novática
UPGRADE Revista de la Asociación de Técnicos de Informática

Nº 184, noviembre-diciembre 2006, año XXXII

sumario

Monografía del próximo número: "*Buscadores en la Web*"

Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática), organización que edita también la revista **REICIS** (Revista Española de Innovación, Calidad e Ingeniería del Software). ***Novática*** edita asimismo **UP**GRADE, revista digital de **CEPIS** (*Council of European Professional Informatics Societies*), en lengua inglesa, y es miembro fundador de **UP**ENET (**UP**GRADE ***E**uropean **NET**work*).

<http://www.ati.es/novatica/>
<http://www.ati.es/reicis/>
<http://www.upgrade-cepis.org/>

ATI es miembro fundador de **CEPIS** (*Council of European Professional Informatics Societies*) y es representante de España en **IFIP** (*International Federation for Information Processing*); tiene un acuerdo de colaboración con **ACM** (*Association for Computing Machinery*), así como acuerdos de vinculación o colaboración con **AdaSpain**, **AI2**, **ASTIC**, **RITSI** e **Hispalinux**, junto a la que participa en **ProInnova**.

Coordinación Editorial, Redacción Central y Redacción ATI Madrid
Padilla 66, 3º, dcha., 28006 Madrid
Tlfn.914029391; fax.913093685 <novatica@ati.es>
Composición, Edición y Redacción ATI Valencia
Av. del Reino de Valencia 23, 46005 Valencia
Tlfn./fax 963330392 <secreval@ati.es>
Administración y Redacción ATI Cataluña
Via Laietana 46, ppal. 1ª, 08018 Barcelona
Tlfn.934125235, fax 934127713 <secregen@ati.es>
Redacción ATI Andalucía
Isaac Newton, s/n, Ed. Sadiel,
Isla Cartuja 41092 Sevilla, Tlfn./fax 954460779 <secreand@ati.es>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza.
Tlfn./fax 976235181 <secreara@ati.es>
Redacción ATI Asturias-Cantabria <gp-astucant@ati.es>
Redacción ATI Castilla-La Mancha <gp-clmancha@ati.es>
Suscripción y Ventas
<http://www.ati.es/novatica/interes.html>, o en ATI Cataluña o ATI Madrid
Publicidad
Padilla 66, 3º, dcha., 28006 Madrid
Tlnf.914029391; fax.913093685 <novatica@ati.es>
Imprenta
Derra S.A., Juan de Austria 66, 08005 Barcelona.
Depósito legal: B 15.154-1975 -- ISSN: 0211-2124; CODEN NOVAEC
Portada: "Alan Turing & friends" (variaciones sobre una foto tomada de www.tuning.org).
RFCalvo / © Rafael Fernández Calvo 2007
Diseño: Fernando Agresta / © ATI 2006

Programación de Aplicaciones Gráficas con OpenGL y Java

Óscar Belmonte Fernández
Dpt. Lenguajes y Sistemas Informáticos, Universitat Jaume I, Castellón

<oscar.belmonte@uji.es>

Resumen: *en la actualidad la mayor parte de aplicaciones gráficas se desarrollan utilizando el lenguaje de programación C/C++. El número de aplicaciones gráficas utilizando este lenguaje de programación es abrumadoramente superior a las aplicaciones construidas con otros lenguajes de programación. No obstante, hoy en día es posible utilizar las principales bibliotecas gráficas desde otros lenguajes de programación, como por ejemplo Java. Las características de Java han hecho que el número de aplicaciones basadas en este lenguaje haya crecido de modo espectacular. Y las aplicaciones gráficas basadas en OpenGL no han sido menos. Este artículo muestra cómo programar OpenGL desde Java utilizando la Java API for OpenGL (JOGL abreviadamente). Se mostrará que esta API ofrece un estilo de programación de aplicaciones gráficas con OpenGL muy similar al que se utiliza con C/C++ y glut. Se verá, asimismo, como integrar el lienzo para dibujar gráficos OpenGL con componentes Swing, para crear interfaces de usuario ricas.*

Palabras clave: *gráficos interactivos, Java, OpenGL.*

1. Introducción

No cabe duda que la mayor parte de las aplicaciones gráficas se desarrollan, o han sido desarrolladas, utilizando principalmente el lenguaje de programación C/C++. Esto se debe a que las principales bibliotecas para la creación de aplicaciones gráficas (OpenGL y DirectX) se han desarrollado utilizando principalmente estos lenguajes de programación.

