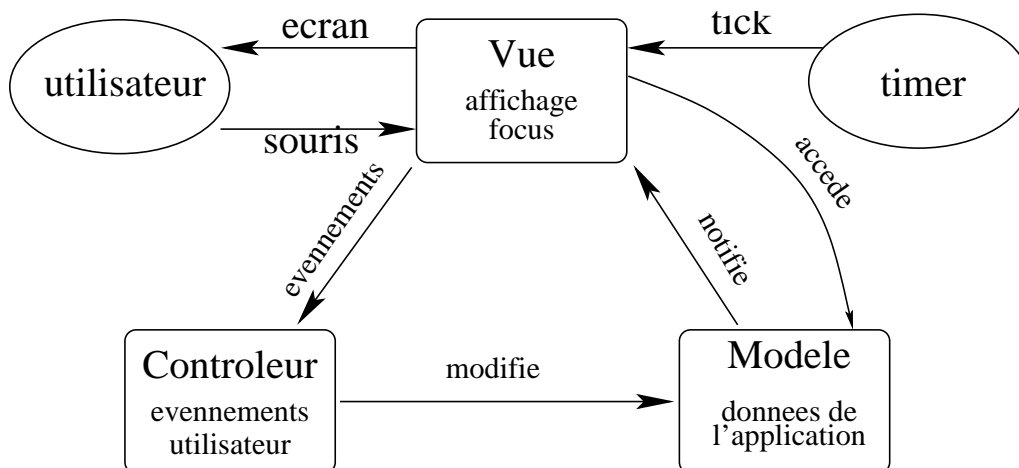


FIGURE 2.1: Le pattern MVC

**Exemple 2.** Voici un autre exemple avec affichage d'une sphère texturée (quadrique avec chargement de la texture à partir d'un fichier *PNG*). La sphère tourne sur elle-même sous l'effet d'une animation gérée par un timer. A cet effet, une méthode `stepForward` a été créée dans la classe gérant l'affichage.

Le rafraîchissement de la vue peut être géré de deux manières : soit par des appels explicites au `repaint` du canvas, soit par un `Animator` qui provoque le rafraîchissement de la vue périodiquement de manière automatique. Dans l'exemple suivant, le rafraîchissement de la vue est géré par un timer qui rafraîchit la vue à travers un modèle observable.



## 2.2 La vue

La vue est ici une fen tre swing avec un affichage JOGL.

exemplesLatex/CoursJOGL2/src/app/AppFrm2.java

```

1  /**
2   *
3   */
4  package app;
5
6
7  import java.util.Observable;
8  import java.util.Observer;
9
10 import javax.swing.JFrame;
11
12 import javax.media.opengl.awt.GLCanvas;
13
14 import swingView.GLDisplayEx2;
15 import mdl.AppMdl2;
16 import ctrl.AnimationTimer2;
17 import ctrl.CtrlMouse2;
18
19
20 /** Fen tre principale de l'application
21  * Cr e toutes les donn es et instances de classes de l'application et la vue.
22  */
23 public class AppFrm2 extends JFrame implements Observer {
24
25     /**
26     * serialVersionUID (sert pour la serialisation, obligatoire pour une JFrame
27     */
28     private static final long serialVersionUID = 9167791876718956063L;
29
30     /** Canvas, c'est le composant utilis  pour dessiner avec JOGL */
31     private static GLCanvas canvas=null;
32
33
34     /**
35     * @param args non utilis 
36     */
37     public static void main(String[] args) {
38         // TODO Auto-generated method stub
39
40         AppFrm2 frame = new AppFrm2();
41         frame.setVisible(true);
42     }
43
44     /*_____*/
45     /** permet d'acc der au canvas
46     * @return le canvas de la vue
47     */
48     public static GLCanvas getCanvas(){
49         return canvas;
50     }
51     /*_____*/

```

```

52  /** Constructeur de la vue. Crée les instances du modèle, du contrôleur ,
    *      démarre le timer, etc...
53  */
54  public AppFrm2() {
55      // TODO Auto-generated constructor stub
56      canvas = new GLCanvas();
57      AppMdl2 mdl = new AppMdl2();
58      mdl.addObserver(this);
59      GLDisplayEx2 glDisplayEx2 = new GLDisplayEx2(canvas, mdl);
60      canvas.addGLEventListener(glDisplayEx2);
61      setSize(500, 500);
62
63      MyWindowAdapter winAdapt = new MyWindowAdapter();
64      addWindowListener(winAdapt);
65      CtrlMouse2 ctrl = new CtrlMouse2(mdl);
66      add(canvas);
67      canvas.addMouseListener(ctrl);
68      canvas.addMouseListener(ctrl);
69      AnimationTimer2.getInstance().start(30, mdl);
70      JFrame.setDefaultLookAndFeelDecorated(true);
71  }
72
73
74  /** Méthode update utilisée dans une gestion sans animator
75   * La méthode update de l'interface Observer est appelée automatiquement
76   * lors d'une notification par le modèle. (voir pattern MVC)
77   * redessine le canvas
78   */
79  @Override
80  public void update(Observable arg0, Object arg1)
81  {
82      // PENSER à IMPLEMENTER Auto-generated method stub
83      System.err.println("Update");
84      canvas.repaint();
85  }
86  }

```

La méthode `update` de l'interface `Observer` permet de rafraichir la vue suivant le pattern `Observable`. Pour cela, nous verrons que le modèle doit notifier la vue lorsque ses données ont changé.

L'interface permet de visualiser la scène sous différents angles en utilisant la souris. C'est le sens des angles `elevation` et `azimuth`. Pour cela, à la place du `gluLookAt`, on a fait les transformations suivantes pour le changement de repère vers la vue.

1. Faire une rotation d'un angle (`angle_azimuth`) autour de l'axe des  $y$  ;
2. puis dans le nouveau repère faire une rotation d'un angle `angle_elevation` autour de l'axe des  $x$  ;
3. et enfin dans le nouveau repère faire une translation le long de l'axe des  $Z$  d'une certaine distance (définir les variables correspondantes) (voir la figure 2.2).

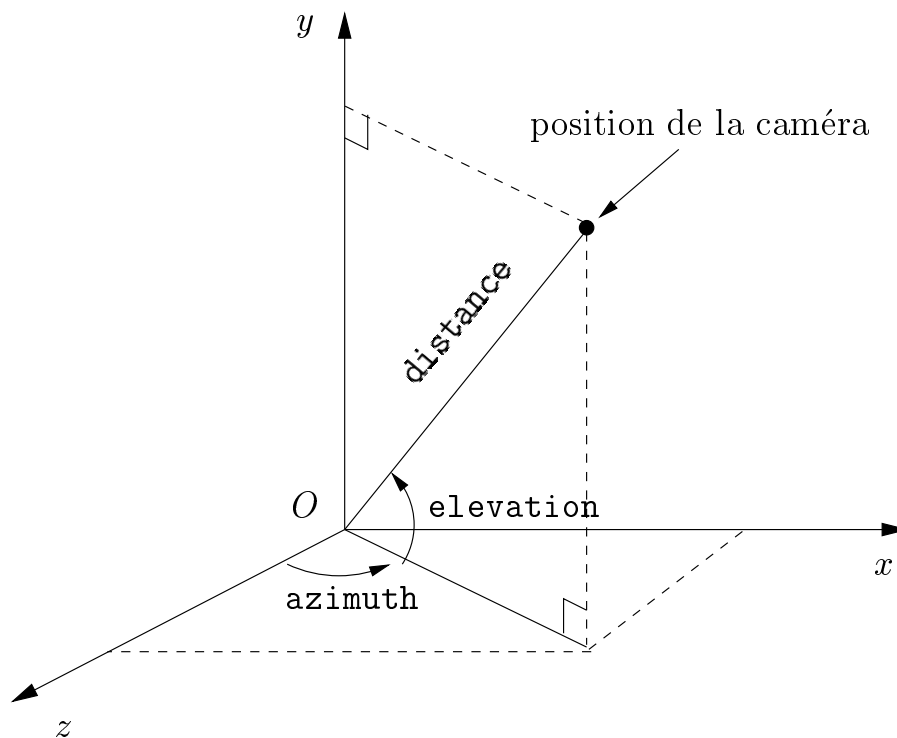


FIGURE 2.2: Les angles azimuth et elevation

```

1 package swingView;
2
3 import java.awt.image.BufferedImage;
4 import java.awt.image.DataBufferByte;
5 import java.io.File;
6
7
8 import javax.imageio.ImageIO;
9 import javax.media.opengl.glu.GLUquadric;
10 import javax.media.opengl.glu.gl2.GLUgl2;
11
12 import javax.media.opengl.GL2;
13 import javax.media.opengl.GLAutoDrawable;
14 import javax.media.opengl.GLEventListener;
15
16 import mdl.AppMdl2;
17
18 import java.nio.Buffer;
19 import java.nio.ByteBuffer;
20
21 /**
22  * Classe gérant les événements d'affichage OpenGL (mais pas la GLUT)
23  *
24  */
25 public class GLDisplayEx2 implements GLEventListener{
26
27     /** ID de texture */
28     final int [] texId = new int [1];
29

```

```

30  /** objet de type quadrique */
31  GLUquadric qobj;
32
33  /** référence du GLCanvas */
34  GLAutoDrawable drawable;
35
36  /** Modèle de l'application contenant les angles caractéristiques de la vue
37  */
38  AppMdl2 mdl;
39
40  /**
41   * Constructeur mémorisant le canvas
42   * @param glDrawable le canvas
43   * @param mdl
44   */
45  public GLDisplayEx2(GLAutoDrawable glDrawable, AppMdl2 mdl){
46      this.drawable=glDrawable;
47      this.mdl = mdl;
48  }
49
50  /** Méthode d'affichage (appelée lors d'un appel explicite au repaint() du
51   * canvas ou par l'Animator)
52   * pour rafraichir la vue. Contient tout le code d'affichage
53   */
54  @Override
55  public void display(GLAutoDrawable glDrawable) {
56      // TODO Auto-generated method stub
57      final GL2 gl = glDrawable.getGL().getGL2();
58      GLUgl2 glu = new GLUgl2();
59
60      gl.glClear(GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BIT);
61      gl.glMatrixMode(GL2.GL_MODELVIEW);
62      gl.glLoadIdentity();
63
64      gl.glTranslatef(0.0f,0.0f,-mdl.getDistance());
65      gl.glRotatef(mdl.getAngleElevation(),1.0f,0.0f,0.0f);
66      gl.glRotatef(mdl.getAngleAzimuth(),0.0f,1.0f,0.0f);
67
68      gl.glRotatef(mdl.getVitesse()*mdl.getParametreAnimation(),0,1,0);
69
70      gl.glBindTexture(GL2.GL_TEXTURE_2D, texId[0]);
71
72      glu.gluSphere(qobj, 5, 45, 20);
73
74      gl.glFlush();
75  }
76
77  /** Appelé lors de l'initialisation de la vue. Définir le ViewPoint et la
78   * projection en perspective */
79  @Override
80  public void init(GLAutoDrawable glDrawable) {
81      // TODO Auto-generated method stub
82      final GL2 gl = glDrawable.getGL().getGL2();
83      GLUgl2 glu = new GLUgl2();

```

```

83
84     initViewPortProjectionandLight(glDrawable, 0, 0, 500, 500);
85
86     gl.glColor3f(0.0f, 0.0f, 0.0f);
87     gl.glLineWidth(1);
88
89     qobj = glu.gluNewQuadric();
90     glu.gluQuadricDrawStyle(qobj, GLUgl2.GLU_FILL); /* smooth shaded */
91     glu.gluQuadricNormals(qobj, GLUgl2.GLU_SMOOTH);
92     glu.gluQuadricTexture(qobj, true);
93     gl.glEnable(GL2.GL_TEXTURE_2D);
94
95     gl.glGenTextures(1, texId, 0);
96
97     gl.glBindTexture(GL2.GL_TEXTURE_2D, texId[0]);
98     gl.glTexParameteri(GL2.GL_TEXTURE_2D, GL2.GL_TEXTURE_MIN_FILTER, GL2.
99         GL_LINEAR);
100
101     gl.glTexParameteri(GL2.GL_TEXTURE_2D, GL2.GL_TEXTURE_MAG_FILTER, GL2.
102         GL_LINEAR);
103
104     try {
105         BufferedImage image = ImageIO.read(new File("fichiersTextures/textthe.
106             png"));
107         ByteBuffer dbb = (ByteBuffer)image.getRaster().getDataBuffer()
108             ;
109         byte[] data = dbb.getData();
110         ByteBuffer pixels = ByteBuffer.allocate(data.length);
111         pixels.put(data);
112         Buffer buff = pixels.flip();
113
114         gl.glTexImage2D(GL2.GL_TEXTURE_2D, 0, GL2.GL_RGB, image.getWidth(),
115             image.getHeight(), 0, GL2.GL_RGB, GL2.GL_UNSIGNED_BYTE, buff);
116     } catch(Throwable t) {
117         t.printStackTrace();
118     }
119 }
120
121 /** Appel  lors d'un redimensionnement de la fen tre. D finir le ViewPoint et
122     la projection en perspective */
123 @Override
124 public void reshape(GLAutoDrawable glDrawable, int x, int y, int width,
125     int height) {
126     // TODO Auto-generated method stub
127     initViewPortProjectionandLight(glDrawable, x, y, width, height);
128 }
129
130 /** M thode d'initialisation de la sc ne, cr ation des objets, des sources
131     lumineuses...
132     * @param glDrawable r f rence du GLCanvas
133     * @param x non utilis 
134     * @param y non utilis 
135     * @param width largeur de la fen tre graphique
136     * @param height hauteur de la fen tre graphique
137     */
138 private void initViewPortProjectionandLight(GLAutoDrawable glDrawable, int x,
139     int y, int width,

```

```

132     int height){
133         final GL2 gl = drawable.getGL().getGL2();
134         GLUgl2 glu = new GLUgl2();
135
136         gl.glMatrixMode(GL2.GL_PROJECTION);
137         gl.glLoadIdentity();
138         glu.gluPerspective(45.0f, (float)width/(float)height, 0.1, 200.0);
139         gl.glMatrixMode(GL2.GL_MODELVIEW);
140         gl.glLoadIdentity();
141         float mat_ambient[] =
142         { 0.2f, 0.2f, 0.2f, 1.0f };
143         float mat_diffuse[] =
144         { 1.0f, 1.0f, 1.0f, 1.0f };
145         float mat_specular[] =
146         { 1.0f, 1.0f, 1.0f, 1.0f };
147         float mat_shininess = 110.0f;
148         float light_position[] =
149         { 0f, 0f, 300f, 0.0f };
150         float model_ambient[] =
151         { 0.2f, 0.2f, 0.2f, 1.0f };
152         gl.glClearColor(1.0f, 1.0f, 1.0f, 0.0f);
153         gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_AMBIENT, mat_ambient, 0);
154         gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_DIFFUSE, mat_diffuse, 0);
155         gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_SPECULAR, mat_specular, 0);
156         gl.glMaterialf(GL2.GL_FRONT, GL2.GL_SHININESS, mat_shininess);
157         gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_POSITION, light_position, 0);
158         gl.glLightModelfv(GL2.GL_LIGHT_MODEL_AMBIENT, model_ambient, 0);
159         gl.glEnable(GL2.GL_LIGHTING);
160         gl.glEnable(GL2.GL_LIGHT0);
161         gl.glEnable(GL2.GL_DEPTH_TEST);
162         gl.glShadeModel(GL2.GL_SMOOTH);
163     }
164
165     @Override
166     public void dispose(GLAutoDrawable arg0) {
167         // TODO Auto-generated method stub
168
169     }
170 }

```

## 2.3 Le modèle

Ici le modèle est très simple et contient simplement les données qui définissent le repère de la caméra (distances, angles, etc...). Dans la suite, nous verrons un exemple où le modèle contient aussi les objets à afficher et leurs transformations du modèle.

