Filthy Rich Android Clients

Romain Guy
UI Toolkit Engineer, Android
Google
Overall Presentation Goal

Learn how to apply Filthy Rich Clients techniques to the Android platform.
Speaker’s qualifications

- Romain works on the Android UI toolkit at Google
- Romain co-authored the book Filthy Rich Clients
- Romain enjoy writing Filthy Rich Client applications
- Romain knows how to use Keynote
Filthy Rich Clients are not specific to any particular platform or software stack.

They are a set of techniques applicable across many platforms and toolkits.

Android is a modern mobile operating system offering advanced features for graphical effects.

Let’s discover some of these features.
Agenda

- Architecture
- Graphics
- Animation
- Performance
Agenda

- Architecture
- Graphics
- Animation
- Performance
Glossary

- **Canvas**: 2D drawing context
- **Drawable**: Abstract painter
- **PixelFlinger**: Rasterizer (OpenGL JIT for ARM)
- **SGL**: 2D drawing API (Skia)
- **Surface**: Drawing buffer
- **SurfaceFlinger**: Surface manager
- **View**: UI widget
- **ViewGroup/Layout**: UI widget container
Architecture

Application

Canvas

SGL

OpenGL

Surface
Architecture

- OpenGL
- GPU
- PixelFlinger
Architecture

Surface

Surface

Surface

SurfaceFlinger

OpenGL

GPU

PixelFlinger
How to draw

```java
1 public class CustomView extends View {
2     @Override
3     protected void onDraw(Canvas canvas) {
4         // draw stuff
5     }
6 }
```
How to draw

1 public class CustomDrawable extends Drawable {
2       @Override
3       public void draw(Canvas canvas) {
4           // draw stuff
5       }
6 }
Redrawing

ViewRoot

invalidate()
draw()

ViewGroup

ViewGroup

View

View
Drawing sequence

- ViewGroup
  - background.draw(Canvas)
  - onDraw(Canvas)
  - dispatchDraw(Canvas)

- Drawable

- View
- View
Agenda

- Architecture
- Graphics
- Animation
- Performance
DEMO

3D Reflection
Fundamentals

- Paints
- Gradients
- Transfer modes
- 3D Transformations
- Shadows
About paints

- Canvas is mostly stateless
- Transformation matrix
- Paint contains the state
  - Opacity, color and color filter
  - Transfer mode, mask filter and shader
  - Anti-aliasing, filtering and dithering
- Stroke and fill
DEMO

Color filter in Home

Screen transfer mode in Shelves

Faded edges in lists
Gradients

- Shader
  - Horizontal span of colors
- LinearGradient
- RadialGradient
- SweepGradient
Gradients

```java
1 Paint mPaint = new Paint();
2 mPaint.setShader(new LinearGradient(0, 0, 0, 20.0f, 0xFFFFFFFF, 0, TileMode.CLAMP));

6 // in onDraw(Canvas)
7 canvas.drawRect(0.0f, 0.0f, 20.0f, 20.0f, mPaint);
```
Transfer modes

- In Java2D, AlphaComposite
- Does more
- Modes
  - Porter-Duff (SrcOver, Atop, DstOut, etc.)
  - Color blending (Screen, Darken, Multiply, etc.)
Transfer modes

```java
Shader gradientShader = new LinearGradient(0, 0, 0, 1,
    0xFF000000, 0, TileMode.CLAMP);

Shader bitmapShader = new BitmapShader(mBitmap,
    TileMode.CLAMP, TileMode.CLAMP);

Shader composeShader = new ComposeShader(
    bitmapShader, gradientShader,
    new PorterDuffXfermode(Mode.DST_OUT));

Paint mPaint = new Paint();
mPaint.setShader(composeShader);
```
3D transformations

- 2D Canvas transformations
  - scale(), translate(), rotate()
- Canvas uses a 4x4 transformation matrix
- 3D transformations
  - Use android.graphics.Camera
3D transformations

1 Camera mCamera = new Camera();
2 // Z translation
3 mCamera.translate(0.0f, 0.0f, 350.0f);
4 // rotation around the Y axis in degrees
5 mCamera.rotateY(45);
6
7 // in onDraw(Canvas)
8 canvas.save();
9 canvas.concat(mCamera.getMatrix());
10 canvas.drawBitmap(bitmap, 0.0f, 0.0f, null);
11 canvas.restore();
3D transition
Shadows

```java
Paint mShadow = new Paint();

// radius=10, y-offset=2, color=black
mShadow.setShadowLayer(10.0f, 0.0f, 2.0f, 0xFF000000);

// in onDraw(Canvas)
canvas.drawBitmap(bitmap, 0.0f, 0.0f, mShadow);
```
Agenda

