# Multimedia en Android

Android nos proporciona multitud de herramientas para trabajar elementos de **multimedia**, desde los más reproductores de audio a efectos de animaciones y gráficos 2D y 3D. Teóricamente, estos conocimientos de multimedia podrían servirte para el desarrollo de videojuegos; pero ni que decir tiene que nosotros vamos a ver solo la punta del iceberg. Para el desarrollo de videojuegos se usan herramientas más especializadas como Unity, y requiere de un periodo de aprendizaje de años.

Nosotros vamos a ver ejemplos razonablemente sencillos que poder usar en nuestras Apps y lograr efectos sencillos y pequeñas animaciones para logos, iconos que giran, etc.

En general, podemos hacer un índice que abarque todo lo que consideraríamos multimedia de la siguiente forma:

1) Gráficos 2D en Android
    a. Canvas y Paint
    b. Vector Drawables
    c. Frame Animations

2) Animaciones en Android
    a. View Animations (En desuso)
    b. Property Animations (Moderno)
    c. Motion Layout (Recomendado)

3) Gráficos 3D en Android
    a. OpenGL ES
    b. Sceneform / Filament (para modelos 3D actuales)
    c. Renderización de imágenes y vídeo

4) Sonido
    a. SoundPool (Para efectos rápidos).
    b. MediaPlayer (Para audios largos, música, etc).
    c. Control del volumen y recursos.

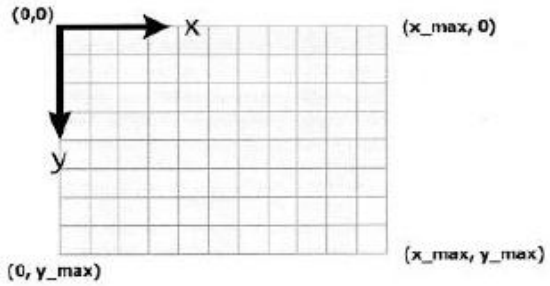
**Gráficos 2D - El Canvas en Android**

Utilizado cuando queremos generar gráficos simples en 2D de forma dinámica (desde código), podemos recurrir a los **Canvas**. Podemos entender un **canvas** como un "lienzo" en el que podemos ir escribiendo diferentes formas geométricas. El canvas tiene la particularidad de que se repinta regularmente, de forma que también podríamos aprovechar ese repintado para hacer una animación sencilla.

El objeto **Canvas** tiene asociado un **bitmap**, que es lo que *se pinta* realmente en la pantalla. Para mostrar un Canvas necesitamos:

- Las coordenadas (X, Y) donde se va a *pintar* el objeto.
- Qué dibujar: **Rect** (rectángulo), **Text** (texto), **Bitmap** (gráfico)…
- Un objeto **Paint** para describir los colores y estilo para el dibujo.

El sistema de **coordenadas** de Android está pensado para ser independientemente del tamaño de pantalla que se esté utilizando.




Por tanto, se hace necesario conocer los valores de x_max e y_max, que dependen del tamaño de la pantalla del dispositivo. Para hacerlo, nos basamos en el objeto display.

```
Display display = getWindowManager().getDefaultDisplay();
Point point = new Point();
display.getSize(point);
int maxX = point.x;
int maxY = point.y;
```

Canvas dispone de muchos métodos para **dibujar** elementos. Por ejemplo:

- drawCircle (): Dibuja un círculo de determinado radio.
- drawLine (): Dibuja una línea.
- drawText (): Dibuja un texto.

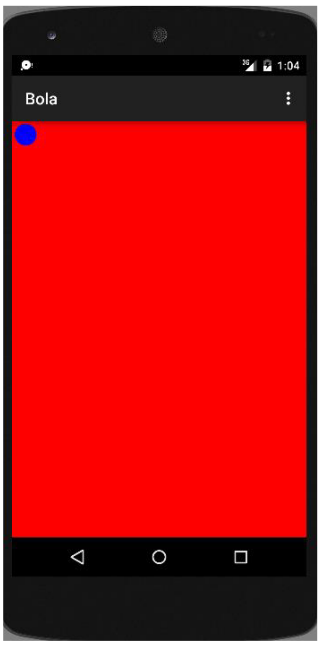
Para dar colores y estilos utilizamos un **Paint**. Algunos de los métodos disponibles son:

- setColor(Color.BLACK): Para asignar el negro por defecto.
- setColor(Color.argb(alfa, rojo, verde, azul): Tu propio color.
- Usar el recurso colors.xml para crear tu propio color