Durante mucho tiempo se ha mantenido el prejuicio de que Java es un lenguaje de programación lento en la ejecución porque es interpretado. Nada más lejos de la realidad. Desde sus inicios, acercar la velocidad de ejecución del Java a otros lenguajes de programación que podríamos llamar clásicos, como C/C++ y Fortran, ha sido uno de los principales objetivos de sus desarrolladores. Tal es así que en la actualidad existen test que muestran que el rendimiento de Java se acerca al de C/C++ o Fortran.

En la actualidad es posible programar OpenGL desde infinidad de lenguajes de programación. En <http://nehe.gamedev.net> se puede encontrar una buena cantidad de ejemplos de aplicaciones OpenGL programadas con distintos lenguajes de programación.

En este artículo se muestra como programar aplicaciones gráficas con OpenGL desde Java. En la **sección 2** se muestra cual es el estado actual de Java y sus principales características. En la **sección 3** se dan referencias de algunos ejemplos de aplicaciones gráficas desarrolladas en OpenGL y Java. La **sección 4** muestra como desarrollar una aplicación en OpenGL y Java utilizando el método de rellamada, como combinar componentes Swing con OpenGL y finalmente como convertir estas aplicaciones en applets. La **sección 5** es una recapitulación de consejos que se deben tener en cuenta al programar OpenGL desde Java. Finalmente, la **sección 6** indica al lector interesado donde conseguir más información.

2. Estado Actual de Java

Java es un excelente lenguaje de programación para desarrollar todo tipo de aplicaciones, desde pequeñas aplicaciones para equipos de sobremesa hasta grandes aplicaciones corporativas basadas en Internet, pasando por aplicaciones para dispositivos móviles, como teléfonos o PDAs.

En el momento de escribir este artículo la versión estable de Java 2 Standard Edition es las 1.5, también llamada 5.0, y desde la página web de Sun <java.sun.com> se puede descargar la nueva versión 6.0 (Mustang) en su versión beta.

2.1. Características de Java

Las principales características de Java son [1]:

- Java es multiplataforma, el mismo código compilado Java se puede ejecutar bajo una gran cantidad de sistemas operativos (OS X, Solaris, Linux, Microsoft XP, etcétera), y una gran cantidad de plataformas hardware (PowerPC, Intel, Sun, etcétera).
- Java es orientado a objetos. Todo en Java es un objeto (excepto los tipos de datos primitivos, pero para estos tipos Java proporciona los recubridores, (*wrappers* en inglés).
- Java hace control estricto de tipos.
- Java es seguro, tanto desde el punto de vista del programador, como desde el punto de vista del usuario.
- La concurrencia es una de las características del lenguaje, no es necesario importar librería externas; además, Java está orientado a la programación en red desde sus inicios.

Existe un sinfín de falsas creencias alrededor de Java, que han quedado en el inconsciente colectivo de los programadores desde su lanzamiento, y que se han ido heredando de generación en generación de programadores, la más extendida es que Java es un lenguaje de programación lento porque es interpretado. Esto era cierto para las primeras versiones de Java, pero desde la incorporación de la tecnología JIT (del inglés *just in time compilation*) [2] se superó el problema de la lentitud de ejecución. Brevemente, la tecnología JIT consiste en que la primera vez que se llama a un método su código es compilado, en tiempo de ejecución, a código nativo de la máquina donde se ejecuta la aplicación, y se almacena, de tal modo que la próxima vez que se llame al mismo método ya se ejecuta el código nativo del método. Si nuestras aplicaciones están constantemente ejecutando un pequeño conjunto de métodos (como en el caso de los juegos por ordenador por ejemplo) la tecnología JIT da excelentes resultados desde el punto de vista del tiempo de ejecución. Es más, ya que la compilación se realiza en tiempo de ejecución, algunos autores [9] sostienen que la velocidad de ejecución de Java superará a lenguajes como C/C++ y Fortran ya que se dispone de más información, en el momento de la compilación, que con la compilación "estática" de los lenguajes "clásicos".

