MÓVILES.
Tema: Cubo3D con Imágenes.

INTRODUCCIÓN.
OpenGL® (Open Graphics Library), consiste en una especificación estándar multiplataforma definida en una API, la cual permite generar y configurar gráficas en 2D y 3D para dibujar escenas complejas con primitivas geométricas simples, tales como un cuadrado, triángulo o círculo. Estas formas primitivas representan las representaciones más simples que se pueden reproducir y ser diseñadas con OpenGL.

OpenGL® ES (OpenGL for Embedded Systems) es una variante de la API OpenGL, que fue diseñada para sistemas integrados, tales como dispositivos móviles o videoconsolas, y que Android se encarga de mantenerla. Debido a que está dirigida a dispositivos anidados, y que son dispositivos menos potentes que una PC, se diferencian de OpenGL debido a son más cortos los tiempos de cálculo de OpenGL® ES.

El término renderizar se refiere al proceso de generar una imagen o vídeo mediante el cálculo de la iluminación global o GI, a partir de un modelo 3D.

Clases fundamentales para implantar OpenGL en Android:

`GLSurfaceView`	Clase que proporciona una vista o `View`, para la representación y manipulación de objetos mediante el uso de la API OpenGL.
`GLSurfaceView.Renderer`	Interfaz que implanta los métodos para construir los gráficos en la vista proporcionada por la clase `GLSurfaceView`.

Métodos:

`onSurfaceCreated()`	Invocado para configurar el entorno de OpenGL ES de la vista.
`onDrawFrame()`	Invocado para cada redibujado de la vista.
`OnSurfaceChanged()`	Invocado si la geometría de la vista cambia, por ejemplo, cuando cambia la orientación de la pantalla del dispositivo.

Principales características de la API OpenGL ES
- Distribución libre. Es independiente del hardware.
- Permite dibujar figuras geométricas básicas.
- Proporciona las herramientas necesarias para realizar transformaciones sobre los vértices: traslaciones (cambiar la posición de una figura en un plano), rotaciones (cambio de orientación de un cuerpo con respecto a un punto tomado como centro) y escalados (aumento o disminución del tamaño de una figura sin deformarlo).
- Incorporada con la API de Android, sin necesidad de añadir ninguna librería o instalar cualquier otra herramienta.
- API multiplataforma y multilenguaje, con todas las funciones 2D y 3D para sistemas embebidos o integrados (consolas, electrodomésticos, móviles etc.).
- Las versiones 1.X están destinadas a hardware de función fija, ofreciendo aceleración, calidad de imagen y rendimiento.
- Las versiones 2.X permiten programar gráficos 3D completos.
- La versión OpenGL ES 1.1 es totalmente compatible con OpenGL ES 1.0.
- Aceleración de gráficos gracias a su acceso al hardware dedicado (GPU o unidad de procesamiento gráfico).

Versiones OpenGL ES y paquetes que contienen.
Enseguida se mencionan las diferentes versiones disponibles de OpenGL ES, y los paquetes que implantan cada una.
- **OpenGL ES 1.0 y 1.1**

Especificaciones de API soportada por las versiones de Android 1.0 y superiores. OpenGL ES 1.0 está basada en OpenGL 1.3 y OpenGL 1.1 está basada en OpenGL 1.5.

Paquetes:

`android.opengl`: Paquete que proporciona una interfaz estática a las clases de OpenGL ES 1.0/1.1, y mayor rendimiento que los paquetes de interfaces de javax.microedition.khronos.

- **OpenGL ES 1.0 y 1.1**

Especificaciones de API soportada por las versiones de Android 1.0 y superiores. OpenGL ES 1.0 está basada en OpenGL 1.3 y OpenGL 1.1 está basada en OpenGL 1.5.

Paquetes:

`android.opengl`: Paquete que proporciona una interfaz estática a las clases de OpenGL ES 1.0/1.1, y mayor rendimiento que los paquetes de interfaces de javax.microedition.khronos.

Clases: `GLES10`, `GLES10Ext`, `GLES11`, `GLES11Ext`.

`javax.microedition.khronos.opengles`: Paquete que proporciona la implantación estándar de OpenGL ES 1.0/1.1.

Clases: `GLES10`, `GLES10Ext`, `GLES11`, `GLES11Ext`, `GL11ExtensionPack`.

- **OpenGL ES 2.0**

Especificaciones de API soportada para las versiones de Android 2.2 y superiores (API nivel 8). OpenGL ES 2.0 está basada en OpenGL 2.0.

Paquetes:

`android.opengl.GLES2`: Paquete que proporciona la interfaz para OpenGL ES 2.0. Disponible a partir de Android 2.2 (nivel de API 8).

