

Rapport du Travail d'Etudes

Ajp - Another Java Plotter

Un plotter 3D scriptable.

Par Jean-Pierre Lozi
sous l'encadrement de Gilles Menez

Table des matières

1	Introduction	3
1.1	Présentation	3
1.2	Fonctions implémentées	3
1.3	Installation et lancement	3
1.3.1	Installation de JOGL	3
1.3.2	Lancement	4
1.3.3	Utilisation	4
2	Utilisation et API	5
2.1	Introduction	5
2.2	Classe AjpUserScript	5
2.3	Classe AjpUser3DPoint	5
2.3.1	Constructeur	6
2.3.2	Accesseurs	6
2.3.3	Modificateurs	6
2.4	Classe AjpUserColor	6
2.4.1	Constructeur	6
2.4.2	Accesseurs	7
2.4.3	Modificateurs	7
2.5	Classe AjpUserLight	7
2.5.1	Constructeur	7
2.5.2	Accesseurs	8
2.5.3	Modificateurs	8
2.6	Classe AjpUserParameteredCurve	8
2.7	Classe AjpUserParameteredSurface	9
2.8	Classe AjpUserPlotter	10
2.8.1	Fonction newWindow	10
2.9	Classe AjpUserSurface	10
2.10	Classe AjpUserWindow	11
2.10.1	Accesseurs	11
2.10.2	Modificateurs	12
2.10.3	Fonctions publiques	14
2.10.4	L'interface AjpUserWindowAction	15
3	Exemples	17
3.1	La classe Example1	17
3.2	La classe Example2	22
3.3	La classe Example3	24
4	Diagramme De Classes	27
5	Listings	33

<i>TABLE DES MATIÈRES</i>	2
6 Conclusion	83

Chapitre 1

Introduction

1.1 Présentation

Ajp est un traceur de courbes et de surfaces en trois dimensions, scriptable en *java*, développé à l'aide de *java* et de *JOGL*. Il permet de tracer simplement des courbes, surfaces et autres objets tridimensionnels facilement, de manière transparente, sans que l'utilisateur n'ait à gérer le moindre objet *JOGL* ou *Swing*.

Le choix du système des scripts offre une très grande maniabilité à l'utilisateur, en lui permettant notamment d'accéder à toute l'*API* de *java*.

1.2 Fonctions implémentées

Ajp a été développé de telle manière que :

1. Son interface et son API soient simple d'utilisation :
 - L'utilisateur peut charger des fichiers java ou des classes précompilées.
 - L'utilisateur n'a pas à gérer les threads des différentes fenêtres : l'application s'en occupe pour lui.
 - L'utilisateur n'a besoin, pour effectuer un tracé, que de la fonction à tracer et des bornes. Il peut préciser des paramètres "esthétiques" mais les valeurs par défaut conviennent.
2. La navigation dans les objets en 3 dimensions soit possible :
 - On peut tourner à droite, à gauche, en haut et en bas à l'aide des touches du clavier.
 - On peut zoomer ou revenir en arrière.
3. L'utilisateur n'ait pas besoin de connaître le fonction interne de l'application ni les *APIs* qu'il utilise.

Une idée importante nous a guidés : ce n'est pas parce que l'utilisateur a en ses mains un programme *scriptable* qu'il doit pour autant être *difficile d'utilisation*.

1.3 Installation et lancement

1.3.1 Installation de JOGL

Ajp est basé sur JOGL. Voici où placer les différents fichiers, que l'on peut récupérer sur <https://jogl.dev.java.net/>.

```
$JAVA_HOME/jre/lib/i386/libjogl.so  
$JAVA_HOME/jre/lib/i386/libjogl_cg.so  
$JAVA_HOME/jre/lib/ext/jogl.jar
```



FIG. 1.1 – L'interface de l'application

1.3.2 Lancement

Pour lancer Ajp, il faut s'assurer que `javac` est dans la variable d'environnement `$PATH`. Puis, sous linux, il suffit de lancer le script `cl.sh`. Sous windows, lancer le script `cl.bat`. Pour lancer l'application à la main, il suffit de taper les commandes suivantes :

```
javac -classpath classes -d classes sources/org/lozi/ajp/ajp/scripts/*.java \
      sources/org/lozi/ajp/ajp/*.java
java -classpath classes org.lozi.ajp.ajp.Ajp
```

1.3.3 Utilisation

L'interface de l'application est assez intuitive. Voici une capture d'écran :

L'utilisateur peut charger un script java implémentant l'interface `AjpUserScript`, ou sa version compilée. Puis il peut exécuter le script en cliquant sur `Start`.

Dans chacune des fenêtres créées par le script de l'utilisateur, il est possible de faire tourner la courbe vers la gauche, la droite, le haut ou le bas à l'aide des flèches du clavier. Il est également possible de zoomer ou de dézoomer à l'aide des touches *Page Up* et *Page Down*.

Chapitre 2

Utilisation et API

2.1 Introduction

Ajp permet d'exécuter des *scripts* écrits en *java*. L'utilisateur a à sa disposition l'interface `AjpUserScript`, qui contient une fonction `start`. Cette fonction est exécutée lorsque l'utilisateur clique sur `Start` après avoir chargé la classe.

L'utilisateur est parfaitement libre d'utiliser toutes les classes d'*Ajp* dans cette fonction, toutefois il est recommandé de n'utiliser que les classes destinées à l'utilisateur - dont le nom commence par *AjpUser*, et les classes de l'*API* java.

L'application est conçue de sorte que l'utilisateur n'a pas besoin de connaître l'existence des classes *OpenGL* ou *Swing* pour créer des scripts, celles-ci étant encapsulées dans les classes destinées aux utilisateurs.

Voici la liste des classes/interfaces accessibles à l'utilisateur :

- `AjpUserScript` : Cette interface doit être implémentée par l'utilisateur lorsqu'il crée un script.
- `AjpUser3DPoint` : Cette classe représente un point en 3 dimensions.
- `AjpUserColor` : Cette classe représente une couleur, avec son canal alpha.
- `AjpUserLight` : Cette classe représente une lumière.
- `AjpUserParameteredCurve` : Les classes dérivant de `AjpUserParameteredCurve` représentent des courbes paramétrées.
- `AjpUserParameteredSurface` : Les classes dérivant de `AjpUserParameteredSurface` représentent des surfaces paramétrées.
- `AjpUserPlotter` : Cette classe contient les différentes fonctions de base accessibles à l'utilisateur.
- `AjpUserSurface` : Les classes dérivant de `AjpUserSurface` représentent une surface 3D.
- `AjpUserWindow` : Cette classe représente une fenêtre contenant un contexte *OpenGL*.
- `AjpUserWindowAction` : Les implémentations de cette classe représentent des actions devant être effectuées.

2.2 Classe **AjpUserScript**

Cette classe doit être implémentée par l'utilisateur, les scripts de l'application en sont des implémentations.

2.3 Classe **AjpUser3DPoint**

La classe `AjpUser3DPoint` représente un point en 3 dimensions.

2.3.1 Constructeur

```
/**
 * Creates a new 3D point.
 *
 * @param x The x coordinate.
 * @param y The y coordinate.
 * @param z The z coordinate.
 */
public AjpUser3DPoint( double x, double y, double z )
```

Listing 2.1 – Constructeur de la classe AjpUser3DPoint

Ce constructeur permet de créer un point en 3 dimensions à l'aide de ses trois coordonnées cartésiennes.

2.3.2 Accesseurs

```
/** @return Returns the x coordinate. */
public double getX()
/** @return Returns the y coordinate. */
public double getY()
/** @return Returns the z coordinate. */
public double getZ()
```

Listing 2.2 – Accesseurs de la classe AjpUser3DPoint

Ces accesseurs permettent de récupérer les coordonnées cartésiennes du point.

2.3.3 Modificateurs

```
/** @param x The x coordinate to set. */
public void setX(double x)
/** @param y The y coordinate to set. */
public void setY(double y)
/** @param z The y coordinate to set. */
public void setZ(double z)
```

Listing 2.3 – Modificateurs de la classe AjpUser3DPoint

Ces modificateurs permettent de modifier les coordonnées cartésiennes du point.

2.4 Classe AjpUserColor

Cette classe représente une couleur avec canal alpha.

2.4.1 Constructeur

```
/**
 * Creates an AjpUserColor object.
 *
 * @param red The amount of red. Between 0.0 and 1.0.
 * @param green The amount of green. Between 0.0 and 1.0.
 * @param blue The amount of blue. Between 0.0 and 1.0.
 * @param alpha The alpha channel. Between 0.0 and 1.0.
```

```
*/
public AjpUserColor( float red, float green, float blue, float alpha )
```

Listing 2.4 – Constructeur de la classe AjpUserColor

Ce constructeur permet de créer une couleur dont on donne les quantités de rouge, vert et bleu, ainsi que le canal alpha. Chacun de ces paramètres doit avoir une valeur comprise entre 0.0 (le minimum) et 1.0 (le maximum). Le canal alpha représente la transparence de la couleur (1.0 : opaque, 0.0 : transparent).

2.4.2 Accesseurs

```
/** @return Returns the amount of red. Between 0.0 and 1.0. */
public float getRed()
/** @return Returns the amount of green. Between 0.0 and 1.0. */
public float getGreen()
/** @return Returns the amount of blue. Between 0.0 and 1.0. */
public float getBlue()
/** @return Returns the alpha channel's value. Between 0.0 and 1.0. */
public float getAlpha()
```

Listing 2.5 – Accesseurs de la classe AjpUserColor

Ces accesseurs permettent de récupérer chacune des composantes de la couleur, ainsi que son canal alpha.

2.4.3 Modificateurs

```
/** @param red The amount of red to set. Between 0.0 and 1.0. */
public void setRed(float red)
/** @param green The amount of green to set. Between 0.0 and 1.0. */
public void setGreen(float green)
/** @param blue The amount of blue to set. Between 0.0 and 1.0. */
public void setBlue(float blue)
/** @param alpha The alpha channel to set. Between 0.0 and 1.0. */
public void setAlpha(float alpha)
```

Listing 2.6 – Modificateurs de la classe AjpUserColor

Ces procédures permettent de modifier chacune des composantes de la couleur, ainsi que son canal alpha.

2.5 Classe AjpUserLight

2.5.1 Constructeur

```
/**
 * Creates an AjpUserLight.
 *
 * @param position The light position.
 * @param color The light color.
 */
public AjpUserLight ( AjpUser3DPoint position, AjpUserColor color )
```

Listing 2.7 – Constructeur de la classe AjpUserLight

Ce constructeur permet de créer une lumière, à l'aide d'une position et d'une couleur. *OpenGL* permet de gérer jusqu'à 8 couleurs.

2.5.2 Accesseurs

```
/** @return Returns the color. */
public AjpUserColor getColor()
/** @return Returns the position. */
public AjpUser3DPoint getPosition()
```

Listing 2.8 – Accesseurs de la classe AjpUserLight

Ces accesseurs permettent de récupérer la couleur et la position de la lumière.

2.5.3 Modificateurs

```
/** @param color The color to set. */
public void setColor(AjpUserColor color)
/** @param position The position to set. */
public void setPosition(AjpUser3DPoint position)
```

Listing 2.9 – Modificateurs de la classe AjpUserLight

Ces procédures permettent de modifier la couleur et la position de la lumière.

2.6 Classe AjpUserParameteredCurve

Les classes dérivées d'AjpUserParameteredCurve représentent des courbes paramétrées. L'utilisateur doit implémenter la fonction `init` afin d'initialiser les variables locales.

```
package org.lozi.ajp.ajp;

/**
 * AjpUserParameteredCurve abstract class.
 * The user should derive this class each time he needs a new curve with a
 * single paramater.
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
 */
public abstract class AjpUserParameteredCurve {
    /** The lower bound for the parameter t. */
    protected double tLowerBound;
    /** The upper bound for the parameter t. */
    protected double tUpperBound;
    /** The plot color. */
    protected AjpUserColor color;
    /** The line width. */
    protected float lineWidth;
    /** The step. */
    protected double step;

    /**
     * Evaluates the function for the parameter t.
     * Should return a point, the value of the function at t.
     */
}
```

```

    * @param t The parameter for which the function should be evaluated.
    */
    public abstract AjpUser3DPoint evaluate( double t );

    /**
     * Initializes the local variables.
     */
    public abstract void init ();

    // ...

```

Listing 2.10 – La classe AjpUserParameteredCurve

2.7 Classe AjpUserParameteredSurface

Les classes dérivées d'AjpUserParameteredCurve représentent des surfaces paramétrées. L'utilisateur doit implémenter la fonction `init` afin d'initialiser les variables locales.

```

/**
 * AjpUserParameteredSurface interface.
 * The user should derive this class each time he needs a new curve with a
 *   two parameters.
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
 */
public abstract class AjpUserParameteredSurface {
    /** The lower bound for the parameter t. */
    protected double tLowerBound;
    /** The upper bound for the parameter t. */
    protected double tUpperBound;
    /** The lower bound for the parameter s. */
    protected double sLowerBound;
    /** The upper bound for the parameter s. */
    protected double sUpperBound;
    /** The step for the s parameter. */
    protected float sStep;
    /** The step for the t parameter. */
    protected float tStep;
    /** The plot's color. */
    protected String colorString;

    /**
     * Evaluates the function for the parameters s and t.
     * Should return a tridimensionnal array of double values.
     *
     * @param s The first parameter for which the function should be evaluated
     * .
     * @param t The scond parameter for which the function should be evaluated
     * .
     * @return The evaluated point.
     */
    public abstract AjpUser3DPoint evaluate( double s, double t );

```

Listing 2.11 – La classe AjpUserParameteredSurface

2.8 Classe **AjpUserPlotter**

Cette classe contient les principales fonctions de base fournies à l'utilisateur, comme la création de fenêtres.

2.8.1 Fonction **newWindow**

```
/**
 * Creates a new window.
 *
 * @param title The window title.
 * @param width The window width.
 * @param height The window height.
 * @param x The window x position.
 * @param y The window y position.
 * @param lightsArray The array containing the window's lights (8 max).
 * @return The newly created window.
 */
public AjpUserWindow newWindow( String title, int width, int height, int x,
                                int y, AjpUserLight[] lightsArray )
```

Listing 2.12 – La fonction newWindow

Cette fonction crée une nouvelle fenêtre dont on spécifie le titre, la largeur la hauteur, la position, et un tableau contenant jusqu'à 8 lumières.

2.9 Classe **AjpUserSurface**

Les classes dérivées **AjpParameteredCurve** représentent des courbes paramétrées. L'utilisateur doit implémenter la fonction **init** afin d'initialiser les variables locales.

```
package org.lozi.ajp.ajp;

/**
 * AjpUserSurface abstract class.
 * The user should derive this class each time he needs to plot a a R -> R2
 * function.
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
 */
public abstract class AjpUserSurface {
    /** The x lower bound. */
    protected double xLowerBound;
    /** The x upper bound. */
    protected double xUpperBound;
    /** The y lower bound. */
    protected double yLowerBound;
    /** The y upper bound. */
    protected double yUpperBound;
    /** The grid's size. */
    protected double gridSize;
    /** The plot color. */
    protected String colorString;

    /**
```

```

* Evaluates the curve's function at the (x,y) position.
*
* @param t The parameter for which the function should be evaluated.
* @return The evaluation.
*/
public abstract double evaluate( double x, double y );

/**
 * Initializes the local variables.
*/
public abstract void init ();

    // ...

```

Listing 2.13 – La classe AjpUserSurface

2.10 Classe AjpUserWindow

Cette classe représente une fenêtre de l'application. Une fenêtre peut contenir plusieurs tracés et plusieurs lumières. L'utilisateur a accès à sa couleur de fond et à d'autres paramètres. Cette classe ne doit pas être instanciée via son constructeur (dont la visibilité est restreinte à son paquetage), mais par la fonction `newWindow` de la classe `AjpUserPlotter`.

2.10.1 Accesseurs

La fonction `getParameterizedCurvesArrayList`

```

/**
 * Gets the list of the parametered curves bound to this window.
*/
public ArrayList<AjpUserParameteredCurve> getParameterizedCurvesArrayList()

```

Listing 2.14 – La fonction `getParameterizedCurvesArrayList`

Cette fonction permet de récupérer la liste de courbes paramétrées associées à la fenêtre.

La fonction `getParameterizedSurfacesArrayList`

```

/**
 * Gets the list of the parametered surfaces bound to this window.
*/
public ArrayList<AjpUserParameteredSurface> getParameterizedSurfacesArrayList()

```

Listing 2.15 – La fonction `getParameterizedSurfacesArrayList`

Cette fonction permet de récupérer la liste des surfaces paramétrées associées à la fenêtre.

La fonction `getSurfacesArrayList`

```

/**
 * Gets the list of the surfaces bound to this window.
*/
public ArrayList<AjpUserSurface> getSurfacesArrayList()

```

Listing 2.16 – La fonction `getSurfacesArrayList`

Cette fonction permet de récupérer la liste des surfaces associées à la fenêtre.

2.10.2 Modificateurs

La fonction **setAxesBounds**

```
/**
 * Sets the bounds of the window's axes.
 *
 * @param xLowerBound The x lower bound.
 * @param xUpperBound The x upper bound.
 * @param yLowerBound The y lower bound.
 * @param yUpperBound The y upper bound.
 * @param zLowerBound The z lower bound.
 * @param zUpperBound The z upper bound.
 */
public void setAxesBounds( double xLowerBound, double xUpperBound, double
    yLowerBound, double yUpperBound, double zLowerBound, double zUpperBound
)
```

Listing 2.17 – La fonction `setAxesBounds`

Cette fonction permet de modifier les bornes des 3 axes de la fenêtre.

La fonction **setAxesColor**

```
/**
 * Sets the axes' color.
 *
 * @param color The new axes' color.
 */
public void setAxesColor( AjpUserColor color )
```

Listing 2.18 – La fonction `setAxesColor`

Cette fonction permet de modifier la couleur des axes.

La fonction **setBackgroundColor**

```
/**
 * Sets the background color.
 *
 * @param color The new background color.
 */
public void setBackgroundColor( AjpUserColor color )
```

Listing 2.19 – La fonction `setBackgroundColor`

Cette fonction modifie la couleur de fond de la fenêtre.