Lorsque les données du modèle ont changé suite à des événements, celui-ci notifie la vue suivant le pattern `Observable` dans la méthode `notifyChanges` qui appelle la méthode `Observable.notifyObservers`. Pour cela, le modèle hérite de la classe `Observable`.

exemplesLatex/CoursJOGL2/src/mdl/AppMdl2.java

```

1 /**
2  *
3  */

```

```
4 package mdl;
5
6 import java.util.Observable;
7
8 import ctrl.CtrlMouse2;
9
10 /**
11  * @author remy
12  *
13  */
14 public class AppMdl2 extends Observable {
15
16     /** Angle Twist utilis  pour l'interface de visualisation   la souris */
17     private float angleTwist;
18     /** Angle Elevation utilis  pour l'interface de visualisation   la souris */
19     private float angleElevation;
20     /** Angle Azimuth utilis  pour l'interface de visualisation   la souris */
21     private float angleAzimuth;
22     /** Distance au centre utilis e pour l'interface de visualisation   la souris
23         */
24     private float distance;
25
26     /** pr c dente position X de la souris */
27     private int mouseX;
28     /** pr c dente position Y de la souris */
29     private int mouseY;
30
31     /** vitesse de l'effet de la souris */
32     float vitesse;
33
34     /** Param tre incr ment  par le timer pour l'animation */
35     int parametreAnimation=0;
36
37     /** _____ */
38     /** Constructeur initialisant les angles, distances, vitesse, etc...
39         */
40     public AppMdl2()
41     {
42         setAngleTwist(0.0f);
43         setAngleElevation(30.0f);
44         setAngleAzimuth(30.0f);
45
46         setVitesse(1);
47         setDistance(30);
48
49         setMouseX(0);
50         setMouseY(0);
51     }
52
53     /** _____ */
54     /** Accesseur permet d'obtenir la valeur de distance
55         * @return la valeur du champs distance
56         */
57     public float getDistance() {
58         return distance;
59     }
60 }
```

```

59
60
61
62 /*_____*/
63 /** Permet de fixer la valeur du champs distance
64  * @param distance
65  */
66 public void setDistance(float distance) {
67     this.distance = distance;
68 }
69
70
71
72 /*_____*/
73 /** Accesseur permet d'obtenir la valeur de vitesse
74  * @return la valeur du champs vitesse
75  */
76 public float getVitesse() {
77     return vitesse;
78 }
79
80
81
82 /*_____*/
83 /** Permet de fixer la valeur du champs vitesse
84  * @param vitesse
85  */
86 public void setVitesse(float vitesse) {
87     this.vitesse = vitesse;
88 }
89
90
91 /*_____*/
92 /** Accesseur permet d'obtenir la valeur de l'angle twist
93  * @return la valeur du champs angleTwist
94  */
95 public float getAngleTwist() {
96     return angleTwist;
97 }
98
99
100
101 /*_____*/
102 /** Permet de fixer la valeur du champs angleTwist
103  * @param angleTwist
104  */
105 public void setAngleTwist(float angleTwist) {
106     this.angleTwist = angleTwist;
107 }
108
109
110
111 /*_____*/
112 /** Accesseur permet d'obtenir la valeur de l'angle Elevation
113  * @return la valeur du champs angleElevation
114  */

```



```
115 public float getAngleElevation() {
116     return angleElevation;
117 }
118
119
120
121
122 /*_____*/
123 /** Permet de fixer la valeur du champs angleElevation
124  * @param angleElevation
125  */
126 public void setAngleElevation(float angleElevation) {
127     this.angleElevation = angleElevation;
128 }
129
130
131
132 /*_____*/
133 /** Accesseur permet d'obtenir la valeur de l'angle Azimuth
134  * @return la valeur de angleAzimuth
135  */
136 public float getAngleAzimuth() {
137     return angleAzimuth;
138 }
139
140
141
142 /*_____*/
143 /** Permet de fixer la valeur du champs angleAzimuth
144  * @param angleAzimuth
145  */
146 public void setAngleAzimuth(float angleAzimuth) {
147     this.angleAzimuth = angleAzimuth;
148 }
149
150
151
152 /*_____*/
153 /** Accesseur permet d'obtenir la valeur du champs parametreAnimation
154  * @return la valeur du champs parametreAnimation
155  */
156 public int getParametreAnimation() {
157     return parametreAnimation;
158 }
159
160
161
162 /*_____*/
163 /** Permet de fixer la valeur du champs parametreAnimation
164  * @param parametreAnimation
165  */
166 public void setParametreAnimation(int parametreAnimation) {
167     this.parametreAnimation = parametreAnimation;
168 }
169
170 /*_____*/
```

```

171  /** Accesseur permet d'obtenir la valeur de la coordonnée x de la souris
172  * @return la valeur du champs mouseX
173  */
174  public int getMouseX() {
175      return mouseX;
176  }
177
178
179
180  /*_____*/
181  /** Permet de fixer la valeur du champs mouseX
182  * @param mouseX
183  */
184  public void setMouseX(int mouseX) {
185      this.mouseX = mouseX;
186  }
187
188
189
190
191  /*_____*/
192  /** Accesseur permet d'obtenir la valeur de la coordonnée y de la souris
193  * @return la valeur du champs mouseY
194  */
195  public int getMouseY() {
196      return mouseY;
197  }
198
199
200
201  /*_____*/
202  /** Permet de fixer la valeur du champs mouseY
203  * @param mouseY
204  */
205  public void setMouseY(int mouseY) {
206      this.mouseY = mouseY;
207  }
208
209  /*_____*/
210  /** Méthode appelée par le contrôleur lorsque la souris se déplace avec un
    bouton enfoncé
211  * @param newMouseX nouvelle coordonnée X de la souris
212  * @param newMouseY nouvelle coordonnée Y de la souris
213  * @param buttonPressed tableau indiquant l'état des boutons
214  */
215  public void moveDragged(int newMouseX, int newMouseY, boolean buttonPressed [])
    {
216      if (buttonPressed [CtrlMouse2.LEFT_BTN])
217      {
218          System.err.println ("Modif de elevation et azimuth");
219          angleElevation += vitesse /5.0*(newMouseY–mouseY);
220          angleAzimuth += vitesse /5.0*(newMouseX–mouseX);
221      }
222
223      if (buttonPressed [CtrlMouse2.MIDDLE_BTN])
224      {

```

```

225     System.err.println("Modif de distance");
226     distance += vitesse*(newMouseY-mouseY);
227 }
228     if (buttonPressed [ CtrlMouse2.LEFT_BTN] || buttonPressed [ CtrlMouse2.
        MIDDLE_BTN])
229 {
230     mouseX = newMouseX;
231
232     mouseY = newMouseY;
233     notifyChanges ();
234 }
235 }
236
237 /** M thode pour notifier la vue qu'elle doit se rafraichir (si gestion sans
    Animator) */
238 private void notifyChanges () {
239     System.err.println("Notification");
240     setChanged ();
241     notifyObservers ();
242 }
243
244
245
246
247
248 /*
249 /** M thode faisant avancer l'animation et appel e r guli rement par la
    m thode run du TimerTask
250 */
251 public void stepForward () {
252     // TODO Auto-generated method stub
253     parametreAnimation += vitesse;
254     if (parametreAnimation >= 360)
255         parametreAnimation -=360;
256     notifyChanges ();
257 }
258
259
260
261
262 }

```

## 2.4 Le Contr leur

Le Contr leur permet de r ceptionner les  v nements issus de la souris :

exemplesLatex/CoursJOGL2/src/ctrl/CtrlMouse2.java

```

1 package ctrl;
2
3 import java.awt.event.MouseEvent;
4 import java.awt.event.MouseListener;
5 import java.awt.event.MouseMotionListener;
6
7 import mdl.AppMdl2;

```

```

8
9 /*_____*/
10 /** Contrôleur de l'application (voir pattern MVC)
11  * Réceptionne les événements souris (AWT) et modifie le modèle en conséquence
12  */
13 public class CtrlMouse2 implements MouseListener, MouseMotionListener {
14
15     /** Modèle contenant les données de l'application */
16     private AppMdl2 mdl;
17
18     /** ID des boutons de la souris (indices dans le tableau buttonPressed) */
19     public final static int LEFT_BTN=0;
20     /** ID des boutons de la souris (indices dans le tableau buttonPressed) */
21     public final static int MIDDLE_BTN=1;
22     /** ID des boutons de la souris (indices dans le tableau buttonPressed) */
23     public final static int RIGHT_BTN=2;
24
25     /** Tableau indiquant l'état des boutons de la souris (vrai si bouton
26      *   enfoncé */
27     private boolean buttonPressed [] = {false, false, false};
28
29 /*_____*/
30 /** Constructeur initialisant le modèle
31  * @param mdl Modèle contenant les données de l'application
32  */
33 public CtrlMouse2(AppMdl2 mdl) {
34     // TODO Auto-generated constructor stub
35     this.mdl = mdl;
36 }
37
38 /** Appelé lors d'un click de souris */
39 @Override
40 public void mouseClicked(MouseEvent arg0) {
41     // TODO Auto-generated method stub
42     System.err.println("MouseClicked");
43 }
44
45 /** Appelé lorsque la souris entre dans un composant */
46 @Override
47 public void mouseEntered(MouseEvent arg0) {
48     // TODO Auto-generated method stub
49     System.err.println("MouseEntered");
50 }
51
52 /** Appelé lorsque la souris sort d'un un composant */
53 @Override
54 public void mouseExited(MouseEvent arg0) {
55     // TODO Auto-generated method stub
56 }
57 }
58
59 /** Appelé lorsqu'on enfonce un bouton */
60 @Override
61 public void mousePressed(MouseEvent arg0) {
62     // TODO Auto-generated method stub

```

```

63
64 System.err.println("MousePressed[" +arg0.getX()+" ,"+arg0.getY()+"");
65 if (arg0.getButton()== MouseEvent.BUTTON1)
66     buttonPressed[LEFT_BTN] = true;
67 if (arg0.getButton()== MouseEvent.BUTTON2)
68     buttonPressed[MIDDLE_BTN] = true;
69 if (arg0.getButton()== MouseEvent.BUTTON3)
70     buttonPressed[RIGHT_BTN] = true;
71
72     mdl.setMouseX(arg0.getX());
73     mdl.setMouseY(arg0.getY());
74 }
75
76 /** appel  lorsqu'on lib re un bouton */
77 @Override
78 public void mouseReleased(MouseEvent arg0) {
79     // TODO Auto-generated method stub
80     System.err.println("MouseReleased");
81     if (arg0.getButton()== MouseEvent.BUTTON1)
82         buttonPressed[LEFT_BTN] = false;
83     if (arg0.getButton()== MouseEvent.BUTTON2)
84         buttonPressed[MIDDLE_BTN] = false;
85     if (arg0.getButton()== MouseEvent.BUTTON3)
86         buttonPressed[RIGHT_BTN] = false;
87 }
88
89 /** appel  lorsque la souris a un bouton enfonc  et est d plac e */
90 @Override
91 public void mouseDragged(MouseEvent e) {
92     // TODO Auto-generated method stub
93     System.err.println("MouseDragged[" +e.getX()+" ,"+e.getY()+"");
94     mdl.moveDragged(e.getX(), e.getY(), buttonPressed);
95 }
96
97 /** appel  lorsque la souris se d place sans aucun bouton enfonc  */
98 @Override
99 public void mouseMoved(MouseEvent e) {
100     // TODO Auto-generated method stub
101     System.err.println("MouseMoved");
102 //     mdl.moveMouse(e.getX(), e.getY(), buttonPressed);
103 }
104 }
105
106 }

```

La classe g rant le timer correspond   une impl mentation *Java* classique.

exemplesLatex/CoursJOGL2/src/ctrl/AnimationTimer2.java

```

1 /**
2  *
3  */
4 package ctrl;
5
6 import java.util.Timer;
7 import java.util.TimerTask;
8

```

```

9 import mdl.AppMdl2;
10
11 /**
12  * @author remy
13  *
14  */
15
16 /*_____*/
17 /** Classe contenant le timer utilisé pour les animations.
18  * suit le pattern du singleton.
19  */
20
21 public class AnimationTimer2 {
22
23     /** Timer utilisé pour la démo. */
24     Timer timer;
25     /** booléen indiquant si le time est en cours ou a été annulé */
26     boolean isRunning=false;
27
28     /** Référence sur l'unique instance de DemoTimer suivant le pattern du singleton
29     */
30     public static AnimationTimer2 instance=null;
31
32     /*_____*/
33     /** Constructeur créant un timer (qui n'a aucune tâche programmée).
34     */
35     private AnimationTimer2()
36     {
37         timer = new Timer();
38         isRunning=false;
39     }
40
41     /*_____*/
42     /** Programme une exécution de UpdateAnimation.run()
43     * @param milisecons intervalle en milisecondes entre deux appels de
44     * UpdateAnimation.run()
45     * @param mdl modèle à mettre à jour
46     */
47     public void start(long milisecons , AppMdl2 mdl)
48     {
49         if (!isRunning){
50             isRunning=true;
51             timer.schedule(new UpdateAnimation(mdl) , 500, milisecons);
52         }
53
54     /*_____*/
55     /** Annulation des tâches programmées.
56     */
57     public void cancel()
58     {
59         if (isRunning)
60         {
61             isRunning=false;
62             timer.cancel();

```

```
63         timer= new Timer();
64     }
65 }
66
67 /*_____*/
68 /** Retourne l'unique instance de AnimationTimer suivant le pattern du singleton
69
70 * @return l'unique instance de AnimationTimer.
71 */
72 public static AnimationTimer2 getInstance()
73 {
74     if (instance==null)
75         instance = new AnimationTimer2();
76     return instance;
77 }
78
79 /*_____*/
80 /** Classe de t che du timer
81 * @see Timer
82 * @see TimerTask
83 */
84 class UpdateAnimation extends TimerTask {
85     /** Mod le de donn es   mettre   jour lors d'un  v nement timer */
86     private AppMdl2 mdl;
87     /*_____*/
88     /**
89     * @param mdl mod le   mettre   jour
90     */
91     public UpdateAnimation(AppMdl2 mdl)
92     {
93         this.mdl = mdl;
94     }
95     /** fonction ex cut e lors d'un  v nement timer() */
96     public void run() {
97         mdl.stepForward();
98     }
99 }
```

Le timer doit  tre cr e avant le d but de l'animation (voir la classe `AppMdl2`).

# Chapitre 3

## Développement d'un Framework Métier pour la 3D

Nous développons dans l'exemple suivant des classes pour représenter les sources lumineuses, les caméras, les objets 3D, etc. Nous illustrons par un exemple comment utiliser ces classes dans une application avec modèle MVC pour la gestion de la vue et des événements. La mise à jour de la vue est cette fois gérée avec un `animator`.

### 3.1 Transformations géométriques

Pour gérer les transformations géométriques du modèle et de la vue, nous créons tout d'avord une classe de vecteurs et une classe de matrices.

exemplesLatex/CoursJOGL3/src/view3D/Vector3D.java

```
1  /**
2   *
3   */
4  package view3D;
5
6  /**
7   * @author remy
8   * Classe Vector3D. Classe immuable implémentant les vecteurs et points 3D.
9   */
10 public class Vector3D {
11     /** coordonnées catésiennes du vecteur (ou du point) */
12     private float [] coord;
13
14     /*_____*/
15     /** Constructeur pouvant prendre 3 ou 4 coordonnées
16      * @param coord le tableau de coordonnées
17      * @throws IllegalArgumentException si la longueur du tableau n'est ni 3 ni 4
18      */
19     public Vector3D(float [] coord) throws IllegalArgumentException {
20         if (coord.length>4 || coord.length<3)
21             throw new IllegalArgumentException("Erreur, un vecteur doit être
22                 initialisé à partir de 3 ou 4 coordonnées");
23         if (coord.length==3)
24             this.coord=coord.clone();
25         else {
```



```

25     if (Math.abs(coord[3])<1e-10f)
26         throw new IllegalArgumentException("Erreur , la quatri me coordonn e d'un
           vecteur en coordonn es homog nes doit  tre non nulle");
27     this.coord = new float[3];
28     for (int i = 0 ; i < 3 ; i++)
29     {
30         this.coord[i] = coord[i]/coord[3];
31     }
32 }
33 }
34
35 @Override
36 public String toString()
37 {
38     // PENSER   IMPLEMENTER Auto-generated method stub
39     return new String("(" + coord[0] + ", " + coord[1] + ", " + coord[2] + ")");
40 }
41
42 /*_____*/
43 /** Cr e un vecteur nul (0,0,0)
44 */
45 public Vector3D() {
46     // TODO Auto-generated constructor stub
47     coord = new float[3];
48     coord[0]=0; coord[1]=0; coord[2]=0;
49 }
50
51 /*_____*/
52 /** Constructeur   partir de trois coordonn es
53 * @param x
54 * @param y
55 * @param z
56 */
57 public Vector3D(float x, float y, float z) {
58     // TODO Auto-generated constructor stub
59     coord = new float[3];
60     coord[0]=x; coord[1]=y; coord[2]=z;
61 }
62
63 /*_____*/
64 /** convertit le vecteur en coordonn es homog nes
65 * @return les coordonn es homog nes du vecteur dans un tableau
66 */
67 public float[] getHomogeneousCoord(){
68     float [] hc = new float[4];
69     for (int i = 0 ; i < 3 ; i++)
70     {
71         hc[i] = coord[i];
72     }
73     hc[3]=1;
74     return hc;
75 }
76
77 /*_____*/
78 /** retourne une copie (clone) des coordonn es du vecteur
79 * @return la copie des coordonn es du vecteur

```

```

80     */
81     public float [] getCoord(){
82         return coord.clone();
83     }
84
85     /*_____*/
86     /**
87      * @return la première coordonnée du vecteur
88      */
89     public float getX(){
90         return coord[0];
91     }
92
93
94     /*_____*/
95     /**
96      * @return la deuxième coordonnée du vecteur
97      */
98     public float getY(){
99         return coord[1];
100    }
101
102    /*_____*/
103    /**
104     * @return la troisième coordonnée du vecteur
105     */
106    public float getZ(){
107        return coord[2];
108    }
109
110    /*_____*/
111    /** Addition de deux vecteurs
112     * @param v1 vecteur à additionner
113     * @param v2 vecteur à additionner
114     * @return la somme de v1 et v2
115     */
116    public static Vector3D add(Vector3D v1, Vector3D v2){
117        Vector3D result = new Vector3D(v1.getX()+v2.getX(), v1.getY()+v2.getY(), v1.
118            getZ()+v2.getZ());
119        return result;
120    }
121
122    /*_____*/
123    /** Soustraction de deux vecteurs
124     * @param v1 vecteur à additionner
125     * @param v2 vecteur à additionner
126     * @return la différence v1-v2 de v1 et v2
127     */
128    public static Vector3D soustr(Vector3D v1, Vector3D v2){
129        Vector3D result = new Vector3D(v1.getX()-v2.getX(), v1.getY()-v2.getY(), v1.
130            getZ()-v2.getZ());
131        return result;
132    }
133
134    /*_____*/
135    /**