Dentro de la comunidad Java se ha hecho un gran esfuerzo por mostrar las prestaciones de este lenguaje frente a otros lenguajes de programación. Para una visión general se pueden consultar las referencias [3] [4] [5] [7] [8].

3. Ejemplos de aplicaciones gráficas desarrollados en Java

Existe gran cantidad de herramientas para la construcción de aplicaciones gráficas, orientadas a la visualización de datos <http://www.kdnuggets.com/software/visualization.html>, a la creación de juegos para ordenador <http://ogre4j.org/drupal>, o a la manipulación de imágenes, por citar algunas de las más comunes.

Andrew Davison ha publicado un excelente libro titulado *Killer Game Programming in Java* [6]. En el capítulo de introducción

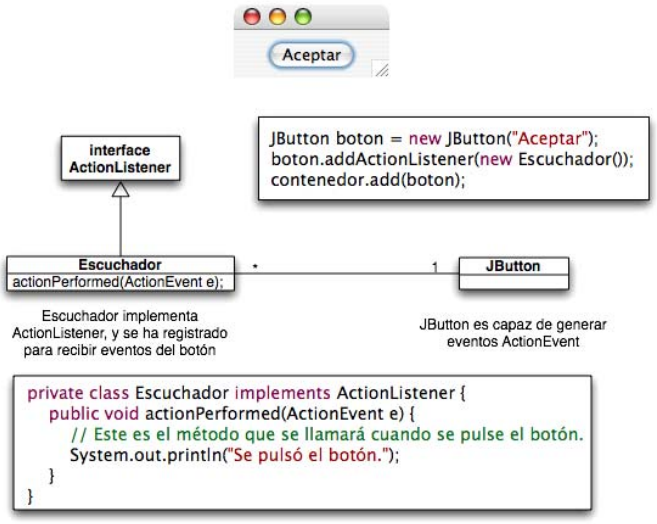


Figura 1. Un ejemplo sencillo de GUI. La clase que recibe los eventos del botón (Escuchador) debe implementar la interfaz ActionListener y registrarse para recibir los eventos generados por el botón. Cada vez que se pulse sobre el botón, se llamará al método actionPerformed(ActionEvent e) de esta clase.

muestra enlaces a numerosos juegos por ordenador desarrollados en Java.

4. Cómo se construye una aplicación OpenGL con Java

Sun, ha hecho disponible un API con el que poder programar OpenGL desde Java, y esta API se ha desarrollado siguiendo un *Java Specification Request*, lo que significa que el proceso está abierto a todo aquel desarrollador interesado en contribuir en la especificación. La última versión es la JSR-231 beta 0.5. <https://jogl.dev.java.net/>. Este API proporciona acceso a la especificación OpenGL 2.0, así como a extensiones de los fabricantes (incluye el lenguaje para shaders Cg de Nvidia).

Los paquetes de esta API incluyen funciones de las bibliotecas GLU y GLUT conocidas a los programadores de OpenGL, excepto aquellas relacionadas con el sistema de ventanas y la gestión de eventos, que evidentemente en Java se hace con AWT o Swing.

En esta especificación se puede programar OpenGL en modo de rellamada o en modo inmediato. En este artículo se muestra cómo programar OpenGL desde Java utilizando el modo de rellamada.

4.1. Aplicaciones independientes

La idea subyacente a la programación de OpenGL desde Java en modo de rellamada es la misma que cuando se escriben aplicaciones de interfaz gráfica basadas en AWT o Swing, por un lado utilizamos componentes capaces de recibir eventos, en este caso del usuario, y por otro lado construimos clases que reciben esos eventos gracias a que implementan la interfaz necesaria y se han registrado para recibir los eventos. La **figura 1** muestra el procedimiento en el caso de componentes Swing.

En el caso de OpenGL, por un lado las dos clases capaces de generar eventos son GLCanvas y GLJPanel. La primera de ella es análoga a las clases de interfaz gráfico en el paquete AWT (hay que tener cuidado al combinarlas con clases Swing); la segunda de ellas se integra perfectamente con las clases del paquete Swing sacrificando velocidad durante la fase de dibujo, es decir, dibujar utilizando GLCanvas es más rápido que utilizando GLJPanel.