La fonction **setRotationSteps**

```
/**
 * Rotates the window contents.
 *
 * @param leftRightStep The left/right step.
```

```

* @param topBottomStep The top/bottom step.
*/
public void setRotationSteps( double leftRightStep, double topBottomStep )

```

Listing 2.20 – La fonction setRotationSteps

Cette fonction permet de modifier les pas de rotation de la fenêtre (qui valent 0.0 par défaut). Le pas de rotation leftRightStep est la distance vers la gauche (resp. vers le bas) que parcourt la caméra à chaque affichage - la valeur est donc dépendante de la machine qui exécute le script.

La fonction setVisible

```

/**
 * Shows/Hides the window.
 *
 * @param visibility True to show the window, false to hide it.
 */
public void setVisible( boolean visibility )

```

Listing 2.21 – La fonction setVisible

Cette fonction permet d'afficher/masquer la fenêtre.

La fonction setParameteredCurvesArrayList

```

/**
 * Sets the list of the parametered curves bound to this window.
 *
 * @param parameteredCurvesArrayList The list of parametered curves to set.
 */
public void setParameteredCurvesArrayList( ArrayList<
    AjpUserParameteredCurve> parameteredCurvesArrayList )

```

Listing 2.22 – La fonction setParameteredCurvesArrayList

Cette procédure permet de modifier la liste de courbes paramétrées associées à la fenêtre.

La fonction setParameteredSurfacesArrayList

```

/**
 * Sets the list of the parametered surfaces bound to this window.
 *
 * @param parameteredSurfacesArrayList The list parametered surfaces to set
 *
 */
public void setParameteredSurfacesArrayList( ArrayList<
    AjpUserParameteredSurface> parameteredSurfacesArrayList )

```

Listing 2.23 – La fonction setParameteredSurfacesArrayList

Cette procédure permet de modifier la liste des surfaces associées à la fenêtre.

La fonction setSurfacesArrayList

```

/**
 * Sets the list of the surfaces bound to this window.
 *
 * @param surfacesArrayList The list of surfaces to set.
 */
public void setSurfacesArrayList( ArrayList<AjpUserSurface>
    surfacesArrayList )

```

Listing 2.24 – La fonction setSurfacesArrayList

2.10.3 Fonctions publiques

La fonction `animate`

```

/**
 * Animates the window.
 */
public void animate ()

```

Listing 2.25 – La fonction `animate`

Cette fonction lance l’animation dans la fenêtre. Elle doit être appelée à la fin de la fonction `start`.

La fonction `addParameteredCurve`

```

/**
 * Binds a parametered curve to the current window.
 *
 * @param curve The parametered curve to bind.
 */
public void addParameteredCurve( AjpUserParameteredCurve curve )

```

Listing 2.26 – La fonction `addParameteredCurve`

Cette fonction permet d’ajouter une courbe paramétrée à la fenêtre.

La fonction `addSurface`

```

/**
 * Binds a surface to the current window.
 *
 * @param surface The surface to bind.
 */
public void addSurface( AjpUserSurface surface )

```

Listing 2.27 – La fonction `addSurface`

Cette fonction permet d’ajouter une surface à la fenêtre.

La fonction `addParameteredSurface`

```

/**
 * Binds a parametered surface to the current window.
 *

```

```

* @param curve The parametered curve to bind.
*/
public void addParameteredSurface( AjpUserParameteredSurface surface ) {
    listener.addParameteredSurface( surface );
}

```

Listing 2.28 – La fonction addParameteredSurface

Cette fonction permet d'ajouter une surface paramétrée à la fenêtre.

La fonction setInitialCameraPositionAndDirection

```

/**
 * Sets the initial camera position and direction.
 *
 * @param Ajp3DUserPoint position The camera position.
 * @param xRot The x initial rotation.
 * @param yDir The y initial rotation.
 * @param zDir The z initial rotation.
 */
public void setInitialCameraPositionAndDirection( AjpUser3DPoint position,
    double xRot, double yRot, double zRot ) {
    listener.setInitialCameraPositionAndDirection( position, xRot, yRot, zRot
    );
}

```

Listing 2.29 – La fonction setInitialCameraPositionAndDirection

Cette fonction permet d'ajuster la position de départ de la caméra.

La fonction addAction

```

/**
 * Set the window's action.
 *
 * @param action The action to bind.
 */
public void setAction( AjpUserWindowAction action ) {
    listener.setAction( action );
}

```

Listing 2.30 – La fonction addAction

Cette fonction permet d'ajouter une fonction à la fenêtre.

2.10.4 L'interface AjpUserWindowAction

```

/**
 * AjpUserWindowAction interface.
 * The user should implement this class each time he needs to bind an
 * action to
 * the window.
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
 */
public interface AjpUserWindowAction {

```



```
/**
 * This function is called each time a frame is drawn.
 *
 * @param window The window in which the frame is drawn.
 */
public abstract void run ( AjpUserWindow window );
}
```

Listing 2.31 – L’interface AjpUserWindowAction

Les classes qui implémentent AjpUserWindowAction représentent des actions à effectuer à chaque fois que le contexte *OpenGL* est redessiné.

Chapitre 3

Exemples

Cette section contient différents exemples de script pour *Ajp*.

3.1 La classe **Example1**

Cette classe crée deux fenêtre avec diverses surfaces paramétrées et fonctions.

```
package org.lozi.ajp.ajp.scripts;

import org.lozi.ajp.ajp.*;

public class Example1 implements AjpUserScript
{
    public void start( AjpUserPlotter plotter ) {
        // We create the lights.
        AjpUserLight[] lights = new AjpUserLight[] {
            new AjpUserLight( new AjpUser3DPoint( 0, 0, 5 ),
                new AjpUserColor( .5f, .5f, .5f, 1f ) ) };

        // We create the window.
        AjpUserWindow window = plotter.newWindow( "First test window", 320, 240,
            10, 10, lights );
        // We set the background color.
        window.setBackgroundColor( new AjpUserColor( 0f, 0f, 0f, 1.0f) );

        // We add three surfaces.
        window.addSurface( new AjpUserSurface() {
            public double evaluate( double x, double y ) {
                return Math.cos( x ) + Math.cos( y );
            }
        }

        public void init() {
            this.colorString = "Blue";
            this.xLowerBound = -10;
            this.xUpperBound = 10;
            this.yLowerBound = -10;
            this.yUpperBound = 10;
            this.gridSize = 0.3;
        }
    }
}
```

```

});

window.addSurface( new AjpUserSurface() {
    public double evaluate( double x, double y ) {
        return Math.sin( x * y );
    }

    public void init () {
        this.colorString = "Red";
        this.xLowerBound = -5;
        this.xUpperBound = 5;
        this.yLowerBound = -5;
        this.yUpperBound = 5;
        this.gridSize = 0.1;
    }
});

window.addSurface( new AjpUserSurface() {
    public double evaluate( double x, double y ) {
        if( x * y > 0 )
            return Math.sqrt( x * y );
        else return Math.sqrt( - x * y );
    }

    public void init() {
        this.colorString = "Yellow";
        this.xLowerBound = -6;
        this.xUpperBound = 6;
        this.yLowerBound = -6;
        this.yUpperBound = 6;
        this.gridSize = 0.5;
    }
});

// We set the axes' bounds.
window.setAxesBounds( -10.0, 10.0, -10.0, 10.0, -10.0, 10.0 );

// We set the rotation steps.
window.setRotationSteps( 0.1, 0.0 );

// We animate the window.
window.animate();

// Now we create another window.
// We create two lights.
lights = new AjpUserLight[] {
    new AjpUserLight( new AjpUser3DPoint( 0, 0, 2 ),
        new AjpUserColor( .8f, .8f, .8f, 1f ) ) };

// We create the window itself.
window = plotter.newWindow( "Second test window", 320, 240, 400, 10,
    lights );

```

```

// We set its background color.
window.setBackgroundColor( new AjpUserColor( 0.1f, 0.1f, 0.2f, 1.0f) );

// We add two surfaces and two parametered curves.
window.addSurface( new AjpUserSurface() {
    public double evaluate( double x, double y ) {
        return Math.cos(x + y);
    }

    public void init() {
        this.colorString = "Purple";
        this.xLowerBound = -10;
        this.xUpperBound = 10;
        this.yLowerBound = -10;
        this.yUpperBound = 10;
        this.gridSize = 0.3;
    }
});

window.addSurface( new AjpUserSurface() {
    public double evaluate( double x, double y ) {
        return Math.cos( x - y );
    }

    public void init() {
        this.colorString = "Cyan";
        this.xLowerBound = -10;
        this.xUpperBound = 10;
        this.yLowerBound = -10;
        this.yUpperBound = 10;
        this.gridSize = 0.3;
    }
});

window.addParameteredCurve( new AjpUserParameteredCurve() {
    public AjpUser3DPoint evaluate( double t ) {
        double R=1.0;
        double r=0.2;
        double n=17.0/2;
        return new AjpUser3DPoint( ( R + r * Math.cos( n * t ) ) * Math.cos( t
            ) * 15,
            ( R + r * Math.cos( n * t ) ) * Math.sin( t ) * 15,
            ( r * Math.sin( n * t ) * 15 ) );
    }

    public void init() {
        this.color = new AjpUserColor( .9f, 0.8f, 0.9f, 1.0f );
        this.tLowerBound = -7;
        this.tUpperBound = 7;
        this.step = 0.01;
    }
});

```

```

window.addParameteredCurve( new AjpUserParameteredCurve() {
    public AjpUser3DPoint evaluate( double t ) {
        double R=1.0;
        double r=0.2;
        double n=17.0/2;
        return new AjpUser3DPoint( ( R + r * Math.cos( n * t + 0.2 ) ) * Math.
            cos( t + 0.2 ) * 15,
            ( R + r * Math.cos( n * t + 0.2 ) ) * Math.sin( t + 0.2 ) *
                15,
            ( r * Math.sin( n * t + 0.2 ) * 15 ) );
    }

    public void init() {
        this.color = new AjpUserColor( .8f, .8f, .9f, 1.0f );
        this.tLowerBound = -7;
        this.tUpperBound = 7;
        this.step = 0.01;
    }
});

window.addParameteredCurve( new AjpUserParameteredCurve() {
    public AjpUser3DPoint evaluate( double t ) {
        double R=1.0;
        double r=0.2;
        double n=17.0/2;
        return new AjpUser3DPoint( ( R + r * Math.cos( n * t + 0.4 ) ) * Math.
            cos( t + 0.4 ) * 15,
            ( R + r * Math.cos( n * t + 0.4 ) ) * Math.sin( t + 0.4 ) *
                15,
            ( r * Math.sin( n * t + 0.4 ) * 15 ) );
    }

    public void init() {
        this.color = new AjpUserColor( .8f, .9f, .8f, 1.0f );
        this.tLowerBound = -7;
        this.tUpperBound = 7;
        this.step = 0.01;
    }
});

window.addParameteredCurve( new AjpUserParameteredCurve() {
    public AjpUser3DPoint evaluate( double t ) {
        double R=1.0;
        double r=0.2;
        double n=17.0/2;
        return new AjpUser3DPoint( ( R + r * Math.cos( n * t + 0.6 ) ) * Math.
            cos( t + 0.6 ) * 15,
            ( R + r * Math.cos( n * t + 0.6 ) ) * Math.sin( t + 0.6 ) *
                15,
            ( r * Math.sin( n * t + 0.6 ) * 15 ) );
    }

    public void init() {
        this.color = new AjpUserColor( .9f, .8f, .9f, 1.0f );
    }
});

```

```
    this.tLowerBound = -7;
    this.tUpperBound = 7;
    this.step = 0.01;
  }
});

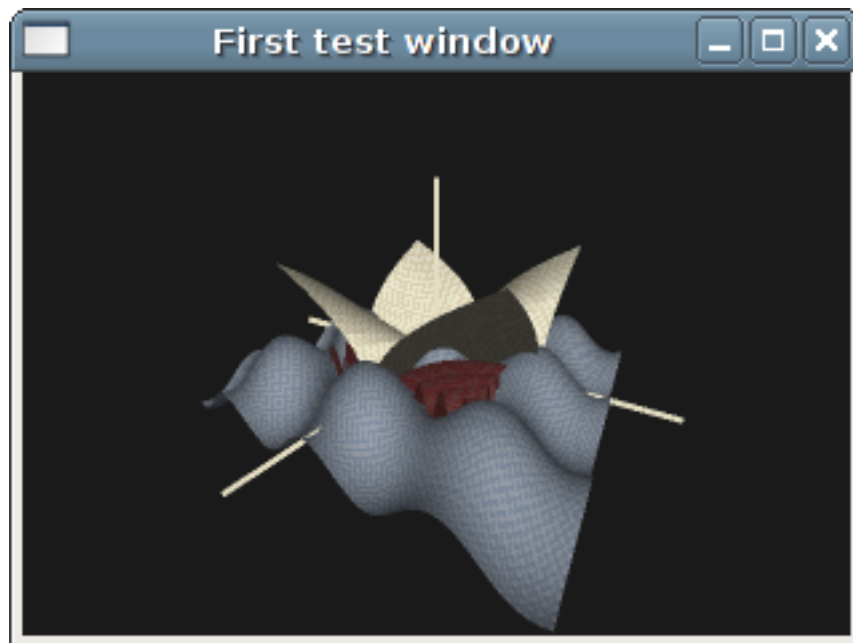
// We set the axes' bounds.
window.setAxesBounds( -10.0, 10.0, -10.0, 10.0, -10.0, 10.0 );

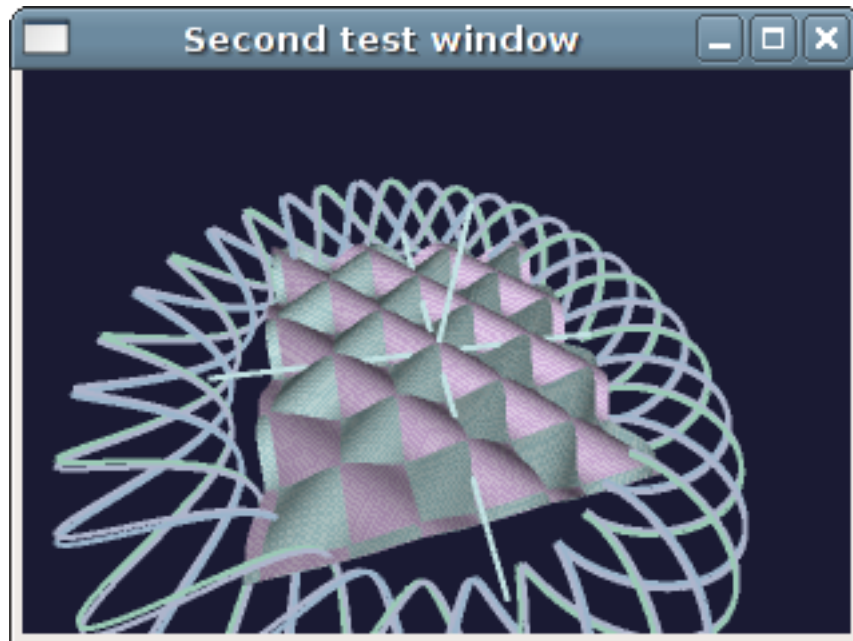
// We set the rotation steps.
window.setRotationSteps( 0.2, 0.1 );

// We animate the window.
window.animate();
}
```

Listing 3.1 – L'exemple Example1.java

Cet exemple produit le résultat suivant :





3.2 La classe **Example2**

Cette classe crée une surface et une courbe paramétrée animées, à l'aide d'une action associée à la fenêtre.

```
package org.lozi.ajp.ajp.scripts;

import org.lozi.ajp.ajp.*;

import java.util.ArrayList;

public class Example2 implements AjpUserScript
{
    public void start( AjpUserPlotter plotter ) {
        // We create the lights.
        AjpUserLight[] lights = new AjpUserLight[] {
            new AjpUserLight( new AjpUser3DPoint( 0, 0, 5 ),
                new AjpUserColor( .5f, .5f, .5f, 1f ) ) };

        // We create the window.
        AjpUserWindow window = plotter.newWindow( "First test window", 320, 240,
            10, 10, lights );
        // We set the background color.
        window.setBackgroundColor( new AjpUserColor( 0.1f, 0.1f, 0.1f, 1.0f ) );
        // We add a surface to the window.
        window.addSurface( new MySurface() );
        // We add a parametered curve.
        window.addParameteredCurve( new MyParameteredCurve() );
        // We set the window's action.
        window.setAction( new AjpUserWindowAction() {
            public void run ( AjpUserWindow window ) {
                MySurface surface = (MySurface)window.getSurfacesArrayList().get( 0 )
                ;
            }
        } );
    }
}
```

```

        MyParameteredCurve curve = (MyParameteredCurve>window.
            getParameteredCurvesArrayList().get( 0 );
        // We increment the parameter T of the first surface.
        surface.setT( surface.getT() + 0.1 );
        curve.setU( curve.getU() + 0.01 );
    }
});

// We set the rotation steps.
window.setRotationSteps( 0.1, 0 );
// We set the axes' bounds.
window.setAxesBounds( -20, 20, -20, 20, -20, 20 );
// We animate the window.
window.animate();
}

class MySurface extends AjpUserSurface {
    // We set a parameter that we can update.
    private double t;

    public double evaluate( double x, double y ) {
        return Math.cos( Math.sqrt( x * x + y * y ) + t );
    }

    public void init() {
        this.colorString = "Orange";
        this.xLowerBound = -15;
        this.xUpperBound = 15;
        this.yLowerBound = -15;
        this.yUpperBound = 15;
        this.gridSize = 0.3;
    }

    public double getT() {
        return t;
    }

    public void setT( double t ) {
        this.t = t;
    }
}

class MyParameteredCurve extends AjpUserParameteredCurve {
    // We set a second parameter.
    private double u;

    public AjpUser3DPoint evaluate( double t ) {
        double R=1.0;
        double r=0.2;
        double n=17.0/2;
        return new AjpUser3DPoint( ( R + r * Math.cos( n * t + u ) ) * Math.cos(
            t + u ) * 15,
            ( R + r * Math.cos( n * t + u ) ) * Math.sin( t + u ) * 15,

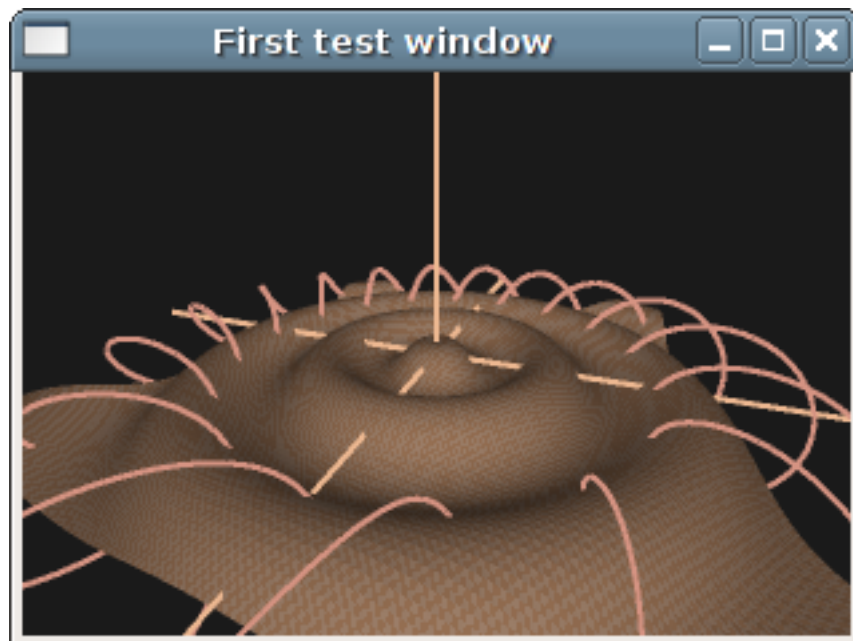
```



```
        ( r * Math.sin( n * t + u ) * 15 ) );  
    }  
  
    public void init() {  
        this.color = new AjpUserColor( .9f, 0.8f, 0.9f, 1.0f );  
        this.tLowerBound = -7;  
        this.tUpperBound = 7;  
        this.step = 0.01;  
    }  
  
    public double getU() {  
        return u;  
    }  
  
    public void setU( double u ) {  
        this.u = u;  
    }  
}  
}
```

