

JOGL: JAVA E OPENGL

Per programmare in OpenGL tramite Java, in un modo molto simile a come si fa in C++ con la libreria GLUT, si può utilizzare Jogl, una piccola libreria scritta appunto a questo scopo. Se invece siamo interessati a funzioni di più alto livello possiamo anche utilizzare direttamente le API di Java3D.

Un tipico programma risulta composto primaditutto da una classe contenente il main, la quale crea un Frame e un GLCanvas per la visualizzazione, aggiunge il canvas alla finestra e setta i parametri per l'inizializzazione:

dal file MainWindow.java:

```
//creazione del Frame e del GLCancas
    Frame frame = new Frame("CMSC427 - Project 1");
    GLCanvas canvas = GLDrawableFactory.getFactory
() .createGLCanvas(new GLCapabilities());

// aggiungiamo l'oggetto canvas alla finestra
    frame.add(canvas);

// settaggi iniziali del frame
    frame.setSize(400, 400);
    frame.setLocation(0, 0);
    frame.setBackground(Color.white);

// aggiungiamo un listener per l'evento di chiusura del frame
    frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });

/*a questo punto prima di terminare il metodo con la visualizzazione del frame
aggiungiamo un listener per gli eventi opengl collegato all'oggetto canvas, il listener è
un oggetto della classe JoglEventManager, una classe scritta appositamente da noi e che
farà tutto il lavoro*/
    JoglEventManager listener = new
JoglEventManager(canvas);
    canvas.addGLEventListener(listener);
    canvas.addKeyListener(listener);
    canvas.addMouseListener(listener);
    canvas.addMouseMotionListener(listener);

/*terminiamo con la visualizzazione del frame e la chiamata requestFocus che equivale
al "glutMainLoop"*/
    frame.setVisible(true);
    frame.show();
    canvas.requestFocus();
```

dal file JoglEventManager.java:

A questo punto ci possiamo occupare di realizzare la classe che gestisce gli eventi opengl, ovvero JoglEventManager. La classe implementa alcune interfacce che la caratterizzano quale listener di eventi opengl:

```
public class JoglEventManager
    implements GLEventListener, KeyListener,
    MouseListener, MouseMotionListener {
```

```
/*tra le variabili private che la classe memorizza è utile mantenere un riferimento alla
classe canvas sorgente degli eventi, riferimento che verrà passato al costruttore di
questa classe*/
```

```
    private GLCanvas _canvas = null;
```

```
/*Altra variabile privata molto importante è l'oggetto CommandMediator che in
sostanza fa da mediatore tra questa classe che gestisce gli eventi e i singoli oggetti che
devono essere visualizzati*/
```

```
    private CommandMediator _commandMediator;
```

```
/*nel costruttore gli viene passato un riferimento all'oggetto canvas sorgente degli
eventi, inoltre crea l'oggetto CommandMediator che contiene tutti gli effettivi oggetti da
renderizzare */
```

```
    public JoglEventManager(GLCanvas canvas) {
        this._canvas = canvas;
        this._commandMediator = new CommandMediator();
    }
```

```
/*display è il metodo che si occupa del rendering degli oggetti, è l'equivalente del
gestore del rendering chiamato tramite glutPostRedisplay() e settato con
glutDisplayFunc.
```

```
In questo caso riceve un parametro che specifica su quale oggetto deve effettuare il
rendering e sarà chiaramente il canvas sul quale JoglEventManager si è registrato nella
funzione main vista in precedenza*/
```

```
    public void display(GLDrawable drawable) {
        this._commandMediator.draw(drawable);
    }
```

```
/*L'equivalente di glutPostRedisplay(): questa funzione chiama display sul canvas
forzando in questo modo un evento di display*/
```

```
    private void refreshDisplay() {
        this._canvas.display();
    }
```

```
/*La funzione seguente viene chiamata quando si cambia il "displayMode",
correntemente non è implementata in Jogl, quindi viene lasciata vuota:*/
```

```
    public void displayChanged(GLDrawable drawable,
    boolean modeChanged, boolean deviceChanged) {
    }
```

```

/*Funzione di inizializzazione che viene chiamata solo la prima volta*/
    public void init(GLDrawable gLDrawable) {
        this._commandMediator.initalize(gLDrawable);
    }

/*Funzione che si occupa di gestire l'evento di Reshape, può chiamare direttamente
GL.glViewport(int, int, int, int), ma in questo caso passa la palla a CommandMediator*/
    public void reshape(GLDrawable gLDrawable, int x, int
y, int width, int height) {
        this._commandMediator.reshape(gLDrawable, x, y,
width, height);
    }

/*Poi ovviamente ci sono tutte le funzioni che gestiscono gli altri eventi per cui questa
classe si è aggiunta come listener nel main*/
    public void keyPressed(KeyEvent e) [...]

    public void keyReleased(KeyEvent e) [...]

    public void keyTyped(KeyEvent e) [...]

    public void mouseClicked(MouseEvent mouse) [...]

    public void mouseEntered(MouseEvent mouse) [...]

    public void mouseExited(MouseEvent mouse) [...]

    public void mousePressed(MouseEvent mouse) [...]

    public void mouseReleased(MouseEvent mouse) [...]

    public void mouseDragged(MouseEvent mouse) [...]

    public void mouseMoved(MouseEvent mouse) [...]

/*Infine presento un esempio di funzione di conversione delle coordinate del mouse da
MouseEvent al riferimento di Jogl, non strettamente necessaria, ma utile in molti casi.*/
    private Point2D.Float screenToJOGLWorld(MouseEvent
mouse) {
        Point2D.Float world = new Point2D.Float();

        Rectangle bounds = this._canvas.getBounds();
        float x = mouse.getX();
        float y = mouse.getY();

        world.x = ((x / (float) bounds.getMaxX()) *
2.0f) - 1.0f;
        world.y = (((float) bounds.getMaxY() - y) /
(float) bounds.getMaxY()) * 2.0f - 1.0f;

        return world;
    }