Por otro lado, si queremos que una clase reciba los eventos que generan las dos anteriores clases debe implementar la interfaz GLEventListener. Esta interfaz declara cuatro métodos, que son los que se rellamarán cada vez que se produzca un evento sobre GLCanvas.

Estos cuatro métodos son:
init(GLAutoDrawable drawable); Este método se llama inmediatamente después a la creación de un contexto OpenGL.
reshape(GLAutoDrawable drawable, int x, int y, int width, int height); Este método se llama durante el primer dibujo de la escena y después que el componente a fijado su tamaño. También se llama cada vez que la ventana cambia de tamaño.
display(GLAutoDrawable drawable); Este es el evento al que debe responder el cliente cada vez que se procede a dibujar la escena.
displayChanged(GLAutoDrawable drawable, boolean modeChanged, boolean deviceChanged); Este método es llamado cada vez que el dispositivo de visualización cambia.

Vamos a crear una sencilla aplicación que muestre una esfera en modo alambre para ilustrar como programar OpenGL desde Java. La **figura 2** muestra el esquema que guía la construcción de esta aplicación.

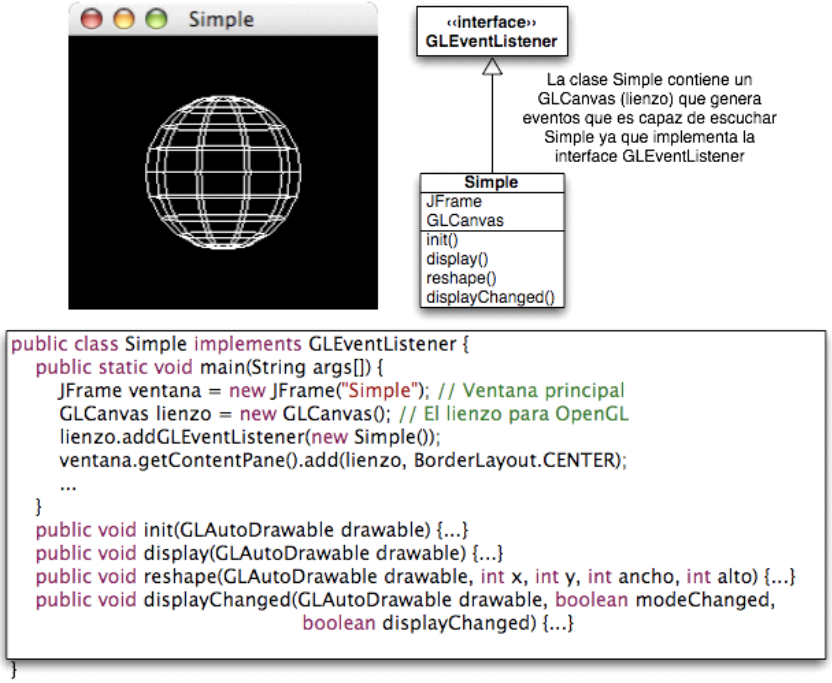


Figura 2. En esta figura se muestra el marco para programar OpenGL desde Java. La idea básica es que debemos utilizar un lienzo que nos de acceso a OpenGL (clase GLCanvas) y crear una clase que responda a los eventos de GLCanvas, implementando la interfaz GLEventListener y registrándose como escuchador de estos eventos con el método addGLEventListener().

```
import javax.media.opengl.GL;
import javax.media.opengl.glu.GLU;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLCanvas;

import javax.swing.JFrame;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.BorderLayout;

import com.sun.opengl.util.GLUT;

public class Simple implements GLEventListener {
   public static void main(String args[]) {
      JFrame ventana = new JFrame("Simple"); // Ventana principal
      GLCanvas lienzo = new GLCanvas(); // El lienzo para OpenGL
      lienzo.addGLEventListener(new Simple());
      ventana.getContentPane().add(lienzo, BorderLayout.CENTER);
      ventana.setSize(200, 200);
      ventana.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            System.exit(0);
         }
      });
      ventana.setVisible(true);
   }

   public void init(GLAutoDrawable drawable) {
      GL gl = drawable.getGL();

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

   public void reshape(GLAutoDrawable drawable, int x, int y, int ancho, int alto) {
      GL gl = drawable.getGL(); // Recupero el contexto
      float h = (float)alto / (float)ancho; // Relación alto-ancho

      gl.glMatrixMode(GL.GL_PROJECTION); // Selecciono la matriz de proyección
      gl.glLoadIdentity(); // Cargo la identidad sobre la matriz de proyección
      gl.glFrustum(-1.0f, 1.0f, -h, h, 5.0f, 60.0f); // Fijo el tamaño del frustrum
      gl.glMatrixMode(GL.GL_MODELVIEW); // Selecciono la matriz modelo-vista
      gl.glLoadIdentity(); // Cargo sobre la matriz modelo-vista la identidad
   }

   public void display(GLAutoDrawable drawable) {
      GL gl = drawable.getGL();
      GLU glu = new GLU();
      GLUT glut = new GLUT();

      gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
      gl.glPushMatrix();
      glu.gluLookAt(0.0, 0.0, -10.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
      gl.glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
      glut.glutWireSphere(1.0, 10, 10);
      gl.glPopMatrix();
   }

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

Figura 3. Esta figura muestra el listado completo para una aplicación gráfica en Java utilizando JOGL.