Listing 3.2 – L'exemple Example2.java

Cet exemple produit le résultat suivant :



3.3 La classe **Example3**

Cette classe crée une surface paramétrée.

```
package org.lozi.ajp.ajp.scripts;  
  
import org.lozi.ajp.ajp.*;  
  
public class Example3 implements AjpUserScript  
{
```

```

public void start( AjpUserPlotter plotter ) {
    // We create the lights.
    AjpUserLight[] lights = new AjpUserLight[] {
        new AjpUserLight( new AjpUser3DPoint( 0, 0, 5 ),
            new AjpUserColor( .5f, .5f, .5f, 1f ) ) };

    // We create the window.
    AjpUserWindow window = plotter.newWindow( "First test window", 320, 240,
        10, 10, lights );
    // We set the background color.
    window.setBackgroundColor( new AjpUserColor( 0.1f, 0.1f, 0.1f, 1.0f ) );

    // We add the parametered surface.
    window.addParameteredSurface( new AjpUserParameteredSurface() {
        public AjpUser3DPoint evaluate( double s, double t ) {
            return new AjpUser3DPoint( s, t, s*s );
        }
    }

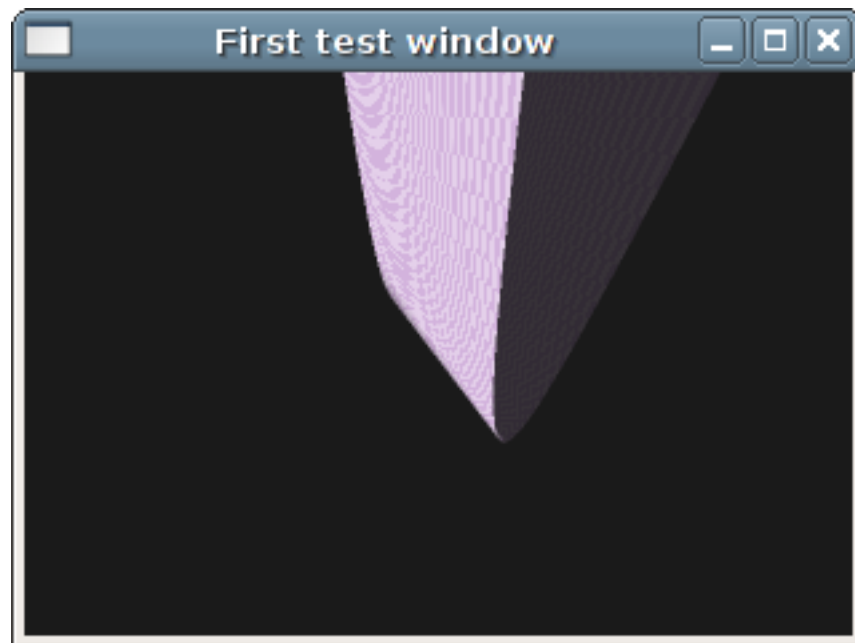
    public void init() {
        this.colorString = "Purple";
        this.sLowerBound = -10;
        this.sUpperBound = 10;
        this.tLowerBound = -10;
        this.tUpperBound = 10;
        this.sStep = 0.3f;
        this.tStep = 0.6f;
    }
    });

    window.setRotationSteps( 0.1, 0.0 );
    window.animate();
}
}

```

Listing 3.3 – L'exemple Example3.java

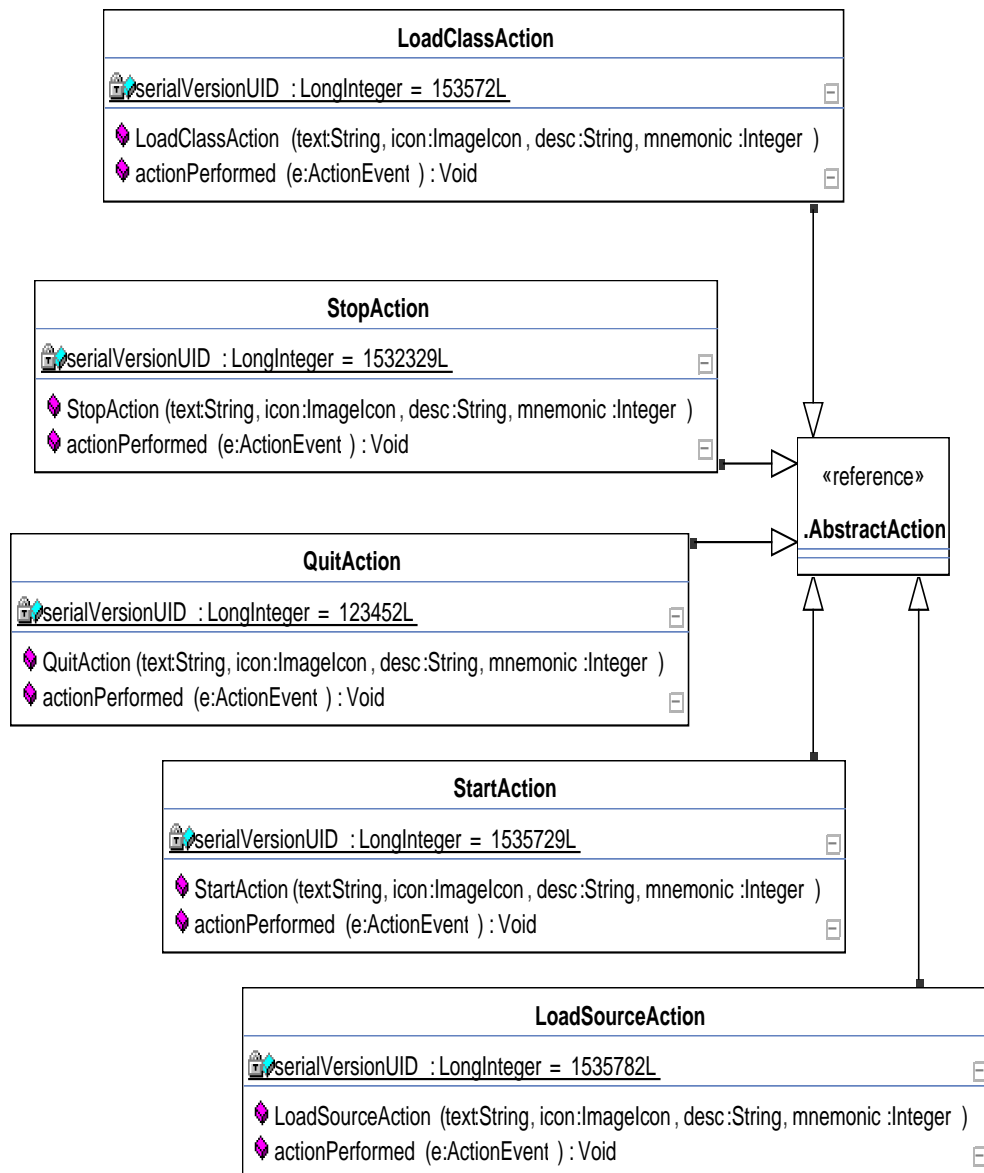
Cet exemple produit le résultat suivant :

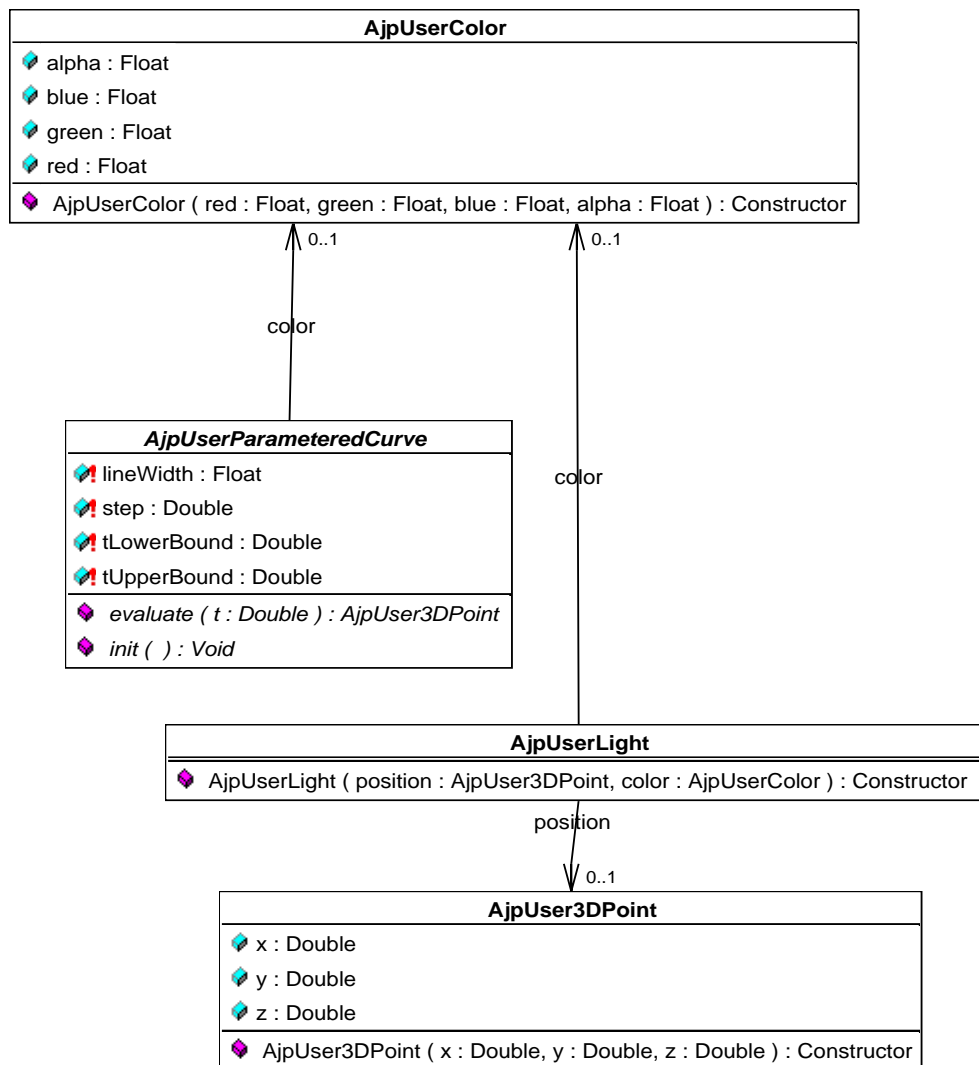


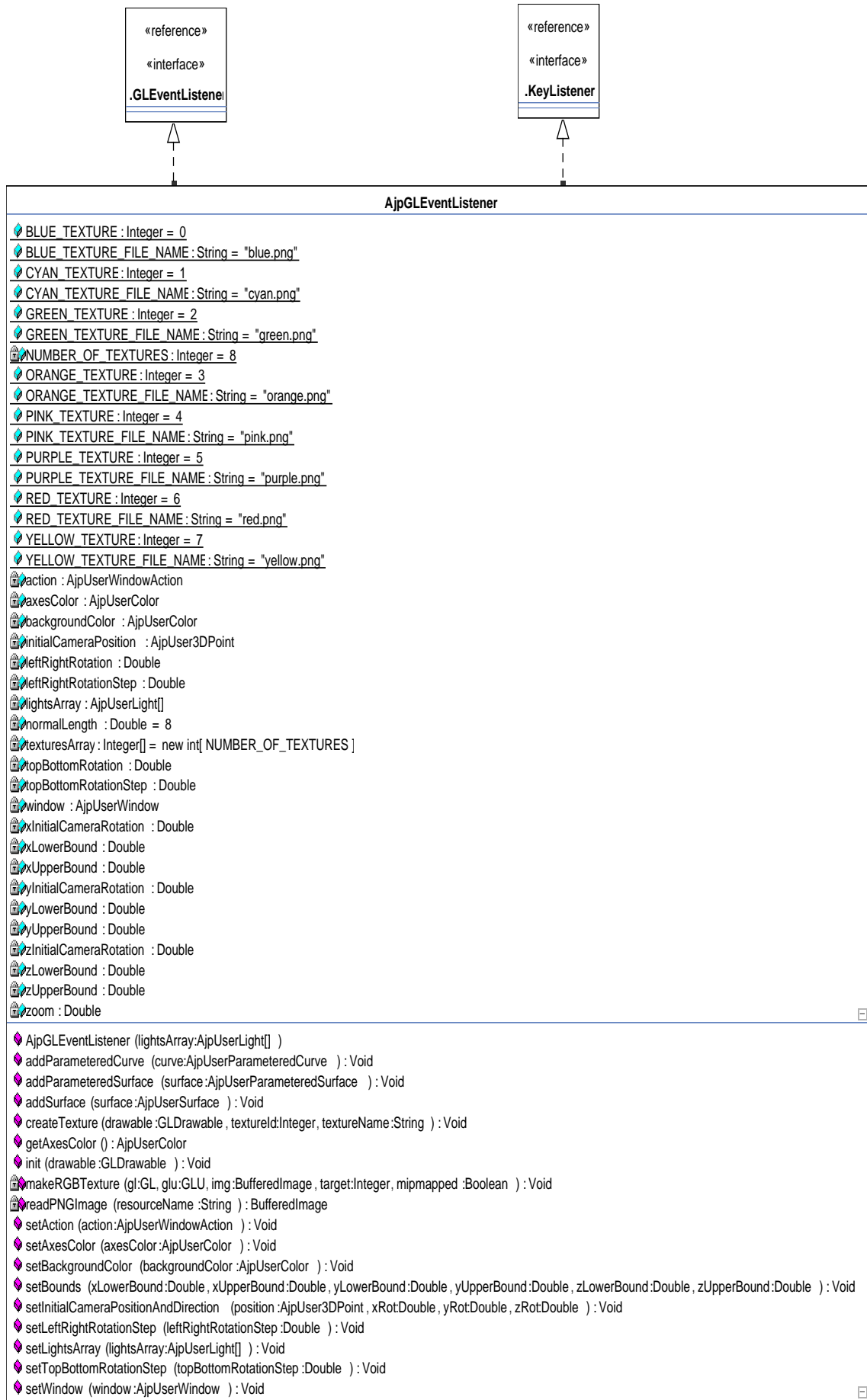
Chapitre 4

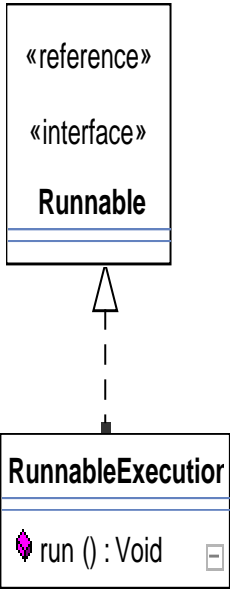
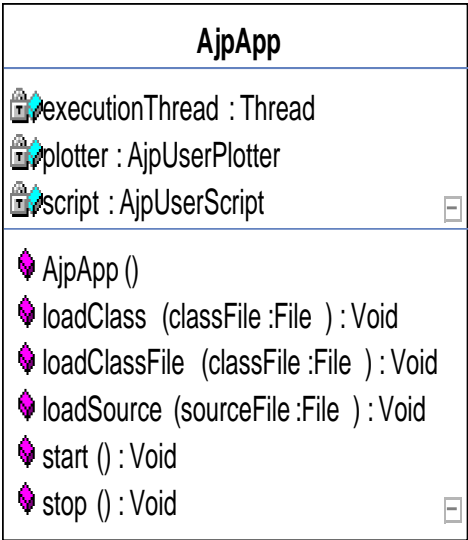
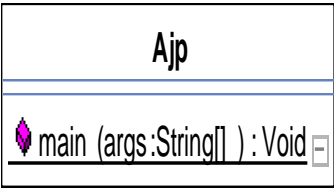
Diagramme De Classes

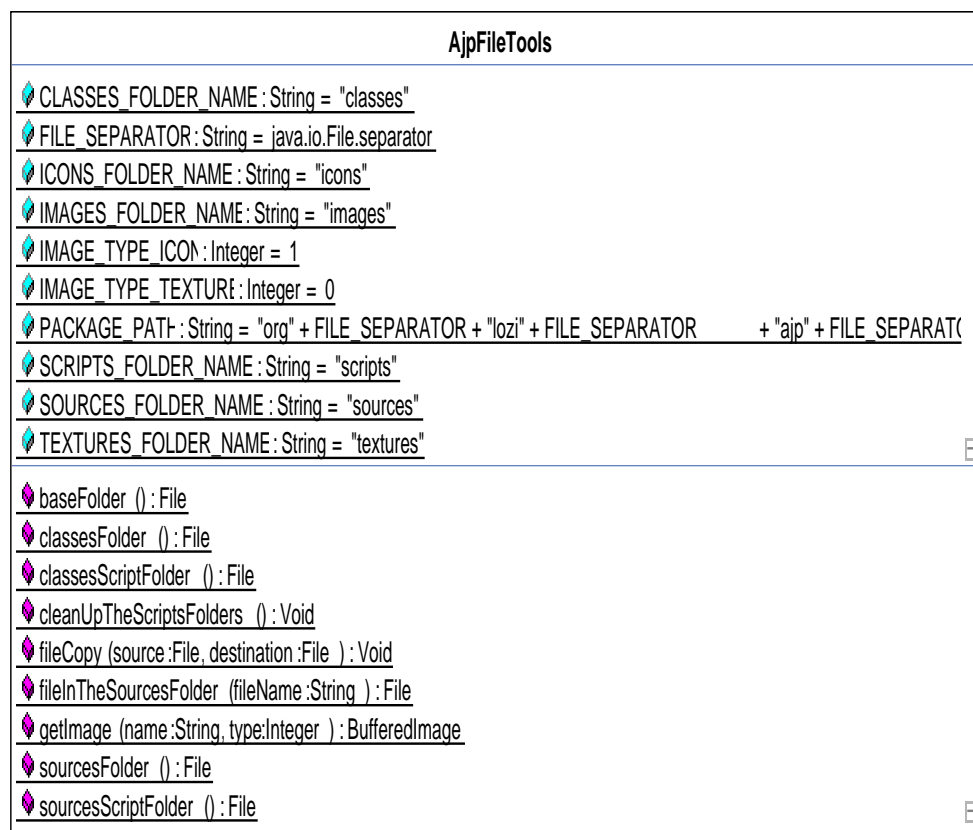
Ce chapitre contient le diagramme de classes de l'application.