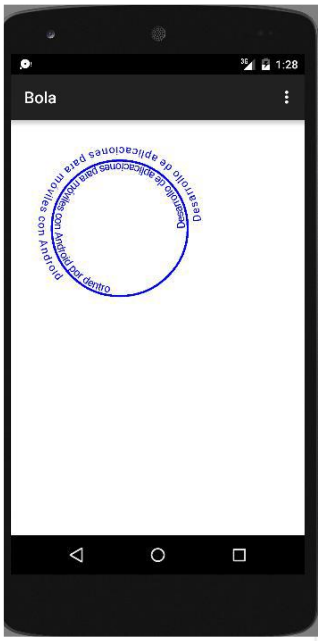
  <color name="Azul">#500000FF</color>

Un ejemplo de Canvas [JAVA]: Creamos un objeto View de forma dinámica, donde dibujamos una esfera azul sobre un fondo rojo. En el MainActivity, no incluimos un Layout. Simplemente, generaremos un nueva **view** y se la incluimos al

```java
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate( savedInstanceState );

        // En lugar de asignarle un Layout...
        // setContentView( R.layout.activity_main );

        // ... nos lo creamos dinámicamente...
        BolaView bolaView = new BolaView( context: this);
        setContentView (bolaView);
    }
}

public class BolaView extends View {

    public BolaView( Context context ) {
        super( context );
    }

    protected void onDraw (Canvas canvas ) {
        super.onDraw( canvas );

        Paint myPaint = new Paint();
        canvas.drawColor( Color.RED );
        myPaint.setColor( Color.BLUE );
        canvas.drawCircle( cx: 50, cy: 50, radius: 40, myPaint );
    }
}
```



Por supuesto, el nivel de sofisticación que podemos alcanzar con Canvas depende únicamente de nuestra imaginación y habilidad.




```
protected void onDraw (Canvas canvas ) {
    super.onDraw( canvas );

    Path trazo = new Path();
    trazo.addCircle ( x: 400 , y: 400 , radius: 250 ,Path.Direction.CCW);
    canvas.drawColor (Color.WHITE);
    Paint pincel = new Paint ();
    pincel.setColor (Color.BLUE);
    pincel.setStrokeWidth(8);
    pincel.setStyle (Paint.Style.STROKE);
    canvas.drawPath (trazo, pincel);

    pincel.setStrokeWidth(1);
    pincel.setStyle (Paint.Style.FILL);
    pincel.setTextSize (40);
    pincel.setTypeface (Typeface.SANS_SERIF);
    canvas.drawTextOnPath ( text: "Desarrollo de aplicaciones para móviles con Android ",
            trazo,  hOffset: 20 ,  vOffset: 50 , pincel);
    canvas.drawTextOnPath ( text: "Desarrollo de aplicaciones para móviles con Android por dentro",
            trazo,  hOffset: 0,  vOffset: -10 , pincel);
}
```

**Gráficos 2D – Drawable**

**Drawable** significa cualquier cosa que se puede dibujar. Usaremos un Drawable cuando queramos usar una imagen guardada en un fichero de nuestra App. Android es capaz de manejar ficheros del tipo .png, .jpg y .gif, los cuales deberán estar cargados en la carpeta de recursos **drawable**. Estos elementos pueden ser referenciados en recursos/drawable normalmente mediante **R.drawable.nombre_fichero**, o si se desean usar en un Layout mediante **@drawable/nombre_fichero**.

Para usar un Drawable, podríamos utilizar un **bitmap** y cargarlo en un canvas mediante la siguiente instrucción:

```
bitmap = BitmapFactory.decodeResource (getResources(), R.drawable.imagen);
canvas.drawBitmap (bitmap, 500, 500, null);
```

Los diferentes tipos de Drawables son:

- BitmapDrawable: Son imágenes png o jpg que están en la carpeta drawable.
- GradientDrawable: Son degradados de color que se suelen usar de fondo.
- TransitionDrawable: Utilizados para pasar de una imagen a otra.
- ShapeDrawable: Permiten dibujar gráficos a partir de formas básicas.
- AnimationDrawable: Animación frame a frame a partir de una serie de otros objetos de tipo Drawable.

Un ejemplo de **GradientDrawable** podría ser el que observas en la imagen para usar de fondo en nuestras Apps. Para generar uno, creamos el fichero **degradado.xml** en Drawable, con el siguiente contenido:

```
<shape xmlns:android =="http://schemas.android.com/apk/res/android">
<gradient
    android:startColor="#FFFFF"
    android:endColor="#000000"
    android:angle="270"/>
</shape>
```

Para mostrar el degradado podríamos indicarlo:

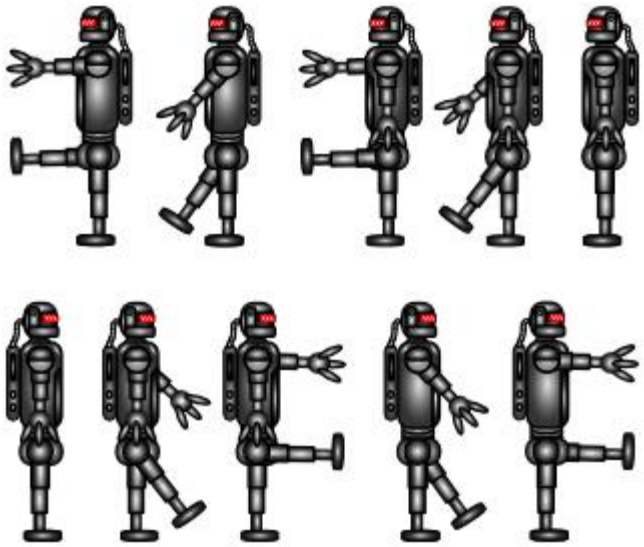
- En el onCreate: setBackgroundResource(R.drawable.degradado)
- En el layout: android:background="@drawable/degrada"

Sprites

Los **Sprites** son imágenes con fondo transparente que se utilizan para superponerlos a un fondo o escena. Para poder crear un Sprite podría ser necesario usar un programa de edición de imágenes tipo Gimp. En la siguiente dirección podríamos descargarnos muchos bitmaps y Sprites libres para nuestras App:

http://opengameart.org

Un uso simple de Sprites podría ser una **animación** a la antigua usanza. Partiendo de una plantilla de imágenes similar a ésta, podríamos ir mostrando una tras otra en secuencia para dar la sensación de un robot en movimiento; o podríamos simplemente hacer una transición entre dos imágenes (similar a un fundido).



Un **TransitionDrawable** permite hacer esto último. Generamos un fichero en **res/drawable/transicion.xml** con la configuración de los Sprites:

```
<?xml version="1.0" encoding="utf-8"?>
<transition xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable = "@drawable/robot1"/>
    <item android:drawable = "@drawable/robot2"/>
</transition>
```

En este caso, se va a realizar un fundido del Sprite **robot1** al Sprite **robot2**. Nótese que estos Sprites son imágenes que se cargan desde **res/drawable**.

Ahora, sólo hay que cargar la transición:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate( savedInstanceState );
    setContentView( R.layout.activity_main );

    ImageView imageView = new ImageView( context: this);
    setContentView (imageView);

    TransitionDrawable transitionDrawable = (TransitionDrawable) getDrawable (R.drawable.transition);
    imageView.setImageDrawable(transitionDrawable);
    transitionDrawable.startTransition( durationMillis: 2000);
}
```

Finalmente, el **AnimationDrawable** Es una secuencia de imágenes (frames) que se reproducen una detrás de otra, como un gif**.** Para crear uno, defines los Sprites en un XML (en res/drawable) mediante la etiqueta **<animation-list>**. Cada uno de los elementos de la lista será una de las imágenes que se irán intercalando en la animación, que permanecerán visibles tantos milisegundos como indiquemos en la propiedad **android:duration**. Añadiendo el atributo **android:oneshot="false"** podemos hacer que la animación se repita continuamente.

Por ejemplo: para animar un robot en un ImageView, creamos el fichero **Animación_robot.xml** de la siguiente forma:

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <item android:drawable="@drawable/robot1" android:duration="200" />
    <item android:drawable="@drawable/robot2" android:duration="200" />
    <item android:drawable="@drawable/robot3" android:duration="200" />
    <item android:drawable="@drawable/robot4" android:duration="200" />
    <item android:drawable="@drawable/robot5" android:duration="200" />
    <item android:drawable="@drawable/robot6" android:duration="200" />
    <item android:drawable="@drawable/robot7" android:duration="200" />
    <item android:drawable="@drawable/robot8" android:duration="200" />
    <item android:drawable="@drawable/robot9" android:duration="200" />
    <item android:drawable="@drawable/robot10" android:duration="200" />
</animation-list>
```

En el Activity:

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    super.onCreate (savedInstanceState);
    setContentView R.layout.activity_main;