En este primer ejemplo se detalla todo el proceso de construcción y se muestra el código completo de la aplicación. En los ejemplos siguientes sólo se mostrará las partes del código que sean de interés. El código completo de este primer ejemplo aparece en la **figura 3**.

Lo primero que se necesita es importar las clases que se van a utilizar. JOGL (*Java API for OpenGL*) está formado por seis paquetes, de ellos se importan clases de los paquetes: javax.media.opengl, javax.media.opengl.GLU, y com.sun. opengl.util, el resto de paquetes (javax.awt, javax.awt.event y javax.swing) incorporan las clases necesarias para construir la interfaz gráfica de usuario (GUI).

La única clase que forma la aplicación es la que recibe los eventos de GLCanvas, por lo tanto debe implementar la interfaz GLEventListener. Como ya sabemos esta interfaz declara cuatro métodos, que serán los que debemos definir para responder a cada uno de los cuatro eventos que GLCanvas es capaz de generar (tres en realidad, displayChanged() no se implementa).

Veamos qué se hace en cada uno de estos métodos. El primero que aparece en el listado es init() donde vamos a definir las propiedades de OpenGL para esta aplicación. En este ejemplo tan sencillo lo único que definimos es el color de borrado mediante el método glClearColor() de la clase GL. Y aquí aparece el primer detalle en el que debe fijarse el programador de OpenGL al utilizar Java y el modelo de rellamada: el acceso a los métodos de OpenGL se hace a través de la clase GL y ***la instancia de esa clase se debe recuperar del parámetro GLAutoDrawable con que se llama a init(), y nunca crear una instancia de la clase GL nosotros mismos, menos aún declararla atributo de la clase para poder acceder a ella desde nuestros propios métodos***. Esto se debe a que GLAutoDrawable está ligada al contexto gráfico correcto con cada llamada a los métodos de rellamada. Si creamos nosotros mismos una instancia de la clase GL no estará ligada a ningún contexto gráfico válido y se producirá un error en nuestra aplicación al intentar utilizar la instancia.

El siguiente método de la interfaz GLEventListener que se define es reshape(). En este método, al igual que en init(), accedemos a la instancia de la clase GL recuperándola del atributo GLAutoDrawable. Los otros parámetros del método indican la posición de la esquina superior izquierda, el ancho y alto de la ventana. En este método definimos el tipo de proyección que vamos a utilizar, perspectiva en este caso; iniciamos la matriz modelo-vista con la identidad, y finalmente dejamos cargada esta matriz como la actual.

El tercer método de este sencillo ejemplo es display() donde realizamos todas las tareas de dibujo. De nuevo, lo primero que hacemos es recuperar una instancia de la clase GL a través del parámetro GLAutoDrawable. Y creamos dos instancias, locales a este método, de las clases GLU y GLUT. Como ya he comentado, a través de estas clases se tiene la misma funcionalidad que a través de las funciones que en C empiezan por glu* y glut* respectivamente, excepto las relacionadas con la gestión de ventanas. Y aquí es donde aparece el segundo detalle que debemos fijar al programar OpenGL utilizando Java y el modelo de rellamada: ***nunca debemos declarar como atributos de nuestra clase instancias de GLU y GLUT para acceder a ellas desde los métodos de rellamada. Siempre deberemos crear instancias de ellas en estos métodos***. El resto del código es muy sencillo de interpretar para los programadores de OpenGL, simplemente se borra el buffer de color con gl.glClear(), se aíslan las transformaciones para la siguiente llamada al mismo método mediante el par gl.glPushMatrix()-gl.glPopMatrix(), definimos la posición del punto de vista con glu.

gluLookAt() y una transformación con gl.glRotatef(), y finalmente dibujamos una esfera en modo alambre con glut.glut WireSphere().

