Grabación de audio/vídeo y gráficos avanzados en Android

Índice

1 Grabando vídeo y audio..... 2
1.1 Usando Intents para capturar vídeo..... 2
1.2 Usando la clase MediaRecorder..... 3
1.3 Configurando y controlando la grabación de vídeo..... 4
1.4 Previsualización..... 5
2 Sintetizador de voz de Android..... 6
3 Gráficos 3D..... 8

En esta sesión continuamos examinando las capacidades multimedia de Android presentando el sintetizados de voz *Text to Speech*, el cual permitirá que una actividad reproduzca por los altavoces la lectura de un determinado texto. Se trata de un componente relativamente sencillo de utilizar que puede mejorar la accesibilidad de nuestras aplicaciones en gran medida.

Las últimas versiones del SDK de Android permiten emular la captura de video y audio en nuestros dispositivos virtuales. En concreto, es posible realizar esta simulación por medio de una webcam, que se utilizará para captar lo que se supone que estaría captando la cámara del dispositivo real. Desafortunadamente esta característca no parace estar disponible aun en sistemas Mac.

Otro objetivo de la sesión será hacer una introducción a la generación de gráficos 3D en Android. En sesiones anteriores hemos visto como modificar la interfaz de usuario de nuestra aplicación para mostrar nuestros propios componentes, que mostraban unos gráficos y un comportamiento personalizados. En esta sesión veremos cómo incorporar contenido 3D en esos componentes propios por medio de OpenGL.

1. Grabando vídeo y audio

Android ofrece dos alternativas para la grabación de vídeo o audio en nuestra aplicación. La solución más sencilla consiste en la utilización de *Intents* para lanzar la aplicación nativa de cámara de vídeo. Esto permite especificar en qué lugar se guardará el vídeo o audio resultante, así como indicar la calidad de grabación, dejando a la aplicación nativa el resto de tareas, como por ejemplo el manejo de errores. En el caso en el que se desee reemplazar la aplicación nativa de vídeo o se quiera tener más control sobre la grabación sería posible utilizar la clase `MediaRecorder`.

Hemos de tener en cuenta que para que nuestra aplicación pueda grabar audio o vídeo en Android es necesario incluir los permisos necesarios en el *Manifest*:

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.RECORD_VIDEO"/>
```

1.1. Usando Intents para capturar vídeo

La manera más sencilla de comenzar a grabar vídeo es mediante la constante `ACTION_VIDEO_CAPTURE` definida en la clase `MediaStore`, que deberá utilizarse conjuntamente con un `Intent` que se pasará como parámetro a `startActivityForResult`:

```
startActivityForResult(new Intent(MediaStore.ACTION_VIDEO_CAPTURE),
GRABAR_VIDEO);
```

Esto lanzará la aplicación nativa de grabación de vídeos en Android, permitiendo al

usuario comenzar o detener la grabación, revisar lo que se ha grabado, y volver a comenzar la grabación en el caso en el que se desee. La ventaja como desarrolladores será la misma de siempre: al utilizar el componente nativo nos ahorramos el tener que desarrollar una actividad para la captura de vídeo desde cero.

La acción de captura de vídeo que se pasa como parámetro al `Intent` acepta dos párametros extra opcionales, cuyos identificadores se definen como constantes en la clase `MediaStore`:

- `EXTRA_OUTPUT`: por defecto el vídeo grabado será guardado en el *Media Store*. Para almacenarlo en cualquier otro lugar indicaremos una URI como paráemtro extra utilizando este identificador.
- `EXTRA_VIDEO_QUALITY`: mediante un entero podemos especificar la calidad del vídeo capturado. Sólo hay dos valores posibles: 0 para tomar vídeos en baja resolución y 1 para tomar vídeos en alta resolución (este último valor es el que se toma por defecto).

A continuación se puede ver un ejemplo en el que se combinan todos estos conceptos vistos hasta ahora:

```
private static int GRABAR_VIDEO = 1;
private static int ALTA_CALIDAD = 1;
private static int BAJA_CALIDAD = 0;

private void guardarVideo(Uri uri) {
        Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);

        // Si se define una uri se especifica que se desea almacenar el
        // vídeo en esa localización. En caso contrario se hará uso
        // del Media Store
        if (uri != NULL)
                intent.putExtra(MediaStore.EXTRA_OUTPUT, output);

        // En la siguiente línea podríamos utilizar cualquiera de las
        // dos constantes definidas anteriormente: ALTA_CALIDAD o
BAJA_CALIDAD
        intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, ALTA_CALIDAD);

        startActivityForResult(intent, GRABAR_VIDEO);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
        if (requestCode == GRABAR_VIDEO) {
                Uri videoGrabado = data.getData();
                // Hacer algo con el vídeo
        }
}
```

1.2. Usando la clase MediaRecorder

Otra alternativa para guardar audio o vídeo que más tarde puede ser utilizado en nuestra aplicación o almacenado en el *Media Store* consiste en hacer uso de la clase `MediaRecorder`. La creación de un objeto de esta clase es sencilla:

```
MediaRecorder mediaRecorder = new MediaRecorder();
```

La clase `MediaRecorder` permite especificar el origen del audio o vídeo, el formato del fichero de salida y los codecs a utilizar. Como en el caso de la clase `MediaPlayer`, la clase `MediaRecorder` maneja la grabación mediante una máquina de estados. Esto quiere decir que el orden en el cual se inicializa y se realizan operaciones con los objetos de este tipo es importante. En resumen, los pasos para utilizar un objeto `MediaRecorder` serían los siguientes:

- Crear un nuevo objeto `MediaRecorder`.
- Asignarle las fuentes a partir de las cuales grabar el contenido.
- Definir el formato de salida.
- Especificar las características del vídeo: codec, framerate y resolución de salida.
- Seleccionar un fichero de salida.
- Prepararse para la grabación.
- Realizar la grabación.
- Terminar la grabación.

Una vez finalizamos la grabación hemos de hacer uso del método `release` del objeto `MediaRecorder` para liberar todos sus recursos asociados:

```
mediaRecorder.release();
```

1.3. Configurando y controlando la grabación de vídeo

Como se ha indicado anteriormente, antes de grabar se deben especificar la fuente de entrada, el formato de salida, el codec de audio o vídeo y el fichero de salida, en ese estricto orden.

Los métodos `setAudioSource` y `setVideoSource` permiten especificar la fuente de datos por medio de constantes estáticas definidas en `MediaRecorder.AudioSource` y `MediaRecorder.VideoSource`, respectivamente. El siguiente paso consiste en especificar el formato de salida por medio del método `setOutputFormat` que recibirá como parámetro una constante entre las definidas en `MediaRecorder.OutputFormat`. A continuación usamos el método `setAudioEnconder` o `setVideoEncoder` para especificar el codec usado para la grabación, utilizando alguna de las constantes definidas en `MediaRecorder.AudioEncoder` o `MediaRecorder.VideoEncoder`, respectivamente. Es en este punto en el que podremos definir el framerate o la resolución de salida si se desea. Finalmente indicamos la localización del fichero donde se guardará el contenido grabado por medio del método `setOutputFile`. El último paso antes de la grabación será la invocación del método `prepare`.

El siguiente código muestra cómo configurar un objeto `MediaRecorder` para capturar audio y vídeo del micrófono y la cámara usando un codec estándar y grabando el resultado en la tarjeta SD:

```
MediaRecorder mediaRecorder = new MediaRecorder();
```

```
// Configuramos las fuentes de entrada
mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
mediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);

// Seleccionamos el formato de salida
mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);

// Seleccionamos el codec de audio y vídeo
mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
mediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.DEFAULT);

// Especificamos el fichero de salida
mediaRecorder.setOutputFile("/mnt/sdcard/mificherodesalida.mp4");