    ImageView imageView = (ImageView ) findViewById (R.id.imgRobot);
    imageView.setBackgroundResource(R.drawable.animationDrawable);
    AnimationDrawable animationDrawable = (AnimationDrawable) imageView.getBackground();
}
```

**Animaciones en Android - Property Animations**

Android incorpora un potente sistema de animaciones que permiten realizar cualquier tipo de animaciones automáticas sencillas sobre Vistas, Superficies u otros objetos. Para ello emplea la clase **ObjectAnimator**. La clase **AnimationSet** permite para gestionar conjuntos de animaciones, y gestionarlas de forma simultánea o en secuencia.

Tienes un ejemplo en: **UD10 - Property Animations**

En el ejemplo, animamos un simple cuadrado. Pero recuerda que no tiene por qué ser un cuadrado: puede ser cualquier una imagen, un botón, etc.

Puedes animar propiedades como:

- Transparencia
- Rotación
- Rotación eje X y Rotación eje Y, para crear efectos 3D falsos
- Translación eje X y Traslación eje Y, para desplazar el elemento
- Escala X y Escala Y (zoom)
- Filtro de color
- Elevación (sombra)
- BackgroundColor

Un ejemplo más elaborado requiere el uso de **interpoladores**. Una **interpolación** define cómo cambia la animación con el tiempo. Por defecto, las animaciones usan LinearInterpolator que les da una velocidad constante; pero nada te impide hacer que la animación acelere, desacelere, rebote, etc**.** Android tiene interpoladores predefinidos para ello.

Tienes un ejemplo de esto en: **UD10 - Property Animations Interpolators**

**Animaciones en Android - MotionLayout**

El **MotionLayout**, que es la herramienta de animaciones más potente de Android a día de hoy. Permite realizar animaciones complejas y transiciones entre estados de layout, soportando animaciones de todo tipo, interpolaciones, eventos de ratón y swipe, y animaciones basadas en gestos y scroll.

Piensa en MotionLayout como un "ConstraintLayout que sabe animar solo".

Primero se define el **MotionLayout** que queremos animar. En muchas ocasiones, este MotionLayout substituye al clásico ConstraintLayout que asignamos a una Actividad. A continuación, generamos el fichero **res/xml/scene.xml** donde configuramos las transiciones que van a sucederse en el **MotionLayout**. Por ejemplo, podemos definir un estado inicial para una imagen, un estado final, y un desencadenante (evento). Al cargarse el MotionLayout, la imagen se encontrará colocada donde se indique en su estado inicia. Cuando suceda el evento, la imagen terminará donde lo indique el estado final. De esta forma, conseguiremos la animación de una imagen desplazándose por la pantalla.

Tienes un ejemplo en: **UD10 - Motion Layout Example**

## Gráficos 3D – OpenGL ES

**OpenGL** significa Open Graphics Library. Es un estándar multiplataforma para generar gráficos 2D y 3D. Permite a los programas dibujar formas, imágenes, texturas y animaciones usando la GPU. OpenGL (y su versión para móviles, **OpenGL ES**) es muy usado en videojuegos, tanto en móviles como en otras plataformas.

Usando OpenGL podemos dibujar:

- Triángulos, cuadrados, polígonos…
- Modelos 3D complejos
- Animaciones y rotaciones
- Texturas, luces y sombreado

OpenGL **no tiene layouts**. Solo dibuja en una superficie (**GLSurfaceView**) que se genera dinámicamente sobre la que vamos añadiendo propiedades y elementos.

Debido a que es **muy complejo** aprender OpenGL ES, vamos a mostrar unos pocos ejemplos funcionales simples para ver su potencial. Por ejemplo, es relativamente sencillo mostrar **un triángulo 2D** por pantalla con **OpenGL ES**. Esto es el equivalente al 'Hola, ¡Mundo!' del OpenGL (sí, ya sé, en 2D).

Tienes un ejemplo en: **UD10 - OpenGLBasic**

El resultado final es un sencillo triángulo rojo centrado en un fondo negro, sin animaciones de ningún tipo. Pero date cuenta de que se ha generado mediante **gráficos 2D/3D reales**, no mediante una interfaz de usuario.

Pero el verdadero potencial de OpenGL ES se consigue con la animación. Por ejemplo, podemos mostrar un triángulo 2D rotando en animación 3D. El efecto se consigue haciendo que el triángulo azul rote sobre su eje X dando un efecto visual de 3D.

Tienes un ejemplo en: **UD10 - OpenGL3DRotation**

Finalmente, es posible añadir eventos que afecten al triángulo. Por ejemplo, podemos controlar la rotación del triángulo mediante el dedo tocando la pantalla. Al arrastrar el dedoo conseguimos que lo haga en uno u otro eje.

Tienes un ejemplo en: **UD10 - OpenGL3DRotationFinger**

**Sonido – SoundPool**

SoundPool es una **API de Android** para reproducir sonidos cortos. Nos estamos refiriendo a efectos, notificaciones, clicks, disparos, etc. Está optimizada para baja latencia, a diferencia de MediaPlayer, que se usa para música o streaming.

Permite:

- Cargar varios sonidos en memoria
- Reproducirlos simultáneamente
- Controlar **volumen, pitch y looping**

Tienes por internet muchas páginas con efectos de sonido, como por ejemplo:

https://pixabay.com/es/sound-effects/search/clicks/

Tienes un ejemplo en: **UD10 - SoundPoolBasic**