El cuarto de los métodos que debemos definir displayChanged(), como he comentado anteriormente no está implementado en la versión actual de JOGL, sencillamente lo dejamos vacío.

Nótese que todos estos métodos ya son conocidos por todo aquel que utiliza C/C++ para programar OpenGL, y que la interfaz de programación para OpenGL utilizando el modelo de rellamada es prácticamente igual que en el caso de C, a excepción del par de detalles que se debe fijar, por lo que la curva de aprendizaje de programación de OpenGL desde Java resulta muy suave para los programadores de OpenGL en C.

El código completo de los ejemplos que aparecen en este artículo se puede encontrar en <http://www4.uji.es/~belfern/JOGL/Ejemplos/>.

4.2. Uso de JOGL y Swing

Nuestras aplicaciones gráficas, en general, no sólo muestran un lienzo OpenGL, sino que además nos permiten interaccionar con la aplicación a través de una interfaz gráfica de usuario, que puede estar compuesta por botones, barras de deslizamiento, cajas de selección y todo el arsenal de componentes gráficos que Swing pone a nuestra disposición.

Utilizar lienzos OpenGL e interfaces de usuario Swing es muy sencillo siempre que se fijen, como en el apartado anterior, unas sencillas reglas de programación. Como ejemplo se va a mostrar una sencilla aplicación con un lienzo OpenGL que muestre una esfera sólida rotando en torno al eje de las equis y que se controla mediante una muy sencilla interfaz programada utilizando Swing. La rotación se iniciará cuando se pulse el botón *Inicio* (se creará un hilo, instancia de la clase Thread, encargado de ir incrementando el ángulo de rotación) y se detendrá al pulsar el botón *Parar* (se destruirá el hilo).

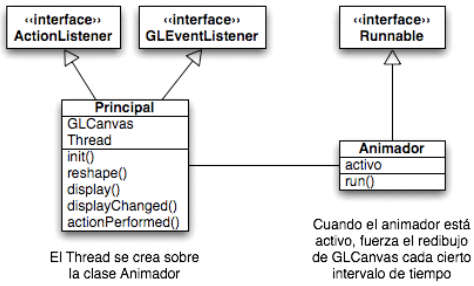
En la **figura 4** se muestra la arquitectura de esta aplicación. En esta figura sólo se muestra el código relevante al tema que estamos tratando. Lo ideal en estos casos es utilizar un patrón Modelo-Vista-Controlador para desarrollar la aplicación, pero no lo haremos para que el lector pueda centrar su atención en cómo combinar Swing con JOGL.

La primera diferencia con respecto al sencillo ejemplo del caso anterior es que nuestra nueva clase Principal implementa no solamente la interfaz GLEventListener, sino también la interfaz ActionListener, de este modo se podrá registrar como escuchadora de los eventos que produzcan los botones que incluye la interfaz de usuario. Por otro lado, la clase Animador sirve exclusivamente para crear sobre ella un Hilo que controle la rotación de la esfera, y es ella la que interaccionará con el lienzo OpenGL cada vez que haya que dibujar la escena.

Como la clase Principal implementa la interfaz ActionListener, estamos obligados a definir el método actionPerformed(), y a registrar la clase Principal como escuchadora de los eventos de los botones. Este método se llama cada vez que se pulsa uno de los dos botones (registramos Principal como escuchadora de los dos botones), y simplemente lanza un hilo de control de la rotación, si es que no existe ninguno, o detiene el hilo de control de la rotación, si es que existe uno.

El detalle verdaderamente importante lo contiene la clase Animador en su método run(). En este método se fuerza el dibujo del lienzo OpenGL, pero al hacerlo, no se llama directamente al método display() de la clase Principal, sino que se fuerza el dibujo llamando al método repaint() de GLCanvas.

Este es el nuevo detalle que debemos fijar al programar OpenGL desde Java en modo de rellamada: ***Nunca debemos llamar a los métodos de la interfaz GLEventListener desde hilos distintos al que controla a GLEventListener.*** En general, nunca llamaremos de modo explícito a los métodos de la interfaz GLEventListener.