- **OpenGL ES 3.0**

Especificaciones de API soportada para las versiones de Android 4.3 y superiores (API nivel 18).

Paquetes:

`android.opengl.GLES30`: Paquete que proporciona la interfaz para OpenGL ES 3.0. Disponible a partir de Android 4.3 (nivel de API 18).

DESARROLLO.

1. Crear un proyecto nuevo en el IDE de Android Studio. Crear tres clases, con lenguaje Java, y actualizar sus contenidos con el código correspondiente de cada una de ellas, como se indica a continuación.

==========

Archivo: MainActivity.java

==========

```
import android.app.Activity;
import android.opengl.GLSurfaceView;
import android.os.Bundle;
public class MainActivity extends Activity {
    private GLSurfaceView glsv;
    @Override
    protected void onCreate(Bundle b) {
        super.onCreate(b);
        glsv = new GLSurfaceView(this);
        glsv.setRenderer(new Renderizador(this));
        this.setContentView(glsv);
    }
    @Override
    protected void onPause() {
        super.onPause();
        glsv.onPause();
    }
    @Override
    protected void onResume() {
        super.onResume();
        glsv.onResume();
    }
}
```

==========
Archivo: MiCubo.java
==========

```
import javax.microedition.khronos.opengles.GL10;
import android.content.Context;
import android.graphics.*;
import android.opengl.*;
import java.nio.*;
public class MiCubo {
    private FloatBuffer vb;
    private FloatBuffer tb;
    private int n = 6;
    private int[] imgs = { R.drawable.escom, R.drawable.ipn, R.drawable.f18,
            R.drawable.f15, R.drawable.mexico, R.drawable.marte };
    private int[] id = new int[n];
    private Bitmap[] bm = new Bitmap[n];
    private float tam = 1.2f;
    public MiCubo(Context cx) {
        ByteBuffer vbb = ByteBuffer.allocateDirect(12 * 4 * n);
        vbb.order(ByteOrder.nativeOrder());
        vb = vbb.asFloatBuffer();
        for (int f = 0; f < n; f++) {
            bm[f]                                                                    =
BitmapFactory.decodeStream(cx.getResources().openRawResource(imgs[f]));
            int ancho = bm[f].getWidth();
            int alto = bm[f].getHeight();
            float width = 2.0f;
            float height = 2.0f;
            if (ancho > alto) {
                height = height * alto/ancho;
            } else {
                width = width * ancho/alto;
            }
            float izq = -width / 2;
            float der = -izq;
            float tope = height / 2;
            float base = -tope;
            float[] vertices = {
                    izq, base, 0.0f,  // 0 = izq-base-frente
                    der, base, 0.0f,  // 1 = der-base-frente
                    izq, tope, 0.0f,  // 2 = izq-tope-frente
                    der, tope, 0.0f,  // 3 = der-tope-frente
            };
            vb.put(vertices);  // Poblar
        }
        vb.position(0);    // Repetir
        float[] coordenadas = {
                0.0f, 1.0f,  // A = izq-base
                1.0f, 1.0f,  // B = der-base
                0.0f, 0.0f,  // C = izq-tope
                1.0f, 0.0f   // D = der-tope
        };
        ByteBuffer bb = ByteBuffer.allocateDirect(coordenadas.length * 4 * n);
        bb.order(ByteOrder.nativeOrder());
        tb = bb.asFloatBuffer();
        for (int c = 0; c < n; c++) {
            tb.put(coordenadas);
```

```
        }
        tb.position(0);   // Rewind
    }
    public void draw(GL10 gl) {
        gl.glFrontFace(GL10.GL_CCW);
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vb);
        gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, tb);
        // frente
        gl.glPushMatrix();
        gl.glTranslatef(0f, 0f, tam);
        gl.glBindTexture(GL10.GL_TEXTURE_2D, id[0]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
        gl.glPopMatrix();
        // izquierda
        gl.glPushMatrix();
        gl.glRotatef(270.0f, 0f, 1f, 0f);
        gl.glTranslatef(0f, 0f, tam);
        gl.glBindTexture(GL10.GL_TEXTURE_2D, id[1]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 4, 4);
        gl.glPopMatrix();
        // atrás
        gl.glPushMatrix();
        gl.glRotatef(180.0f, 0f, 1f, 0f);
        gl.glTranslatef(0f, 0f, tam);
        gl.glBindTexture(GL10.GL_TEXTURE_2D, id[2]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 8, 4);
        gl.glPopMatrix();
        // derecha
        gl.glPushMatrix();
        gl.glRotatef(90.0f, 0f, 1f, 0f);
        gl.glTranslatef(0f, 0f, tam);
        gl.glBindTexture(GL10.GL_TEXTURE_2D, id[3]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 12, 4);
        gl.glPopMatrix();
        // tope
        gl.glPushMatrix();
        gl.glRotatef(270.0f, 1f, 0f, 0f);
        gl.glTranslatef(0f, 0f, tam);
        gl.glBindTexture(GL10.GL_TEXTURE_2D, id[4]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 16, 4);
        gl.glPopMatrix();
        // base
        gl.glPushMatrix();
        gl.glRotatef(90.0f, 1f, 0f, 0f);
        gl.glTranslatef(0f, 0f, tam);
        gl.glBindTexture(GL10.GL_TEXTURE_2D, id[5]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 20, 4);
        gl.glPopMatrix();
        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
    }
    public void loadTexture(GL10 gl) {
        gl.glGenTextures(6, id, 0); // Generar arreglo de IDs de texture-ID de 6 IDs
        for (int face = 0; face < n; face++) {
            gl.glBindTexture(GL10.GL_TEXTURE_2D, id[face]);
            gl.glTexParameterf(GL10.GL_TEXTURE_2D,            GL10.GL_TEXTURE_MIN_FILTER,
GL10.GL_NEAREST);
```

```
            gl.glTexParameterf(GL10.GL_TEXTURE_2D,            GL10.GL_TEXTURE_MAG_FILTER,
GL10.GL_LINEAR);
            GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bm[face], 0);
            bm[face].recycle();
        }
    }
}
```