```

```

134     * @return l'oppos  du vecteur
135     */
136     public Vector3D oppose(){
137         Vector3D result = new Vector3D(-getX(), -getY(), getZ());
138         return result;
139     }
140
141     /*_____*/
142     /** multiplication d'un vecteur par un scalaire
143     * @param lambda
144     * @return lambda multipli  par le vecteur
145     */
146     public Vector3D mult(float lambda){
147         Vector3D result = new Vector3D(lambda*getX(), lambda*getY(), lambda*getZ());
148         return result;
149     }
150
151     /*_____*/
152     /** produit scalaire de deux vecteurs
153     * @param v1 un vecteur
154     * @param v2 un vecteur
155     * @return le produit scalaire de v1 et v2
156     */
157     public static float dot(Vector3D v1, Vector3D v2){
158         return v1.getX()*v2.getX()+v1.getY()*v2.getY()+v1.getZ()*v2.getZ();
159     }
160
161     /*_____*/
162     /** produit vectoriel de deux vecteurs
163     * @param v1 un vecteur
164     * @param v2 un vecteur
165     * @return le produit vectoriel de v1 et v2
166     */
167     public static Vector3D cross(Vector3D v1, Vector3D v2){
168         Vector3D result = new Vector3D(v1.getY()*v2.getZ()-v1.getZ()*v2.getY(),
169                                     v1.getZ()*v2.getX()-v1.getX()*v2.getZ(),
170                                     v1.getX()*v2.getY()-v1.getY()*v2.getX());
171         return result;
172     }
173
174     /*_____*/
175     /** permet de modifier une coordonn e du vecteur
176     * @param index indice 0,1 ou 2 de la coordonn e   modifier
177     * @param f nouvelle valeur
178     * @throws ArrayIndexOutOfBoundsException si l'indice n'est pas 0,1 ou 2
179     */
180     public void set(int index, float f) throws ArrayIndexOutOfBoundsException
181     {
182         // PENSER   IMPLEMENTER Auto-generated method stub
183         if (index<0 || index>3)
184             throw new ArrayIndexOutOfBoundsException("Un vecteur 3D n'a que 3
185                 coordonn es.");
186         coord[index]=f;
187     }
188     /*_____*/

```

```

189  /** norme du vecteur
190  * @return la norme du vecteur
191  */
192  public float norm()
193  {
194      // PENSER à IMPLEMENTER Auto-generated method stub
195      return (float)Math.sqrt(getX()*getX()+getY()*getY()+getZ()*getZ());
196  }
197
198  /*_____*/
199  /** Divise un vecteur par sa norme pour le normaliser
200  * @throws ArithmeticException si la norme est nulle (ou proche de 0)
201  */
202  public void normalize() throws ArithmeticException
203  {
204      // PENSER à IMPLEMENTER Auto-generated method stub
205      float norme = norm();
206      if (norme<1e-7)
207          throw new ArithmeticException("La norme doit être non nulle pour pouvoir
208          normaliser un vecteur");
209      coord[0]/=norme; coord[1]/=norme; coord[2]/=norme;
210  }

```

exemplesLatex/CoursJOGL3/src/view3D/Matrix3D.java

```

1  /**
2  *
3  */
4  package view3D;
5
6  import java.nio.FloatBuffer;
7
8  import javax.media.opengl.GL2;
9
10
11 /**
12 * @author remy
13 * Classe matrice3D implémentant les transformations affines
14 * avec les coordonnées homogènes.
15 */
16 public class Matrix3D {
17     /** Coefficients de la matrices rangés en colonnes
18     * colonne0, puis colonne1, puis colonne2, puis colonne3 */
19     private float [] coeffs;
20
21     /*_____*/
22     /** Constructeur initialisant la matrice à l'identité
23     */
24     public Matrix3D() {
25         // TODO Auto-generated constructor stub
26
27         coeffs = new float [16];
28         for (int i=0 ; i<4 ; i++)
29             for (int j=0 ; j<4 ; j++)
30                 {

```

```

31     if (i==j)
32         coeffs[4*j+i] = 1;
33     else
34         coeffs[4*j+i] = 0;
35     }
36 }
37
38 /*_____*/
39 /** Constructeur   partir des coefficients dans un tableau
40  * utilise la r f rence aux coefficients sans faire de copie
41  * @param mat
42  * @throws IllegalArgumentException
43  */
44 public Matrix3D(float [] mat) throws IllegalArgumentException{
45     if (mat.length!=16)
46         throw new IllegalArgumentException("Erreur , coefficients de la matrice
47         mal allou s , 16  l ments demand s");
48     coeffs=mat;
49 }
50 /*_____*/
51 /** Inverse la transformation en supposant que c'est une isom trie
52  * (l'inverse de la partie lin aire est alors  gale   sa transpos e)
53  * @return la matrice de la transformation inverse.
54  */
55 public Matrix3D inverseIsometry(){
56     float [] linearPart = coeffs.clone();
57     linearPart[12]=0;
58     linearPart[13]=0;
59     linearPart[14]=0;
60     Matrix3D invLinearPart = new Matrix3D(linearPart);
61     invLinearPart.transpose();
62     Matrix3D translInv = Matrix3D.translationMatrix(new Vector3D(-getCoeff
63     (0,3) , -getCoeff(1,3) , -getCoeff(2,3)));
64     Matrix3D result = product(invLinearPart , translInv);
65     System.err.println("Identit  = \n"+product(this , result));
66     return result;
67 }
68 /*_____*/
69 /** Application de la transformation   un point
70  * @param v point   transformer
71  * @return la position du point apr s transformation.
72  */
73 public Vector3D multiply(Vector3D v){
74     float [] w = new float [4];
75     float [] vec = v.getHomogeneousCoord();
76     for (int i=0 ; i<4 ; i++)
77     {
78         w[i] = getCoeff(i, 0)*vec[0]+getCoeff(i, 1)*vec[1]+getCoeff(i, 2)*vec
79         [2]+getCoeff(i, 3)*vec[3];
80     }
81     return new Vector3D(w);
82 }
83 /*_____*/

```

```

84  /** Construit la matrice de rotation autour d'un axe passant par l'origine
85  * et d'un certain angle.
86  * @param axis vecteur directeur de l'axe de rotation
87  * @param angle angle de la rotation en degrés.
88  * @return la matrice de rotation
89  */
90  public static Matrix3D rotationMatrix(Vector3D axis, float angle){
91      double radians = angle*Math.PI/180.0;
92      float [] coeffsRotZ = {(float)Math.cos(radians), (float)Math.sin(radians), 0,
93                             0,
94                             -(float)Math.sin(radians), (float)Math.cos(radians), 0, 0,
95                             0, 0, 1, 0,
96                             0, 0, 0, 1};
97
98      Vector3D nonParallele = new Vector3D(0,0,1);
99      Vector3D v = Vector3D.cross(axis, nonParallele);
100     if (v.norm()<1e-5)
101     {
102         nonParallele = new Vector3D(0,1,0);
103         v = Vector3D.cross(axis, nonParallele);
104     }
105     if (v.norm()<1e-5)
106     {
107         nonParallele = new Vector3D(1,0,0);
108         v = Vector3D.cross(axis, nonParallele);
109     }
110     Vector3D w = Vector3D.cross(axis, v);
111     float [] changRepere = {
112         v.getX(), v.getY(), v.getZ(), 0,
113         w.getX(), w.getY(), w.getZ(), 0,
114         axis.getX(), axis.getY(), axis.getZ(), 0,
115         0, 0, 0, 1
116     };
117     Matrix3D transpose = new Matrix3D(changRepere.clone());
118     transpose.transpose();
119     return product(product(new Matrix3D(changRepere), new Matrix3D(coeffsRotZ)),
120                    transpose);
121 }
122
123  /** _____ */
124  /** Construit la matrice de translation d'un certain vecteur.
125  * @param v vecteur de translation
126  * @return la matrice de la translation.
127  */
128  public static Matrix3D translationMatrix(Vector3D v){
129      float [] coeffsTrans = new float [16];
130      coeffsTrans [0]=1; coeffsTrans [4]=0; coeffsTrans [8]=0; coeffsTrans [12]=v.getX();
131      coeffsTrans [1]=0; coeffsTrans [5]=1; coeffsTrans [9]=0; coeffsTrans [13]=v.getY();
132      coeffsTrans [2]=0; coeffsTrans [6]=0; coeffsTrans [10]=1; coeffsTrans [14]=v.getZ();
133      ;
134      coeffsTrans [3]=0; coeffsTrans [7]=0; coeffsTrans [11]=0; coeffsTrans [15]=1;
135
136      return new Matrix3D(coeffsTrans);
137  }
138  /** _____ */

```

```

137  /** transpose la matrice sans faire de copie.
138  * Faire un clone en cas de besoin.
139  */
140  public void transpose()
141  {
142  // PENSER   IMPLEMENTER Auto-generated method stub
143  float [] newCoeffs = new float [16];
144  for (int i = 0 ; i<4 ; i++)
145  for (int j = 0 ; j < 4 ; j++){
146  newCoeffs [4*j+i] = coeffs [4*i+j];
147  }
148  coeffs=newCoeffs;
149  }
150
151  /*_____*/
152  /** Construit la matrice d'un changement d' chelle sur les axes.
153  * @param v vecteur donnant les trois facteurs sur chaque axe
154  * @return la matrice du changement d' chelle.
155  */
156  public static Matrix3D scaleMatrix(Vector3D v){
157  float [] coeffsScal = {v.getX(),0,0,0,
158  0, v.getY(),0, 0,
159  0, 0, v.getZ(), 0,
160  0, 0, 0, 1};
161  return new Matrix3D(coeffsScal);
162  }
163
164  /*_____*/
165  /** Calcul du produit de matrices (compos e des transformations)
166  * @param m1 une matrice
167  * @param m2 une matrice
168  * @return la matrice produit de m1 par m2.
169  */
170  public static Matrix3D product(Matrix3D m1, Matrix3D m2){
171  float [] coeffsRes = new float [16];
172  for (int i=0 ; i<4 ; i++)
173  for (int j=0 ; j<4 ; j++){
174  coeffsRes [4*j+i]=0f;
175
176  for (int k=0 ; k<4 ; k++)
177  coeffsRes [4*j+i] += m1.getCoeff(i,k)*m2.getCoeff(k,j);
178  }
179  return new Matrix3D(coeffsRes);
180  }
181
182  /*_____*/
183  /** permet d'acc der   un coefficient de la matrice
184  * @param i indice de ligne
185  * @param j indice de colonne
186  * @return le coefficient ligne i colonne j de la matrice.
187  */
188  public float getCoeff(int i, int j)
189  {
190  // PENSER   IMPLEMENTER Auto-generated method stub
191  return coeffs [4*j+i];
192  }

```

```

193
194 /*_____*/
195 /** permet de charger la matrice dans le GL_MODELVIEW ou le GL_PROJECTION
196 * @param my_gl contexte JOGL
197 * @param matrix_mode GL_MODELVIEW ou GL_PROJECTION
198 * @throws IllegalArgumentException si le mode est incorrect.
199 */
200 public void load(GL2 gl, int matrix_mode) throws IllegalArgumentException{
201     if (matrix_mode != GL2.GL_MODELVIEW && matrix_mode!= GL2.GL_PROJECTION){
202         throw new IllegalArgumentException("Erreur, le mode doit être GL_MODELVIEW
203             ou GL_PROJECTION");
204     }
205     gl.glMatrixMode(matrix_mode);
206     gl.glLoadIdentity();
207     gl.glLoadMatrixf(coeffs,0);
208 }
209 /*_____*/
210 /** permet de multiplier GL_MODELVIEW ou le GL_PROJECTION (à droite) par la
211     matrice.
212 * @param my_gl contexte JOGL
213 * @param matrix_mode GL_MODELVIEW ou GL_PROJECTION
214 * @throws IllegalArgumentException si le mode est incorrect.
215 */
216 public void multiply(GL2 gl, int matrix_mode) throws IllegalArgumentException{
217     if (matrix_mode != GL2.GL_MODELVIEW && matrix_mode!= GL2.GL_PROJECTION){
218         throw new IllegalArgumentException("Erreur, le mode doit être GL_MODELVIEW
219             ou GL_PROJECTION");
220     }
221     gl.glMatrixMode(matrix_mode);
222     gl.glMultMatrixf(FloatBuffer.wrap(coeffs));
223 }
224 /*_____*/
225 /** Permet de récupérer la matrice GL_MODELVIEW courante dans une Matrix3D
226 * @param my_gl
227 * @return la matrice GL_MODELVIEW courante au format Matrix3D
228 * @throws IllegalArgumentException
229 */
230 public static Matrix3D retrieveModelView(GL2 gl) throws
231     IllegalArgumentException{
232     float [] mat = new float [16];
233     gl.glGetFloatv(GL2.GL_MODELVIEW_MATRIX, mat, 0);
234     return new Matrix3D(mat);
235 }
236 /*_____*/
237 /** Permet de récupérer la matrice GL_PROJECTION courante dans une Matrix3D
238 * @param my_gl
239 * @return la matrice GL_PROJECTION courante au format Matrix3D
240 * @throws IllegalArgumentException
241 */
242 public static Matrix3D retrieveProjection(GL2 gl) throws
243     IllegalArgumentException{
244     float [] mat = new float [16];

```



```

244     gl.glGetFloatv(GL2.GL_PROJECTION_MATRIX, mat, 0);
245     return new Matrix3D(mat);
246 }
247
248 /*_____*/
249 /** Permet d'éviter que la composée d'isométrie ne dérive
250  * et ne s'éloigne d'une isométrie par les erreurs d'approximation folttantes
251  * en calculant une isométrie proche de la matrice.
252  */
253 public void normalizeIsometry(){
254     Vector3D line1 = new Vector3D(coeffs[0], coeffs[1], coeffs[2]);
255     Vector3D line2 = new Vector3D(coeffs[4], coeffs[5], coeffs[6]);
256     Vector3D line3;// = new Vector3D(coeffs[8], coeffs[9], coeffs[10]);
257     line1.normalize();
258     line3=Vector3D.cross(line1, line2);
259     line3.normalize();
260     line2 = Vector3D.cross(line3, line1);
261     line2.normalize();
262     coeffs[0]=line1.getX(); coeffs[1]=line1.getY(); coeffs[2]=line1.getZ();
263     coeffs[4]=line2.getX(); coeffs[5]=line2.getY(); coeffs[6]=line2.getZ();
264     coeffs[8]=line3.getX(); coeffs[9]=line3.getY(); coeffs[10]=line3.getZ();
265 }
266
267 @Override
268 public String toString()
269 {
270     // PENSER à IMPLEMENTER Auto-generated method stub
271     StringBuffer buf = new StringBuffer();
272     for (int i=0 ; i<4 ; i++){
273         buf.append("|");
274         for (int j=0 ; j<4 ; j++){
275             buf.append(" "+getCoeff(i, j)+" ");
276         }
277         buf.append("\n");
278     }
279     return buf.toString();
280 }
281 }

```

## 3.2 Hiérarchie d'objets 3D

Un pattern Stratégie ou un pattern Composite s'impose pour représenter les objets 3D.

exemplesLatex/CoursJOGL3/src/geometry/Objet3D.java

```

1 /**
2  *
3  */
4 package geometry;
5
6 import javax.media.opengl.GL2;
7 import javax.media.opengl.GLContext;
8
9 import shading.Material;
10 import view3D.Matrix3D;

```

```

11 import view3D.Vector3D;
12
13 /**
14  * @author remy
15  *
16  */
17 public class Objet3D {
18     /** Transformation du modèle appliquée à l'objet juste avant affichage */
19     Matrix3D modelTransform;
20     /** matériau de l'objet (ambient, diffus, spéculaire, éventuelle texture...)
21         */
22     Material material;
23
24     /**
25      * Constructeur de l'objet avec transformations de base et matériau
26      * @param axisRotation Axe de rotation de l'objet sur lui-même
27      * @param angle angle de rotation de l'objet sur lui-même avant translation
28      * @param vScale changement d'échelle de l'objet avant rotation
29      * @param vTranslate translation de l'objet après rotation
30      * @param matAmbient paramètre du matériau
31      * @param matDiffuse paramètre du matériau
32      * @param matSpecular paramètre du matériau
33      * @param matShininess paramètre du matériau
34      * @param pathToTexture chemin vers la texture. peut être null ou "" si pas de
35      * texture
36      * @param gl GL courant
37      */
38     Objet3D(Vector3D axisRotation, float angle, Vector3D vScale, Vector3D
39         vTranslate, float [] matAmbient, float [] matDiffuse,
40         float [] matSpecular, float matShininess, String pathToTexture){
41         modelTransform = new Matrix3D();
42         translate(vTranslate);
43         rotate(axisRotation, angle);
44         scale(vScale);
45         material = new Material(matAmbient, matDiffuse, matSpecular, matShininess,
46             pathToTexture);
47     }
48
49     /**
50      * Constructeur de l'objet avec transformations de base, matériau par défaut
51      * @param axisRotation Axe de rotation de l'objet sur lui-même
52      * @param angle angle de rotation de l'objet sur lui-même avant translation
53      * @param vScale changement d'échelle de l'objet avant rotation
54      * @param vTranslate translation de l'objet après rotation
55      * @param pathToTexture chemin vers la texture. peut être null ou "" si pas de
56      * texture
57      * @param gl GL courant
58      */
59     protected Objet3D(Vector3D axisRotation, float angle, Vector3D vScale,
60         Vector3D vTranslate, String pathToTexture){
61         modelTransform = new Matrix3D();
62         translate(vTranslate);
63         rotate(axisRotation, angle);
64         scale(vScale);
65         material = new Material(pathToTexture);
66     }
67 }
68
69 /**
70  */