```
public class Principal implements GLEventListener, ActionListener {
   private GLCanvas lienzo;
   private Animador animador;
   private Thread threadAnimador;
   private float angulo = 90.0f;
   private GLU glu; // Se puede crear como atributo de la clase.
   private GLUT glut; // También como atributo de la clase, pero sólo se pueden utilizar
                      // en los métodos de callback, nunca se pueden llamar directamente.
   ...
   public void actionPerformed(ActionEvent e) {
      String boton = e.getActionCommand();
      if(boton.equals("inicio") && animador == null) {
         iniciaAnimador();
         System.out.println("Se pulsó inicio.");
      }
      else if(boton.equals("para") && animador != null) {
         detenAnimador();
         System.out.println("Se pulsó para.");
      }
   }
```

```
public class Animador implements Runnable {
   ....
   public void run() {
      activo = true;
      while(activo) {
         principal.incAngulo();
         principal.redibuja();
         try { Thread.sleep(50); }
         catch(InterruptedException e) { System.out.println(e.getMessage()); }
      }
      System.out.println("Elimado el animador");
   }
```

Figura 4. Modelo de eventos para la clase GLCanvas. Para recibir los eventos que genera GLCanvas se debe crear una clase que implemente la interfaz GLEventListener. También se muestra la clase Animador, encargada de iniciar y detener la animación.

```
public class Principal extends JApplet implements GLEventListener, ActionListener {
   private GLCanvas lienzo;
   ....
   private void creaGUI(Container contenedor) {
      lienzo = new GLCanvas(); // El lienzo para OpenGL
      lienzo.addGLEventListener(this);
      JPanel panel = new JPanel();
      panel.setLayout(new BorderLayout());
      JPanel panelBotones = new JPanel();
      JButton inicio = new JButton("Inicio");
      inicio.addActionListener(this);
      inicio.setActionCommand("inicio");
      ....
   }

