Android Graphics

Sang Shin
Michèle Garoche
www.javapassion.com
"Learn with Passion!"

Disclaimer

- Portions of this presentation are modifications based on work created and shared by the Android Open Source Project
  - http://code.google.com/policies.html
- They are used according to terms described in the Creative Commons 2.5 Attribution License
  - http://creativecommons.org/licenses/by/2.5/

Topics

- Android Graphics
- Drawing 2D Graphics

Android Graphics

Android 2D & 3D Graphics Support

- Android graphics are powered by
  - A custom 2D graphics library
  - OpenGL ES 1.0 for high performance 3D graphics.
- The most common 2D graphics APIs can be found in the *drawable* package.
- OpenGL APIs are available from the Khronos OpenGL ES package, plus some Android OpenGL utilities
  - (We will cover 3D in another presentation)

Drawing 2D Graphics

Two Options of Drawing 2D Graphics

- Option #1 - Drawing into a View (or ImageView)
  - Draw to the background of a View or to the content of an ImageView
  - The drawing (and any animation) of your graphics is handled by the system's normal View hierarchy drawing process — you simply define the graphics to go inside the View.
- Option #2 – Draw your graphics directly to a Canvas yourself
  - You personally call the appropriate class's *draw()* method (passing it your Canvas), or one of the Canvas *draw...()* methods (like *drawPicture()*)

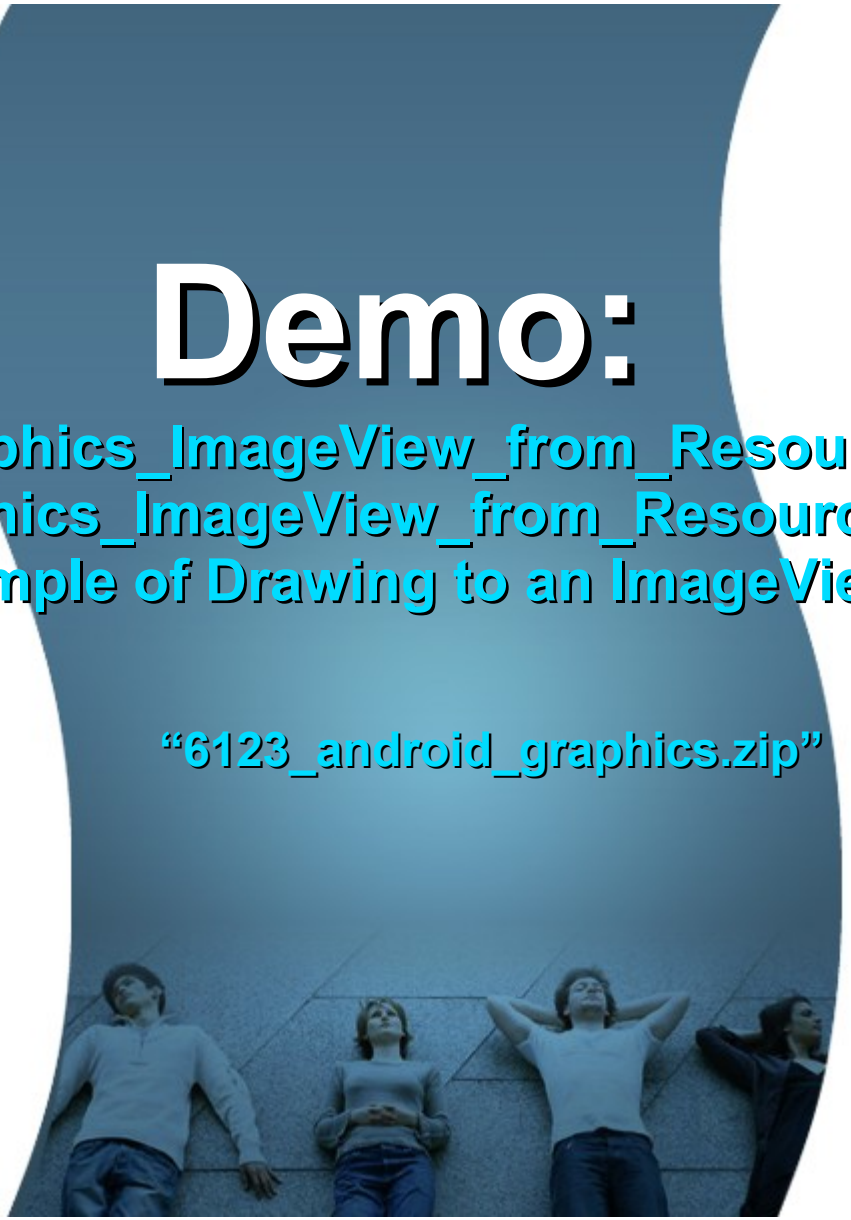
Option #1 - Drawing to a View (or ImageView)