```

```

61  /** Applique une rotation sur la matrice modelTransform.
62  * Multiplication   droite : la rotation est appliqu e en premier
63  * @param axis axe de rotation
64  * @param angle angle de rotation
65  */
66  public void rotate(Vector3D axis, float angle){
67      modelTransform = Matrix3D.product(modelTransform, Matrix3D.rotationMatrix(
        axis, angle));
68  }
69
70  /*_____*/
71  /** Applique une translation sur la matrice modelTransform.
72  * Multiplication   droite : la translation est appliqu e en premier
73  * @param v vecteur de translation
74  */
75  public void translate(Vector3D v){
76      modelTransform = Matrix3D.product(modelTransform, Matrix3D.translationMatrix
        (v));
77  }
78
79  /*_____*/
80  /** Applique un changement d' chelle sur la matrice modelTransform.
81  * Multiplication   droite : le changement d' chelle est appliqu  en premier
82  * @param v trois coeffs de changement d' chelle sur les trois axes
83  */
84  public void scale(Vector3D v){
85      modelTransform = Matrix3D.product(modelTransform, Matrix3D.scaleMatrix(v));
86  }
87
88  /*_____*/
89  /** Applique une transformation affine sur la matrice modelTransform.
90  * Multiplication   droite : la transformation affine en question est
        appliqu e en premier
91  * @param mat matrice en coordonn es homog nes de la transformation
92  */
93  public void transform(Matrix3D mat){
94      modelTransform = Matrix3D.product(modelTransform, mat);
95  }
96
97  /*_____*/
98  /** M thode d'affichage. A rappeler dans les classes d riv es
99  * applique la transformation modelTransform et s lectionne le mat riau
100  * @param gl
101  */
102  public void display(){
103      GL2 gl = GLContext.getCurrentGL().getGL2();
104      modelTransform.multiply(gl, GL2.GL_MODELVIEW);
105      material.select();
106  }
107  }

```

exemplesLatex/CoursJOGL3/src/geometry/Quad.java

```

1  /*_____*/
2  /**
3  * Fichier : Quad.java

```

```

4  *
5  * créé le 22.02.2011 à 14:40:55
6  *
7  * Auteur : Remy Malgouyres
8  */
9  package geometry;
10
11 import java.nio.FloatBuffer;
12
13 import javax.media.opengl.GL2;
14 import javax.media.opengl.GLContext;
15
16 import view3D.Vector3D;
17
18 /*_____*/
19 /** Classe quadrilataire planeaire
20  */
21 public class Quad extends Objet3D
22 {
23     /** Sommets du quadrilataire */
24     Vector3D [] vertices = new Vector3D[4];
25     /** normale du quadrilataire */
26     Vector3D normale;
27
28
29     /*_____*/
30     /**
31      * @param vertex1 sommet du quadrilataire
32      * @param vertex2 sommet du quadrilataire
33      * @param vertex3 sommet du quadrilataire
34      * @param vertex4 sommet du quadrilataire
35      * @param axisRotation paramètres de Objet3D
36      * @param angle
37      * @param vScale
38      * @param vTranslate
39      * @param matAmbient
40      * @param matDiffuse
41      * @param matSpecular
42      * @param matShininess
43      * @param pathToTexture
44      * @param gl contexte JOGL
45      */
46     public Quad(Vector3D vertex1, Vector3D vertex2, Vector3D vertex3, Vector3D
         vertex4,
47         Vector3D axisRotation, float angle, Vector3D vScale,
48         Vector3D vTranslate, float [] matAmbient, float [] matDiffuse,
49         float [] matSpecular, float matShininess, String pathToTexture)
50     {
51         super(axisRotation, angle, vScale, vTranslate, matAmbient, matDiffuse,
52             matSpecular, matShininess, pathToTexture);
53         // PENSER à IMPLEMENTER Auto-generated constructor stub
54
55         vertices[0]=vertex1;
56         vertices[1]=vertex2;
57         vertices[2]=vertex3;
58         vertices[3]=vertex4;

```

```

59
60     normale = Vector3D.cross(Vector3D.soustr(vertex2, vertex1), Vector3D.soustr(
        vertex3, vertex1));
61     if (Vector3D.dot(Vector3D.soustr(vertex4, vertex1), normale)>1e-5){
62         System.err.println("Warning: le quad est non planaire!");
63     }
64     normale.normalize();
65
66 }
67
68
69 /** Méthode d'affichage */
70 @Override
71 public void display()
72 {
73     // PENSER à IMPLEMENTER Auto-generated method stub
74     GL2 gl = GLContext.getCurrentGL().getGL2();
75
76     gl.glPushMatrix();
77     super.display();
78     gl.glBegin(GL2.GL_QUADS);
79     for (int i=0 ; i<4 ; i++){
80         gl.glTexCoord2f((i/2)%2, ((i+1)/2)%2);
81         gl.glNormal3f(normale.getX(), normale.getY(), normale.getZ());
82         gl.glVertex3fv(FloatBuffer.wrap(vertices[i].getCoord()));
83     }
84     gl.glEnd();
85     gl.glPopMatrix();
86 }
87
88 }
89
90 /*_____*/
91 /* Fin du fichier Quad.java
92 /*_____*/

```

exemplesLatex/CoursJOGL3/src/geometry/Boite.java

```

1 /*_____*/
2 /**
3  * Fichier : Boite.java
4  *
5  * créé le 22.02.2011 à 15:44:57
6  *
7  * Auteur : Remy Malgouyres
8  */
9 package geometry;
10
11 import javax.media.opengl.GL2;
12 import javax.media.opengl.GLContext;
13
14 import view3D.Matrix3D;
15 import view3D.Vector3D;
16
17 /*_____*/
18 /**

```

```

19  */
20  public class Boite extends Objet3D
21  {
22      /** Xmin wall of the box */
23      Quad quadXmin;
24      /** Xmax wall of the box */
25      Quad quadXmax;
26      /** Ymin wall of the box */
27      Quad quadYmin;
28      /** Ymax wall of the box */
29      Quad quadYmax;
30      /** Zmin wall of the box */
31      Quad quadZmin;
32      /** Zmax wall of the box */
33      Quad quadZmax;
34
35      /*_____*/
36      /** Construit une boîte
37       * @param xmin limites de la boîte
38       * @param xmax limites de la boîte
39       * @param ymin limites de la boîte
40       * @param ymax limites de la boîte
41       * @param zmin limites de la boîte
42       * @param zmax limites de la boîte
43       * @param axisRotation paramètres de Objet3D
44       * @param angle
45       * @param vScale
46       * @param vTranslate
47       * @param matAmbient
48       * @param matDiffuse
49       * @param matSpecular
50       * @param matShininess
51       * @param pathToTexture
52       * @param gl JOGL context
53       */
54      public Boite(float xmin, float xmax, float ymin, float ymax, float zmin, float
          zmax,
55                  Vector3D axisRotation, float angle, Vector3D vScale,
56                  Vector3D vTranslate, float [] matAmbient, float [] matDiffuse,
57                  float [] matSpecular, float matShininess, String pathToTexture)
58      {
59          super(axisRotation, angle, vScale, vTranslate, matAmbient, matDiffuse,
60              matSpecular, matShininess, pathToTexture);
61          modelTransform = new Matrix3D();
62          // PENSER à IMPLEMENTER Auto-generated constructor stub
63          quadXmin = new Quad(new Vector3D(xmin, ymin, zmin), new Vector3D(xmin, ymin,
          zmax), new Vector3D(xmin, ymax, zmax), new Vector3D(xmin, ymax, zmin),
64              axisRotation, angle, vScale, vTranslate, matAmbient, matDiffuse,
65              matSpecular, matShininess, pathToTexture);
66          quadXmax = new Quad(new Vector3D(xmax, ymin, zmin), new Vector3D(xmax, ymax,
          zmin), new Vector3D(xmax, ymax, zmax), new Vector3D(xmax, ymin, zmax),
67              axisRotation, angle, vScale, vTranslate, matAmbient, matDiffuse,
68              matSpecular, matShininess, pathToTexture);
69          quadYmin = new Quad(new Vector3D(xmin, ymin, zmin), new Vector3D(xmax, ymin,
          zmin), new Vector3D(xmax, ymin, zmax), new Vector3D(xmin, ymin, zmax),
70              axisRotation, angle, vScale, vTranslate, matAmbient, matDiffuse,

```

```

71     matSpecular, matShininess, pathToTexture);
72     quadYmax = new Quad(new Vector3D(xmin, ymax, zmin), new Vector3D(xmin, ymax,
73         zmax), new Vector3D(xmax, ymax, zmax), new Vector3D(xmax, ymax, zmin),
74         axisRotation, angle, vScale, vTranslate, matAmbient, matDiffuse,
75         matSpecular, matShininess, pathToTexture);
76     quadZmin = new Quad(new Vector3D(xmin, ymin, zmin), new Vector3D(xmin, ymax,
77         zmin), new Vector3D(xmax, ymax, zmin), new Vector3D(xmax, ymin, zmin),
78         axisRotation, angle, vScale, vTranslate, matAmbient, matDiffuse,
79         matSpecular, matShininess, pathToTexture);
80     quadZmax = new Quad(new Vector3D(xmin, ymin, zmax), new Vector3D(xmax, ymin,
81         zmax), new Vector3D(xmax, ymax, zmax), new Vector3D(xmin, ymax, zmax),
82         axisRotation, angle, vScale, vTranslate, matAmbient, matDiffuse,
83         matSpecular, matShininess, pathToTexture);
84 }
85
86 /** Méthode d'affichage */
87 @Override
88 public void display()
89 {
90     // PENSER à IMPLEMENTER Auto-generated method stub
91     GL2 gl = GLContext.getCurrentGL().getGL2();
92
93     gl.glPushMatrix();
94     super.display();
95     quadXmin.display();
96     quadXmax.display();
97     quadYmin.display();
98     quadYmax.display();
99     quadZmin.display();
100    quadZmax.display();
101    gl.glPopMatrix();
102 }
103 }
104
105 /* _____ */
106 /* Fin du fichier Boite.java */
107 /* _____ */

```

exemplesLatex/CoursJOGL3/src/geometry/Quadrique.java

```

1  /**
2   *
3   */
4  package geometry;
5
6  import javax.media.opengl.glu.GLUquadric;
7  import javax.media.opengl.glu.gl2es1.*;
8
9
10 import view3D.Vector3D;
11
12 /**
13  * @author remy
14  * Une quadrique est un objet 3D.
15  */

```

```

16 public class Quadrique extends Objet3D {
17
18     /** Objet quadrique OpenGL */
19     GLUquadric qobj=null;
20
21     /*_____*/
22     /** Constructeur de quadrique (avec les paramètres de l'objet 3D)
23      * @param axisRotation Axe de rotation de l'objet sur lui-même
24      * @param angle angle de rotation de l'objet sur lui-même avant translation
25      * @param vScale changement d'échelle de l'objet avant rotation
26      * @param vTranslate translation de l'objet après rotation
27      * @param matAmbient paramètre du matériau
28      * @param matDiffuse paramètre du matériau
29      * @param matSpecular paramètre du matériau
30      * @param matShininess paramètre du matériau
31      * @param pathToTexture chemin vers la texture. peut être null ou "" si pas de
32         texture
33      * @param gl GL courant
34     */
35     Quadrique(Vector3D axisRotation, float angle, Vector3D vScale,
36              Vector3D vTranslate, float [] matAmbient, float [] matDiffuse,
37              float [] matSpecular, float matShininess, String pathToTexture) {
38         super(axisRotation, angle, vScale, vTranslate, matAmbient, matDiffuse,
39              matSpecular, matShininess, pathToTexture);
40         // TODO Auto-generated constructor stub
41         GLUgl2es1 glu = new GLUgl2es1();
42
43         qobj = glu.gluNewQuadric();
44         glu.gluQuadricDrawStyle(qobj, GLUgl2es1.GLU_FILL); /* smooth shaded */
45         glu.gluQuadricNormals(qobj, GLUgl2es1.GLU_SMOOTH);
46         if (material.isTextured())
47             glu.gluQuadricTexture(qobj, true);
48     }
49     /*_____*/
50     /** Constructeur de quadrique (avec les paramètres de l'objet 3D)
51      * @param axisRotation Axe de rotation de l'objet sur lui-même
52      * @param angle angle de rotation de l'objet sur lui-même avant translation
53      * @param vScale changement d'échelle de l'objet avant rotation
54      * @param vTranslate translation de l'objet après rotation
55      * @param pathToTexture chemin vers la texture. peut être null ou "" si pas de
56         texture
57      * @param gl GL courant
58     */
59     Quadrique(Vector3D axisRotation, float angle, Vector3D vScale,
60              Vector3D vTranslate, String pathToTexture) {
61         super(axisRotation, angle, vScale, vTranslate, pathToTexture);
62         // TODO Auto-generated constructor stub
63         GLUgl2es1 glu = new GLUgl2es1();
64
65         qobj = glu.gluNewQuadric();
66         glu.gluQuadricDrawStyle(qobj, GLUgl2es1.GLU_FILL); /* smooth shaded */
67         glu.gluQuadricNormals(qobj, GLUgl2es1.GLU_SMOOTH);
68         if (material.isTextured())
69             glu.gluQuadricTexture(qobj, true);
70     }
71 }

```



```

70  /** M thode d'affichage (appelle simplement la m thode d'Objet3D) */
71  @Override
72  public void display() {
73      // TODO Auto-generated method stub
74      super.display();
75  }
76
77
78  }

```

exemplesLatex/CoursJOGL3/src/geometry/Sphere.java

```

1  package geometry;
2
3  import javax.media.opengl.GL2;
4  import javax.media.opengl.GLContext;
5  import javax.media.opengl.glu.gl2es1.*;
6
7  import view3D.Vector3D;
8
9  /*_____*/
10 /** Une sph re est une quadrique.
11  */
12 public class Sphere extends Quadrique {
13     /** Nombre de parall las dans le maillage */
14     private int parallels;
15     /** Nombre de m ridiens dans le maillage */
16     private int meridians;
17     /** rayon de la sph re */
18     private float radius;
19
20     /*_____*/
21     /** Constructeur de sph re avec les param tres d'objet3D
22     * @param radius rayon de la sph re
23     * @param parallels Nombre de parall las dans le maillage
24     * @param meridians Nombre de m ridiens dans le maillage
25     * @param axisRotation Axe de rotation de l'objet sur lui-m me
26     * @param angle angle de rotation de l'objet sur lui-m me avant translation
27     * @param vScale changement d' chelle de l'objet avant rotation
28     * @param vTranslate translation de l'objet apr s rotation
29     * @param matAmbient param tre du mat riau
30     * @param matDiffuse param tre du mat riau
31     * @param matSpecular param tre du mat riau
32     * @param matShininess param tre du mat riau
33     * @param pathToTexture chemin vers la texture. peut  tre null ou "" si pas de
34     *   texture
35     * @param gl GL courant
36     * @throws IllegalArgumentException
37     */
38     public Sphere(float radius, int parallels, int meridians, Vector3D
39     axisRotation, float angle, Vector3D vScale,
40     Vector3D vTranslate, float [] matAmbient, float [] matDiffuse,
41     float [] matSpecular, float matShininess, String pathToTexture) throws
42     IllegalArgumentException {
43         super(axisRotation, angle, vScale, vTranslate, matAmbient, matDiffuse,
44             matSpecular, matShininess, pathToTexture);

```

```

42     // TODO Auto-generated constructor stub
43
44     setParallels(parallels);
45     setMeridians(meridians);
46     setRadius(radius);
47 }
48
49 /*_____*/
50 /** Constructeur de sphère avec les paramètres d'objet3D
51  * @param radius rayon de la sphère
52  * @param parallels Nombre de parallèles dans le maillage
53  * @param meridians Nombre de méridiens dans le maillage
54  * @param axisRotation Axe de rotation de l'objet sur lui-même
55  * @param angle angle de rotation de l'objet sur lui-même avant translation
56  * @param vScale changement d'échelle de l'objet avant rotation
57  * @param vTranslate translation de l'objet après rotation
58  * @param pathToTexture chemin vers la texture. peut être null ou "" si pas de
59     texture
60  * @param gl GL courant
61  * @throws IllegalArgumentException
62  */ public Sphere(float radius, int parallels, int meridians, Vector3D
63     axisRotation, float angle, Vector3D vScale,
64     Vector3D vTranslate, String pathToTexture) throws IllegalArgumentException
65     {
66     super(axisRotation, angle, vScale, vTranslate, pathToTexture);
67     // TODO Auto-generated constructor stub
68
69     setParallels(parallels);
70     setMeridians(meridians);
71     setRadius(radius);
72 }
73
74 /*_____*/
75 /** Setter
76  * @param parallels
77  * @throws IllegalArgumentException
78  */
79 public void setParallels(int parallels) throws IllegalArgumentException{
80     if (parallels < 3)
81         throw new IllegalArgumentException();
82     this.parallels = parallels;
83 }
84
85 /*_____*/
86 /** Setter
87  * @param meridians
88  * @throws IllegalArgumentException
89  */
90 public void setMeridians(int meridians) throws IllegalArgumentException{
91     if (meridians < 3)
92         throw new IllegalArgumentException();
93     this.meridians = meridians;
94 }
95
96 /*_____*/
97 /**

```

```

95     * @param radius
96     * @throws IllegalArgumentException
97     */
98     public void setRadius(float radius) throws IllegalArgumentException{
99         if (radius<0)
100             throw new IllegalArgumentException();
101         this.radius = radius;
102     }
103
104     /** M thode d'affichage. Appelle super.display pour appliquer la
105         transformation et le mat riau */
106     @Override
107     public void display() {
108         // TODO Auto-generated method stub
109         GL2 gl = GLContext.getCurrentGL().getGL2();
110         gl.glPushMatrix();
111         super.display();
112         GLUgl2es1 glu = new GLUgl2es1();
113
114         glu.gluSphere(qobj, radius, meridians, parallels);
115         gl.glPopMatrix();
116     }

```

### 3.3 Gestion de la vue et des cam ras

exemplesLatex/CoursJOGL3/src/view3D/Camera.java

```

1  /**
2  *
3  */
4  package view3D;
5
6
7  import javax.media.opengl.GL2;
8  import javax.media.opengl.GLContext;
9  import javax.media.opengl.glu.gl2es1.*;
10 import app.AppFrm3;
11
12 /**
13  * @author remy
14  * Classe cam ra permettant de g rer la vue (projection , position de l'
15     observateur ...
16  */
17 public abstract class Camera {
18
19     /** Matrice vue contenant le changement de rep re de la cam ra */
20     protected Matrix3D view;
21     /** Matrice projection contenant la projection (perspective , orthogonale , etc
22         ...*/
23     protected Matrix3D projection;
24     /** angle d'ouverture en Y */
25     double angleY;
26     /** rapport hauteur/largeur de la fen tre */