==========
Archivo: Renderizador.java
==========

```
import javax.microedition.khronos.egl.*;
import javax.microedition.khronos.opengles.*;
import android.content.*;
import android.opengl.GLSurfaceView;
import android.opengl.GLU;
class Renderizador implements GLSurfaceView.Renderer {
    private MiCubo mc;
    private static float angulo = 0, velocidad = 0.5f;
    public Renderizador(Context cx) {
        mc = new MiCubo(cx);
    }
    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig eglc) {
        gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
        gl.glClearDepthf(1.0f);
        gl.glEnable(GL10.GL_DEPTH_TEST);
        gl.glDepthFunc(GL10.GL_LEQUAL);
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST);
        gl.glShadeModel(GL10.GL_SMOOTH);
        gl.glDisable(GL10.GL_DITHER);
        mc.loadTexture(gl);
        gl.glEnable(GL10.GL_TEXTURE_2D);
    }
    @Override
    public void onSurfaceChanged(GL10 gl, int w, int h) {
        if (h == 0) h = 1;
        float spct = (float) w/h;
        gl.glViewport(0, 0, w, h);
        gl.glMatrixMode(GL10.GL_PROJECTION);
        gl.glLoadIdentity();
        GLU.gluPerspective(gl, 45, spct, 0.1f, 100.f);
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
    }
    @Override
    public void onDrawFrame(GL10 gl) {
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
        gl.glLoadIdentity();
        gl.glTranslatef(0.0f, 0.0f, -6.0f);
        gl.glRotatef(angulo, 0.15f, 1.0f, 0.3f);
        mc.draw(gl);
        angulo += velocidad;
    }
}
```

2. Agregar en la carpeta res/drawable del repositorio las imágenes correspondientes a cada cara. Se pueden incluir las imágenes de su preferencia con el nombre en letra minúscula. Las seis imágenes PNG se incluyen en una carpeta anexa: escom, ipn, f18, f15, mexico y marte.

3. Ejecutar la aplicación para mostrarla como se indica en la figura 1, que contiene el cubo en 3D con una imagen en cada cara, incluyendo su giro.

Figura 1. Cubo en 3D con imágenes en sus 6 caras.

REFERENCIAS.

https://mastermoviles.gitbook.io/graficos-y-multimedia/graficos-de-alto-rendimiento-en-android
https://www.opengl.org/
https://www.khronos.org/opengles/
https://developer.android.com/reference/android/opengl/GLSurfaceView.html
https://developer.android.com/reference/android/opengl/GLSurfaceView.Renderer.html

EJERCICIOS.

1. Unir las aristas que separan las imágenes de las caras del cubo.
2. Detener el giro del cubo y girarlo manualmente.
3. Mostrar un mensaje `Toast` que indique cual cara se ha seleccionado al momento de digitar sobre ella.

NOTA. Generar un reporte con las imágenes obtenidas en un documento con la sintaxis AlumnoCubo3DimagenesGrupo.doc y enviarlo al sitio indicado por el profesor.