- Best choice when you want to draw simple graphics (images, shapes, colors, pre-defined animations, etc.) that do not need to change dynamically and are not part of a performance-intensive game.
  - For example, you should draw your graphics into a View or ImageView when you want to display a static graphic

Demo:

Run “graphics_ImageView_from_Resource1” &
“graphics_ImageView_from_Resource2”
Example of Drawing to an ImageView

“6123_android_graphics.zip”



Option #2 – Draw with a Canvas

- Use it when your application needs to regularly re-draw itself.
  - Basically, any video game should be drawing to the Canvas on its own.
- Two ways of performing Option #2
  - Scheme #a: Create a custom View
    - In the same thread as your UI Activity, wherein you create a custom View component in your layout, call invalidate() and handle the onDraw() callback..
  - Scheme #b: Extend SurfaceView
    - In a separate thread, wherein you manage a SurfaceView and perform draws to the Canvas as fast as your thread is capable (you do not need to request invalidate()).

Canvas Class

- The Canvas class holds the "draw" calls.
- To draw something, you need 4 basic components:
  - A Bitmap to hold the pixels
  - A Canvas to host the draw calls (writing into the bitmap)
  - A drawing primitive (e.g. Rect, Path, text, Bitmap)
  - A Paint (to describe the colors and styles for the drawing).

Scheme #a: Create a Custom View

- Override *onDraw(Canvas canvas)* callback method
  - Android framework will provide you with a pre-defined Canvas to which you will place your drawing calls
  - Use the Canvas given to you for all your drawing, using various Canvas.draw...() methods, or other class draw() methods that take your Canvas as an argument.
  - Example of calling Canvas.draw...() method
    - *Canvas.drawBitmap (Bitmap bitmap, float left, float top, Paint paint)*
  - Example of calling draw() method of a Drawable class
    - *mDrawable.draw(canvas);*

Demo:

Run “graphics_CustomView_AlphaBitmap1”

Example of drawing to a Canvas
Example of calling Canvas.drawBitmap() method

“6123_android_graphics.zip”

Demo:

Run “graphics_CustomView_ShapeDrawable”

Example of drawing to a Canvas
Example of calling draw() method of a Drawable class

“6123_android_graphics.zip”

Scheme #a: onDraw() & invalidate()