```

```

25 double aspect;
26 /** profondeur du plan de clipping proche */
27 double znear;
28 /** profondeur du plan de clipping éloigné */
29 double zfar;
30
31 /** Accesseur de la matrice vue
32  * @return la matrice vue
33  *
34  */
35 public Matrix3D getView(){
36     return view;
37 }
38
39 /** Accesseur de la matrice projection
40  * @return la matrice projection
41  *
42  */
43 public Matrix3D getProjection(){
44     return projection;
45 }
46 /**
47  * Constructeur de la caméra avec les paramètres analogues à gluPerspective
48  * initialise la projection
49  * @param angleY angle d'ouverture en Y de la caméra
50  * @param aspect rapport h/w de la vue
51  * @param znear profondeur du plan de clipping proche
52  * @param zfar profondeur du plan de clipping éloigné
53  */
54 public Camera(double angleY, double aspect, double znear, double zfar){
55     projection = new Matrix3D();
56     view = new Matrix3D();
57     this.angleY=angleY;
58     this.aspect=aspect;
59     this.znear=znear;
60     this.zfar=zfar;
61     gluPerpective(angleY, aspect, znear, zfar);
62 }
63 /**
64  * Constructeur de la caméra avec les paramètres analogues à gluPerspective et
65     à gluLookAt
66  * initialise la projection et la vue (changement de repère)
67  * @param angleY angle d'ouverture en Y de la caméra
68  * @param aspect rapport h/w de la vue
69  * @param znear profondeur du plan de clipping proche
70  * @param zfar profondeur du plan de clipping éloigné
71  * @param eyex position de l'observateur
72  * @param eyez position de l'observateur
73  * @param centerx point de visée
74  * @param centery point de visée
75  * @param centerz point de visée
76  * @param upx vecteur 3D apparaissant comme vertical après projection
77  * @param upy vecteur 3D apparaissant comme vertical après projection
78  * @param upz vecteur 3D apparaissant comme vertical après projection
79  */

```

```

80 public Camera(double angleY, double aspect, double znear, double zfar, double
      eyex, double eyey, double
81      eyez, double centerx, double centery, double
82      centerz, double upx, double upy, double upz){
83     projection = new Matrix3D();
84     this.angleY=angleY;
85     this.aspect=aspect;
86     this.znear=znear;
87     this.zfar=zfar;
88     view = new Matrix3D();
89     gluPerspective(angleY, aspect, znear, zfar);
90     gluLookAt(eyex, eyey, eyez,
91              centerx, centery, centerz,
92              upx, upy, upz);
93 }
94
95 /*_____*/
96 /**
97  * Constructeur de la cam ra avec les param tres analogues   gluPerspective
98  * initialise la projection et la vue (changement de rep re)
99  * @param angleY
100  * @param aspect
101  * @param znear
102  * @param zfar
103  * @param azimuth
104  * @param elevation
105  * @param distance
106  */
107 public Camera(double angleY, double aspect, double znear, double zfar, float
      azimuth, float elevation, float distance){
108     projection = new Matrix3D();
109     gluPerspective(angleY, aspect, znear, zfar);
110     view = Matrix3D.product(Matrix3D.translationMatrix(new Vector3D(0,0,-
      distance)),Matrix3D.product(Matrix3D.rotationMatrix(new Vector3D(0f,0f
      ,0f), elevation), Matrix3D.rotationMatrix(new Vector3D(0f,1f,0f),
      azimuth)));
111     view = Matrix3D.translationMatrix(new Vector3D(0,0,-distance));
112     this.angleY=angleY;
113     this.aspect=aspect;
114     this.znear=znear;
115     this.zfar=zfar;
116 }
117
118 /*_____*/
119 /** Red finit la projection en perspective
120  * @param angleY angle d'ouverture en Y de la cam ra
121  * @param aspect rapport h/w de la vue
122  * @param znear profondeur du plan de clipping proche
123  * @param zfar profondeur du plan de clipping  loign 
124  */
125 public void gluPerspective(double angleY, double aspect, double znear, double
      zfar){
126     GL2 gl = GLContext.getCurrentGL().getGL2();
127     GLUgl2es1 glu = new GLUgl2es1();
128     gl.glMatrixMode(GL2.GL_PROJECTION);
129     gl.glPushMatrix();

```

```

130     gl.glLoadIdentity();
131     glu.gluPerspective(angleY, aspect, znear, zfar);
132     projection=Matrix3D.retrieveProjection(gl);
133     gl.glPopMatrix();
134 }
135
136 /*_____*/
137 /** Change le repère de la caméra
138  * @param eyex position de l'observateur
139  * @param eyez position de l'observateur
140  * @param centerx point de visée
141  * @param centery point de visée
142  * @param centerz point de visée
143  * @param upx vecteur 3D apparaissant comme vertical après projection
144  * @param upy vecteur 3D apparaissant comme vertical après projection
145  * @param upz vecteur 3D apparaissant comme vertical après projection
146  */
147
148 public void gluLookAt(double eyex, double eyez, double
149     eyez, double centerx, double centery, double
150     centerz, double upx, double upy, double upz){
151     GL2 gl = AppFrm3.getCanvas().getGL().getGL2();
152     GLUgl2es1 glu = new GLUgl2es1();
153     gl.glMatrixMode(GL2.GL_MODELVIEW);
154     gl.glPushMatrix();
155     gl.glLoadIdentity();
156     glu.gluLookAt(eyex, eyez, eyez,
157         centerx, centery, centerz,
158         upx, upy, upz);
159     view=Matrix3D.retrieveModelView(gl);
160     gl.glPopMatrix();
161 }
162
163 /*_____*/
164 /** Charge la projection et la vue de la caméra dans GL_MODELVIEW et
165     GL_PROJECTION
166  * @param gl GL actif
167  */
168 public void select(){
169     GL2 gl = GLContext.getCurrentGL().getGL2();
170     projection.load(gl, GL2.GL_PROJECTION);
171     view.load(gl, GL2.GL_MODELVIEW);
172 }
173
174 /*_____*/
175 /** Charge ma matrice view dans GL_MODELVIEW
176  * @param gl
177  */
178 public void resetView(GL2 gl){
179     view.load(gl, GL2.GL_MODELVIEW);
180 }
181
182 /*_____*/
183 /** méthode virtuelle implémentant la réaction de la caméra au mouvement
184  * de souris avec au moins un bouton enfoncé.

```

```

185 * L'implémentation dépend dumodèle de caméra (Centrale, Navigation,...)
186 * @param deltaX
187 * @param deltaY
188 * @param buttonPressed
189 */
190 public abstract void mouseDragged(float deltaX, float deltaY, boolean []
      buttonPressed);
191
192 }

```

exemplesLatex/CoursJOGL3/src/view3D/CameraCentrale.java

```

1 /*
2 /**
3  * Fichier : CameraNavigation.java
4  *
5  * créé le 22.02.2011 à 16:10:13
6  *
7  * Auteur : Remy Malgouyres
8  */
9 package view3D;
10
11
12 /*
13 /** La caméra centrale est un omdèle de caméra dans lequel
14  * la caméra tourne autour d'un point de focus appelé centre.
15  * La caméra est toujours dirigée vers ce centre
16  */
17 public class CameraCentrale extends Camera
18 {
19     /** Vitesse de translation de la caméra en fonction de la souris */
20     private final float vitesseCamTranslat=0.5f;
21     /** Vitesse de rotation de la caméra en fonction de la souris */
22     private final float vitesseCamRotat=1f;
23
24     /** Centre de la vue suivant le modèle de caméra central */
25     Vector3D centre;
26
27     /*
28     /** Constructeur appelant le constructeur de Camera initialisant
29     * la projection et la vue avec gluPerspective et gluLookAt.
30     * @param angleY
31     * @param aspect
32     * @param znear
33     * @param zfar
34     * @param eyex
35     * @param eyey
36     * @param eyez
37     * @param centerx
38     * @param centery
39     * @param centerz
40     * @param upx
41     * @param upy
42     * @param upz
43     */
44     public CameraCentrale(

```

```

45     double angleY, double aspect, double znear,
46     double zfar, double eyex, double eyey, double eyez, double centerx,
47     double centery, double centerz, double upx, double upy, double upz)
48     {
49     super(angleY, aspect, znear, zfar, eyex, eyey, eyez, centerx, centery,
50         centerz, upx, upy, upz);
51     // PENSER à IMPLEMENTER Auto-generated constructor stub
52     centre = new Vector3D((float)centerx, (float)centery, (float)centerz);
53     }
54
55     /** Implémente la réaction de la caméra aux événements souris */
56     @Override
57     public void mouseDragged(float deltaX, float deltaY, boolean[] buttonPressed)
58     {
59         Matrix3D transfo;
60
61         if (buttonPressed[0]){
62             Vector3D positionCamera = view.inverseIsometry().multiply(new Vector3D
63                 (0,0,0));
64             Vector3D centrePos = Vector3D.soustr(centre, positionCamera);
65             System.err.println("positionCaméra_=" + positionCamera + ", Centre_=" + centre
66                 + ", Centrepos_=" + centrePos + "\n");
67             Matrix3D translat = Matrix3D.translationMatrix(new Vector3D(0,0,centrePos.
68                 norm()));
69             Matrix3D translatInv = Matrix3D.translationMatrix(new Vector3D(0,0,-
70                 centrePos.norm()));
71             transfo = Matrix3D.product(Matrix3D.product(translatInv, Matrix3D.product(
72                 Matrix3D.rotationMatrix(new Vector3D(1,0,0), vitesseCamRotat*deltaY),
73                 Matrix3D.rotationMatrix(new Vector3D(0,1,0), vitesseCamRotat*deltaX)))
74                 ,translat);
75             view = Matrix3D.product(transfo, view);
76         }
77         if (buttonPressed[2]){
78             transfo = Matrix3D.rotationMatrix(new Vector3D(0,0,1), vitesseCamRotat*
79                 deltaX);
80             view = Matrix3D.product(transfo, view);
81         }
82         if (buttonPressed[1]){
83             transfo = Matrix3D.translationMatrix(new Vector3D(0,0,-vitesseCamTranslat*
84                 deltaY));
85             view = Matrix3D.product(transfo, view);
86         }
87         view.normalizeIsometry();
88     }
89 }
90
91 /*_____*/
92 /* Fin du fichier CameraNavigation.java
93 /*_____*/

```

exemplesLatex/CoursJOGL3/src/view3D/CameraNavigation.java

```

1 /*_____*/
2 /**
3  * Fichier : CameraNavigation.java

```



```
4  *
5  * cr e le 22.02.2011   16:10:13
6  *
7  * Auteur : Remy Malgouyres
8  */
9  package view3D;
10
11
12  /*_____*/
13  /** Le mod le de cam ra pour la navigation est tel que les rotations
14  * de la cam ra ont lieu dans le rep re de la cam ra et autour
15  * d'axes passant par la position de la cam ra, ce qui correspond
16  *   un changement de direction d'un avatar. La translation
17  * de la cam ra est le long de l'axe des Z du rep re de la cam ra.
18  */
19  public class CameraNavigation extends Camera
20  {
21
22      /** Vitesse de translation de la cam ra en fonction de la souris */
23      private final float vitesseCamTranslat=0.5f;
24      /** Vitesse de rotation de la cam ra en fonction de la souris */
25      private final float vitesseCamRotat=1f;
26
27      /*_____*/
28      /**
29       * @param angleY
30       * @param aspect
31       * @param znear
32       * @param zfar
33       */
34      public CameraNavigation(double angleY, double aspect, double znear,
35                             double zfar)
36      {
37          super(angleY, aspect, znear, zfar);
38          // PENSER   IMPLEMENTER Auto-generated constructor stub
39      }
40
41      /*_____*/
42      /**
43       * @param angleY
44       * @param aspect
45       * @param znear
46       * @param zfar
47       * @param eyex
48       * @param eyey
49       * @param eyez
50       * @param centerx
51       * @param centery
52       * @param centerz
53       * @param upx
54       * @param upy
55       * @param upz
56       */
57      public CameraNavigation(double angleY, double aspect, double znear,
58                             double zfar, double eyex, double eyey, double eyez, double centerx,
59                             double centery, double centerz, double upx, double upy, double upz)
```

```

60 {
61     super(angleY, aspect, znear, zfar, eyex, eyey, eyez, centerx, centery,
62           centerz, upx, upy, upz);
63     // PENSER à IMPLEMENTER Auto-generated constructor stub
64 }
65
66 /*_____*/
67 /**
68  * @param angleY
69  * @param aspect
70  * @param znear
71  * @param zfar
72  * @param azimuth
73  * @param elevation
74  * @param distance
75  */
76 public CameraNavigation(double angleY, double aspect, double znear,
77                         double zfar, float azimuth, float elevation, float distance)
78 {
79     super(angleY, aspect, znear, zfar, azimuth, elevation, distance);
80     // PENSER à IMPLEMENTER Auto-generated constructor stub
81 }
82
83 /** Implémente la réaction de la caméra aux événements souris */
84 @Override
85 public void mouseDragged(float deltaX, float deltaY, boolean [] buttonPressed)
86 {
87     Matrix3D transfo;
88
89     if (buttonPressed[0]){
90         transfo = Matrix3D.product(
91             Matrix3D.rotationMatrix(new Vector3D(0,1,0), vitesseCamRotat*deltaX),
92             Matrix3D.rotationMatrix(new Vector3D(1,0,0),vitesseCamRotat*deltaY));
93         view = Matrix3D.product(transfo, view);
94     }
95     if (buttonPressed[2]){
96         transfo = Matrix3D.rotationMatrix(new Vector3D(0,0,1), vitesseCamRotat*
97             deltaX);
98         view = Matrix3D.product(transfo, view);
99     }
100    if (buttonPressed[1]){
101        transfo = Matrix3D.translationMatrix(new Vector3D(0,0,-vitesseCamTranslat*
102            deltaY));
103        view = Matrix3D.product(transfo, view);
104    }
105    view.normalizeIsometry();
106 }
107
108
109
110 }
111
112 /*_____*/
113 /* Fin du fichier CameraNavigation.java

```

```
114 /*_____*/
```

### 3.4 Gestion de l' clairnement

exemplesLatex/CoursJOGL3/src/shading/Material.java

```

1 package shading;
2
3
4 import java.awt.image.BufferedImage;
5 import java.awt.image.DataBufferByte;
6 import java.io.File;
7 import java.nio.Buffer;
8 import java.nio.ByteBuffer;
9
10 import javax.imageio.ImageIO;
11 import javax.media.opengl.GL;
12 import javax.media.opengl.GL2;
13 import javax.media.opengl.GLContext;
14
15
16 /*_____*/
17 /** Classe mat riau : permet de stocker le mat riau d'un (ou plusieurs) objet(s)
18 */
19 public class Material {
20     /** coefficient de r flexion de la lumi re ambiante */
21     private float mat_ambient[];
22     /** coefficient de r flexion diffuse */
23     private float mat_diffuse[];
24     /** coefficient de r flexion sp culaire */
25     private float mat_specular[];
26     /** brillance (exposant de la r flexion sp culaire) */
27     private float mat_shininess = 4f;
28
29     /** ID de l' ventuelle texture (null si pas de texture) */
30     int [] texId=null;
31
32     /*_____*/
33     /** Constructeur du mat riau
34     * @param matAmbient coefficient de r flexion de la lumi re ambiante
35     * @param matDiffuse coefficient de r flexion diffuse
36     * @param matSpecular coefficient de r flexion sp culaire
37     * @param matShininess brillance (exposant de la r flexion sp culaire)
38     * @param pathToTexture chemin sur le disque vers l'image de texture
39     * @param gl GL courant
40     */
41     public Material(float [] matAmbient, float [] matDiffuse ,
42         float [] matSpecular, float matShininess, String pathToTexture) {
43         GL2 gl = GLContext.getCurrentGL().getGL2();
44         setMat_ambient(matAmbient);
45         setMat_diffuse(matDiffuse);
46         setMat_specular(matSpecular);
47         setMat_shininess(matShininess);

```

```

48     if (pathToTexture!=null&& pathToTexture.length ()>0){
49         gl.glEnable (GL.GL_TEXTURE_2D);
50         texId = new int [1];
51         gl.glGenTextures (1, texId, 0);
52
53         gl.glBindTexture (GL.GL_TEXTURE_2D, texId [0]);
54         gl.glTexParameteri (GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MIN_FILTER, GL.
55             GL_LINEAR);
56         gl.glTexParameteri (GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MAG_FILTER, GL.
57             GL_LINEAR);
58
59         try {
60             BufferedImage image = ImageIO.read (new File (pathToTexture));
61             ByteBuffer dbb = (ByteBuffer) image.getRaster ().getDataBuffer
62                 ();
63             byte [] data = dbb.getData ();
64             ByteBuffer pixels = ByteBuffer.allocate (data.length);
65             pixels.put (data);
66             Buffer buff = pixels.flip ();
67
68             gl.glTexImage2D (GL2.GL_TEXTURE_2D, 0, GL2.GL_RGB, image.getWidth (),
69                 image.getHeight (), 0, GL2.GL_RGB, GL2.GL_UNSIGNED_BYTE, buff);
70         } catch (Throwable t) {
71             System.err.println ("Le fichier de texture "+pathToTexture+" ne peut être
72                 ouvert!");
73             texId=null;
74         }
75     }
76 }
77
78 /* _____ */
79 /** Constructeur du matériau avec valeurs par défaut
80  * @param pathToTexture chemin sur le disque vers l'image de texture
81  * @param gl GL courant
82  */
83 public Material (String pathToTexture) {
84     // TODO Auto-generated constructor stub
85     GL2 gl = GLContext.getCurrentGL ().getGL2 ();
86
87     setMat_ambient (getQuadruple (0.2f, 0.2f, 0.2f, 1.0f));
88     setMat_diffuse (getQuadruple (0.4f, 0.4f, 0.4f, 1.0f));
89     setMat_specular (getQuadruple (0.3f, 0.3f, 0.3f, 1.0f));
90     setMat_shininess (128.0f);
91     if (pathToTexture!=null&& pathToTexture.length ()>0){
92         gl.glEnable (GL.GL_TEXTURE_2D);
93         texId = new int [1];
94         gl.glGenTextures (1, texId, 0);
95
96         gl.glBindTexture (GL.GL_TEXTURE_2D, texId [0]);
97         gl.glTexParameteri (GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MIN_FILTER, GL.
98             GL_LINEAR);
99         gl.glTexParameteri (GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MAG_FILTER, GL.
100             GL_LINEAR);
101
102         try {