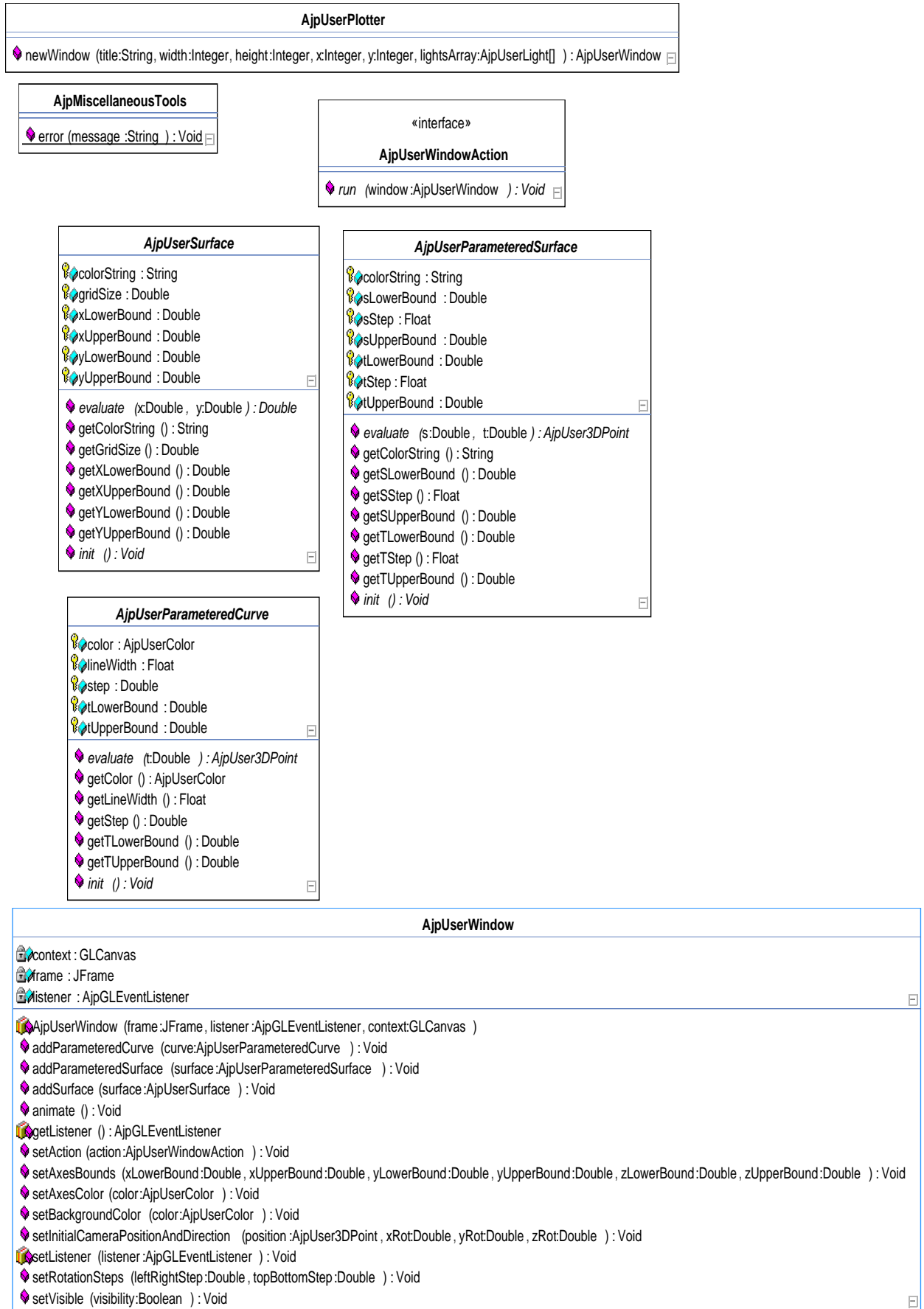












Chapitre 5

Listings

Ce chapitre contient l'intégralité du code de l'application.

```
package org.lozi.ajp.ajp;

/**
 * Ajp class.
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
 * @version 1.0
 */
public class Ajp
{
    public static void main( String[] args )
    {
        // We create the application.
        new AjpApp();
    }
}
```

Listing 5.1 – Le fichier Ajp.java

```
package org.lozi.ajp.ajp;

import org.lozi.ajp.ajp.scripts.*;
import java.io.File;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLClassLoader;
import java.util.ArrayList;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import javax.swing.JFrame;

/**
 * AjpApp class.
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
 * @version 1.0
 */
```

```

public class AjpApp {
    /** The current script to execute. */
    private AjpUserScript script;
    /** The thread where the script is executed. */
    private Thread executionThread;
    /** The plotter that we will pass to the user. */
    private AjpUserPlotter plotter;
    /** The frame list. */
    private static ArrayList<AjpUserWindow> windowList;

    /**
     * Returns an instance of the application.
     */
    public AjpApp() {
        // We first clean up the two scripts folders.
        AjpFileTools.cleanUpTheScriptsFolders();
        // We create the GUI.
        AjpGUI gui = new AjpGUI();
        // The GU needs a pointer to the application in order to
        // call the application functions.
        gui.init( this );
        // Now we initialize the thread.
        executionThread = new Thread( new RunnableExecution() );

        windowList = new ArrayList<AjpUserWindow>();

        plotter = new AjpUserPlotter();

        // Before we exit...
        Runtime.getRuntime().addShutdownHook(new Thread () {
            public void run() {
                // ...we clean up the scripts folders.
                AjpFileTools.cleanUpTheScriptsFolders();
            }
        });
    }

    /**
     * Returns the script.
     *
     * @return Returns the script.
     */
    public AjpUserScript getScript() {
        return script;
    }

    /**
     * Sets the script.
     *
     * @param script The script to set.
     */
    public void setScript( AjpUserScript script ) {
        this.script = script;
    }
}

```

```

/**
 * Loads the given class.
 *
 * @param classFile The class to load.
 */
public void loadClass( File classFile ) {
    // The file where the class will be copied.
    File newClassFile = new File( AjpFileTools.classesScriptFolder()
        + AjpFileTools.FILE_SEPARATOR + classFile.getName() );

    try {
        // We try to copy the file to the scripts folder.
        AjpFileTools.fileCopy( classFile, newClassFile );
    }
    catch( IOException ex ) {
        // If we could not copy the file we can not continue.
        AjpMiscellaneousTools.error( commentstyle"commentstyleUnable
            commentstyle commentstyleto commentstyle commentstylecopy
            commentstyle commentstylethe commentstyle commentstyleclass
            commentstyle commentstylefile commentstyle.commentstyle
            commentstylePlease commentstyle commentstylecheck commentstyle
            commentstylethat commentstyle commentstyleyou commentstyle
            commentstylehave commentstyle commentstylethe commentstyle
            commentstylewriting commentstyle commentstylerights commentstyle
            commentstylethe commentstyle commentstylescripts commentstyle
            commentstylefolder commentstyle.commentstyle" );
        return;
    }

    loadClassFile( newClassFile );
}

/**
 * Loads the given source file.
 *
 * @param sourceFile The source file to load.
 */
public void loadSource( File sourceFile ) {

    // The file where the source will be copied.
    File newSourceFile = new File( AjpFileTools.sourcesScriptFolder()
        + AjpFileTools.FILE_SEPARATOR + sourceFile.getName() );

    try {
        // We try to copy the file to the scripts folder.
        AjpFileTools.fileCopy( sourceFile, newSourceFile );
    }
    catch( IOException ex ) {
        // If we could not copy the file we can not continue.
        AjpMiscellaneousTools.error( commentstyle"commentstyleUnable
            commentstyle commentstyleto commentstyle commentstylecopy
            commentstyle commentstylethe commentstyle commentstylesource
            commentstyle commentstylefile commentstyle.commentstyle

```

```

        commentstylePleasecommentstyle commentstylecheckcommentstyle
        commentstylethatcommentstyle commentstyleyoucommentstyle
        commentstylehavecommentstyle commentstylethecommentstyle
        commentstylewritingcommentstyle commentstylerightscommentstyle
        commentstylethecommentstyle commentstylescriptscommentstyle
        commentstylefoldercommentstyle.commentstyle" );
    }
    return;
}

Process compilation = null;

try {

    System.out.flush();
    // We try compile the source file.
    compilation = Runtime.getRuntime().exec( new String[] { commentstyle"
        commentstylejavacommentstyle", commentstyle"commentstyle-
        commentstyleedcommentstyle", AjpFileTools.classesFolder().getPath(),
        commentstyle"commentstyle-commentstyleclasspathcommentstyle",
        AjpFileTools.classesFolder().getPath(),
        AjpFileTools.sourcesScriptFolder().toString() + AjpFileTools.
        FILE_SEPARATOR + newSourceFile.getName().toString() } , null,
        new File( AjpFileTools.baseFolder().getPath() ) );

    // We try to get the compilation errors.
    try {
        BufferedReader br = new BufferedReader( new InputStreamReader(
            compilation.getErrorStream() ) );

        String line = null;
        String errors = commentstyle"commentstyle";

        while ( (line = br.readLine()) != null)
            errors += line + commentstyle"commentstyle\commentstylen
                commentstyle";

        // Now we display the compilation errors (if any).
        if( errors.toString().length() > 1) {
            System.out.println(commentstyle"commentstyle####commentstyle
                commentstyleCOMPILATIONcommentstyle commentstyleERRORS
                commentstyle commentstyle####\commentstylencommentstyle" +
                errors );
            AjpMiscellaneousTools.error( commentstyle"commentstyleUnable
                commentstyle commentstyletocommentstyle commentstylecompile
                commentstyle commentstylethecommentstyle commentstylesource
                commentstyle commentstylefilecommentstyle!\commentstylen
                commentstyle\commentstylencommentstyle" + errors );
        }

    } catch ( Exception ex ) {
        // If we could not get the compilation error, we display a simple
        message.
        AjpMiscellaneousTools.error( commentstyle"commentstyleUnable
            commentstyle commentstyletocommentstyle commentstyleget

```

```

        commentstyle commentstylethecommentstyle commentstylecompilation
        commentstyle commentstyleerrorscommentstyle.commentstyle" );
    }

    // We wait for the command to end.
    compilation.waitFor();

} catch (Exception e) {
    // If we could not compile the file we can not continue.
    AjpMiscellaneousTools.error( commentstyle"commentstyleUnable
        commentstyle commentstyletocommentstyle commentstylecompile
        commentstyle commentstylethecommentstyle commentstylesource
        commentstyle commentstylefilecommentstyle!commentstyle" );

    return;
}

// We try to get the compilation errors.
try {
    StringBuffer errorStr = new StringBuffer( commentstyle"commentstyle" );

    // If the compilation object was initialized...
    if( compilation != null ) {
        byte[] buffer = new byte[1024];

        // We read the error stream.
        while( compilation.getErrorStream().read( buffer ) != -1 )
        {
            errorStr.append( buffer );
        }
    }

    // Now we display the compilation errors (if any).
    if( errorStr.toString().length() > 1 ) {
        System.out.println(commentstyle"commentstyle##commentstyle" +
            errorStr );
        AjpMiscellaneousTools.error( commentstyle"commentstyleUnable
            commentstyle commentstyletocommentstyle commentstylecompile
            commentstyle commentstylethecommentstyle commentstylesource
            commentstyle commentstylefilecommentstyle!\commentstylen
            commentstyle\commentstylencommentstyle" + errorStr );
    }
} catch ( Exception ex ) {
    // If we could not get the compilation error, we display a simple
    message.
    AjpMiscellaneousTools.error( commentstyle"commentstyleUnable
        commentstyle commentstyletocommentstyle commentstyleget
        commentstyle commentstylethecommentstyle commentstylecompilation
        commentstyle commentstyleerrorscommentstyle.commentstyle" );
}

// Now we get the class file.
File classFile = new File( AjpFileTools.classesScriptFolder()

```

```

        + AjpFileTools.FILE_SEPARATOR + sourceFile.getName().
        substring( 0, sourceFile.getName().length() -
        commentstyle"commentstyle.commentstylejavacommentstyle".
        length() ) + commentstyle"commentstyle.commentstyleclass
        commentstyle" );

    loadClassFile( classFile );
}

/**
 * Loads the given class file, supposing it is in the right directory.
 *
 * @param classFile The class file to load.
 */
public void loadClassFile( File classFile ) {
    // Now we create a class loader.
    URLClassLoader classLoader;

    try {
        // We create a new class loader.
        classLoader = URLClassLoader.newInstance( new URL[] { new URL (
            commentstyle"commentstylefilecommentstyle:commentstyle" +
            AjpFileTools.classesFolder().getPath() + commentstyle"commentstyle/
            commentstyle" ) } );

    } catch (MalformedURLException e) {
        // If we could not, we display an error and return.
        AjpMiscellaneousTools.error( commentstyle"commentstyleUnable
            commentstyle commentstyletocommentstyle commentstyleinstantiate
            commentstyle commentstyleacommentstyle commentstyleclass
            commentstyle commentstyleloadercommentstyle.commentstyle" );
        return;
    }

    Class<?> newClass;

    try {
        // We load the new class.
        newClass = classLoader.loadClass( commentstyle"commentstyleorg
            commentstyle.commentstylelozicommentstyle.commentstyleajp
            commentstyle.commentstyleajpcommentstyle.commentstylescripts
            commentstyle.commentstyle" + classFile.getName().substring( 0,
            classFile.getName().length() - commentstyle"commentstyle.
            commentstyleclasscommentstyle".length() ) );
        //newClass = Class.forName( "org.lozi.ajp.ajp.Example1" );
    } catch (ClassNotFoundException e) {
        // If we could not, we display an error and return.
        AjpMiscellaneousTools.error( commentstyle"commentstyleUnable
            commentstyle commentstyletocommentstyle commentstyleload
            commentstyle commentstylethecommentstyle commentstylenew
            commentstyle commentstyleclasscommentstyle.commentstyle" + e.
            getMessage() );
        return;
    }
}

```

```

Object script;

try {
    // We create an instance of this class.
    script = newClass.newInstance();

    // Then we set our script to the newly created object.
    this.script = ( AjpUserScript )script;
} catch ( Exception ex ) {
    AjpMiscellaneousTools.error( commentstyle"commentstyleUnable
        commentstyle commentstyletocommentstyle commentstyleinstantiate
        commentstyle commentstylethecommentstyle commentstylenew
        commentstyle commentstyleclasscommentstyle.commentstyle" + ex.
        getMessage() );
    return;
}

// Hey! Did this *** user try to load a class which is not an
// AjpUserScript?
if( ! ( script instanceof org.lozi.ajp.ajp.scripts.AjpUserScript ) ) {
    // **** off!
    AjpMiscellaneousTools.error( commentstyle"commentstyleThiscommentstyle
        commentstyleclasscommentstyle commentstyledoescommentstyle
        commentstylenotcommentstyle commentstyleimplementcommentstyle
        commentstyleAjpUserScriptcommentstyle!commentstyle commentstyle" +
        script.getClass().getName() );
    // Such a mofo >_<
    return;
}

/**
 * Starts the execution.
 */
public void start () {
    // Pretty self explanatory...
    executionThread.run();
}

/**
 * Stops the execution.
 */
public void stop () {
    for( AjpUserWindow frame : windowList )
        frame.close();
}

/**
 * This class
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
 */
public class RunnableExecution implements Runnable {

```



```

/**
 * This function runs the thread.
 */
public void run() {
    script.start( plotter );
}
}

/**
 * Adds a frame.
 *
 * @param frame The frame to add.
 */
public static void addFrame ( AjpUserWindow frame ) {
    windowList.add( frame );
}
}