- Architecture
- Graphics
- Animation
- Performance
Animation

- Why?
  - Better visual feedback
  - UI appears more responsive

- How?
  - Animation
  - LayoutAnimation
Bring life to your application

- Life is restless
  - Transitions, highlights, progress, motion, etc.
- Animate changes
  - Adding/removing views
- Keep animations short and simple
Animation features

- Start delay
- Start time
- Duration
- Repeat mode
- Repeat count
- Interpolation
- Fill before/after
- Defined in XML or code
Inside animations

- Subclass of Animation
- Tied to a View
  - View.setAnimation()/startAnimation()
- Not driven by a timer
  - But time driven
- Driven by the drawing code
  - View.getDrawingTime()
Inside animations

- Fixed set of animated properties
  - AlphaAnimation
  - RotateAnimation
  - ScaleAnimation
  - TranslateAnimation
- View itself is not animated
  - Only a bitmap copy is
- Drawing cache API
DEMO

Animation in Home

Animation in Shelves
Defining the animation

res/anim/slide_in.xml

```xml
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <translate
    android:fromYDelta="0"
    android:toYDelta="100%"
    android:duration="200" />
  <alpha
    android:fromAlpha="1.0"
    android:toAlpha="0.0"
    android:duration="200" />
</set>
```
Playing the animation

1 Animation animation;
2 animation = AnimationUtils.loadAnimation(
3 context, R.anim.slide_in);
4 view.startAnimation(animation);
Layout animations

- Apply to a ViewGroup’s children
  - One animation
  - Each child has the same animation
  - Each child has a different start delay

- Layout animation controller
  - Defines the start delay for each child
  - Based on the index, position, column, row, etc.
Layout animations
Defining the layout animation

```
res/anim/layout_fade

1  <gridLayoutAnimation
2    android:columnDelay="50%"
3    android:directionPriority="row"
4    android:direction="right_to_left|bottom_to_top"
5    android:animation="@anim/fade" />
```
Playing the layout animation

1. `<GridView
2.   android:layoutAnimation="@anim/layout_fade"
3.   android:layout_width="fill_parent"
4.   android:layout_height="fill_parent"/>

Transitions

- Long operations
- Long-press for contextual actions
- Changes
  - Avoid jarring effect
- TransitionDrawable
  - Contains 2 drawables
  - Fade between them
Defining a transition

res/drawable/transition

1  <transition>
2   <item android:drawable="@drawable/start" />
3   <item android:drawable="@drawable/end" />
4  </transition>
Playing a transition

1 `TransitionDrawable` drawable;
2 `drawable = getDrawable(R.drawable.transition);`
3 `view.setBackgroundDrawable(drawable);`
4 `drawable.startTransition(1000);`
Transition in Home
Transition in Shelves
Agenda

- Architecture
- Graphics
- Animation
- Performance
Performance

- G1 hardware
- ~384 Mhz CPU
- 16 MB of RAM per process
- ATI Imageon GPU
- Interpreted VM
- Simple Garbage Collector
- SGL is not hardware accelerated
- Native code is not supported (yet)
General optimizations

- Do not allocate at drawing time
- Avoid method calls
- Especially interface calls
- Avoid invalidate()
- Invalidate only what you need
  - invalidate(left, top, right, bottom)
- Flatten the view hierarchy
DEMO

HierarchyViewer

DDMS
Bitmaps

- Drawable stretch bitmaps
- Size your bitmap accordingly
- `Bitmap.createScaledBitmap()`
- `BitmapFactory.Options.inSampleSize`
- Dithering at drawing time is costly
- Pre-dither bitmaps (Photoshop plugin)
- `BitmapFactory.Options.inDither`
Backgrounds

- Remove unnecessary backgrounds
- No “opaque view” optimization
- getWindow().setBackgroundDrawable(null)
- For instance: Home, Google Maps, Shelves
- Prefer ColorDrawable
Drawing cache

- Intermediate bitmap
- Special API
  - View.setDrawingCacheEnabled()
  - View.buildDrawingCache()
  - View.getDrawingCache()
- Sometimes managed automatically
  - ViewGroup (animations)
  - ListView (scrolling)
Home
ListView
Concluding statement

Filthy Rich Clients are possible on today’s mobile devices. Powerful APIs and hardware open new possibilities that have barely been explored.
Q&A

romainguy@android.com
Thanks for your attention!

http://www.android.com
http://source.android.com
http://code.google.com/android