// Nos preparamos para grabar
mediaRecorder.prepare();
```

Aviso:

Recuerda que los métodos que hemos visto en el ejemplo anterior deben invocarse en ese orden concreto, ya que de lo contrario se lanzará una excepción de tipo *Illegal State Exception*.

Para comenzar la grabación, una vez inicializados todos los parámetros, utilizaremos el método `start`:

```
mediaRecorder.start();
```

Cuando se desee finalizar la grabación se deberá hacer uso en primer lugar del método `stop`, y a continuación invocar el método `reset`. Una vez seguidos estos pasos es posible volver a utilizar el objeto invocando de nuevo a `setAudioSource` y `setVideoSource`. Llama a `release` para liberar los recursos asociados al objeto `MediaRecorder` (el objeto no podrá volver a ser usado, se tendrá que crear de nuevo):

```
mediaRecorder.stop();
mediaRecorder.reset();
mediaRecorder.release();
```

1.4. Previsualización

Durante la grabación de vídeo es recomendable mostrar una previsualización de lo que se está captando a través de la cámara en tiempo real. Para ello utilizaremos el método `setPreviewDisplay`, que nos permitirá asignar un objeto `Surface` sobre el cual mostrar dicha previsualización.

El comportamiento en este caso es muy parecido al de la clase `MediaPlayer` para la reproducción de vídeo. Debemos definir una actividad que incluya una vista de tipo `SurfaceView` en su interfaz y que implemente la interfaz `SurfaceHolder.Callback`. Una vez que el objeto `SurfaceHolder` ha sido creado podemos asignarlo al objeto `MediaRecorder` invocando al método `setPreviewDisplay`, tal como se puede ver en el siguiente código. El vídeo comenzará a previsualizarse tan pronto como se haga uso del método `prepare`.

```
public class MyActivity extends Activity implements SurfaceHolder.Callback
{

        private MediaRecorder mediaRecorder;

        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);

                SurfaceView surface =
(SurfaceView)findViewById(R.id.surface);
                SurfaceHolder holder = surface.getHolder();
                holder.addCallback(this);
                holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
                holder.setFixedSize(400, 300);
        }

        public void surfaceCreated(SurfaceHolder holder) {
                if (mediaRecorder != null) {
                        try {
mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
mediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);

mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);

mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
mediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.DEFAULT);

mediaRecorder.setOutputFile("/sdcard/myoutputfile.mp4");

                                // Asociando la previsualización a la
superficie
mediaRecorder.setPreviewDisplay(holder.getSurface());
                                mediaRecorder.prepare();
                        } catch (IllegalArgumentException e) {
                                Log.d("MEDIA_PLAYER", e.getMessage());
                        } catch (IllegalStateException e) {
                                Log.d("MEDIA_PLAYER", e.getMessage());
                        } catch (IOException e) {
                                Log.d("MEDIA_PLAYER", e.getMessage());
                        }
                }
        }

        public void surfaceDestroyed(SurfaceHolder holder) {
                mediaRecorder.release();
        }

        public void surfaceChanged(SurfaceHolder holder,
                                        int format, int width, int height)
{ }
}
```

2. Sintetizador de voz de Android

Android incorpora desde la versión 1.6 un motor de síntesis de voz conocido como *Text to Speech*. Mediante su API podremos hacer que nuestros programas "lean" un texto al usuario. Es necesario tener en cuenta que por motivos de espacio en disco los paquetes de lenguaje pueden no estar instalados en el dispositivo. Por lo tanto, antes de que nuestra

aplicación utilice *Text to Speech* se podría considerar una buena práctica de programación el comprobar si dichos paquetes están instalados. Para ello podemos hacer uso de un `Intent` como el que se muestra a continuación:

```
Intent intent = new Intent(TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
startActivityForResult(intent, TTS_DATA_CHECK);
```

El método `onActivityResult()` recibirá un `CHECK_VOICE_DATA_PASS` si todo está correctamente instalado. En caso contrario deberemos iniciar una nueva actividad por medio de un nuevo `Intent` implícito que haga uso de la acción `ACTION_INSTALL_TTS_DATA` del motor *Text to Speech*.

Una vez comprobemos que todo está instalado deberemos crear e inicializar una instancia de la clase `TextToSpeech`. Como no podemos utilizar dicha instancia hasta que esté inicializada (la inicialización se hace de forma asíncrona), la mejor opción es pasar como parámetro al constructor un manejador `onInitListener` de tal forma que en dicho método se especifiquen las tareas a llevar a cabo por el sintetizador de voz una vez esté inicializado.

```
boolean ttsIsInit = false;
TextToSpeech tts = null;

tts = new TextToSpeech(this, new OnInitListener() {
        public void onInit(int status) {
                if (status == TextToSpeech.SUCCESS) {
                        ttsIsInit = true;
                        // Hablar
                }
        }
});
```

Una vez que la instancia esté inicializada se puede utilizar el método `speak` para sintetizar voz por medio del dispositivo de salida por defecto. El primer parámetro será el texto a sintetizar y el segundo podrá ser o bien `QUEUE_ADD`, que añade una nueva salida de voz a la cola, o bien `QUEUE_FLUSH`, que elimina todo lo que hubiera en la cola y lo sustituye por el nuevo texto.

```
tts.speak("Hello, Android", TextToSpeech.QUEUE_ADD, null);
```

Otros métodos de interés de la clase `TextToSpeech` son:

- `setPitch` y `setSpeechRate` permiten modificar el tono de voz y la velocidad. Ambos métodos aceptan un parámetro real.
- `setLanguage` permite modificar la pronunciación. Se le debe pasar como parámetro una instancia de la clase `Locale` para indicar el país y la lengua a utilizar.
- El método `stop` se debe utilizar al terminar de hablar; este método detiene la síntesis de voz.
- El método `shutdown` permite liberar los recursos reservados por el motor de *Text to Speech*.

El siguiente código muestra un ejemplo en el que se comprueba si todo está correctamente instalado, se inicializa una nueva instancia de la clase `TextToSpeech`, y se

utiliza dicha clase para decir una frase en español. Al llamar al método `initTextToSpeech` se desencadenará todo el proceso.

```
private static int TTS_DATA_CHECK = 1;
private TextToSpeech tts = null;
private boolean ttsIsInit = false;

private void initTextToSpeech() {
        Intent intent = new Intent(Engine.ACTION_CHECK_TTS_DATA);
        startActivityForResult(intent, TTS_DATA_CHECK);
}

protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
        if (requestCode == TTS_DATA_CHECK) {
                if (resultCode == Engine.CHECK_VOICE_DATA_PASS) {
                        tts = new TextToSpeech(this, new OnInitListener()
{
                                public void onInit(int status) {
                                        if (status ==
TextToSpeech.SUCCESS) {
                                                ttsIsInit = true;
                                                Locale loc = new
Locale("es","","");
                                                if
(tts.isLanguageAvailable(loc) >=
TextToSpeech.LANG_AVAILABLE)
tts.setLanguage(loc);
                                                tts.setPitch(0.8f);
                                                tts.setSpeechRate(1.1f);
                                                speak();
                                        }
                                }
                        });
                } else {
                        Intent installVoice = new
Intent(Engine.ACTION_INSTALL_TTS_DATA);
                        startActivity(installIntent);
                }
        }
}

private void speak() {
        if (tts != null && ttsIsInit) {
                tts.speak("Hola Android", TextToSpeech.QUEUE_ADD, null);
        }
}