```

Listing 5.2 – Le fichier AjpApp.java

```

package org.lozi.ajp.ajp;

import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.net.URL;
import javax.imageio.ImageIO;

/**
 * AjpFileTools class. Contains miscellaneous tools for the Ajp project.
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
 * @version 1.0
 */
public class AjpFileTools {
    /** The OS dependant file separator. */
    public static final String FILE_SEPARATOR = java.io.File.separator;
    /** The package's path. */
    public static final String PACKAGE_PATH = commentstyle"commentstyleorg
commentstyle" + FILE_SEPARATOR + commentstyle"commentstylelozi
commentstyle" + FILE_SEPARATOR
        + commentstyle"commentstyleajpcommentstyle" +
        FILE_SEPARATOR + commentstyle"commentstyleajp
commentstyle";
    /** The name of the folder containing the scripts. */
    public static final String SCRIPTS_FOLDER_NAME = commentstyle"
commentstylescriptscommentstyle";
    /** The name of the folder containing the classes. */

```

```

public static final String CLASSES_FOLDER_NAME = commentstyle"
    commentstyleclassescommentstyle";
/** The name of the folder containing the sources. */
public static final String SOURCES_FOLDER_NAME = commentstyle"
    commentstylesourcescommentstyle";
/** The name of the folder containing the images. */
public static final String IMAGES_FOLDER_NAME = commentstyle"
    commentstyleimagescommentstyle";
/** The name of the folder containing the textures. */
public static final String TEXTURES_FOLDER_NAME = commentstyle"
    commentstyletexturescommentstyle";
/** The name of the folder containing the icons. */
public static final String ICONS_FOLDER_NAME = commentstyle"
    commentstyleiconscommentstyle";
/** The "texture" image type. */
public static final int IMAGE_TYPE_TEXTURE = 0;
/** The "icon" image type. */
public static final int IMAGE_TYPE_ICON = 1;

/**
 * Gets an image from its name. Automatically resolves the image file,
 * and returns the Image object.
 *
 * @param name The image name.
 * @param type The image type.
 * @return The BufferedImage object, null if it could not be found.
 */
public static BufferedImage getImage( String name, int type ) {
    // The images folder is in the classes folder's parent.
    URL imageURL;
    try {
        // The images folder is in the classes folder's parent.
        imageURL = new URL( commentstyle"commentstylefilecommentstyle:
            commentstyle" + baseFolder().getPath() + FILE_SEPARATOR +
            IMAGES_FOLDER_NAME
            + FILE_SEPARATOR + ( type == IMAGE_TYPE_TEXTURE ?
                TEXTURES_FOLDER_NAME : ICONS_FOLDER_NAME )
            + FILE_SEPARATOR + name );

        // We create the image and return it.
        return ImageIO.read( imageURL );
    } catch (Exception e) {
        // We return null if we could not locate the file.
        return null;
    }
}

/**
 * Returns the "sources" folder.
 *
 * @return The "sources" folder.
 */
public static File sourcesFolder() {

```

```

// We first get the classes' base URL.
URL classesBaseURL = ClassLoader.getResource(commentstyle"
    commentstyle");
// Then we return the File object of the "sources" folder.
return new File( classesBaseURL.getPath().replaceAll(commentstyle"
    commentstyle%20commentstyle",commentstyle"commentstyle commentstyle")
    .substring( 0, classesBaseURL.getPath().replaceAll(commentstyle"
    commentstyle%20commentstyle",commentstyle"commentstyle commentstyle")
    .length() - CLASSES_FOLDER_NAME.length() - 1 )
    + FILE_SEPARATOR + SOURCES_FOLDER_NAME );
}

/**
 * Returns the "scripts" folder within the "sources" folder.
 *
 * @return The "scripts" folder within the "sources" folder.
 */
public static File sourcesScriptFolder() {
    return new File( sourcesFolder().getPath() + FILE_SEPARATOR +
        PACKAGE_PATH + FILE_SEPARATOR + SCRIPTS_FOLDER_NAME );
}

/**
 * Returns the "classes" folder.
 *
 * @return The "classes" folder.
 */
public static File classesFolder() {
    // We first get the classes' base URL.
    URL classesBaseURL = ClassLoader.getResource(commentstyle"
        commentstyle");
    // Then we return the File object of the "classes" folder.
    return new File( classesBaseURL.getPath().replaceAll(commentstyle"
        commentstyle%20commentstyle",commentstyle"commentstyle commentstyle")
        .substring( 0, classesBaseURL.getPath().replaceAll(commentstyle"
        commentstyle%20commentstyle",commentstyle"commentstyle commentstyle")
        .length() - CLASSES_FOLDER_NAME.length() - 1 )
        + FILE_SEPARATOR + CLASSES_FOLDER_NAME );
}

/**
 * Returns the "scripts" folder within the "classes" folder.
 *
 * @return The "scripts" folder within the "classes" folder.
 */
public static File classesScriptFolder() {
    return new File( classesFolder().getPath() + FILE_SEPARATOR +
        SCRIPTS_FOLDER_NAME );
}

/**
 * Returns the "ajp" folder.
 *
 * @return The "ajp" folder.

```

```

*/
public static File baseFolder() {
    // We first get the classes' base URL.
    URL classesBaseURL = ClassLoader.getResource(commentstyle"
        commentstyle");

    // Then we return the File object of the "ajp" folder.
    return new File( classesBaseURL.getPath().replaceAll(commentstyle"
        commentstyle%20commentstyle",commentstyle"commentstyle commentstyle")
        .substring( 0, classesBaseURL.getPath().replaceAll(commentstyle"
        commentstyle%20commentstyle",commentstyle"commentstyle commentstyle")
        .length() - CLASSES_FOLDER_NAME.length() - 1 ) );
}

/**
 * Returns the file which name is given as a parameter, assuming that the
 * file
 * belongs to the "sources" folder.
 *
 * @param fileName The file name.
 * @return The file object corresponding to the file.
 */
public static File fileInTheSourcesFolder( String fileName ) {
    // Then we return the File object of the file whose name was given as a
    parameter.

    return new File( sourcesFolder().getPath() + fileName );
}

/**
 * Copies a file from a destination to another.
 *
 * @param source The file to copy.
 * @param destination The destination file.
 * @throws IOException
 */
public static void fileCopy( File source, File destination ) throws
    IOException {

    // We first create the reader and the writer.
    BufferedReader reader = new BufferedReader( new FileReader( source ) );
    BufferedWriter writer = new BufferedWriter( new FileWriter( destination
        ) );

    // We initialize the buffer used to copy the file.
    char[] buffer = new char[32768];

    int read;

    // We simply copy the file.
    while( ( read = reader.read( buffer ) ) != -1 )
        writer.write( buffer, 0, read );

    // Then we close the buffers.

```

```

    reader.close();
    writer.close();
}

/**
 * Cleans up the scripts' folders in the project path.
 * This function should be called when the application starts and when it
 *   exits.
 */
public static void cleanUpTheScriptsFolders() {

    // We get the sources' scripts folder.
    File sourcesScriptsFolder = new File( sourcesFolder().getPath()
        + FILE_SEPARATOR + PACKAGE_PATH + FILE_SEPARATOR +
        SCRIPTS_FOLDER_NAME );
    // We get the classes' scripts folder.
    File classesScriptsFolder = new File( classesFolder().getPath()
        + FILE_SEPARATOR + PACKAGE_PATH + FILE_SEPARATOR +
        SCRIPTS_FOLDER_NAME );

    if( !sourcesScriptsFolder.isDirectory() ) AjpMiscellaneousTools.error(
        commentstyle"commentstyleUnablecommentstyle commentstyleto
        commentstyle commentstylefindcommentstyle commentstylethe
        commentstyle commentstylescriptscommentstyle commentstylefolder
        commentstyle commentstylewithincommentstyle commentstylethe
        commentstyle commentstylesourcescommentstyle commentstyledirectory
        commentstyle.commentstyle" );
    if( !classesScriptsFolder.isDirectory() ) AjpMiscellaneousTools.error(
        commentstyle"commentstyleUnablecommentstyle commentstyleto
        commentstyle commentstylefindcommentstyle commentstylethe
        commentstyle commentstylescriptscommentstyle commentstylefolder
        commentstyle commentstylewithincommentstyle commentstylethe
        commentstyle commentstyleclassescommentstyle commentstyledirectory
        commentstyle.commentstyle" );

    if( sourcesScriptsFolder.listFiles() != null ) {
        // We delete each file in the first folder.
        for( File currentFile : sourcesScriptsFolder.listFiles() )
            if( !currentFile.getName().equals(commentstyle"
                commentstyleAjpUserScriptcommentstyle.commentstylejava
                commentstyle") && !currentFile.getName().equals(commentstyle"
                commentstyleAjpScriptLoadercommentstyle.commentstylejava
                commentstyle") )
                currentFile.delete();
    }

    if( classesScriptsFolder.listFiles() != null ) {
        // We delete each file in the second folder.
        for( File currentFile : classesScriptsFolder.listFiles() )
            if( !currentFile.getName().equals(commentstyle"
                commentstyleAjpUserScriptcommentstyle.commentstyleclass
                commentstyle") && !currentFile.getName().equals(commentstyle"
                commentstyleAjpScriptLoadercommentstyle.commentstyleclass
                commentstyle") )

```

```

        currentFile.delete();
    }

}
}

```

Listing 5.3 – Le fichier AjpFileTools.java

```

package org.lozi.ajp.ajp;

import static java.lang.Math.*;
import net.java.games.jogl.*;
import static net.java.games.jogl.GL.*;

import net.java.games.jogl.util.*;

import java.awt.event.*;
import java.awt.image.*;
import java.nio.*;
import java.util.ArrayList;

/**
 * AjpGLEventListener class.
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
 * @version 1.0
 */
public class AjpGLEventListener implements GLEventListener, KeyListener
{
    /** The number of textures. */
    private static final int NUMBER_OF_TEXTURES = 8;

    /** The id of the blue texture. */
    public static final int BLUE_TEXTURE = 0;
    /** The file name of the blue texture. */
    public static final String BLUE_TEXTURE_FILE_NAME = commentstyle"
        commentstylebluecommentstyle.commentstylepngcommentstyle";
    /** The id of the cyan texture. */
    public static final int CYAN_TEXTURE = 1;
    /** The file name of the cyan texture. */
    public static final String CYAN_TEXTURE_FILE_NAME = commentstyle"
        commentstylecyancommentstyle.commentstylepngcommentstyle";
    /** The green texture id. */
    public static final int GREEN_TEXTURE = 2;
    /** The file name of the green texture. */
    public static final String GREEN_TEXTURE_FILE_NAME = commentstyle"
        commentstylegreengcommentstyle.commentstylepngcommentstyle";
    /** The orange texture id. */
    public static final int ORANGE_TEXTURE = 3;
    /** The file name of the orange texture. */
    public static final String ORANGE_TEXTURE_FILE_NAME = commentstyle"
        commentstyleorangecommentstyle.commentstylepngcommentstyle";
    /** The pink texture id. */
    public static final int PINK_TEXTURE = 4;

```

```

/** The file name of the pink texture. */
public static final String PINK_TEXTURE_FILE_NAME = commentstyle"
    commentstylepinkcommentstyle.commentstylepngcommentstyle";
/** The purple texture id. */
public static final int PURPLE_TEXTURE = 5;
/** The file name of the purple texture. */
public static final String PURPLE_TEXTURE_FILE_NAME = commentstyle"
    commentstylepurplecommentstyle.commentstylepngcommentstyle";
/** The red texture id. */
public static final int RED_TEXTURE = 6;
/** The file name of the red texture. */
public static final String RED_TEXTURE_FILE_NAME = commentstyle"
    commentstyleredcommentstyle.commentstylepngcommentstyle";
/** The yellow texture id. */
public static final int YELLOW_TEXTURE = 7;
/** The file name of the yellow texture. */
public static final String YELLOW_TEXTURE_FILE_NAME = commentstyle"
    commentstyleyellowcommentstyle.commentstylepngcommentstyle";

/** This array contains the different textures. */
private int[] texturesArray = new int[ NUMBER_OF_TEXTURES ];

/** The left/right rotation step. */
private double leftRightRotationStep;
/** The top/bottom rotation step. */
private double topBottomRotationStep;
/** The zoom step. */
private double zoom;
/** The surfaces list. */
private ArrayList<AjpUserSurface> surfacesArrayList;
/** The list of the parametered curves. */
private ArrayList<AjpUserParameteredCurve> parameteredCurvesArrayList;
/** The list of the parametered surfaces. */
private ArrayList<AjpUserParameteredSurface> parameteredSurfacesArrayList;
/** The list of the parametered surfaces. */
private ArrayList<AjpUserDot> dotsArrayList;
/** The x lower bound. */
private double xLowerBound;
/** The x upper bound. */
private double xUpperBound;
/** The y lower bound. */
private double yLowerBound;
/** The y upper bound. */
private double yUpperBound;
/** The z lower bound. */
private double zLowerBound;
/** The z upper bound. */
private double zUpperBound;
/** The normals' lengths. */
private double normalLength = 8;
/** The background color. */
private AjpUserColor backgroundColor;
/** The axes' color. */
private AjpUserColor axesColor;

```

```

/** The left/right rotation. */
private double leftRightRotation;
/** The top/bottom rotation. */
private double topBottomRotation;
/** The lights' array. */
private AjpUserLight[] lightsArray;
/** The window. */
private AjpUserWindow window;
/** The window's action. */
private AjpUserWindowAction action;
/** The initial camera position. */
private AjpUser3DPoint initialCameraPosition;
/** The x initial rotation. */
private double xInitialCameraRotation;
/** The y initial rotation. */
private double yInitialCameraRotation;
/** The z initial rotation. */
private double zInitialCameraRotation;

/**
 * Builds a new AjpGLEventListener.
 *
 * @param lightsArray The lights.
 */
public AjpGLEventListener( AjpUserLight[] lightsArray ) {
    this.lightsArray = lightsArray;
}

/**
 * Adds the given surface.
 *
 * @param surface The surface to add.
 */
public synchronized void addSurface ( AjpUserSurface surface ) {
    surfacesArrayList.add( surface );
}

/**
 * Adds the given parametric curve (one parameter).
 *
 * @param curve The curve to add.
 */
public synchronized void addParameteredCurve ( AjpUserParameteredCurve
    curve ) {
    parameteredCurvesArrayList.add( curve );
}

/**
 * Adds the given parametric surface (two parameters).
 *
 * @param surface The surface to add.
 */
public synchronized void addParameteredSurface ( AjpUserParameteredSurface
    surface ) {

```



```

parameteredSurfacesArrayList.add( surface );
}

/**
 * Adds the given parametric dot.
 *
 * @param dot The dot to add.
 */
public synchronized void addDot ( AjpUserDot dot ) {
    dotsArrayList.add( dot );
}

/**
 * Sets the drawing bounds.
 *
 * @param xLowerBound The x lower bound.
 * @param xUpperBound The x upper bound.
 * @param yLowerBound The y lower bound.
 * @param yUpperBound The y upper bound.
 * @param zLowerBound The z lower bound.
 * @param zUpperBound The z upper bound.
 */
public void setBounds( double xLowerBound, double xUpperBound, double
    yLowerBound, double yUpperBound, double zLowerBound,
    double zUpperBound ) {
    this.xLowerBound = xLowerBound;
    this.xUpperBound = xUpperBound;
    this.yLowerBound = yLowerBound;
    this.yUpperBound = yUpperBound;
    this.zLowerBound = zLowerBound;
    this.zUpperBound = zUpperBound;
}

/**
 * @param backgroundColor The background color to set.
 */
public void setBackgroundColor(AjpUserColor backgroundColor) {
    this.backgroundColor = backgroundColor;
}

/**
 * This function quickly creates a texture.
 *
 * @param drawable The drawable.
 * @param textureID The texture id.
 * @param textureName The texture name.
 */
public void createTexture( GLDrawable drawable, int textureId, String
    textureName ) {
    drawable.getGL().glBindTexture( GL_TEXTURE_2D, texturesArray[ textureId
    ] );
    drawable.getGL().glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
        GL_LINEAR );

```

```

drawable.getGL().glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
    GL_LINEAR );
makeRGBTexture( drawable.getGL(), drawable.getGLU(), readPNGImage(
    textureName ), GL_TEXTURE_2D, true );
}

/**
 * Initializes the event listener.
 *
 * @param drawable The GLDrawable where to draw.
 */
public void init ( GLDrawable drawable )
{
    // We initialize the surfaces array.
    surfacesArrayList = new ArrayList<AjpUserSurface>();
    parameteredCurvesArrayList = new ArrayList<AjpUserParameteredCurve>();
    parameteredSurfacesArrayList = new ArrayList<AjpUserParameteredSurface
        >();
    dotsArrayList = new ArrayList<AjpUserDot>();

    if( initialCameraPosition == null ) initialCameraPosition = new
        AjpUser3DPoint( 0, 0, -20 );
    xInitialCameraRotation = -65;
    yInitialCameraRotation = 0;
    zInitialCameraRotation = -45;
    if( backgroundColor == null ) backgroundColor = new AjpUserColor( 1.0f,
        1.0f, 1.0f, 1.0f );
    // We first get the GL and GLU objects.
    final GL gl = drawable.getGL();
    final GLU glu = drawable.getGLU();

    // We need to listen to the key events.
    drawable.addKeyListener(this);

    // We set miscellaneous properties.
    gl.glClearColor( 0.0f, 0.0f, 0.0f, 0.0f );
    gl.glShadeModel( GL_SMOOTH );
    gl.glClearDepth( 1.0f );
    gl.glEnable( GL_DEPTH_TEST );
    gl.glDepthFunc( GL_LEQUAL );
    gl.glHint( GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST );

    int i = 0;

    for( AjpUserLight light : lightsArray ) {
        // No more than 7 lights!
        if( i == 7 ) break;

        // Lighting
        float LightAmbient[] = { 1f, 1f, 1f, 1.0f };
        float LightDiffuse[] = { light.getColor().getRed(), light.getColor().
            getGreen(), light.getColor().getBlue(), light.getColor().getAlpha()
        };
    }
}

```

```

    float LightPosition[] = { (float)light.getPosition().getX(), (float)
        light.getPosition().getY(), (float)light.getPosition().getZ(), 1.0f
        };

    int lightNumber;
    // Well the values provided by OpenGL are not in an array, so we have
    no choice.
    switch( i ) {
    case 0 : lightNumber = GL_LIGHT1; break;
    case 1 : lightNumber = GL_LIGHT2; break;
    case 2 : lightNumber = GL_LIGHT3; break;
    case 3 : lightNumber = GL_LIGHT4; break;
    case 4 : lightNumber = GL_LIGHT5; break;
    case 5 : lightNumber = GL_LIGHT6; break;
    case 6 : lightNumber = GL_LIGHT7; break;
    case 7 : lightNumber = GL_LIGHT0; break;
    default : lightNumber = GL_LIGHT1; break;
    }
    gl.glLightfv( lightNumber, GL_AMBIENT, LightAmbient );
    gl.glLightfv( lightNumber, GL_DIFFUSE, LightDiffuse );
    gl.glLightfv( lightNumber, GL_POSITION, LightPosition );
    gl.glEnable( lightNumber );

    i++;
}

gl.glEnable( GL_LIGHTING );
gl.glEnable( GL_TEXTURE_2D );

// Now we create the different textures.
gl.glGenTextures( NUMBER_OF_TEXTURES, this.texturesArray );
// One for each color :)
createTexture( drawable, BLUE_TEXTURE, BLUE_TEXTURE_FILE_NAME );
createTexture( drawable, CYAN_TEXTURE, CYAN_TEXTURE_FILE_NAME );
createTexture( drawable, GREEN_TEXTURE, GREEN_TEXTURE_FILE_NAME );
createTexture( drawable, ORANGE_TEXTURE, ORANGE_TEXTURE_FILE_NAME );
createTexture( drawable, PINK_TEXTURE, PINK_TEXTURE_FILE_NAME );
createTexture( drawable, PURPLE_TEXTURE, PURPLE_TEXTURE_FILE_NAME );
createTexture( drawable, RED_TEXTURE, RED_TEXTURE_FILE_NAME );
createTexture( drawable, YELLOW_TEXTURE, YELLOW_TEXTURE_FILE_NAME );

//gl.glColor4f(.5f,0.5f,0.5f,0.9f); // Full Brightness, 50% Alpha ( NEW )
//gl.glBlendFunc(GL_SRC_ALPHA, GL_ONE); // Blending Function For
    Translucency Based On Source Alpha Value ( NEW )

//gl.glEnable(GL_BLEND); // Turn Blending On
//gl.glDisable(GL_DEPTH_TEST);
}

/**
 * Converts a color string to a texture.
 *
 * @param color The color string to consider.