```

```

98  /*      BufferedImage image = ImageIO.read(new File(pathToTexture));
99      DataBufferByte dbb = (DataBufferByte)image.getRaster().getDataBuffer();
100     byte [] data = dbb.getData();
101     ByteBuffer pixels = Buffers.newDirectByteBuffer(data.length);
102     pixels.put(data);
103     pixels.flip();
104     gl.glTexImage2D(GL.GL_TEXTURE_2D, 0, GL.GL_RGB, image.getWidth(), image.
        getHeight(), 0, GL.GL_RGB, GL.GL_UNSIGNED_BYTE, pixels);
105 */      } catch(Throwable t) {
106         texId=null;
107     }
108     }else
109         texId=null;
110 }
111
112 /*_____*/
113 /** Utilitaire permettant de construire un quadruplet   partir de quatre
        flottant
114  * @param a
115  * @param b
116  * @param c
117  * @param d
118  * @return le quadruplet
119  */
120 public static float [] getQuadruple(float a, float b, float c, float d){
121     float [] result = new float [4];
122     result [0] = a; result [1] = b; result [2] = c; result [3] = d;
123     return result;
124 }
125
126 /*_____*/
127 /** retourne vrai si le mat rieau est textur 
128  * @return vrai si le mat rieau est textur 
129  */
130 public boolean isTextured() {
131     // TODO Auto-generated method stub
132     return (texId!=null);
133 }
134
135 /*_____*/
136 /** Setter de coefficient de r flexion de la lumi re ambiante
137  * @param matAmbient
138  * @throws IllegalArgumentException
139  */
140 public void setMat_ambient(float [] matAmbient) throws IllegalArgumentException
        {
141     if (matAmbient.length!=4)
142         throw new IllegalArgumentException("Erreur , le mat rieau_ambient doit avoir
             3 coordonn es");
143
144     this.mat_ambient = matAmbient;
145 }
146
147 /*_____*/
148 /** Setter de coefficient de r flexion diffuse
149  * @param matDiffuse

```

```

150     * @throws IllegalArgumentException
151     */
152     public void setMat_diffuse(float [] matDiffuse) throws IllegalArgumentException
153     {
154         if (matDiffuse.length!=4)
155             throw new IllegalArgumentException("Erreur , le matériau diffus doit avoir
156             3 coordonnées");
157         this.mat_diffuse = matDiffuse;
158     }
159     /* _____ */
160     /** Setter de coefficient de réflexion spéculaire
161     * @param matSpecular
162     * @throws IllegalArgumentException
163     */
164     public void setMat_specular(float [] matSpecular) throws
165     IllegalArgumentException{
166         if (matSpecular.length!=4)
167             throw new IllegalArgumentException("Erreur , le matériau spéculaire doit
168             avoir 3 coordonnées");
169         this.mat_specular = matSpecular;
170     }
171     /* _____ */
172     /** Setter de brillance
173     * @param matShininess
174     */
175     public void setMat_shininess(float matShininess) {
176         this.mat_shininess = matShininess;
177     }
178     /* _____ */
179     /** Sélectionne le matériau comme matériau courant
180     * @param gl
181     */
182     public void select(){
183         GL2 gl = GLContext.getCurrentGL().getGL2();
184         if (texId != null){
185             gl.glBindTexture (GL2.GL_TEXTURE_2D, texId [0]);
186             gl.glEnable(GL2.GL_TEXTURE_2D);
187         }else
188             gl.glDisable(GL2.GL_TEXTURE_2D);
189         gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_AMBIENT, mat_ambient, 0);
190         gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_DIFFUSE, mat_diffuse, 0);
191         gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_SPECULAR, mat_specular, 0);
192         gl.glMaterialf(GL2.GL_FRONT, GL2.GL_SHININESS, mat_shininess);
193     }

```

exemplesLatex/CoursJOGL3/src/shading/PointLightSource.java

```

1 /**
2 *
3 */
4 package shading;
5

```

```

6 import javax.media.opengl.GL2;
7 import javax.media.opengl.GLContext;
8
9 import view3D.Matrix3D;
10 import view3D.Vector3D;
11
12
13 /**
14  * @author remy
15  * Classe repr sentant une source lumineuse ponctuelle
16  */
17 public class PointLightSource {
18     /** transformation   appliquer sur la position de la source lumineuse */
19     Matrix3D transform;
20     /** position de la source lumineuse
21      * (subit la transformation ModelView courante et la matrice transform)
22      */
23     Vector3D position;
24
25     /** intensit  diffuse */
26     float [] diffuse;
27     /** intensit  sp culaire */
28     float [] specular;
29
30     /** ID de la source dans OpenGL (GL_LIGHT0, GL_LIGHT1, etc.) */
31     int lightID;
32
33     /** Indique si la source subit une att nuation avec la distance */
34     boolean withAttenuation=false;
35     /** terme d'att nuation constante */
36     float constantAttenuation;
37     /** terme d'att nuation lin aire */
38     float linearAttenuation;
39     /** terme d'att nuation quadratique */
40     float quadraticAttenuation;
41     /*_____*/
42     /** Constructeur initialisant la position et les intensit s
43      * la transformation est initialis e   l'identit 
44      * @param posx position de la source lumineuse
45      * @param posy position de la source lumineuse
46      * @param posz position de la source lumineuse
47      * @param diff intensit  diffuse
48      * @param spec intensit  sp culaire
49      */
50     public PointLightSource(float posx, float posy, float posz,
51                             float [] diff, float [] spec) {
52         // TODO Auto-generated constructor stub
53         position = new Vector3D(posx, posy, posz);
54         diffuse= diff;
55         specular = spec;
56         transform = new Matrix3D();
57     }
58     /*_____*/
59     /** Constructeur initialisant la position et les intensit s
60      * la transformation est initialis e   l'identit 
61      * @param posx position de la source lumineuse

```

```

62     * @param posy position de la source lumineuse
63     * @param posz position de la source lumineuse
64     * @param diff intensité diffuse
65     * @param spec intensité spéculaire
66     * @param constantAttenuation terme d'atténuation constante
67     * @param linearAttenuation terme d'atténuation linéaire
68     * @param quadraticAttenuation terme d'atténuation quadratique
69     */
70     public PointLightSource(float posX, float posy, float posz,
71                             float [] diff, float [] spec,
72                             float constantAttenuation,
73                             float linearAttenuation,
74                             float quadraticAttenuation) {
75         // TODO Auto-generated constructor stub
76         position = new Vector3D(posx, posy, posz);
77         diffuse= diff;
78         specular = spec;
79         transform = new Matrix3D();
80     }
81     /*_____*/
82     /** Active la source lumineuse dans OpenGL sur un ID donné.
83     * @param light_ID ID de la source dans OpenGL (GL_LIGHT0, GL_LIGHT1, etc.)
84     * @param enable dit si l'on active la source lumineuse ou pas
85     * @param gl GL courant
86     */
87     public void activate(int light_ID, boolean enable){
88         GL2 gl = GLContext.getCurrentGL().getGL2();
89
90         setLightID(light_ID);
91         gl.glMatrixMode(GL2.GL_MODELVIEW);
92         gl.glPushMatrix();
93         transform.multiply(gl, GL2.GL_MODELVIEW);
94         gl.glLightfv(light_ID, GL2.GL_POSITION, position.getHomogeneousCoord(),
95                     0);
96         gl.glPopMatrix();
97
98         gl.glLightfv(light_ID, GL2.GL_DIFFUSE, diffuse, 0);
99         gl.glLightfv(light_ID, GL2.GL_SPECULAR, specular, 0);
100        if (withAttenuation)
101        {
102            gl.glLightf(light_ID, GL2.GL_CONSTANT_ATTENUATION, constantAttenuation);
103            gl.glLightf(light_ID, GL2.GL_LINEAR_ATTENUATION, linearAttenuation);
104            gl.glLightf(light_ID, GL2.GL_QUADRATIC_ATTENUATION, quadraticAttenuation);
105        }
106
107        if (enable)
108            gl.glEnable(light_ID);
109        else
110            gl.glDisable(light_ID);
111    }
112    /*_____*/
113    /** Setter de l'ID de la source lumineuse dans OpenGL
114    * @param lightID
115    */

```



```
116 private void setLightID(int lightID){
117     // TODO Auto-generated method stub
118     this.lightID = lightID;
119 }
120
121 /*_____*/
122 /** accesseur de l'ID de la source lumineuse
123  * @return l'ID de la source lumineuse
124  */
125 public int getLightID() {
126     // TODO Auto-generated method stub
127     return lightID;
128 }
129
130 /*_____*/
131 /*_____*/
132 /** Applique une rotation sur la matrice modelTransform.
133  * Multiplication   droite : la rotation est appliqu e en premier
134  * @param axis axe de rotation
135  * @param angle angle de rotation
136  */
137 public void rotate(Vector3D axis, float angle){
138     transform = Matrix3D.product(transform, Matrix3D.rotationMatrix(axis, angle)
139     );
140 }
141
142 /*_____*/
143 /** Applique une translation sur la matrice modelTransform.
144  * Multiplication   droite : la translation est appliqu e en premier
145  * @param v vecteur de translation
146  */
147 public void translate(Vector3D v){
148     transform = Matrix3D.product(transform, Matrix3D.translationMatrix(v));
149 }
150
151 /*_____*/
152 /** Applique un changement d' chelle sur la matrice modelTransform.
153  * Multiplication   droite : le changement d' chelle est appliqu e en premier
154  * @param v trois coeffs de changement d' chelle sur les trois axes
155  */
156 public void scale(Vector3D v){
157     transform = Matrix3D.product(transform, Matrix3D.scaleMatrix(v));
158 }
159
160 /*_____*/
161 /** Applique une transformation affine sur la matrice modelTransform.
162  * Multiplication   droite : la transformation affine en question est
163     appliqu e en premier
164  * @param mat matrice en coordonn es homog enes de la transformation
165  */
166 public void transform(Matrix3D mat){
167     transform = Matrix3D.product(transform, mat);
168 }
169
170 /*_____*/
```

```

169  /** Redéfinit la position pour lui faire subir la transformation ModelView
      courante
170   * @param gl
171   */
172  public void resetPosition() {
173      GL2 gl = GLContext.getCurrentGL().getGL2();
174
175      // TODO Auto-generated method stub
176      gl.glMatrixMode(GL2.GL_MODELVIEW);
177      gl.glPushMatrix();
178      transform.multiply(gl, GL2.GL_MODELVIEW);
179      gl.glLightfv(getLightID(), GL2.GL_POSITION, position.getHomogeneousCoord
      (), 0);
180      gl.glPopMatrix();
181  }
182  }

```

### 3.5 Exemple d'application

Cette fois toutes les données de la scène, y compris les objets et les paramètres de l'animation, sont contenus dans le modèle, conformément au pattern MVC strict.

exemplesLatex/CoursJOGL3/src/ctrl/AnimationTimer3.java

```

1  /**
2   *
3   */
4  package ctrl;
5
6  import java.util.Timer;
7  import java.util.TimerTask;
8
9  import swingView.GLDisplayEx3;
10
11 /**
12  * @author remy
13  *
14  */
15
16 /*_____*/
17 /** Classe contenant le timer utilisé pour les animations.
18  * suit le pattern du singleton.
19  */
20
21 public class AnimationTimer3 {
22
23 /** Timer utilisé pour la démo. */
24 Timer timer;
25 /** booléen indiquant si le time est en cours ou a été annulé */
26 boolean isRunning=false;
27
28 /** Référence sur l'unique instance de DemoTimer suivant le pattern du singleton
29  *
30 public static AnimationTimer3 instance=null;

```

```
31
32 /*_____*/
33 /** Constructeur cr ant un timer (qui n'a aucune t che programm e).
34 */
35 private AnimationTimer3()
36 {
37     timer = new Timer();
38     isRunning=false;
39 }
40
41 /*_____*/
42 /** Programme une ex cution de UpdateAnimation.run()
43 * @param milisecons intervalle en milisecons entre deux appels de
44 * UpdateAnimation.run()
45 * @param glDisplay GLDisplay pour l'affichage
46 */
47 public void start(long milisecons , GLDisplayEx3 glDisplay)
48 {
49     if (!isRunning){
50         isRunning=true;
51         timer.schedule(new UpdateAnimation(glDisplay), 500, milisecons)
52             ;
53     }
54 }
55
56 /*_____*/
57 /** Annulation des t ches programm es.
58 */
59 public void cancel()
60 {
61     if (isRunning)
62     {
63         isRunning=false;
64         timer.cancel();
65         timer= new Timer();
66     }
67 }
68
69 /*_____*/
70 /** Retourne l'unique instance de AnimationTimer suivant le pattern du singleton
71 * @return l'unique instance de AnimationTimer.
72 */
73 public static AnimationTimer3 getInstance()
74 {
75     if (instance==null)
76         instance = new AnimationTimer3();
77     return instance;
78 }
79
80 /*_____*/
81 /** Classe de t che du timer
82 * @see Timer
83 * @see TimerTask
84 */
85 class UpdateAnimation extends TimerTask {
```

```

84 /** Modèle de données à mettre à jour lors d'un événement timer */
85 private GLDisplayEx3 glDisplay;
86 /**
87 *
88 * @param glDisplay GLDisplay pour l'affichage
89 */
90 public UpdateAnimation(GLDisplayEx3 glDisplay)
91 {
92     this.glDisplay = glDisplay;
93 }
94 /** fonction exécutée lors d'un événement timer() */
95 public void run() {
96     glDisplay.stepForward();
97 }
98 }
99 }

```

L'animation est cette fois gérée avec un `animator`, une classe spécialisée de JOGL qui rafraîchit la vue à intervalle régulier. L'`animator` doit être démarré avant le début de l'animation.

exemplesLatex/CoursJOGL3/src/app/AppFrm3.java

```

1 /**
2 *
3 */
4 package app;
5
6 import javax.swing.JFrame;
7
8 import javax.media.opengl.awt.GLCanvas;
9 import swingView.GLDisplayEx3;
10
11 import mdl.AppMdl3;
12
13 import com.jogamp.opengl.util.FPSAnimator;
14
15 import ctrl.AnimationTimer3;
16 import ctrl.CtrlMouse3;
17
18 /** Fenêtre principale de l'application
19 * Créé toutes les données et instances de classes de l'application et la vue.
20 */
21 public class AppFrm3 extends JFrame {
22
23     /**
24     * serialVersionUID (sert pour la serialisation, obligatoire pour une JFrame)
25     */
26     private static final long serialVersionUID = 9167791876718956063L;
27
28     /** Canvas, c'est le composant utilisé pour dessiner avec JOGL */
29     private static GLCanvas canvas=null;
30
31
32     /** Animator pour une gestion du rafraîchissement de la vue périodique*/
33     final FPSAnimator animator;
34
35     /**

```

```

36     * @param args non utilis 
37     */
38     public static void main(String [] args) {
39         // TODO Auto-generated method stub
40
41         AppFrm3 frame = new AppFrm3();
42         frame.setVisible(true);
43         frame.startAnimator();
44     }
45
46     /*_____*/
47     /** permet d'acc der au canvas
48     * @return le canvas de la vue
49     */
50     public static GLCanvas getCanvas(){
51         return canvas;
52     }
53     /*_____*/
54     /** Constructeur de la vue. Cr e les instances du mod le, du contr leur,
55         d marre le timer, etc...
56     */
57     public AppFrm3() {
58         // TODO Auto-generated constructor stub
59         canvas = new GLCanvas();
60         setSize(500, 500);
61         AppMdl3 mdl = new AppMdl3();
62         GLDisplayEx3 glDisplayEx3 = new GLDisplayEx3(canvas, mdl);
63         canvas.addGLEventListener(glDisplayEx3);
64         animator = new FPSAnimator(canvas, 30);
65         MyWindowAdapter3 winAdapt = new MyWindowAdapter3(animator);
66         addWindowListener(winAdapt);
67         add(canvas);
68         CtrlMouse3 ctrl = new CtrlMouse3(mdl);
69         canvas.addMouseMotionListener(ctrl);
70         canvas.addMouseListener(ctrl);
71         AnimationTimer3.getInstance().start(15, glDisplayEx3);
72         JFrame.setDefaultLookAndFeelDecorated(true);
73     }
74     /*_____*/
75     /** M thode qui d marre l'animator (appel e si withAnimator==true).
76     */
77     private void startAnimator() {
78         // TODO Auto-generated method stub
79         animator.start();
80     }
81 }

```

L'animator doit  tre arr t    la fin de l'application (ou  ventuellement   la fin de l'animation pour  viter un gaspillage de ressources).

exemplesLatex/CoursJOGL3/src/app/MyWindowAdapter3.java

```

1 package app;
2 /**
3  *
4  */