@Override
public void onDestroy() {
        super.onDestroy();
        if (tts != null) {
                tts.stop();
                tts.shutdown();
        }

        super.onDestroy();
}
```

3. Gráficos 3D

Para mostrar gráficos 3D en Android contamos con OpenGL ES, un subconjunto de la librería gráfica OpenGL destinado a dispositivos móviles.

Hasta ahora hemos visto que para mostrar gráficos propios podíamos usar un componente que heredase de `View`. Estos componentes funcionan bien si no necesitamos realizar repintados continuos o mostrar gráficos 3D.

Sin embargo, en el caso de tener una aplicación con una gran carga gráfica, como puede ser un videojuego o una aplicación que muestre gráficos 3D, en lugar de `View` deberemos utilizar `SurfaceView`. Esta última clase nos proporciona una superficie en la que podemos dibujar desde un hilo en segundo plano, lo cual libera al hilo principal de la aplicación de la carga gráfica.

Vamos a ver en primer lugar cómo crear subclases de `SurfaceView`, y las diferencias existentes con `View`.

Para crear una vista con `SurfaceView` tendremos que crear una nueva subclase de dicha clase (en lugar de `View`). Pero en este caso no bastará con definir el método `onDraw`, ahora deberemos crearnos un hilo independiente y proporcionarle la superficie en la que dibujar (`SurfaceHolder`). Además, en nuestra subclase de `SurfaceView` también implementaremos la interfaz `SurfaceHolder.Callback` que nos permitirá estar al tanto de cuando la superficie se crea, cambia, o se destruye.

Cuando la superficie sea creada pondremos en marcha nuestro hilo de dibujado, y lo pararemos cuando la superficie sea destruida. A continuación mostramos un ejemplo de dicha clase:

```
public class VistaSurface extends SurfaceView
             implements SurfaceHolder.Callback {
    HiloDibujo hilo = null;

    public VistaSurface(Context context) {
        super(context);

        SurfaceHolder holder = this.getHolder();
        holder.addCallback(this);
    }

    public void surfaceChanged(SurfaceHolder holder, int format,
                               int width, int height) {
        // La superficie ha cambiado (formato o dimensiones)
    }

    public void surfaceCreated(SurfaceHolder holder) {
        hilo = new HiloDibujo(holder, this);
        hilo.start();
    }

    public void surfaceDestroyed(SurfaceHolder holder) {

        hilo.detener();
        try {
            hilo.join();
        } catch (InterruptedException e) { }
    }
}
```

Como vemos, la clase `SurfaceView` simplemente se encarga de obtener la superficie y poner en marcha o parar el hilo de dibujado. En este caso la acción estará realmente en el hilo, que es donde especificaremos la forma en la que se debe dibujar el componente. Vamos a ver a continuación cómo podríamos implementar dicho hilo:

```
class HiloDibujo extends Thread {
    SurfaceHolder holder;
    VistaSurface vista;
    boolean continuar = true;

    public HiloDibujo(SurfaceHolder holder, VistaSurface vista) {
        this.holder = holder;
        this.vista = vista;
        continuar = true;
    }