```

```

    * @return Its integer value.
    */
    public int colorStringToTexture ( String color ) {
        if( color.toLowerCase().equals(commentstyle"commentstyleblue
            commentstyle") ) return BLUE_TEXTURE;
        else if( color.toLowerCase().equals(commentstyle"commentstylecyan
            commentstyle") ) return CYAN_TEXTURE;
        else if( color.toLowerCase().equals(commentstyle"commentstylegreen
            commentstyle") ) return GREEN_TEXTURE;
        else if( color.toLowerCase().equals(commentstyle"commentstyleorange
            commentstyle") ) return ORANGE_TEXTURE;
        else if( color.toLowerCase().equals(commentstyle"commentstylepink
            commentstyle") ) return PINK_TEXTURE;
        else if( color.toLowerCase().equals(commentstyle"commentstylepurple
            commentstyle") ) return PURPLE_TEXTURE;
        else if( color.toLowerCase().equals(commentstyle"commentstylered
            commentstyle") ) return RED_TEXTURE;
        else if( color.toLowerCase().equals(commentstyle"commentstyleyellow
            commentstyle") ) return YELLOW_TEXTURE;
        else return 0;
    }

    /**
     * This function is called each time the drawable is drawn. :)
     *
     * @param drawable The GLDrawable where to draw.
     */
    public synchronized void display ( GLDrawable drawable )
    {
        // We run the action each time the frame is refreshed.
        if( action != null )

            action.run( window );

        // We get the gl object.
        final GL gl = drawable.getGL();

        if ( axesColor == null ) axesColor = new AjpUserColor( 1.0f, 1.0f, 1.0f,
            1.0f );

        gl.glClearColor( backgroundColor.getRed(), backgroundColor.getGreen(),
            backgroundColor.getBlue(), backgroundColor.getAlpha() );

        // We clear the drawing area.
        gl.glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
        // We start from the identity.
        gl.glLoadIdentity();

        // We update the camera.

        gl.glTranslated( initialCameraPosition.getX(), initialCameraPosition.
            getY(), initialCameraPosition.getZ() + (float)zoom );
        gl.glRotated( xInitialCameraRotation,1.0, 0.0, 0.0 );
        gl.glRotated( yInitialCameraRotation,0.0, 1.0, 0.0 );
    }

```

```

gl.glRotated( zInitialCameraRotation,0.0, 0.0, 1.0 );

gl.glRotated(leftRightRotation,0.0,0.0,1.0);
gl.glRotated(topBottomRotation,1.0,0.0,0.0);

// We want light!
gl.glEnable( GL_LIGHTING );

// For each surface...
for( AjpUserSurface surface : surfacesArrayList ) {
    // We initialize the surface.
    surface.init();

    // Those variable will represent the current function's value for each
    // vertex of the square.
    double fBottomLeft, fTopLeft, fBottomRight, fTopRight;

    // We get the grid size in the h variable for more lisibility.
    double h = surface.getGridSize();

    // We bind the right texture to this surface.
    gl.glBindTexture( GL_TEXTURE_2D, texturesArray[colorStringToTexture(
        surface.getColorString() )]);

    // We are going to draw the surface, using triangles.
    gl.glBegin( GL_TRIANGLES );

    // Now we are going to divide the (Ox,Oy) axis into
    // squares and to draw the surfaces using triangles.
    // We start from the x lower bound, and step by the grid size.
    for ( double x = surface.getXLowerBound() ; x <= surface.getXUpperBound
        () ; x += surface.getGridSize() ) {
        // We start from the y lower bound, and step by the grid size.
        for ( double y = surface.getYLowerBound() ; y <= surface.
            getYUpperBound() ; y += surface.getGridSize() ) {
            //
            // Seen from above : [^y >x]
            //
            // fTopLeft +---+ fTopRight
            //           | \ |
            // fBottomLeft +---+ fBottomRight
            //
            // We compute the values of the 4 first points ( the lower square ).
            fBottomLeft = surface.evaluate( x, y );
            fTopLeft = surface.evaluate( x + h, y );
            fBottomRight = surface.evaluate( x, y + h );
            fTopRight = surface.evaluate( x + h, y + h );

            // We compute the dot product of the two first edges of the "square
            // ".
            gl.glNormal3d( -h * ( fTopLeft - fBottomLeft ) * normalLength,
                -h * ( fBottomRight - fBottomLeft ) * normalLength,
                h * h * normalLength );

```

```

    // Then we draw the first triangle.
    gl.glTexCoord2d( 0.0, 0.0 ); gl.glVertex3d( x, y, fBottomLeft );
    gl.glTexCoord2d( 1.0, 0.0 ); gl.glVertex3d( x + h, y, fTopLeft );
    gl.glTexCoord2d( 0.0, 1.0 ); gl.glVertex3d( x, y + h, fBottomRight );
    ;

    // We compute the dot product of the two remaining edges of the "
    square".
    gl.glNormal3d( -h * ( fTopRight - fBottomRight ) * normalLength,
        -h * ( fTopRight - fTopLeft ) * normalLength,
        h * h * normalLength );

    // Then we draw the second triangle.
    gl.glTexCoord2d( 1.0, 0.0 ); gl.glVertex3d( x + h, y, fTopLeft );
    gl.glTexCoord2d( 1.0, 1.0 ); gl.glVertex3d( x + h, y + h, fTopRight );
    );
    gl.glTexCoord2d( 0.0, 1.0 ); gl.glVertex3d( x, y + h, fBottomRight );
    ;
}
}

// We finished drawing the surface.
gl.glEnd();
}

// For each parametric surface...
for( AjpUserParameteredSurface surface : parameteredSurfacesArrayList )
{
    // We initialize the surface.
    surface.init();

    // We bind the right texture to this surface.
    gl.glBindTexture( GL_TEXTURE_2D, texturesArray[colorStringToTexture(
        surface.getColorString() )]);

    // We are going to draw the surface, using triangles.
    gl.glBegin( GL_TRIANGLES );

    // We get the two steps.
    double sStep = surface.getSStep();
    double tStep = surface.getTStep();

    // For each value of the s parameter...
    for ( double s = surface.getSLowerBound() ; s <= surface.getSUpperBound
        () ; s += surface.getSStep() ) {
        // ...and for each value of the t parameter...
        for( double t = surface.getTLowerBound() ; t <= surface.
            getTUpperBound() ; t += surface.getTStep() ) {
            // ...we draw two triangles.
            // We get the four points
            AjpUser3DPoint fBottomLeft = surface.evaluate( s, t );
            AjpUser3DPoint fTopLeft = surface.evaluate( s + sStep, t );
            AjpUser3DPoint fBottomRight = surface.evaluate( s, t + tStep );

```

```

AjpUser3DPoint fTopRight = surface.evaluate( s + sStep, t + tStep );

// v ( x1, y1, z1 )
double x1 = fBottomRight.getX() - fBottomLeft.getX();
double y1 = fBottomRight.getY() - fBottomLeft.getY();
double z1 = fBottomRight.getZ() - fBottomLeft.getZ();

// w ( x2, y2, z2 )
double x2 = fTopLeft.getX() - fBottomRight.getY();
double y2 = fTopLeft.getY() - fBottomRight.getY();
double z2 = fTopLeft.getZ() - fBottomRight.getZ();

// We compute the dot product of the two first edges of the "square
// ".
gl.glNormal3d( ( y1 * z2 - y2 * z1 ) * normalLength,
               ( z1 * x2 - x1 * z2 ) * normalLength,
               ( x1 * y2 - x1 * y1 ) * normalLength );

// Then we draw the three points.
gl.glTexCoord2d( 0.0, 0.0 ); gl.glVertex3d( fBottomLeft.getX(),
                                              fBottomLeft.getY(), fBottomLeft.getZ() );
gl.glTexCoord2d( 1.0, 0.0 ); gl.glVertex3d( fTopLeft.getX(),
                                              fTopLeft.getY(), fTopLeft.getZ() );
gl.glTexCoord2d( 0.0, 1.0 ); gl.glVertex3d( fBottomRight.getX(),
                                              fBottomRight.getY(), fBottomRight.getZ() );

// v ( x1, y1, z1 )
x1 = fTopLeft.getX() - fTopRight.getX();
y1 = fTopLeft.getY() - fTopRight.getY();
z1 = fTopLeft.getZ() - fTopRight.getZ();

// w ( x2, y2, z2 )
x2 = fBottomRight.getX() - fTopRight.getY();
y2 = fBottomRight.getY() - fTopRight.getY();
z2 = fBottomRight.getZ() - fTopRight.getZ();

// We compute the dot product of the two remaning edges of the "
// square".
gl.glNormal3d( ( y1 * z2 - y2 * z1 ) * normalLength,
               ( z1 * x2 - x1 * z2 ) * normalLength,
               ( x1 * y2 - x1 * y1 ) * normalLength );

// Then we draw the three points.
gl.glTexCoord2d( 1.0, 0.0 ); gl.glVertex3d( fTopLeft.getX(),
                                              fTopLeft.getY(), fTopLeft.getZ() );
gl.glTexCoord2d( 1.0, 1.0 ); gl.glVertex3d( fTopRight.getX(),
                                              fTopRight.getY(), fTopRight.getZ() );
gl.glTexCoord2d( 0.0, 1.0 ); gl.glVertex3d( fBottomRight.getX(),
                                              fBottomRight.getY(), fBottomRight.getZ() );

}
}
}

```

```

// We don't want lighting for the curves.
gl.glDisable( GL_LIGHTING );

// For each parametric ( 1 parameter ) curve...
for( AjpUserParameteredCurve curve : parameteredCurvesArrayList ) {
    // We initialize the surface.
    curve.init();
    // We set the drawing color.
    gl.glColor3f( curve.getColor().getRed(), curve.getColor().getGreen(),
        curve.getColor().getBlue() );
    // We first get the drawing step.
    double h = curve.getStep();
    // Then we set the line width.
    gl.glLineWidth( curve.getLineWidth() );
    // We are going to draw lines.
    gl.glBegin( GL_LINE_STRIP );
    // Now we step from a value to the next.
    for( double t = curve.getTLowerBound() ; t < curve.getTUpperBound() ; t
        += h ) {
        // We want a line which starts from this point...
        gl.glVertex3d( curve.evaluate( t ).getX(), curve.evaluate( t ).getY()
            , curve.evaluate( t ).getZ() );
        // ...to the next.
        gl.glVertex3d( curve.evaluate( t + h ).getX(), curve.evaluate( t + h
            ).getY(), curve.evaluate( t + h ).getZ() );
    }
    // Done!
    gl.glEnd();
}

// For each dot...
for( AjpUserDot dot : dotsArrayList ) {
    // We initialize the dot.
    dot.init();
    // We set the drawing color.
    gl.glColor3f( dot.getColor().getRed(), dot.getColor().getGreen(), dot.
        getColor().getBlue() );
    // We are going to draw points.
    gl.glBegin( GL_POINTS );
    // Let's plot :)
    gl.glVertex3d( dot.getCoordinates().getX(), dot.getCoordinates().getY()
        , dot.getCoordinates().getZ() );
    // Done!
    gl.glEnd();
}

// Now we draw the axes.
gl.glColor3f( axesColor.getRed(), axesColor.getGreen(), axesColor.
    getBlue() );

// We want a thick line.
gl.glLineWidth( 1.5f );
// Then we draw the axes.
gl.glBegin( GL_LINES );

```



```

gl.glVertex3d( 0.0, 0.0, zLowerBound );
gl.glVertex3d( 0.0, 0.0, zUpperBound );
gl.glVertex3d( 0.0, yLowerBound * 1.3, 0.0 );
gl.glVertex3d( 0.0, yUpperBound * 1.3, 0.0 );
gl.glVertex3d( xLowerBound * 1.3, 0.0, 0.0 );
gl.glVertex3d( xUpperBound * 1.3, 0.0, 0.0 );
gl.glEnd();

// Then we flush the buffer.
gl.glFlush();

leftRightRotation += leftRightRotationStep;
topBottomRotation += topBottomRotationStep;
}

/**
 * This function is called when the window size changes.
 *
 * @param drawable The drawable.
 * @param i Ignored.
 * @param x Ignored.
 * @param width The new window's width.
 * @param height The new window's height.
 */
public void reshape ( GLDrawable drawable, int i, int x, int width, int
    height )
{
    final GL gl = drawable.getGL();
    final GLU glu = drawable.getGLU();

    if ( height <= 0 ) // To avoid a divide by zero error.
        height = 1;
    final float h = ( float ) width / ( float ) height;
    gl.glViewport( 0, 0, width, height );
    gl.glMatrixMode( GL_PROJECTION );
    gl.glLoadIdentity();
    glu.gluPerspective( 85.0f, h, 1.0, 220.0/*Float.MAX_VALUE*/ );
    gl.glMatrixMode( GL_MODELVIEW );
    gl.glLoadIdentity();
    gl.glTranslatef( 0.0f, 0.0f, 5.0f );
}

/**
 * This function is called when the display has changed.
 */
public void displayChanged ( GLDrawable drawable, boolean modeChanged,
    boolean deviceChanged ) { }

/**
 * This function handles the key released events.
 *
 * @param e The event to handle.
 */
public void keyReleased( KeyEvent e ) {}

```

```

/**
 * This function handles the key pressed events.
 *
 * @param e The event to handle.
 */
public void keyPressed( KeyEvent e )
{
    switch( e.getKeyCode() ) {
        // These are the default actions : rotations and zoom.
        case KeyEvent.VK_LEFT :
            leftRightRotation-=0.5;
            break;
        case KeyEvent.VK_RIGHT :
            leftRightRotation+=0.5;
            break;
        case KeyEvent.VK_UP :
            topBottomRotation-=0.5;
            break;
        case KeyEvent.VK_DOWN :
            topBottomRotation+=0.5;
            break;
        case KeyEvent.VK_PAGE_UP:
            zoom+=0.3;
            break;
        case KeyEvent.VK_PAGE_DOWN:
            zoom-=0.3;
            break;
    }
}

/**
 * This function handles the key typed events.
 *
 * @param e The event to handle.
 */
public void keyTyped(KeyEvent e) {}

/**
 * Reads a PNG image from its ressource name.
 *
 * @param resourceName The ressource name.
 * @return The buffered image.
 */
private BufferedImage readPNGImage( String resourceName )
{
    // We first get the image.
    BufferedImage img = AjpFileTools.getImage( resourceName, AjpFileTools.
        IMAGE_TYPE_TEXTURE );
    // Then we scale it...
    java.awt.geom.AffineTransform tx = java.awt.geom.AffineTransform.
        getScaleInstance( 1, -1 );
    // ...and translate it.
    tx.translate( 0, -img.getHeight( null ) );

```

```

    AffineTransformOp op = new AffineTransformOp( tx, AffineTransformOp.
        TYPE_NEAREST_NEIGHBOR );
    img = op.filter( img, null );
    // Then we return the image.
    return img;
}

/**
 * Makes an RGB texture.
 *
 * @param gl The GL context.
 * @param glu The GLU.
 * @param img The image to start from.
 * @param target The target in the textures' array.
 * @param mipmapped Should we mipmap the texture?
 */
private void makeRGBTexture( GL gl, GLU glu, BufferedImage img, int target
    , boolean mipmapped )
{
    // We initialize the byte buffer.
    ByteBuffer dest = null;
    // Then, we create the texture, depending on the image type.
    switch ( img.getType() ) {
    case BufferedImage.TYPE_3BYTE_BGR:
    case BufferedImage.TYPE_CUSTOM: {
        byte[] data = ( ( DataBufferByte ) img.getRaster().getDataBuffer() ).
            getData();
        dest = ByteBuffer.allocateDirect( data.length );
        dest.order( ByteOrder.nativeOrder() );
        dest.put( data, 0, data.length );
        break;
    }
    case BufferedImage.TYPE_INT_RGB: {
        int[] data = ( ( DataBufferInt ) img.getRaster().getDataBuffer() ).
            getData();
        dest = ByteBuffer.allocateDirect( data.length * BufferUtils.SIZEOF_INT
            );
        dest.order( ByteOrder.nativeOrder() );
        dest.asIntBuffer().put( data, 0, data.length );
        break;
    }
    default:
        throw new RuntimeException( commentstyle"commentstyleUnsupported
            commentstyle commentstyleimagecommentstyle commentstyletype
            commentstyle commentstyle" + img.getType() );
    }

    if ( mipmapped ) {
        glu.gluBuild2DMipmaps( target, GL.GL_RGB8, img.getWidth(), img.
            getHeight(), GL.GL_RGB, GL.GL_UNSIGNED_BYTE, dest );
    } else {
        gl.glTexImage2D( target, 0, GL.GL_RGB, img.getWidth(), img.getHeight(),
            0, GL.GL_RGB, GL.GL_UNSIGNED_BYTE, dest );
    }
}