```

```

5
6
7 import java.awt.event.WindowAdapter;
8 import java.awt.event.WindowEvent;
9
10 import com.jogamp.opengl.util.FPSAnimator;
11
12 /**
13  * @author remy
14  */
15 /**
16  * Classe définissant la réaction de l'application aux événements de fenêtre (
17   * comme la fermeture de la fenêtre)
18  */
19 public class MyWindowAdapter3 extends WindowAdapter {
20
21     /** Référence vers l'animateur de la vue (qui doit être stoppé avant de
22      * terminer l'application) */
23     FPSAnimator animator;
24
25     /** Constructeur initialisant l'animateur
26      * @param animator l'animateur de la vue
27      */
28     public MyWindowAdapter3(FPSAnimator animator) {
29         // TODO Auto-generated constructor stub
30         this.animator = animator;
31     }
32
33     /** Termine l'application après avoir stoppé l'animateur (si gestion avec un
34      * animateur) */
35     @Override
36     public void windowClosing(WindowEvent e) {
37         // TODO Auto-generated method stub
38         new Thread(new Runnable() {
39
40             public void run() {
41                 animator.stop();
42                 System.exit(0);
43             }
44         }).start();
45     }
46 }

```

exemplesLatex/CoursJOGL3/src/ctrl/CtrlMouse3.java

```

1 package ctrl;
2
3 import java.awt.event.MouseEvent;
4 import java.awt.event.MouseListener;
5 import java.awt.event.MouseMotionListener;
6
7 import mdl.AppMdl3;
8
9 /**
10 /** Contrôle de l'application (voir pattern MVC)

```

```

11  * R ceptionne les  v nements souris (AWT) et modifie le mod le en cons quence
12  */
13  public class CtrlMouse3 implements MouseListener, MouseMotionListener {
14
15      /** Mod le contenant les donn es de l'application */
16      private AppMdl3 mdl;
17
18      /** ID des boutons de la souris (indices dans le tableau buttonPressed) */
19      public final static int LEFT_BTN=0;
20      /** ID des boutons de la souris (indices dans le tableau buttonPressed) */
21      public final static int MIDDLE_BTN=1;
22      /** ID des boutons de la souris (indices dans le tableau buttonPressed) */
23      public final static int RIGHT_BTN=2;
24
25      /** Tableau indiquant l' tat des boutons de la souris (vrai si bouton
26          enfonc ) */
27      private boolean buttonPressed [] = {false, false, false};
28
29      /*_____*/
30      /** Constructeur initialisant le mod le
31          * @param mdl Mod le contenant les donn es de l'application
32          */
33      public CtrlMouse3(AppMdl3 mdl) {
34          // TODO Auto-generated constructor stub
35          this.mdl = mdl;
36      }
37
38      /** Appel  lors d'un click de souris */
39      @Override
40      public void mouseClicked(MouseEvent arg0) {
41          // TODO Auto-generated method stub
42          System.err.println("MouseClicked");
43      }
44
45      /** Appel  lorsque la souris entre dans un composant */
46      @Override
47      public void mouseEntered(MouseEvent arg0) {
48          // TODO Auto-generated method stub
49          System.err.println("MouseEntered");
50      }
51
52      /** Appel  lorsque la souris sort d'un un composant */
53      @Override
54      public void mouseExited(MouseEvent arg0) {
55          // TODO Auto-generated method stub
56      }
57
58      /** Appel  lorsqu'on enfonce un bouton */
59      @Override
60      public void mousePressed(MouseEvent arg0) {
61          // TODO Auto-generated method stub
62
63          System.err.println("MousePressed_["+arg0.getX()+" ,"+arg0.getY()+"");
64          if (arg0.getButton()== MouseEvent.BUTTON1)

```

```

66     buttonPressed [LEFT_BTN] = true;
67     if (arg0.getButton() == MouseEvent.BUTTON2)
68         buttonPressed [MIDDLE_BTN] = true;
69     if (arg0.getButton() == MouseEvent.BUTTON3)
70         buttonPressed [RIGHT_BTN] = true;
71
72     mdl.setMouseX(arg0.getX());
73     mdl.setMouseY(arg0.getY());
74 }
75
76 /** appelé lorsque l'on libère un bouton */
77 @Override
78 public void mouseReleased(MouseEvent arg0) {
79     // TODO Auto-generated method stub
80     System.err.println("MouseReleased");
81     if (arg0.getButton() == MouseEvent.BUTTON1)
82         buttonPressed [LEFT_BTN] = false;
83     if (arg0.getButton() == MouseEvent.BUTTON2)
84         buttonPressed [MIDDLE_BTN] = false;
85     if (arg0.getButton() == MouseEvent.BUTTON3)
86         buttonPressed [RIGHT_BTN] = false;
87 }
88
89 /** appelé lorsque la souris a un bouton enfoncé et est déplacée */
90 @Override
91 public void mouseDragged(MouseEvent e) {
92     // TODO Auto-generated method stub
93     System.err.println("MouseDragged_" + e.getX() + ", " + e.getY());
94     mdl.moveDragged(e.getX(), e.getY(), buttonPressed);
95 }
96
97 /** appelé lorsque la souris se déplace sans aucun bouton enfoncé */
98 @Override
99 public void mouseMoved(MouseEvent e) {
100     // TODO Auto-generated method stub
101     System.err.println("MouseMoved");
102     // mdl.moveMouse(e.getX(), e.getY(), buttonPressed);
103 }
104 }
105
106 }

```

exemplesLatex/CoursJOGL3/src/mdl/AppMdl3.java

```

1 /**
2  *
3  */
4 package mdl;
5
6 import geometry.Objet3D;
7
8 import shading.PointLightSource;
9 import view3D.Camera;
10
11 import ctrl.CtrlMouse3;
12

```



```
13 /**
14  * @author remy
15  *
16  */
17 public class AppMdl3{
18
19     /** Angle Twist utilis  pour l'interface de visualisation   la souris */
20     private float angleTwist;
21     /** Angle Elevation utilis  pour l'interface de visualisation   la souris */
22     private float angleElevation;
23     /** Angle Azimuth utilis  pour l'interface de visualisation   la souris */
24     private float angleAzimuth;
25     /** Distance au centre utilis e pour l'interface de visualisation   la souris
26         */
27     private float distance;
28
29     /** pr c dente position X de la souris */
30     private int mouseX;
31     /** pr c dente position Y de la souris */
32     private int mouseY;
33
34     /** vitesse de l'effet de la souris */
35     float vitesse;
36
37     /** Param tre incr ment  par le timer pour l'animation */
38     int parametreAnimation=0;
39
40     /** objet   afficher */
41     Objet3D objet;
42
43     /** Cam ra */
44     Camera camera;
45
46     /** source lumineuse 0 */
47     PointLightSource light0;
48     /** source lumineuse 1 */
49     PointLightSource light1;
50     /** source lumineuse 2 */
51     PointLightSource light2;
52
53     /*_____*/
54     /** accesseur
55      * @return la source lumineuse num ro 0
56      */
57     public PointLightSource getLight0()
58     {
59         return light0;
60     }
61
62     /*_____*/
63     /** setter
64      * @param light0 la source lumineuse num ro 0   utiliser
65      */
66     public void setLight0(PointLightSource light0)
67     {
```

```

68     this.light0 = light0;
69 }
70
71 /*_____*/
72 /** accesseur
73  * @return la source lumineuse numéro 1
74  */
75 public PointLightSource getLight1()
76 {
77     return light1;
78 }
79
80 /*_____*/
81 /** setter
82  * @param light1 la source lumineuse numéro 1 à utiliser
83  */
84 public void setLight1(PointLightSource light1)
85 {
86     this.light1 = light1;
87 }
88
89 /*_____*/
90 /** accesseur
91  * @return la source lumineuse numéro 2
92  */
93 public PointLightSource getLight2()
94 {
95     return light2;
96 }
97
98 /*_____*/
99 /** setter
100  * @param light2 la source lumineuse numéro 2 à utiliser
101  */
102 public void setLight2(PointLightSource light2)
103 {
104     this.light2 = light2;
105 }
106
107 /*_____*/
108 /** accesseur
109  * @return la caméra
110  */
111 public Camera getCamera()
112 {
113     return camera;
114 }
115
116 /*_____*/
117 /** setter
118  * @param camera la caméra à utiliser
119  */
120 public void setCamera(Camera camera)
121 {
122     this.camera = camera;
123 }

```

```
124
125 /*_____*/
126 /** accesseur
127  * @return l'objet 3D utilis  pour l'application
128  */
129 public Objet3D getObjet()
130 {
131     return objet;
132 }
133
134 /*_____*/
135 /** setter
136  * @param objet l'Objet3D   utiliser pour l'application
137  */
138 public void setObjet(Objet3D objet)
139 {
140     this.objet = objet;
141 }
142
143 /*_____*/
144 /** Accesseur permet d'obtenir la valeur de distance
145  * @return la valeur du champs distance
146  */
147 public float getDistance() {
148     return distance;
149 }
150
151 /*_____*/
152 /** Permet de fixer la valeur du champs distance
153  * @param distance
154  */
155 public void setDistance(float distance) {
156     this.distance = distance;
157 }
158
159 /*_____*/
160 /** Accesseur permet d'obtenir la valeur de vitesse
161  * @return la valeur du champs vitesse
162  */
163 public float getVitesse() {
164     return vitesse;
165 }
166
167 /*_____*/
168 /** Permet de fixer la valeur du champs vitesse
169  * @param vitesse
170  */
171 public void setVitesse(float vitesse) {
172     this.vitesse = vitesse;
173 }
174
175 /*_____*/
176 /** Accesseur permet d'obtenir la valeur de l'angle twist
177  * @return la valeur du champs angleTwist
178  */
179 public float getAngleTwist() {
```

```

180     return angleTwist;
181 }
182
183 /*_____*/
184 /** Permet de fixer la valeur du champs angleTwist
185  * @param angleTwist
186  */
187 public void setAngleTwist(float angleTwist) {
188     this.angleTwist = angleTwist;
189 }
190
191 /*_____*/
192 /** Accesseur permet d'obtenir la valeur de l'angle Elevation
193  * @return la valeur du champs angleElevation
194  */
195 public float getAngleElevation() {
196     return angleElevation;
197 }
198
199 /*_____*/
200 /** Permet de fixer la valeur du champs angleElevation
201  * @param angleElevation
202  */
203 public void setAngleElevation(float angleElevation) {
204     this.angleElevation = angleElevation;
205 }
206
207 /*_____*/
208 /** Accesseur permet d'obtenir la valeur de l'angle Azimuth
209  * @return la valeur de angleAzimuth
210  */
211 public float getAngleAzimuth() {
212     return angleAzimuth;
213 }
214
215 /*_____*/
216 /** Permet de fixer la valeur du champs angleAzimuth
217  * @param angleAzimuth
218  */
219 public void setAngleAzimuth(float angleAzimuth) {
220     this.angleAzimuth = angleAzimuth;
221 }
222
223 /*_____*/
224 /** Accesseur permet d'obtenir la valeur du champs parametreAnimation
225  * @return la valeur du champs parametreAnimation
226  */
227 public int getParametreAnimation() {
228     return parametreAnimation;
229 }
230
231 /*_____*/
232 /** Permet de fixer la valeur du champs parametreAnimation
233  * @param parametreAnimation
234  */
235 public void setParametreAnimation(int parametreAnimation) {

```

```
236     this.parametreAnimation = parametreAnimation;
237 }
238
239 /*_____*/
240 /** Accesseur permet d'obtenir la valeur de la coordonnée x de la souris
241  * @return la valeur du champs mouseX
242  */
243 public int getMouseX() {
244     return mouseX;
245 }
246
247 /*_____*/
248 /** Permet de fixer la valeur du champs mouseX
249  * @param mouseX
250  */
251 public void setMouseX(int mouseX) {
252     this.mouseX = mouseX;
253 }
254
255 /*_____*/
256 /** Accesseur permet d'obtenir la valeur de la coordonnée y de la souris
257  * @return la valeur du champs mouseY
258  */
259 public int getMouseY() {
260     return mouseY;
261 }
262
263 /*_____*/
264 /** Permet de fixer la valeur du champs mouseY
265  * @param mouseY
266  */
267 public void setMouseY(int mouseY) {
268     this.mouseY = mouseY;
269 }
270
271 /*_____*/
272 /** Méthode appelée par le contrôleur lorsque la souris se déplace avec un
273     bouton enfoncé
274  * @param newMouseX nouvelle coordonnée X de la souris
275  * @param newMouseY nouvelle coordonnée Y de la souris
276  * @param buttonPressed tableau indiquant l'état des boutons
277  */
278 public void moveDragged(int newMouseX, int newMouseY, boolean buttonPressed[])
279 {
280     if (buttonPressed[CtrlMouse3.LEFT_BTN] || buttonPressed[CtrlMouse3.
281         MIDDLE_BTN])
282     {
283         System.err.println("Modif de elevation et azimuth");
284         camera.mouseDragged(newMouseX-mouseX, newMouseY-mouseY, buttonPressed);
285     }
286     if (buttonPressed[CtrlMouse3.LEFT_BTN] || buttonPressed[CtrlMouse3.
287         MIDDLE_BTN])
288     {
289         mouseX = newMouseX;
```

```

288         mouseY = newMouseY;
289     }
290 }
291
292 /*_____*/
293 /** Constructeur initialisant les angles , distances , vitesse , etc ...
294 */
295 public AppMdl3()
296 {
297     setAngleTwist(0.0f);
298     setAngleElevation(30.0f);
299     setAngleAzimuth(-60.0f);
300
301     setVitesse(1);
302     setDistance(30);
303
304     setMouseX(0);
305     setMouseY(0);
306
307 }
308 }
309
310 /*_____*/
311 /** Méthode faisant avancer l'animation et appelée régulièrement par la
312     méthode run du TimerTask
313 */
314 public void stepForward() {
315     // TODO Auto-generated method stub
316     parametreAnimation += vitesse;
317     if (parametreAnimation >= 360)
318         parametreAnimation -= 360;
319 }

```

exemplesLatex/CoursJOGL3/src/swingView/GLDisplayEx3.java

```

1 package swingView;
2
3 import geometry.Objet3D;
4 import geometry.Sphere;
5
6 import javax.media.opengl.*;
7 import javax.media.opengl.awt.GLCanvas;
8
9 import mdl.AppMdl3;
10
11 import shading.Material;
12 import shading.PointLightSource;
13 import view3D.CameraCentrale;
14 import view3D.Vector3D;
15
16 /**
17 * Classe gérant les événements d'affichage OpenGL (mais pas la GLUT)
18 *
19 */
20 public class GLDisplayEx3 implements GLEventListener{

```

```
21
22 /** modèle de données à afficher */
23 AppMdl3 mdl;
24
25 /** référence du GLCanvas */
26 GLCanvas drawable;
27
28 /**
29  * Constructeur mémorisant le canvas
30  * @param glDrawable
31  * @param mdl le modèle à afficher
32  */
33 public GLDisplayEx3(GLAutoDrawable glDrawable, AppMdl3 mdl){
34     this.drawable=(GLCanvas)glDrawable;
35     this.mdl=mdl;
36 }
37
38 /** Méthode d'affichage (appelée lors d'un appel explicite au repaint() du
39     canvas ou par l'Animator)
40     * pour rafraichir la vue. Contient tout le code d'affichage
41     */
42 @Override
43 public void display(GLAutoDrawable glDrawable) {
44     // TODO Auto-generated method stub
45     final GL2 gl = glDrawable.getGL().getGL2();
46     gl.glClear(GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BIT);
47     gl.glMatrixMode(GL2.GL_MODELVIEW);
48     // On ne refait pas le changement de repère de la caméra à chaque fois
49     // elle est sélectionnée une fois pour toutes
50
51     mdl.getCamera().select();
52
53     // light2 dans le repère du monde
54     mdl.getLight2().resetPosition();
55
56     // rotation pour l'animation
57     gl.glRotatef(mdl.getVitesse()*mdl.getParametreAnimation(), 0,1,0);
58     // light1 sur l'axe des z du repère de la sphère texturée
59     mdl.getLight1().resetPosition();
60     // affichage de l'objet
61     mdl.getObjet().display();
62     gl.glFlush();
63 }
64
65 /** méthode appelée par le timer et faisant avancer l'animation */
66 public void stepForward() {
67     // TODO Auto-generated method stub
68     mdl.setParametreAnimation(mdl.getParametreAnimation()+1);
69     if (mdl.getVitesse()*mdl.getParametreAnimation()>=360)
70         mdl.setParametreAnimation(0);
71     drawable.repaint();
72 }
73
74 /** Appelé lors de l'initialisation de la vue. Définir le ViewPoint et la
75     projection en perspective */
76 @Override
```

```

75 public void init(GLAutoDrawable glDrawable) {
76     // TODO Auto-generated method stub
77     final GL2 gl = glDrawable.getGL().getGL2();
78
79
80     gl.glColor3f(0.0f, 0.0f, 0.0f);
81     gl.glLineWidth(1);
82
83     ///////////////////////////////////////////////////////////////////
84     // création des sources lumineuses
85     // light0 sur l'axe des z du repère de la caméra
86     mdl.setLight0(new PointLightSource(0f, 0f, 300f, // position
87         Material.getQuadruple(0.6f, 0.6f, 0.6f, 1f),
88         // diffuse
89         Material.getQuadruple(0.5f, 0.5f, 0.5f, 1f)
90         // specular
91     ));
92     // light0 dans le repère de la caméra
93     mdl.getLight0().activate(GL2.GL_LIGHT0, true);
94
95     // light1 sur l'axe des z du repère de la sphère texturée
96     mdl.setLight1(new PointLightSource(0f, 0f, -300f, // position
97         Material.getQuadruple(0.3f, 0.3f, 0.3f, 1f), // diffuse
98         Material.getQuadruple(0.5f, 0.5f, 0.5f, 1f) // specular
99     ));
100     mdl.getLight1().activate(GL2.GL_LIGHT1, true);
101     // light2 sur l'axe à 45 degrés entre l'axe des x et l'axe des y du
102     // repère du monde
103     mdl.setLight2(new PointLightSource(300f, -300f, 0f, // position
104         Material.getQuadruple(0.3f, 0.3f, 0.3f, 1f), // diffuse
105         Material.getQuadruple(0.5f, 0.5f, 0.5f, 1f) // specular
106     ));
107     mdl.getLight2().activate(GL2.GL_LIGHT2, true);
108
109     ///////////////////////////////////////////////////////////////////
110     // Création d'une sphère texturée
111     Vector3D axisRotation = new Vector3D(0f,0f,1f);
112     Vector3D vScale = new Vector3D(1f,1f,1f);
113     Vector3D vTranslate = new Vector3D(0f,0f,0f);
114     Objet3D objet = new Sphere(5, 65, 50, axisRotation, 0, vScale,
115         vTranslate,
116         Material.getQuadruple(0.2f, 0.2f, 0.2f, 0.2f),
117         Material.getQuadruple(1f, 1f, 1f, 1f),
118         Material.getQuadruple(0.4f, 0.4f, 0.4f, 1f),
119         128f,
120         "fichiersTextures/textthe.png");
121     mdl.setObjet(objet);
122
123     initViewPortProjectionandLight(glDrawable, 0, 0, 500, 500);
124
125 }
126
127 /** Appelé lors d'un redimensionnement de la fenêtre.
128  * Définir le ViewPoint et la projection en perspective
129  */
130 @Override