   /*
      * Este corresponde al applet
      */
   public void init() {
      creaGUI(getContentPane());
   }
```

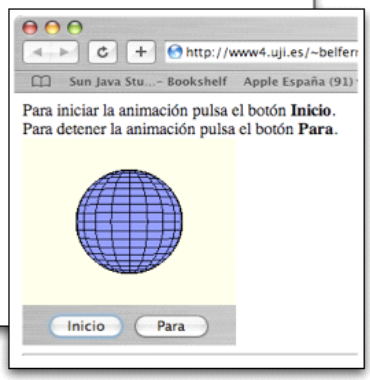


Figura 5. Modificaciones para convertir nuestra aplicación en un applet.

4.3. Applets

Si hemos tenido la precaución de aislar en nuestro código la construcción de la interfaz de usuario, para convertir nuestra aplicación en un applet debemos seguir las mismas reglas que para convertir en applet cualquier otro tipo de aplicación.

Esta vez nuestra clase extenderá a JApplet y en el método init() de la clase JApplet escribiremos el código necesario para crear el lienzo OpenGL y el resto de la interfaz de usuario. En la **figura 5** se muestra los cambios necesarios.

Con estos simples cambios hemos conseguido que nuestra aplicación funcione tanto como una aplicación independiente como un applet para incrustarlo en nuestra páginas web. El único detalle es que un usuario que desee ver nuestra página web, con un applet utilizando JOGL, deberá tener instalado JOGL en su máquina, de lo contrario la máquina virtual de Java no encontrará las clases necesarias para ejecutar la aplicación.

Para evitar el problema de la dependencia de nuestra aplicación de ciertas bibliotecas, podemos utilizar la tecnología Java Web Start de Sun. Esta tecnología, que está incluida en el Java desde la versión 1.4, se encarga de comprobar que en la máquina cliente existen las bibliotecas necesarias para ejecutar la aplicación, y de no ser así es capaz de descargar las bibliotecas ausentes antes de ejecutar la aplicación. Y todo esto de modo transparente al usuario.

5. Resumen

En este artículo se muestra como construir una aplicación gráfica utilizando Java y OpenGL en modo de rellamada. El procedimiento es muy similar al que se utiliza cuando se programa OpenGL desde C/C++ utilizando la biblioteca glut.

La idea es análoga a la utilizada al programar interfaces de usuario basadas en Swing: un lienzo OpenGL (clase GLCanvas) es capaz de generar eventos (init(), reshape(), display() y displayChanged() y para responder a ellos lo hemos de hacer desde una clase que implemente la interfaz GLEventListener.

Las tres máximas que hemos de tener siempre presente son:

- Las instancias de esa clase GLCanvas se deben recuperar a partir del parámetro GLAutoDrawable con que se llama a los métodos declarados en la interfaz GLEventListener, y nunca se deben crear con el operador new().
- Nunca debemos declarar en nuestras clases atributos de tipo GLU o GLUT para acceder a ellas desde los métodos de rellamada. Siempre deberemos crear instancias de estos tipos en los métodos de la interfaz GLEventListener.
- Nunca debemos llamar a los métodos de la interfaz GLEventListener desde hilos distintos al que controla a GLEventListener.

6. Información adicional

La página imprescindible para conocer la Java API for OpenGL (JOGL) es la del proyecto <https://jogl.dev.java.net/>.

En NeHe <http://nehe.gamedev.net> encontraremos código de los ejemplos para aprender OpenGL portado a JOGL.

En la página web de [6] <http://fivedots.coe.psu.ac.th/~ad/jg/> disponemos de un par de capítulos, no incluidos en el libro, que introducen a la programación de JOGL con ejemplos muy descriptivos.

Podemos ver ejemplos impresionantes programados en JOGL y utilizando la tecnología Java Web Start en <https://jogl-demos.dev.java.net/>. Para ver estos ejemplos no es necesario que tengamos la API instalada.

Un buen libro para iniciarse en la programación de JOGL y conteniendo numerosos ejemplos es [10]. Lo podemos comprar directamente al autor desde su página web <http://www.genedavissoftware.com/books/jogl/>

En la página web de Sun <http://java.sun.com/developer/JDCTechTips/2005/tt0208.html#1> encontraremos aún más información sobre estos temas.

Referencias

[1] K. Arnold et al. *El lenguaje de programación Java*. Addison-Wesley, 4ª Edición, 2005. ISBN: 978-0201704334.

[2] J. Aycock. "A Brief History of Just-In-Time". *ACM Computing Surveys, Vol. 35(2), pp: 97-113*, 2003.

[3] R. F. Boisvert, J. Moreira, M. Philippsen, R. Pozo. "Java and Numerical Computing". *IEEE Computing in Science and Engineering, Vol. 3(2), pp: 18-24,* 2001.

[4] J. M. Bull, L. A. Smith, M. D. Westhead, D. S. Henty, R. A. Davey. "A Benchmark Suite for High Performance Java", *Concurrency: Practice and Experience, Vol. 12, pp: 375-388*, 2000.

[5] J. M. Bull, L. A. Smith, L. Pottage, R. Freeman. "Benchmarking Java against C and Fortran for Scientific Application". *Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande, pp: 97-105*, 2001.

[6] A. Davison. *Killer Game Programming in Java*. O'Reilly. 2005. ISBN: 978-0596007300.

[7] Java Grande Forum. <http://www.javagrande.org>.

[8] J.P.Lewis, Ulrich Neumann. *Performance of Java versus C++*. <http://www.idiom.com/~zilla/Computer/javaCbenchmark.html>.

[9] Kirk Reinholtz. "Java will be faster than C++". *ACM SIGPLAN Notice, Vol. 35(2), pp: 25-28,* 2000.

[10] Gene Davis. *Learning Java Bindings for OpenGL (JOGL)*. Authorhouse, 2004. ISBN: 978-1420803624.