



Android y Java para Dispositivos Móviles

Sesión 11: Gráficos avanzados



Puntos a tratar

- Gráficos en Android
- Lienzo y pincel
- Primitivas geométricas
- Texto
- Imágenes
- Elementos *drawables*
- Animaciones
- Gráficos 3D y OpenGL



Gráficos en Android

- Dos formas de mostrar gráficos
 - Bajo nivel
 - Definir componente propio
 - Subclase de `View`
 - Especificamos cómo *pintar* el componente
 - Sobrescribiendo el método `onDraw`
 - Alto nivel
 - Definimos elementos *drawables*
 - En XML o programados
 - Los mostramos en componentes de alto nivel
 - Por ejemplo en `ImageView`

Gráficos a bajo nivel

- Método `onDraw(Canvas canvas)`
 - Recibe parámetro `Canvas` (lienzo)
- Pintamos en el lienzo (`Canvas`)
 - Área de dibujo de nuestro componente

```
public class MiVista extends View {  
    public MiVista(Context context) {  
        super(context);  
    }  
  
    @Override  
    protected void onDraw(Canvas canvas) {  
        // TODO Definir como dibujar el componente  
    }  
}
```

Lienzo y pincel

- Pintamos en el lienzo (`Canvas`)
 - Tiene un tamaño (área que ocupa de la pantalla)
 - Define área de recorte, transformaciones, etc
- Usamos un pincel (`Paint`) para pintar en él
- Define la forma en la que se dibuja
 - Color del pincel
 - Tipo de trazo
 - Otros efectos

```
Paint p = new Paint();  
p.setColor(Color.RED);
```