```



```

127 public void reshape(GLAutoDrawable glDrawable, int x, int y, int width,
128     int height) {
129     // TODO Auto-generated method stub
130     initViewPortProjectionandLight(glDrawable, x, y, width, height);
131 }
132
133 /** Méthode d'initialisation de la scène, création des objets, des sources
134     lumineuses...
135     * @param glDrawable référence du GLCanvas
136     * @param x non utilisé
137     * @param y non utilisé
138     * @param width largeur de la fenêtre graphique
139     * @param height hauteur de la fenêtre graphique
140     */
141 private void initViewPortProjectionandLight(GLAutoDrawable glDrawable, int x,
142     int y, int width,
143     int height){
144     final GL2 gl = drawable.getGL().getGL2();
145
146     ////////////////////////////////////////////////////
147     // initialisation des caractéristiques de la scène (limière ambiante,
148     // Shade Model,...
149     gl.glClearColor(1.0f, 1.0f, 1.0f, 0.0f);
150
151     float model_ambient[] =
152     { 0.2f, 0.2f, 0.2f, 1.0f };
153     gl.glLightModelfv(GL2.GL_LIGHT_MODEL_AMBIENT, model_ambient, 0);
154
155     gl.glEnable(GL2.GL_LIGHTING);
156     gl.glEnable(GL2.GL_DEPTH_TEST);
157     gl.glShadeModel(GL2.GL_SMOOTH);
158
159     ////////////////////////////////////////////////////
160     // réinitialisation du ModelView et du ViewPort
161     gl.glMatrixMode(GL2.GL_MODELVIEW);
162     gl.glLoadIdentity();
163
164     gl.glViewport(0,0,width,height);
165
166     ////////////////////////////////////////////////////
167     // Création d'une caméra
168     mdl.setCamera(new CameraCentrale(45.0, (double)width/(double)height, 0.1,
169         5000.0,10,10,10,0,0,0,0,1));
170     //mdl.setCamera(new CameraNavigation(45.0, (double)width/(double)height,
171         0.1, 5000.0,10,10,10,0,0,0,0,1));
172     mdl.getCamera().gluLookAt(10,10,20,0,0,0,0,1,0);
173
174 }
175
176 @Override
177 public void dispose(GLAutoDrawable arg0) {
178     // TODO Auto-generated method stub
179 }

```



# Chapitre 4

## Graphe de scène

### 4.1 Définition du graphe de scène

Le graphe de scène est une représentation hiérarchique des éléments d'une scène 3D virtuelle. Cette représentation est une arborescence où chaque noeud représente une transformation géométrique (à appliquer à tout le sous-arbre des descendants du noeud) et une liste de noeuds fils qui peuvent être soit des noeuds similaires, soit des feuilles, à savoir des objets 3D simples (boîtes, sphères, maillages, etc.). Cette structure permet de représenter commodément des ensembles d'objets dont la position est définie relativement aux autres objets. Voici la classe *Java* permettant de représenter le graphe de scène :

exemplesLatex/CoursJOGL4/src/mdl/SceneGraphNode.java

```
1 package mdl;
2
3 import geometry.Objet3D;
4
5 import java.util.LinkedList;
6 import java.util.List;
7
8 import javax.media.opengl.GL2;
9 import javax.media.opengl.GLContext;
10
11 import view3D.Vector3D;
12
13 /**
14  * @author remy
15  * Classe pour représenter une hiérarchie d'objets.
16  * Chaque noeud comprend une transformation géométrique (via l'héritage
17  * d'Objet3D), et une liste d'Objet3D fils, pouvant éventuellement être
18  * du même type SceneGraphNode ou des objets simples (boîte, sphere,
19  * maillage, etc.).
20  */
21 public class SceneGraphNode extends Objet3D {
22
23     /**
24     * Liste d'objets
25     */
26     protected List<Objet3D> listeObjets;
27
28     /**
```

```

29     * Constructeur comportant la transformation géométrique et
30     * le contexte
31     * @param axisRotation vecteur directeur de l'axe de rotation
32     * @param angle angle de rotation
33     * @param vScale facteurs de changement d'échelle
34     * @param vTranslate vecteur de translation
35     * @param gl contexte opengl
36     */
37     public SceneGraphNode(Vector3D axisRotation, float angle, Vector3D vScale,
38         Vector3D vTranslate) {
39         super(axisRotation, angle, vScale, vTranslate, null);
40         // TODO Auto-generated constructor stub
41         listeObjets = new LinkedList<Objet3D>();
42     }
43
44     /**
45     * Ajout d'un objet dans la liste des fils
46     * @param obj
47     */
48     public void addChild(Objet3D obj){
49         listeObjets.add(obj);
50     }
51
52     /**
53     * Méthode d'affichage par un parcours récursif de l'arbre
54     */
55     @Override
56     public void display() {
57         // TODO Auto-generated method stub
58         GL2 gl = GLContext.getCurrentGL().getGL2();
59         gl.glPushMatrix();
60         super.display();
61         for (Objet3D obj : listeObjets){
62             obj.display();
63         }
64         gl.glPopMatrix();
65     }
66 }

```

## 4.2 L'exemple d'un robot

Considérons le robot représenté sur la figure 4.1. La hiérarchie d'objets correspondant à ce robot peut être représentée (en simplifiant) comme sur la figure 4.3

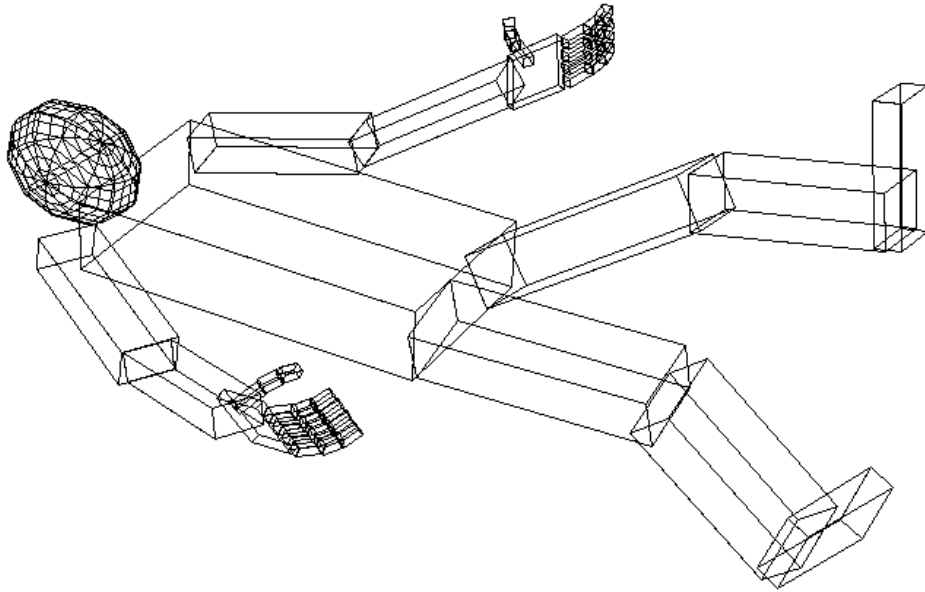


FIGURE 4.1: Exemple de robot construit et dessiné avec *OpenGL*

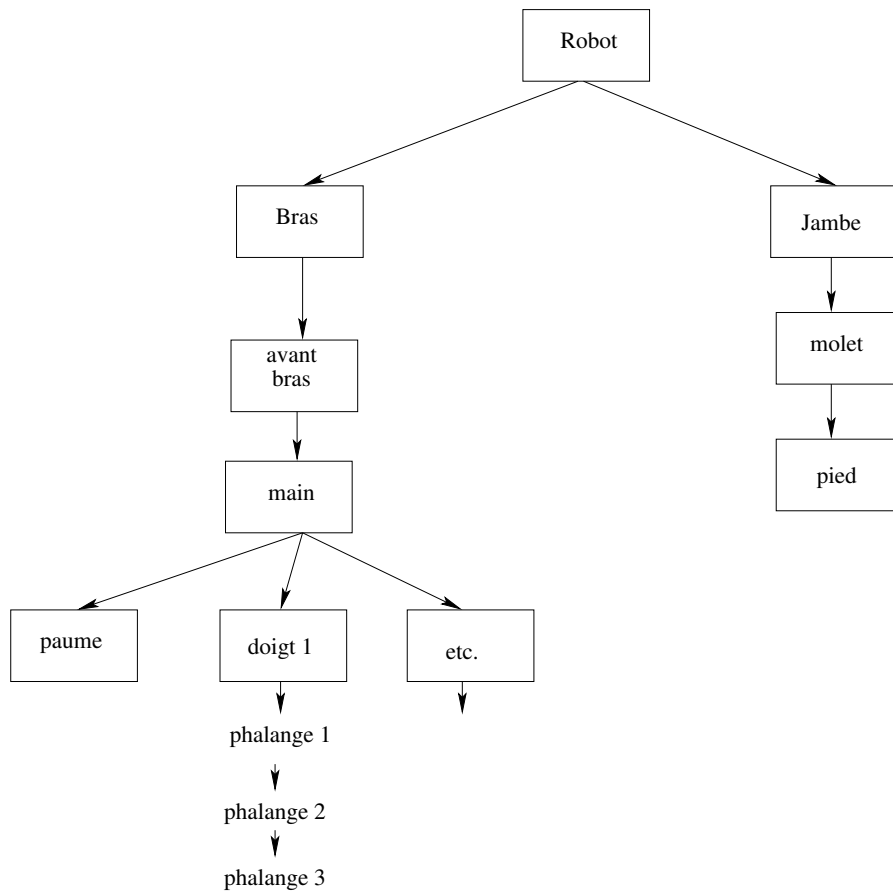


FIGURE 4.2: Exemple de robot construit et dessiné avec *OpenGL*

Voyons maintenant le code de construction du sous-arbre de cette hiérarchie correspondant à une main (voir la figure 4.3). Tous les angles sont paramétrables.

Voyons tout d'abord comment représenter une phalange.

exemplesLatex/CoursJOGL4/src/robot/Phalange.java

```

1 package robot;
2
3 import shading.Material;
4 import view3D.Vector3D;
5 import geometry.Boite;
6 import geometry.Objet3D;
7 import mdl.SceneGraphNode;
8
9 /**
10  *
11  * @author remy
12  * Classe représentant une phalange de la main d'un robot comme Objet3D.
13  */
14 public class Phalange extends SceneGraphNode {
15     float angleDoigt;
16
17     public Phalange(float angleDoigt, Vector3D vTranslate, float longueur) {
18         super(new Vector3D(0f, 1f, 0f), angleDoigt, new Vector3D(1f, 1f, 1f),
19             vTranslate);
20         this.angleDoigt=angleDoigt;
21
22         Objet3D phal = new Boite(-0.0f, longueur, -0.5f, 0.5f, -0.5f, 0.5f,
23             new Vector3D(0f, 0f, 1f), 0f, new Vector3D(1f,1f,1f),
24             new Vector3D(), Material.getQuadruple(0.2f, 0.2f, 0.2f, 1f),
25             Material.getQuadruple(0.4f, 0.4f, 0.4f, 1f),
26             Material.getQuadruple(0.2f, 0.2f, 0.2f, 1f),
27             10f, null);
28         addChild(phal);
29     }
30 }

```

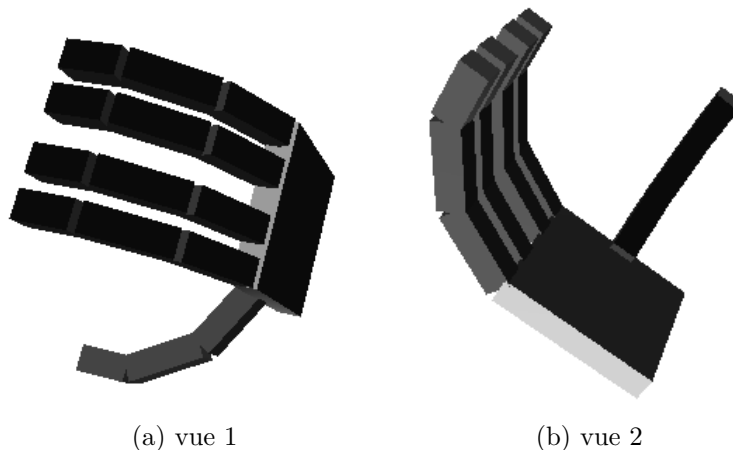


FIGURE 4.3: La main d'un robot construite par le code ci-dessous

Voyons maintenant comment repr senter un doigt constitu  de trois phalanges.

exemplesLatex/CoursJOGL4/src/robot/Doigt.java

```
1 package robot;
2
3 import view3D.Vector3D;
4 import geometry.Objet3D;
5 import mdl.SceneGraphNode;
6 /**
7  *
8  * @author remy
9  * Classe repr santant un doigt d'un Robot comme un Objet3D.
10 */
11 public class Doigt extends SceneGraphNode {
12
13
14     public Doigt(float angleDoigt, Vector3D axisRotation, float angle,
15                 Vector3D vTranslate)
16     {
17         super(axisRotation, angle, new Vector3D(1f, 1f, 1f), vTranslate);
18         // TODO Auto-generated method stub
19         Objet3D phal1 = new Phalange(angleDoigt, new Vector3D(3f, 0f, 0f), 2f);
20         SceneGraphNode phal2 = new Phalange(angleDoigt, new Vector3D(3f, 0f, 0f), 3
21             f);
22         phal2.addChild(phal1);
23         SceneGraphNode phal3 = new Phalange(angleDoigt, new Vector3D(0f, 0f, 0f), 3f
24             );
25         phal3.addChild(phal2);
26         addChild(phal3);
27     }
28 }
```

Voyons enfin comment repr senter la main compl te.

exemplesLatex/CoursJOGL4/src/robot/Main.java

```
1 package robot;
2
3 import shading.Material;
4 import view3D.Vector3D;
5 import geometry.Boite;
6 import geometry.Objet3D;
7 import mdl.SceneGraphNode;
8
9 /**
10  *
11  * @author remy
12  * Classe repr santant la main d'un Robot comme un Objet3D.
13  */
14 public class Main extends SceneGraphNode {
15
16     float anglePoignet;
17
18     public Main(float anglePoignet, float angleTwistMain, float angleDoigt,
19                 Vector3D vTranslate)
```

```
20 {
21     super(new Vector3D(1f, 0f, 0f), angleTwistMain+90, new Vector3D(1f, 1f, 1f),
           vTranslate);
22     // TODO Auto-generated method stub
23     Objet3D doigt1 = new Doigt(angleDoigt, new Vector3D(0f, 1f, 0f), 0f,
           new Vector3D(6f, -2.5f, 0f));
24     Objet3D doigt2 = new Doigt(angleDoigt, new Vector3D(0f, 1f, 0f), 0f,
           new Vector3D(6f, -1f, 0f));
25     Objet3D doigt3 = new Doigt(angleDoigt, new Vector3D(0f, 1f, 0f), 0f,
           new Vector3D(6f, 1f, 0f));
26     Objet3D doigt4 = new Doigt(angleDoigt, new Vector3D(0f, 1f, 0f), 0f,
           new Vector3D(6f, 2.5f, 0f));
27     SceneGraphNode pouce = new SceneGraphNode(new Vector3D(0f, 1f, 0f),
           0f, new Vector3D(1f,1f,1f),
           new Vector3D(3f, 2.5f, -0.5f));
28     Objet3D doigtPouce = new Doigt(angleDoigt, new Vector3D(0f, 0f, 1f),90f,
           new Vector3D(0f, 0f, 0f));
29     pouce.addChild(doigtPouce);
30     Objet3D paume = new Boite(-0f, 6f, -3f, 3f, -0.5f, 0.5f,
           new Vector3D(0f, 0f, 1f), 0f, new Vector3D(1f,1f,1f),
           new Vector3D(), Material.getQuadruple(0.2f, 0.2f, 0.2f, 0.2f),
           Material.getQuadruple(1f, 1f, 1f, 1f),
           Material.getQuadruple(0.3f, 0.3f, 0.3f, 1.0f),
           10f, null);
31     SceneGraphNode main = new SceneGraphNode(new Vector3D(0f, 1f, 0f),
           anglePoignet, new Vector3D(1f,1f,1f),
           vTranslate);
32     main.addChild(paume);
33     main.addChild(doigt1);
34     main.addChild(doigt2);
35     main.addChild(doigt3);
36     main.addChild(doigt4);
37     main.addChild(pouce);
38     addChild(main);
39 }
40 }
41 }
42 }
43 }
44 }
45 }
46 }
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
```