```

```

}

/**
 * Sets the initial camera position and direction.
 *
 * @param Ajp3DUserPoint position The camera position.
 * @param xRot The x initial rotation.
 * @param yDir The y initial rotation.
 * @param zDir The z initial rotation.
 */
public void setInitialCameraPositionAndDirection( AjpUser3DPoint position,
    double xRot, double yRot, double zRot ) {
    this.initialCameraPosition = position;
    this.xInitialCameraRotation = xRot;
    this.yInitialCameraRotation = yRot;
    this.zInitialCameraRotation = zRot;
}

/**
 * @param leftRightRotationStep The leftRightRotationStep to set.
 */
public void setLeftRightRotationStep(double leftRightRotationStep) {
    this.leftRightRotationStep = leftRightRotationStep;
}

/**
 * @param topBottomRotationStep The topBottomRotationStep to set.
 */
public void setTopBottomRotationStep(double topBottomRotationStep) {
    this.topBottomRotationStep = topBottomRotationStep;
}

/**
 * @param lightsArray The lightsArray to set.
 */
public synchronized void setLightsArray(AjpUserLight[] lightsArray) {
    this.lightsArray = lightsArray;
}

/**
 * @return Returns the parameteredCurvesArrayList.
 */
public synchronized ArrayList<AjpUserParameteredCurve>
    getParameteredCurvesArrayList() {
    return parameteredCurvesArrayList;
}

/**
 * @param parameteredCurvesArrayList The parameteredCurvesArrayList to set
 *
 */
public synchronized void setParameteredCurvesArrayList(
    ArrayList<AjpUserParameteredCurve> parameteredCurvesArrayList) {
    this.parameteredCurvesArrayList = parameteredCurvesArrayList;
}

```

```

}

/**
 * @return Returns the parameteredSurfacesArrayList.
 */
public synchronized ArrayList<AjpUserParameteredSurface>
    getParameteredSurfacesArrayList() {
    return parameteredSurfacesArrayList;
}

/**
 * @param parameteredSurfacesArrayList The parameteredSurfacesArrayList to
    set.
 */
public synchronized void setParameteredSurfacesArrayList (
    ArrayList<AjpUserParameteredSurface> parameteredSurfacesArrayList) {
    this.parameteredSurfacesArrayList = parameteredSurfacesArrayList;
}

/**
 * @return Returns the surfacesArrayList.
 */
public synchronized ArrayList<AjpUserSurface> getSurfacesArrayList() {
    return surfacesArrayList;
}

/**
 * @param surfacesArrayList The surfacesArrayList to set.
 */
public synchronized void setSurfacesArrayList (ArrayList<AjpUserSurface>
    surfacesArrayList) {
    this.surfacesArrayList = surfacesArrayList;
}

/**
 * @return Returns the dotsArrayList.
 */
public synchronized ArrayList<AjpUserDot> getDotsArrayList() {
    return dotsArrayList;
}

/**
 * @param dotsArrayList The dotsArrayList to set.
 */
public synchronized void setDotsArrayList (ArrayList<AjpUserDot>
    dotsArrayList) {
    this.dotsArrayList = dotsArrayList;
}

/**
 * @return Returns the axes' color.
 */
public AjpUserColor getAxesColor() {
    return axesColor;
}

```

```

}

/**
 * @param axesColor The axes' color to set.
 */
public void setAxesColor(AjpUserColor axesColor) {
    this.axesColor = axesColor;
}

/**
 * @param action The action to set.
 */
public void setAction(AjpUserWindowAction action) {
    this.action = action;
}

/**
 * @param window The window to set.
 */
public void setWindow(AjpUserWindow window) {
    this.window = window;
}
}

```

Listing 5.4 – Le fichier AjpGLEventListener.java

```

package org.lozi.ajp.ajp;

import javax.swing.*;
import java.awt.event.*;
import java.util.logging.Logger;
import java.net.URL;

/**
 * AjpUI class. The graphical user interface.
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
 * @version 1.0
 */
public class AjpGUI extends JFrame
{
    /**
     * The serial Version UID.
     */
    private static final long serialVersionUID = 1L;
    /** We store the fileChooser in order to always start from the previously
        open directory. */
    private JFileChooser fileChooser;
    /** We keep a pointer to the application. */
    private AjpApp application;

    public void init( AjpApp application ) {

```

```

Logger logger = Logger.getLogger(commentstyle"commentstyleAjpGUI
    commentstyle");
logger.info(commentstyle"commentstyleEnteringcommentstyle
    commentstylethecommentstyle commentstyleAjpGUIcommentstyle
    commentstyleconstructorcommentstyle");

this.application = application;

setTitle( commentstyle"commentstyleAjpcommentstyle commentstyle-
    commentstyle commentstyleAnothercommentstyle commentstyleJava
    commentstyle commentstylePlottercommentstyle" );
// We initialize the file chooser, with the base directory. */
fileChooser = new JFileChooser( AjpFileTools.baseFolder() );

JFrame.setDefaultLookAndFeelDecorated(false);

// We create the actions.
LoadSourceAction loadSourceAction = new LoadSourceAction();
LoadClassAction loadClassAction = new LoadClassAction();
StartAction startAction = new StartAction();
StopAction stopAction = new StopAction();
QuitAction quitAction = new QuitAction();

// We create the menu bar.
JMenuBar menuBar = new JMenuBar();
// We create the File menu.
JMenu fileMenu = new JMenu( commentstyle"commentstyleFilecommentstyle" )
;
menuBar.add( fileMenu );
// We create the menu items for the File menu.
fileMenu.add( new JMenuItem( loadSourceAction ) );
fileMenu.add( new JMenuItem( loadClassAction ) );
fileMenu.addSeparator();
fileMenu.add( new JMenuItem( startAction ) );
fileMenu.add( new JMenuItem( stopAction ) );
fileMenu.addSeparator();
fileMenu.add( new JMenuItem( quitAction ) );
// We create the toolBar.
JToolBar toolBar = new JToolBar();
toolBar.add( loadSourceAction );
toolBar.add( loadClassAction );
toolBar.addSeparator();
toolBar.add( startAction );
toolBar.add( stopAction );
toolBar.addSeparator();
toolBar.add( quitAction );

this.setJMenuBar( menuBar );
this.getContentPane().add( toolBar );

this.pack();
this.getContentPane().repaint();
this.setVisible( true );

```

```

        logger.info(commentstyle"commentstyleLeavingcommentstyle commentstylethe
            commentstyle commentstyleAjpGUIcommentstyle commentstyleconstructor
            commentstyle");
    }

    /**
     * The "Load source" action class.
     */
    class LoadSourceAction extends AbstractAction {

        /**
         * The serial version UID.
         */
        private static final long serialVersionUID = 153572L;

        /**
         * Creates a LoadSource object, with its default values.
         */
        public LoadSourceAction () {
            // We call the constructor with its default values.
            this( commentstyle"commentstyleLoadcommentstyle commentstyleSource
                commentstyle...commentstyle", new ImageIcon( AjpFileTools.getImage(
                    commentstyle"commentstylesourcecommentstyle.commentstylepng
                    commentstyle", AjpFileTools.IMAGE_TYPE_ICON ) ), commentstyle"
                commentstyleLoadscommentstyle commentstyleaccommentstyle
                commentstylejavacommentstyle commentstylesourcecommentstyle
                commentstylefilecommentstyle.commentstyle", new Integer(
                    commentstyle'commentstyleScommentstyle' ));
        }

        /**
         * Creates a LoadSourceAction object.
         *
         * @param text The text.
         * @param icon The icon.
         * @param desc The description.
         * @param mnemonic The mnemonic.
         */
        public LoadSourceAction ( String text, ImageIcon icon,
            String desc, Integer mnemonic) {
            // We call the parent constructor.
            super(text, icon);
            // We set the description and the mnemonic.
            putValue(SHORT_DESCRIPTION, desc);
            putValue(MNEMONIC_KEY, mnemonic);
        }

        /**
         * This function is called when the Load Source action is performed.
         */
        public void actionPerformed(ActionEvent e) {

            // We display the file chooser.

```



```

    int returnValue = fileChooser.showOpenDialog( /*AjpGUI.this*/ null );

    // If the user cancelled, then we just return.
    if( returnValue != JFileChooser.APPROVE_OPTION ) return;

    // Otherwiste, we load the file in the JEditorPane...
    // TODO : load the file in the JEditorPane.

    // ...and we ask the application to load it.
    application.loadSource( fileChooser.getSelectedFile() );
}

}

/**
 * The "Load class" action class.
 */
class LoadClassAction extends AbstractAction {

    /**
     * The serial version UID.
     */
    private static final long serialVersionUID = 153572L;

    /**
     * Creates a LoadClass object, with its default values.
     */
    public LoadClassAction () {
        this(commentstyle"commentstyleLoadcommentstyle commentstyleClass
            commentstyle...commentstyle", new ImageIcon( AjpFileTools.getImage(
                commentstyle"commentstyleclasscommentstyle.commentstylepng
                commentstyle", AjpFileTools.IMAGE_TYPE_ICON ) ), commentstyle"
                commentstyleLoadscommentstyle commentstyleaccommentstyle
                commentstylejavacommentstyle commentstyleclasscommentstyle.
                commentstyle", new Integer( commentstyle'commentstyleOcommentstyle'
                    ));
    }

    /**
     * Creates a LoadClass object.
     *
     * @param text The text.
     * @param icon The icon.
     * @param desc The description.
     * @param mnemonic The mnemonic.
     */
    public LoadClassAction ( String text, ImageIcon icon,
                             String desc, Integer mnemonic) {
        // We call the parent constructor.
        super(text, icon);
        // We set the description and the mnemonic.
        putValue(SHORT_DESCRIPTION, desc);
        putValue(MNEMONIC_KEY, mnemonic);
    }
}

```

```

/**
 * This function is called when the Load Source action is performed.
 */
public void actionPerformed(ActionEvent e) {
    // We display the file chooser.
    int returnValue = fileChooser.showOpenDialog( /*AjpGUI.this*/ null );

    // If the user cancelled, then we just return.
    if( returnValue != JFileChooser.APPROVE_OPTION ) return;

    application.loadClass( fileChooser.getSelectedFile() );
}
}

/**
 * The "Start" action class.
 */
class StartAction extends AbstractAction {

    /**
     * The serial version UID.
     */
    private static final long serialVersionUID = 1535729L;

    /**
     * Creates a Start object, with its default values.
     */
    public StartAction () {
        this( commentstyle"commentstyleStartcommentstyle", new ImageIcon(
            AjpFileTools.getImage( commentstyle"commentstylestartcommentstyle.
            commentstylepngcommentstyle", AjpFileTools.IMAGE_TYPE_ICON ) ),
            commentstyle"commentstyleExecutescommentstyle commentstylethe
            commentstyle commentstylecurrentlycommentstyle commentstyleloaded
            commentstyle commentstyleclasscommentstyle.commentstyle", new
            Integer( commentstyle'commentstyleAcommentstyle' ));
    }

    /**
     * Creates a Start object.
     *
     * @param text The text.
     * @param icon The icon.
     * @param desc The description.
     * @param mnemonic The mnemonic.
     */
    public StartAction ( String text, ImageIcon icon,
                        String desc, Integer mnemonic) {
        // We call the parent constructor.
        super(text, icon);
        // We set the description and the mnemonic.
        putValue(SHORT_DESCRIPTION, desc);
        putValue(MNEMONIC_KEY, mnemonic);
    }
}

```

```

    }

    /**
     * This function is called when the Start action is performed.
     */
    public void actionPerformed(ActionEvent e) {
        application.start();
    }
}

/**
 * The "Stop" action class.
 */
class StopAction extends AbstractAction {

    /**
     * The serial version UID.
     */
    private static final long serialVersionUID = 1532329L;

    /**
     * Creates a Start object, with its default values.
     */
    public StopAction () {
        this( commentstyle"commentstyleStopcommentstyle", new ImageIcon(
            AjpFileTools.getImage( commentstyle"commentstylestopcommentstyle.
            commentstylepngcommentstyle", AjpFileTools.IMAGE_TYPE_ICON ) ),
            commentstyle"commentstyleStopscommentstyle commentstylethe
            commentstyle commentstylecurrentcommentstyle commentstyleexecution
            commentstyle.commentstyle", new Integer( commentstyle'commentstyleP
            commentstyle' ));
    }

    /**
     * Creates a Start object.
     *
     * @param text The text.
     * @param icon The icon.
     * @param desc The description.
     * @param mnemonic The mnemonic.
     */
    public StopAction ( String text, ImageIcon icon,
                        String desc, Integer mnemonic) {
        // We call the parent constructor.
        super(text, icon);
        // We set the description and the mnemonic.
        putValue(SHORT_DESCRIPTION, desc);
        putValue(MNEMONIC_KEY, mnemonic);
    }

    /**
     * This function is called when the Start action is performed.
     */

```

```

        public void actionPerformed(ActionEvent e) {
            application.stop();
        }
    }

    /**
     * The "Quit" action class.
     */
    class QuitAction extends AbstractAction {

        /**
         * The serial version UID.
         */
        private static final long serialVersionUID = 123452L;

        /**
         * Creates a QuitAction object, with its default values.
         */
        public QuitAction () {
            this(commentstyle"commentstyleQuitcommentstyle", new ImageIcon(
                AjpFileTools.getImage( commentstyle"commentstylequitcommentstyle.
                commentstylepngcommentstyle", AjpFileTools.IMAGE_TYPE_ICON ) ),
                commentstyle"commentstyleQuitscommentstyle commentstylethe
                commentstyle commentstyleapplicationcommentstyle.commentstyle", new
                Integer( commentstyle'commentstyleQcommentstyle' ));
        }

        /**
         * Creates a QuitAction object.
         *
         * @param text The text.
         * @param icon The icon.
         * @param desc The description.
         * @param mnemonic The mnemonic.
         */
        public QuitAction ( String text, ImageIcon icon,
                           String desc, Integer mnemonic) {
            // We call the parent constructor.
            super(text, icon);
            // We set the description and the mnemonic.
            putValue(SHORT_DESCRIPTION, desc);
            putValue(MNEMONIC_KEY, mnemonic);
        }

        /**
         * This function is called when the Quit action is performed.
         */
        public void actionPerformed(ActionEvent e) {
            // Exits the application
            System.exit( 0 );
        }
    }

    public final static URL getResource( final String filename )

```

```

{
    // Try to load resource from jar
    URL url = ClassLoader.getResource( filename );
    // If not found in jar, then load from disk
    if ( url == null ) {
        try {
            url = new URL( commentstyle"commentstylefilecommentstyle",
                           commentstyle"commentstylelocalhostcommentstyle", filename );
        } catch ( Exception urlException ) { } // ignore
    }
    return url;
}
}

```

Listing 5.5 – Le fichier AjpGUI.java

```

package org.lozi.ajp.ajp;

import javax.swing.JOptionPane;

/**
 * AjpMiscellaneousTools class. Contains miscellaneous tools for the Ajp
 * project.
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
 * @version 1.0
 */
public class AjpMiscellaneousTools {
    /**
     * A handy error handling function which simply displays an error message.
     *
     * @param message The message to show.
     */
    public static void error ( String message ) {
        JOptionPane.showMessageDialog(null, message, commentstyle"
        commentstyleErrorcommentstyle", JOptionPane.ERROR_MESSAGE);
    }
}

```

Listing 5.6 – Le fichier AjpMiscellaneousTools.java

```

package org.lozi.ajp.ajp;

/**
 * This class represents a 3D point with its alpha channel.
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
 */
public class AjpUser3DPoint
{
    /** The x coordinate. */
    private double x;
    /** The y coordinate. */
    private double y;
}

```

```
/** The z coordinate. */
private double z;

/**
 * Creates a new 3D point.
 *
 * @param x The x coordinate.
 * @param y The y coordinate.
 * @param z The z coordinate.
 */
public AjpUser3DPoint( double x, double y, double z ) {
    this.x = x;
    this.y = y;
    this.z = z;
}

/**
 * @return Returns the x coordinate.
 */
public double getX() {
    return x;
}

/**
 * @return Returns the y coordinate.
 */
public double getY() {
    return y;
}

/**
 * @return Returns the z coordinate.
 */
public double getZ() {
    return z;
}

/**
 * @param x The x coordinate to set.
 */
public void setX(double x) {
    this.x = x;
}

/**
 * @param y The y coordinate to set.
 */
public void setY(double y) {
    this.y = y;
}

/**
 * @param z The z coordinate to set.
 */
```

```
public void setZ(double z) {  
    this.z = z;  
}  
}
```