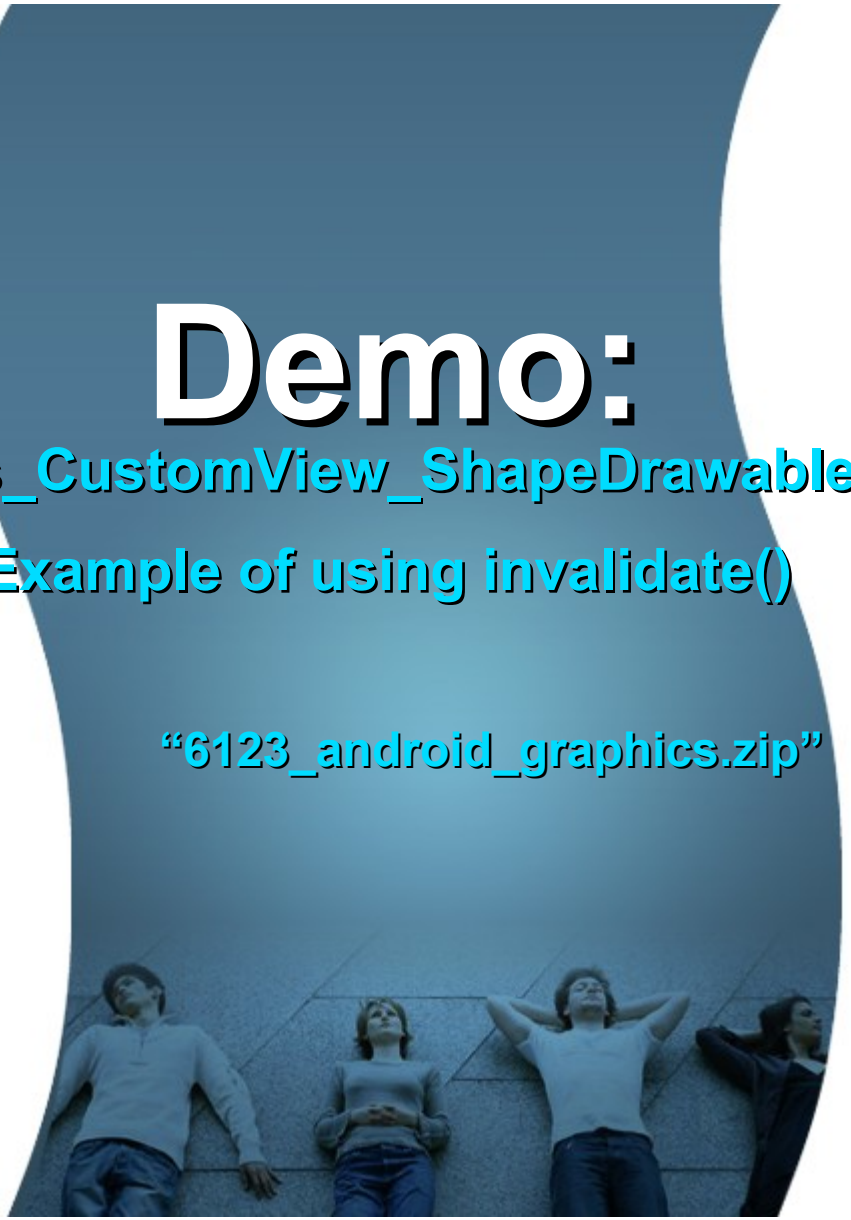
- The Android framework will only call onDraw() as necessary.
- Each time that your application is prepared to be drawn, you must request your View be invalidated by calling invalidate().
- This indicates that you'd like your View to be drawn and Android will then call your onDraw() method (though is not guaranteed that the callback will be instantaneous).

Demo:

Run “graphics_CustomView_ShapeDrawable_invalidate”

Example of using invalidate()

“6123_android_graphics.zip”



Scheme #b: Extend SurfaceView

- The SurfaceView is a special subclass of View that offers a dedicated drawing surface within the View hierarchy.
- The aim is to offer this drawing surface to an application's secondary thread, so that the application isn't required to wait until the system's View hierarchy is ready to draw. Instead, a secondary thread that has reference to a SurfaceView can draw to its own Canvas at its own pace.

Creating a SurfaceView

- Create a new class that extends SurfaceView.
  - The class should also implement *SurfaceHolder.Callback.*
  - This subclass is an interface that will notify you with information about the underlying Surface, such as when it is created, changed, or destroyed

Demo:

Run “graphics_SurfaceView_CameraPreview”

Example of using SurfaceView

“6123_android_graphics.zip”

Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps
"Learn with Passion!"