```

dal file CommandMediator.java:

A questo punto abbiamo creato un frame, aggiunto al frame un canvas opengl e abbiamo creato e registrato correttamente la classe che gestisce tutti gli eventi necessari. Il passo successivo è creare le classi che rappresentano i singoli oggetti da renderizzare e la classe intermedia Command Mediator. Partiamo da quest'ultima.

```
/*la classe contiene una serie di variabili private che costituiscono gli oggetti da
renderizzare*/
public class CommandMediator {
    private TriangleEqil _triangle;

    /*il costruttore istanzia tali oggetti*/
    public CommandMediator() {
        this._triangle = new TriangleEqilState(0.0f, 0.0f,
0.4f, 0.0f,
                                                    GLColor.GRE
EN);
    }

    /*La funzione draw è quella che viene chiamata direttamente da JoglEventMediator e
deve pulire lo schermo e successivamente chiamare i singoli metodi draw so ogni
oggetto*/
    public void draw(GLDrawable glDrawable) {
        final GL gl = glDrawable.getGL();
        gl.glClearColor(0.5f, 0.5f, 0.5f, 1.0f);
        gl.glClear(GL.GL_COLOR_BUFFER_BIT);

        triangle.draw(glDrawable);
    }

    /*funzione che viene chiamata direttamente da JoglEventMediator la prima volta per
l'inizializzazione*/
    public void initalize(GLDrawable glDrawable) {
        final GL gl = glDrawable.getGL();
        final GLU glu = glDrawable.getGLU();

        gl.glMatrixMode(GL.GL_PROJECTION);
        gl.glLoadIdentity();
        glu.gluOrtho2D(-1.0f, 1.0f, -1.0f, 1.0f); //
drawing square
        gl.glMatrixMode(GL.GL_MODELVIEW);
        gl.glLoadIdentity();
    }

    /*funzione chiamata direttamente da JoglEventMediator per l'evento di reshape*/
    public void reshape(GLDrawable glDrawable, int x, int
y, int width, int height) {

        final GL gl = glDrawable.getGL();
```

```

        final GLU glu = GLDrawable.getGLU();
        gl.glViewport(0, 0, width, height); // update the
viewport
    }

```

/*seguono tutte le funzioni che vengono chiamate durante la gestione degli eventi da mouse e tastiera per interagire con gli oggetti, e che devono appunto interagire con questi modificandone eventualmente alcune proprietà. Riporto un esempio di funzione utile per ruotare un generico oggetto ruotabile shape di degrees gradi nella direzione direction*/

```

private void rotateShape(IRotable shape, float
degrees,
                        RotatedDirection direction) {
    float rotation = direction.calculateRotation
(degrees);

    shape.setDegreesRotatedCCW
(shape.getDegreesRotatedCCW() + rotation);
}

```

dai file GLTriangle.java e IGLObject.java:

Ora non dobbiamo fare altro che scrivere una generica interfaccia che rappresenta il nostro oggetto renderizzabile e successivamente una classe per la implementa. Un esempio di interfaccia più generica possibile può essere IGLObject:

```

public interface IGLObject {
    public abstract void draw(GLDrawable glDrawable);
}

```

Eventualmente l'interfaccia può venire estesa da altre interfacce che rappresentano oggetti con proprietà più complesse, come per esempio IRotable:

```

public interface IRotable extends IGLObject {
    public abstract void setDegreesRotatedCCW(float);
    public abstract float getDegreesRotatedCCW();
}

```

Infine termino con questo tutorial con una classe che implementa in modo molto elementare solo la prima interfaccia

```

public class GLTriangle implements IRotable {

    /*variabili private interne che memorizzano le caratteristiche dell'oggetto*/
    private Point2D dot1, dot2, dot3;
    private GLColor color;
}

```

```
public GLTriangle() {
    dot1=new GLDot(0,0);
    dot2=new GLDot(1,1);
    dot3=new GLDot(2,0);
    color=new GLColor(GLColor.black);
}

/*Il fatidico metodo draw che renderizza se stesso*/
public void draw(GLDrawable glDrawable) {
    final GL gl = glDrawable.getGL();
    gl.glColor3fv(color.getColor());

    gl.glBegin(GL.GL_POLYGON);
    gl.glVertex2f(dot1.getX(),dot1.getY());
    gl.glVertex2f(dot2.getX(),dot2.getY());
    gl.glVertex2f(dot3.getX(),dot3.getY());
    gl.glEnd();
}
}
```