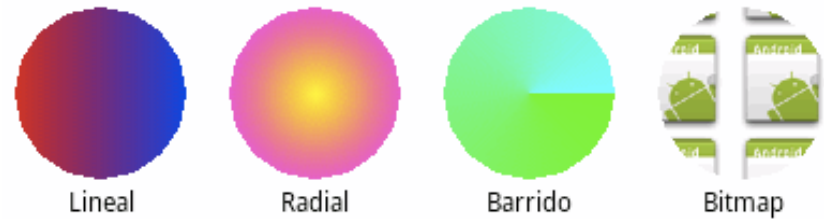


Atributos del pincel

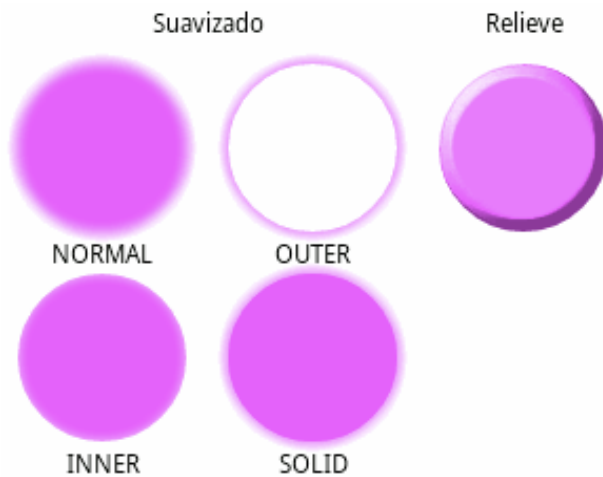
Estilo del pincel



Color sólido o gradiente



Máscaras



Tipo de trazo



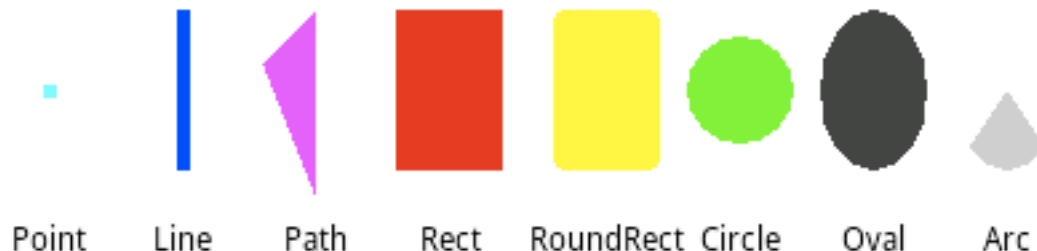
Dithering



Primitivas geométricas

- Se dibujan con métodos de Canvas

```
Paint paint = new Paint();  
paint.setStyle(Style.FILL);  
paint.setStrokeWidth(5);  
paint.setColor(Color.BLUE);  
  
canvas.drawPoint(100, 100, paint);  
canvas.drawLine(10, 300, 200, 350, paint);  
canvas.drawRect(new RectF(180, 20, 220, 80), paint);
```



Texto

- Establecer atributos del texto en el pincel

Normal	<u>Subrayado</u>
Normal lineal	<i>Inclinado</i>
Negrita falsa	Antialiasing
Tachado	Antialiasing subpixel

- Dibujar texto en el lienzo (`drawText`)
- Métricas
 - Mide texto en pixeles
 - Separación recomendada entre líneas
 - Anchura de una cadena



Imágenes

- Clase `Bitmap`
 - Se dibujan en el lienzo con `drawBitmap`
 - Liberar memoria con `recycle`
- Inmutables
 - No se puede modificar su contenido
 - Se crean a partir de un fichero o *array* de pixels
 - `BitmapFactory` para leer GIF, JPEG o PNG
- Mutables
 - Podemos modificar su contenido en el código
 - Se crean vacías, proporcionando ancho y alto



Elementos *drawables*

- Se pueden mostrar en componentes
- Se definen en XML o de forma programática
- Tipos básicos
 - Formas geométricas
 - Gradientes
 - Imágenes
 - *Nine-patch*
 - Animaciones
- Tipos combinados
 - Capas
 - Estados
 - Niveles
 - Transiciones
 - Inserción
 - Recorte
 - Escala

Drawables en XML

- Definimos `drawable/rectangulo.xml`

```
<shape xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <solid android:color="#f00"/>
    <stroke android:width="2dp" android:color="#00f"
        android:dashWidth="10dp" android:dashGap="5dp"/>
</shape>
```

- Lo utilizamos en un componente `ImageView`

```
ImageView visor = (ImageView)findViewById(R.id.visor);
visor.setImageResource(R.drawable.rectangulo);
```

Drawables en código Java

- Definir el objeto Drawable

```
RectShape r = new RectShape();  
ShapeDrawable sd = new ShapeDrawable(r);  
sd.getPaint().setColor(Color.RED);  
sd.setIntrinsicWidth(100);  
sd.setIntrinsicHeight(50);
```

- Mostrar en un componente

```
ImageView visor = (ImageView)findViewById(R.id.visor);  
visor.setImageDrawable(sd);
```



Animaciones

- Bajo nivel
 - Cambiar propiedades de objetos de la vista
 - Llama a `invalidate` para que se repinte
 - Si usamos un hilo, llamar a `postInvalidate`
- Alto nivel
 - Animación por fotogramas (*frames*)
 - Animación por interpolación (*tweened*)
 - Se definen en XML o de forma programática



Animaciones por fotogramas en XML

- Se trata de un *drawable* animado
- Lista de fotogramas (*drawables*) y duración

```
<animation-list
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <item android:drawable="@drawable/spr0"
        android:duration="50" />
    <item android:drawable="@drawable/spr1"
        android:duration="50" />
    <item android:drawable="@drawable/spr2"
        android:duration="50" />
</animation-list>
```



Animaciones por fotogramas en Java

```
BitmapDrawable f1 = (BitmapDrawable)getResources()  
    .getDrawable(R.drawable.sprite0);  
BitmapDrawable f2 = (BitmapDrawable)getResources()  
    .getDrawable(R.drawable.sprite1);  
BitmapDrawable f3 = (BitmapDrawable)getResources()  
    .getDrawable(R.drawable.sprite2);  
AnimationDrawable animFotogramas =  
    new AnimationDrawable();  
  
animFotogramas.addFrame(f1, 50);  
animFotogramas.addFrame(f2, 50);  
animFotogramas.addFrame(f3, 50);  
  
animFotogramas.setOneShot(false);
```