Listing 5.7 – Le fichier AjpUser3DPoint.java

```
package org.lozi.ajp.ajp;  
  
/**  
 * This class represents a color with its alpha channel.  
 *  
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org  
 */  
public class AjpUserColor  
{  
    /** The amount of red in the color. Between 0.0 and 1.0. */  
    private float red;  
    /** The amount of green in the color. Between 0.0 and 1.0. */  
    private float green;  
    /** The amount of blue in the color. Between 0.0 and 1.0. */  
    private float blue;  
    /** The color's alpha channel. Between 0.0 and 1.0. */  
    private float alpha;  
  
    /**  
     * Creates an AjpUserColor object.  
     *  
     * @param red The amount of red. Between 0.0 and 1.0.  
     * @param green The amount of green. Between 0.0 and 1.0.  
     * @param blue The amount of blue. Between 0.0 and 1.0.  
     * @param alpha The alpha channel. Between 0.0 and 1.0.  
     */  
    public AjpUserColor( float red, float green, float blue, float alpha ) {  
        this.red = red;  
        this.green = green;  
        this.blue = blue;  
        this.alpha = alpha;  
    }  
  
    /**  
     * @return Returns the amount of red. Between 0.0 and 1.0.  
     */  
    public float getRed() {  
        return red;  
    }  
  
    /**  
     * @return Returns the amount of green. Between 0.0 and 1.0.  
     */  
    public float getGreen() {  
        return green;  
    }  
}
```

```

/**
 * @return Returns the amount of blue. Between 0.0 and 1.0.
 */
public float getBlue() {
    return blue;
}

/**
 * @return Returns the alpha channel's value. Between 0.0 and 1.0.
 */
public float getAlpha() {
    return alpha;
}

/**
 * @param red The amount of red to set. Between 0.0 and 1.0.
 */
public void setRed(float red) {
    this.red = red;
}

/**
 * @param green The amount of green to set. Between 0.0 and 1.0.
 */
public void setGreen(float green) {
    this.green = green;
}

/**
 * @param blue The amount of blue to set. Between 0.0 and 1.0.
 */
public void setBlue(float blue) {
    this.blue = blue;
}

/**
 * @param alpha The alpha channel to set. Between 0.0 and 1.0.
 */
public void setAlpha(float alpha) {
    this.alpha = alpha;
}
}

```

Listing 5.8 – Le fichier AjpUserColor.java

```

package org.lozi.ajp.ajp;

/**
 * This class represents a simple light.
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
 */
public class AjpUserLight {

```



```

/** The light position. */
public AjpUser3DPoint position;
/** The light color. */
public AjpUserColor color;

/**
 * Creates an AjpUserLight.
 *
 * @param position The light position.
 * @param color The light color.
 */
public AjpUserLight ( AjpUser3DPoint position, AjpUserColor color ) {
    this.position = position;
    this.color = color;
}

/**
 * @return Returns the color.
 */
public AjpUserColor getColor() {
    return color;
}

/**
 * @return Returns the position.
 */
public AjpUser3DPoint getPosition() {
    return position;
}

/**
 * @param color The color to set.
 */
public void setColor(AjpUserColor color) {
    this.color = color;
}

/**
 * @param position The position to set.
 */
public void setPosition(AjpUser3DPoint position) {
    this.position = position;
}
}

```

Listing 5.9 – Le fichier AjpUserLight.java

```

package org.lozi.ajp.ajp;

/**
 * AjpUserParameteredCurve abstract class.
 * The user should derive this class each time he needs a new curve with a
 * single paramater.
 */

```

```

* @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
*/
public abstract class AjpUserParameteredCurve {
    /** The lower bound for the parameter t. */
    protected double tLowerBound;
    /** The upper bound for the parameter t. */
    protected double tUpperBound;
    /** The plot color. */
    protected AjpUserColor color;
    /** The line width. */
    protected float lineWidth;
    /** The step. */
    protected double step;

    /**
     * Evaluates the function for the parameter t.
     * Should return a point, the value of the function at t.
     *
     * @param t The parameter for which the function should be evaluated.
     */
    public abstract AjpUser3DPoint evaluate( double t );

    /**
     * Initializes the local variables.
     */
    public abstract void init ();

    /**
     * @return Returns the color.
     */
    public AjpUserColor getColor() {
        return color;
    }

    /**
     * @return Returns the tLowerBound.
     */
    public double getTLowerBound() {
        return tLowerBound;
    }

    /**
     * @return Returns the tUpperBound.
     */
    public double getTUpperBound() {
        return tUpperBound;
    }

    /**
     * @return Returns the lineWidth.
     */
    public float getLineWidth() {
        return lineWidth;
    }
}

```

```

/**
 * @return Returns the step.
 */
public double getStep() {
    return step;
}
}

```

Listing 5.10 – Le fichier AjpUserParameteredCurve.java

```

package org.lozi.ajp.ajp;

/**
 * AjpUserParameteredSurface interface.
 * The user should derive this class each time he needs a new curve with a
 *   two parameters.
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
 */
public abstract class AjpUserParameteredSurface {
    /** The lower bound for the parameter t. */
    protected double tLowerBound;
    /** The upper bound for the parameter t. */
    protected double tUpperBound;
    /** The lower bound for the parameter s. */
    protected double sLowerBound;
    /** The upper bound for the parameter s. */
    protected double sUpperBound;
    /** The step for the s parameter. */
    protected float sStep;
    /** The step for the t parameter. */
    protected float tStep;
    /** The plot's color. */
    protected String colorString;

    /**
     * Evaluates the function for the parameters s and t.
     * Should return a tridimensionnal array of double values.
     *
     * @param s The first parameter for which the function should be evaluated
     * .
     * @param t The scond parameter for which the function should be evaluated
     * .
     * @return The evaluated point.
     */
    public abstract AjpUser3DPoint evaluate( double s, double t );

    /**
     * Initializes the local variables.
     */
    public abstract void init ();

    /**

```

```
    * @return Returns the sLowerBound.
    */
    public double getSLowerBound() {
        return sLowerBound;
    }

    /**
     * @return Returns the sUpperBound.
     */
    public double getSUpperBound() {
        return sUpperBound;
    }

    /**
     * @return Returns the tLowerBound.
     */
    public double getTLowerBound() {
        return tLowerBound;
    }

    /**
     * @return Returns the tUpperBound.
     */
    public double getTUpperBound() {
        return tUpperBound;
    }

    /**
     * @return Returns the sStep.
     */
    public float getSStep() {
        return sStep;
    }

    /**
     * @return Returns the tStep.
     */
    public float getTStep() {
        return tStep;
    }

    /**
     * @return Returns the colorString.
     */
    public String getColorString() {
        return colorString;
    }
}
```

Listing 5.11 – Le fichier AjpUserParameteredSurface.java

```
package org.lozi.ajp.ajp;
```

```

import javax.swing.JFrame;
import net.java.games.jogl.GLCanvas;
import net.java.games.jogl.GLCapabilities;
import net.java.games.jogl.GLDrawableFactory;

/**
 * AjpUserPlotterTools class. An instance of this class is passed to the
 * functions of
 * the the scripts. All the actions the user can do come from this class.
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
 * @version 1.0
 */
public class AjpUserPlotter {

    /**
     * Creates a new window.
     *
     * @param title The window title.
     * @param width The window width.
     * @param height The window height.
     * @param x The window x position.
     * @param y The window y position.
     * @param lightsArray The array containing the window's lights (8 max).
     * @return The newly created window.
     */
    public AjpUserWindow newWindow( String title, int width, int height, int x
        , int y, AjpUserLight[] lightsArray ) {
        // We create the new window itself.
        JFrame window = new JFrame( title );
        // We create a GLCapabilities object.
        GLCapabilities capabilities = new GLCapabilities();
        // We create the GLCanvas.
        GLCanvas context = GLDrawableFactory.getFactory().createGLCanvas(
            capabilities);
        // We add the context to the window.
        window.getContentPane().add(context);
        // We set the window position.
        window.setLocation( x, y );
        // We set the window size.
        window.setSize( width, height );
        // We create the event listener.
        AjpGLEventListener listener = new AjpGLEventListener( lightsArray );
        // We add a new event listener to the window.
        context.addGLEventListener( listener );
        // We show the window.
        window.setVisible( true );
        // And we return the corresponding AjpUserWindow
        return new AjpUserWindow( window, listener, context );
    }
}

```

Listing 5.12 – Le fichier AjpUserPlotter.java

```

package org.lozi.ajp.ajp;

/**
 * AjpUserSurface abstract class.
 * The user should derive this class each time he needs to plot a a R -> R2
 * function.
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
 */
public abstract class AjpUserSurface {
    /** The x lower bound. */
    protected double xLowerBound;
    /** The x upper bound. */
    protected double xUpperBound;
    /** The y lower bound. */
    protected double yLowerBound;
    /** The y upper bound. */
    protected double yUpperBound;
    /** The grid's size. */
    protected double gridSize;
    /** The plot's color. */
    protected String colorString;

    /**
     * Evaluates the curve's function at the (x,y) position.
     *
     * @param t The parameter for which the function should be evaluated.
     * @return The evaluation.
     */
    public abstract double evaluate( double x, double y );

    /**
     * Initializes the local variables.
     */
    public abstract void init ();

    /**
     * @return Returns the color string.
     */
    public String getColorString() {
        return colorString;
    }

    /**
     * @return Returns the xLowerBound.
     */
    public double getXLowerBound() {
        return xLowerBound;
    }

    /**
     * @return Returns the xUpperBound.
     */
    public double getXUpperBound() {

```

```

    return xUpperBound;
}

/**
 * @return Returns the yLowerBound.
 */
public double getYLowerBound() {
    return yLowerBound;
}

/**
 * @return Returns the yUpperBound.
 */
public double getYUpperBound() {
    return yUpperBound;
}

/**
 * @return Returns the gridSize.
 */
public double getGridSize() {
    return gridSize;
}
}

```

Listing 5.13 – Le fichier `AjpUserSurface.java`

```

package org.lozi.ajp.ajp;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.ArrayList;

import javax.swing.JFrame;
import net.java.games.jogl.*;

/**
 * AjpUserWindow class.
 * AjpUserWindow's objects represent the Ajp application's windows.
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
 */
public class AjpUserWindow {
    /** The corresponding JFrame. */
    private JFrame frame;
    /** The event listener. */
    private AjpGLEventListener listener;
    /** The canvas. */
    private GLCanvas context;
    /** The animator. */
    private Animator animator;

    /**

```

```

    * Creates an AjpUserWindow.
    *
    * @param JFrame The corresponding JFrame.
    * @param listener The window listener.
    * @param firstLight The first light.
    * @param secondLight The second light.
    */
AjpUserWindow( JFrame frame, AjpGLEventListener listener, GLCanvas context
    ) {
    this.frame = frame;
    this.listener = listener;
    this.context = context;
    this.listener.setWindow( this );
    animator = new Animator(context);

    AjpApp.addFrame( this );

    frame.addWindowListener(new AjpUserWindowAdapter( animator ) );
}

/**
 * Animates the window.
 */
public void animate () {
    animator.start();
}

/**
 * Gets the list of the parametered curves bound to this window.
 */
public ArrayList<AjpUserParameteredCurve> getParameteredCurvesArrayList ()
{
    return listener.getParameteredCurvesArrayList();
}

/**
 * Gets the list of the parametered surfaces bound to this window.
 */
public ArrayList<AjpUserParameteredSurface>
    getParameteredSurfacesArrayList () {
    return listener.getParameteredSurfacesArrayList();
}

/**
 * Gets the list of the surfaces bound to this window.
 */
public ArrayList<AjpUserSurface> getSurfacesArrayList () {
    return listener.getSurfacesArrayList();
}

/**

```



```

    * Gets the list of the dots bound to this window.
    */
    public ArrayList<AjpuUserDot> getDotsArrayList() {
        return listener.getDotsArrayList();
    }

    /**
     * Rotates the window contents.
     *
     * @param leftRightStep The left/right step.
     * @param topBottomStep The top/bottom step.
     */
    public void setRotationSteps( double leftRightStep, double topBottomStep )
    {
        this.listener.setLeftRightRotationStep( leftRightStep );
        this.listener.setTopBottomRotationStep( topBottomStep );
    }

    /**
     * Sets the bounds of the window's axes.
     *
     * @param xLowerBound The x lower bound.
     * @param xUpperBound The x upper bound.
     * @param yLowerBound The y lower bound.
     * @param yUpperBound The y upper bound.
     * @param zLowerBound The z lower bound.
     * @param zUpperBound The z upper bound.
     */
    public void setAxesBounds( double xLowerBound, double xUpperBound, double
        yLowerBound, double yUpperBound,
        double zLowerBound, double zUpperBound ) {
        listener.setBounds( xLowerBound, xUpperBound, yLowerBound, yUpperBound,
            zLowerBound, zUpperBound );
    }

    /**
     * Sets the axes' color.
     *
     * @param color The new axes' color.
     */
    public void setAxesColor( AjpuUserColor color ) {
        listener.setAxesColor( color );
    }

    /**
     * Shows/Hides the window.
     *
     * @param visibility True to show the window, false to hide it.
     */
    public void setVisible( boolean visibility ) {
        this.frame.setVisible( visibility );
    }

    /**

```

```

    * Sets the background color.
    *
    * @param color The new background color.
    */
    public void setBackgroundColor( AjpUserColor color ) {
        listener.setBackgroundColor( color );
    }

    /**
     * Sets the list of the parametered curves bound to this window.
     *
     * @param parameteredCurvesArrayList The list of parametered curves to set
     * .
     */
    public void setParameteredCurvesArrayList( ArrayList<
        AjpUserParameteredCurve> parameteredCurvesArrayList ) {
        listener.setParameteredCurvesArrayList( parameteredCurvesArrayList );
    }

    /**
     * Sets the list of the parametered surfaces bound to this window.
     *
     * @param parameteredSurfacesArrayList The list parametered surfaces to
        set.
     */
    public void setParameteredSurfacesArrayList( ArrayList<
        AjpUserParameteredSurface> parameteredSurfacesArrayList ) {
        listener.setParameteredSurfacesArrayList( parameteredSurfacesArrayList )
        ;
    }

    /**
     * Sets the list of the surfaces bound to this window.
     *
     * @param surfacesArrayList The list of surfaces to set.
     */
    public void setSurfacesArrayList( ArrayList<AjpUserSurface>
        surfacesArrayList ) {
        listener.setSurfacesArrayList( surfacesArrayList );
    }

    /**
     * Binds a parametered curve to the current window.
     *
     * @param curve The parametered curve to bind.
     */
    public void addParameteredCurve( AjpUserParameteredCurve curve ) {
        listener.addParameteredCurve( curve );
    }

    /**
     * Binds a parametered surface to the current window.
     *
     * @param curve The parametered curve to bind.

```

```

    */
    public void addParameteredSurface( AjpUserParameteredSurface surface ) {
        listener.addParameteredSurface( surface );
    }

    /**
     * Binds a surface to the current window.
     *
     * @param surface The surface to bind.
     */
    public void addSurface( AjpUserSurface surface ) {
        listener.addSurface( surface );
    }

    /**
     * Binds a dot to the current window.
     *
     * @param surface The surface to bind.
     */
    public void addDot( AjpUserDot dot ) {
        listener.addDot( dot );
    }

    /**
     * Sets the initial camera position and direction.
     *
     * @param Ajp3DUserPoint position The camera position.
     * @param xRot The x initial rotation.
     * @param yDir The y initial rotation.
     * @param zDir The z initial rotation.
     */
    public void setInitialCameraPositionAndDirection( AjpUser3DPoint position,
        double xRot, double yRot, double zRot ) {
        listener.setInitialCameraPositionAndDirection( position, xRot, yRot,
            zRot );
    }

    /**
     * Set the window's action.
     *
     * @param action The action to bind.
     */
    public void setAction( AjpUserWindowAction action ) {
        listener.setAction( action );
    }

    /**
     * @return Returns the listener.
     */
    AjpGLEventListener getListener() {
        return listener;
    }

```

```

/**
 * @param listener The listener to set.
 */
void setListener(AjpGLEventListener listener) {
    this.listener = listener;
}

/**
 * Closes the window.
 */
void close () {
    animator.stop();
    frame.setVisible( false );
}

/**
 * The AjpUserWindowAdapter which stops the animation when the window is
 * closed.
 *
 * @author jeep - jean-pierre@lozi.org
 */
public class AjpUserWindowAdapter extends WindowAdapter {
    /** The animator. */
    private Animator animator;

    /**
     * The constructor.
     *
     * @param animator The window's animator.
     */
    public AjpUserWindowAdapter ( Animator animator ) {
        this.animator = animator;
    }

    /**
     * This function is called when the window is close.
     *
     * @param winEvt The window event.
     */
    public void windowClosing(WindowEvent winEvt) {
        animator.stop();
    }
}
}

```

Listing 5.14 – Le fichier AjpUserWindow.java

```

package org.lozi.ajp.ajp;

/**
 * AjpUserWindowAction interface.
 * The user should implement this class each time he needs to bind an
 * action to

```

```
* the window.
*
* @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org
*/
public interface AjpUserWindowAction {

    /**
     * This function is called each time a frame is drawn.
     *
     * @param window The window in which the frame is drawn.
     */
    public abstract void run ( AjpUserWindow window );
}
```

Listing 5.15 – Le fichier AjpUserWindowAction.java

```
package org.lozi.ajp.ajp.scripts;

import org.lozi.ajp.ajp.*;

/**
 * AjpUserScript interface.
 *
 * @author Jean-Pierre Lozi - mailto:jean-pierre@lozi.org , Amir Maalej -
 *         mailto:maaleja@echo.unice.fr
 */
public interface AjpUserScript
{
    /** The main script function which is run when the script is executed. */
    public abstract void start( AjpUserPlotter plotter );
}
```

Listing 5.16 – Le fichier AjpUserScript.java

Chapitre 6

Conclusion

Ce projet permet à des utilisateurs de scripter de manière très simple des courbes en 3 dimensions, sans avoir à manipuler le moindre objet *OpenGL*. Il touche à plusieurs notions non abordées cette année, comme *OpenGL* avec *JOGL* ou le chargement dynamique de classes. Il est facilement extensible et il devrait être assez simple d'ajouter la gestion d'autres objets, comme les cubes ou les sphères.

Le développement ayant pris la plus grande partie de notre temps, le rapport et la présentation n'ont peut-être pas la qualité espérée - toutefois, nous avons préféré privilégier le développement d'un logiciel fonctionnel et réutilisable.

La comparaison de la longueur d'un script et du même programme réalisé directement avec l'API *JOGL* peut varier du simple au quintuple. Ainsi, ce programme est particulièrement destiné à ceux qui désirent tracer des courbes et surfaces en 3 dimensions rapidement. Le système des actions permet de modifier les surfaces, les courbes, les couleurs, les lumières, ainsi que tous les autres paramètres durant l'animation. Ceci permet notamment de visualiser des algorithmes dont l'évolution temporelle est un facteur indispensable.

Ce projet a donc, en plus de ses vertus pédagogiques, une certaine utilité pour les utilisateurs en quête de traceurs de courbes particulièrement personnalisables.