    public void detener() {
        continuar = false;
    }

    @Override
    public void run() {
        while (continuar) {
            Canvas c = null;
            try {
                c = holder.lockCanvas(null);
                synchronized (holder) {
                    // Dibujar aqui los graficos
                    c.drawColor(Color.BLUE);
                }
            } finally {
                if (c != null) {
                    holder.unlockCanvasAndPost(c);
                }
            }
        }
    }
}
```

Podemos ver que en el bucle principal de nuestro hilo obtenemos el lienzo (`Canvas`) a partir de la superficie (`SurfaceHolder`) mediante el método `lockCanvas`. Esto deja el lienzo bloqueado para nuestro uso, por ese motivo es importante asegurarnos de que siempre se desbloquee. Para tal fin hemos puesto `unlockCanvasAndPost` dentro del bloque `finally`. Además debemos siempre dibujar de forma sincronizada con el objeto `SurfaceHolder`, para así evitar problemas de concurrencia en el acceso a su lienzo.

Para aplicaciones como videojuegos 2D sencillos un código como el anterior puede ser suficiente (la clase `View` sería demasiado lenta para un videojuego). Sin embargo, lo realmente interesante es utilizar `SurfaceView` junto a OpenGL, para así poder mostrar gráficos 3D, o escalados, rotaciones y otras transformaciones sobre superficies 2D de forma eficiente.

El estudio de la librería OpenGL queda fuera del ámbito de este curso. A continuación veremos un ejemplo de cómo utilizar OpenGL (concretamente OpenGL ES) vinculado a nuestra `SurfaceView`.

Realmente la implementación de nuestra clase que hereda de `SurfaceView` no cambiará, simplemente modificaremos nuestro hilo, que es quien realmente realiza el dibujado. Toda la inicialización de OpenGL deberá realizarse dentro de nuestro hilo (en el método `run`), ya que sólo se puede acceder a las operaciones de dicha librería desde el mismo hilo en el que se inicializó. En caso de que intentásemos acceder desde otro hilo obtendríamos un error indicando que no existe ningún contexto activo de OpenGL.

En este caso nuestro hilo podría contener el siguiente código:

```
public void run() {
    initEGL();
    initGL();

    Triangulo3D triangulo = new Triangulo3D();
    float angulo = 0.0f;

    while(continuar) {
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
                   GL10.GL_DEPTH_BUFFER_BIT);

        // Dibujar gráficos aquí
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        gl.glTranslatef(0, 0, -5.0f);
        gl.glRotatef(angulo, 0, 1, 0);

        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        triangulo.dibujar(gl);

        egl.eglSwapBuffers(display, surface);
        angulo += 1.0f;
    }
}
```

En primer lugar debemos inicializar la interfaz EGL, que hace de vínculo entre la plataforma nativa y la librería OpenGL:

```
EGL10 egl;
GL10 gl;
EGLDisplay display;
EGLSurface surface;
EGLContext contexto;
EGLConfig config;

private void initEGL() {
    egl = (EGL10)EGLContext.getEGL();
    display = egl.eglGetDisplay(EGL10.EGL_DEFAULT_DISPLAY);

    int [] version = new int[2];
    egl.eglInitialize(display, version);

    int [] atributos = new int[] {
        EGL10.EGL_RED_SIZE, 5,
        EGL10.EGL_GREEN_SIZE, 6,
        EGL10.EGL_BLUE_SIZE, 5,
        EGL10.EGL_DEPTH_SIZE, 16,
        EGL10.EGL_NONE
    };

    EGLConfig [] configs = new EGLConfig[1];
    int [] numConfigs = new int[1];
```

```
    egl.eglChooseConfig(display, atributos, configs,
                        1, numConfigs);

    config = configs[0];
    surface = egl.eglCreateWindowSurface(display,
                        config, holder, null);
    contexto = egl.eglCreateContext(display, config,
                        EGL10.EGL_NO_CONTEXT, null);
    egl.eglMakeCurrent(display, surface, surface, contexto);

    gl = (GL10)contexto.getGL();
}
```

A continuación debemos proceder a la inicialización de la interfaz de la librería OpenGL:

```
private void initGL() {
    int width = vista.getWidth();
    int height = vista.getHeight();
    gl.glViewport(0, 0, width, height);
    gl.glMatrixMode(GL10.GL_PROJECTION);
    gl.glLoadIdentity();

    float aspecto = (float)width/height;
    GLU.gluPerspective(gl, 45.0f, aspecto, 1.0f, 30.0f);
    gl.glClearColor(0.5f, 0.5f, 0.5f, 1);
}
```

Una vez hecho esto, ya sólo nos queda ver cómo dibujar una malla 3D. Vamos a ver como ejemplo el dibujo de un triángulo:

```
public class Triangulo3D {

    FloatBuffer buffer;

    float[] vertices = {
        -1f, -1f, 0f,
        1f, -1f, 0f,
        0f, 1f, 0f };


    public Triangulo3D() {
        ByteBuffer bufferTemporal = ByteBuffer
                    .allocateDirect(vertices.length*4);
        bufferTemporal.order(ByteOrder.nativeOrder());
        buffer = bufferTemporal.asFloatBuffer();
        buffer.put(vertices);
        buffer.position(0);
    }

    public void dibujar(GL10 gl) {
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, buffer);
        gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);
    }
}
```

Para finalizar, es importante que cuando la superficie se destruya se haga una limpieza de los recursos utilizados por OpenGL:

```
private void cleanupGL() {
    egl.eglMakeCurrent(display, EGL10.EGL_NO_SURFACE,
            EGL10.EGL_NO_SURFACE, EGL10.EGL_NO_CONTEXT);
    egl.eglDestroySurface(display, surface);
    egl.eglDestroyContext(display, contexto);
```

```
    egl.eglTerminate(display);
}
```

Podemos llamar a este método cuando el hilo se detenga (debemos asegurarnos que se haya detenido llamando a `join` previamente).

A partir de Android 1.5 se incluye la clase `GLSurfaceView`, que ya incluye la inicialización del contexto GL y nos evita tener que hacer esto manualmente. Esto simplificará bastante el uso de la librería. Vamos a ver a continuación un ejemplo de cómo trabajar con dicha clase.

En este caso ya no será necesario crear una subclase de `GLSurfaceView`, ya que la inicialización y gestión del hilo de OpenGL siempre es igual. Lo único que nos interesará cambiar es lo que se muestra en la escena. Para ello deberemos crear una subclase de `GLSurfaceViewRenderer` que nos obliga a definir los siguientes métodos:

```
public class MiRenderer implements GLSurfaceView.Renderer {

    Triangulo3D triangulo;
    float angulo;

    public MiRenderer() {
        triangulo = new Triangulo3D();
        angulo = 0;
    }

    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
    }

    public void onSurfaceChanged(GL10 gl, int w, int h) {
        // Al cambiar el tamaño cambia la proyección
        float aspecto = (float)w/h;
            gl.glViewport(0, 0, w, h);

        gl.glMatrixMode(GL10.GL_PROJECTION);
        gl.glLoadIdentity();
        GLU.gluPerspective(gl, 45.0f, aspecto, 1.0f, 30.0f);
    }

    public void onDrawFrame(GL10 gl) {
        gl.glClearColor(0.5f, 0.5f, 0.5f, 1.0f);
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
                   GL10.GL_DEPTH_BUFFER_BIT);

        // Dibujar gráficos aquí
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        gl.glTranslatef(0, 0, -5.0f);
        gl.glRotatef(angulo, 0, 1, 0);

        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        triangulo.dibujar(gl);

        angulo += 1.0f;
    }
}
```

Podemos observar que será el método `onDrawFrame` en el que deberemos escribir el código para mostrar los gráficos. Con hacer esto será suficiente, y no tendremos que

encargarnos de crear el hilo ni de inicializar ni destruir el contexto.

Para mostrar estos gráficos en la vista deberemos proporcionar nuestro *renderer* al objeto `GLSurfaceView`:

```
vista = new GLSurfaceView(this);
vista.setRenderer(new MiRenderer());
setContentView(vista);
```

Por último, será importante transmitir los eventos `onPause` y `onResume` de nuestra actividad a la vista de OpenGL, para así liberar a la aplicación de la carga gráfica cuando permanezca en segundo plano. El código completo de la actividad quedaría como se muestra a continuación:

```
public class MiActividad extends Activity {
    GLSurfaceView vista;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        vista = new GLSurfaceView(this);
        vista.setRenderer(new MiRenderer());
        setContentView(vista);
    }

    @Override
    protected void onPause() {
        super.onPause();
        vista.onPause();
    }

    @Override
    protected void onResume() {
        super.onResume();
        vista.onResume();
    }
}
```