Se repite
indefinidamente

Mostrar animación por fotogramas

- Si está en XML, recuperarla en Java

```
AnimationDrawable animFotogramas =  
    getResources().getDrawable(R.drawable.animacion);
```

- Vincularla a un componente


```
ImageView visor = (ImageView)findViewById(R.id.visor);  
visor.setBackgroundDrawable(animFotogramas)
```

- Ejecutar `animFotogramas.start();`

No se puede hacer
en onCreate

- Detener `animFotogramas.stop();`

Animación por interpolación

- Anima una vista entera
 - Traslaciones
 - Rotaciones
 - Escalados Combinaciones (`set`)
- Escala de tiempo (*interpolator*)
 - Lineal
 - Aceleración/deceleración
 - Sinusoidal, ida y vuelta
 - Personalizados



Animación por interpolación en XML

- Definimos `anim/rotacion.xml`

```
<set xmlns:android=  
    "http://schemas.android.com/apk/res/android"  
    android:shareInterpolator="false">  
    <rotate  
        android:fromDegrees="0"  
        android:toDegrees="360"  
        android:pivotX="50%"  
        android:pivotY="50%"  
        android:duration="5000" />  
</set>
```

Animación por interpolación en Java

- Definimos la animación en Java

```
RotateAnimation rotacion = new RotateAnimation(0, 360,  
    RotateAnimation.RELATIVE_TO_SELF, 0.5f,  
    RotateAnimation.RELATIVE_TO_SELF, 0.5f);  
rotacion.setDuration(5000);
```

- O la cargamos del XML

```
Animation rotacion = AnimationUtils  
    .loadAnimation(this, R.anim.rotacion);
```

- La reproducimos en una vista

```
vista.startAnimation(rotacion);
```



Gráficos 3D

- `View` es útil para mostrar gráficos sencillos
- Es poco eficiente para
 - Gráficos 3D
 - Tasas elevadas de actualización
- Para aplicaciones con alta carga gráfica
 - Utilizaremos `SurfaceView`
 - Se dibuja en hilo independiente
 - No bloquea hilo principal de eventos
 - OpenGL para gráficos 3D
 - A partir de 1.5, tenemos `GLSurfaceView`



SurfaceView

```
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) {
        // Detener hilo
    }
}
```

Heredamos de
SurfaceView e
implementamos
SurfaceHolder
.Callback

Obtenermos el *holder* de
la superficie y registramos
el *callback*

Al crearse la
superficie
ejecutamos el
hilo de dibujo

Al destruirse
lo paramos



Hilo de dibujo

```
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);  
            }  
        }  
    }  
}
```

Obtenemos el lienzo a partir del *holder*, y lo bloqueamos

Debemos dibujar de forma sincronizada con el *holder*

Desbloqueamos el lienzo y mostramos en pantalla lo dibujado



GLSurfaceView

- Se encarga de:
 - Inicialización y destrucción del contexto OpenGL
 - Gestión del hilo de *render*
- No hace falta sobrescribir la clase
- Debemos definir un objeto `Renderer`

```
public class MiRenderer implements GLSurfaceView.Renderer {  
  
    public void onSurfaceCreated(GL10 gl,  
                                EGLConfig config) { ... }  
    public void onSurfaceChanged(GL10 gl, int w,  
                                int h) { ... }  
    public void onDrawFrame(GL10 gl) { ... }  
}
```



Creación de la vista

```
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();  
    }  
}
```

Proporcionamos
nuestro *renderer*

Comunicamos a la vista
de OpenGL los eventos
de pausa y reanudación



¿Preguntas...?