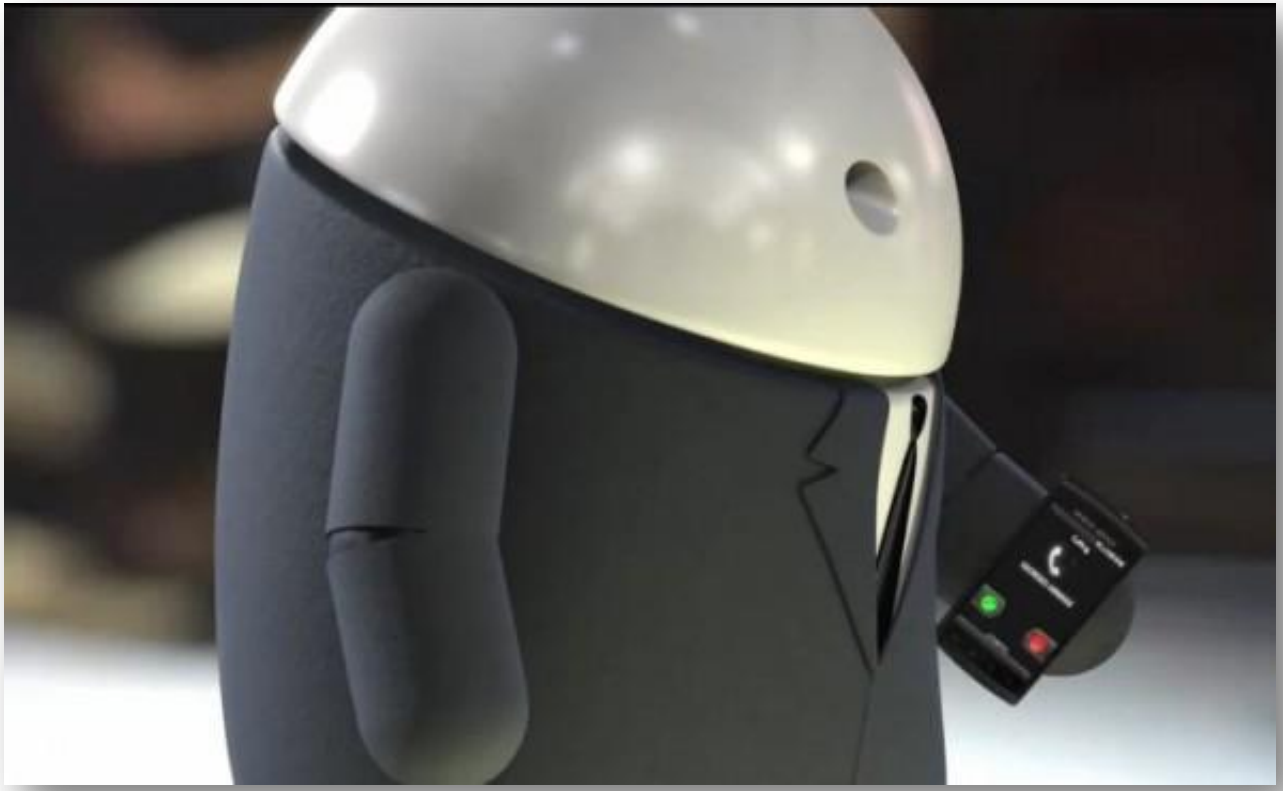
Game coder in the world of Android UI

Few words on optimisation from different angle

What's going on?

Android is a „business" thing



Meanwhile in the games world

What do we have?

- A CPU
  - 1 GHz
- A GPU
  - 22 M tris/s, 133 M pix/s, 128 MHz
- Memory
  - 512 MB
- Screen
  - 800 x 480

Nexus One


Google
htc
11:03 PM
Google
Maps
Gallery
YouTube
Market
Phone
Gmail
Browser
Voice
Helping Grandpa Get His Tech On
New York Times
61°

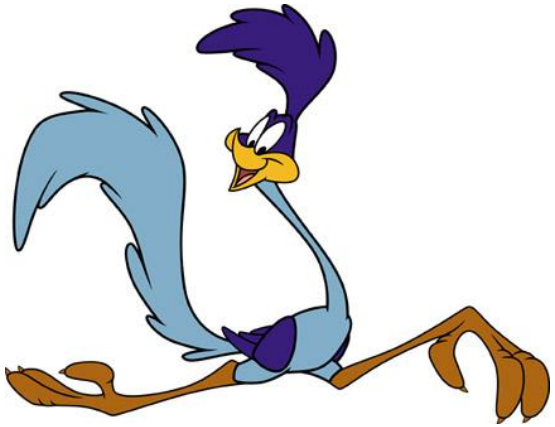
ARM is fast!

Qualcomm Adreno 200

- 22 M tris/s
- 33 M pix/s
- 48?x 128 MHz
- 32 MB RAM

NVidia GeForce GT 240M

- 4.4 G pix/s
- 48x 1.2 Ghz
- 1+1.7 GB RAM



Adreno 200, Nexus One

- 1 full-screen blending
  - 800*480*4*60 = 88 MHz = **69%** of core clock (1 of 48?)
- 1 full-screen draw with filtering
  - 800*480*5*60 = 110 Mp/s = **83%** of GPU fillrate
- 1 full-screen image
  - 800*480*4 = 1.5 MB = **5%** of app memory

Adreno 330, Nexus 5

- 1 full-screen blending
  - 1920*1080*4*60 = 0.5 GHz = **105%** of core clock (1 of 128?)
- 1 full-screen draw with filtering
  - 1920*1080*5*60 = 0.6 Gp/s = **16%** of GPU fillrate
- 1 full-screen image
  - 1920*1080*4 = 8 MB = **12%** of app memory

GeForce GT 240M, ASUS N61VN

- 1 full-screen blending
  - 1366*768*4*60 = 240 MHz = **20%** of core clock (1 of 48)
- 1 full-screen draw with filtering
  - 1366*768*5*60 = 0.3 Gp/s = **7%** of GPU fillrate
- 1 full-screen image
  - 1366*768*4 = 4 MB = **0,4%** of app memory

Blending

$$c_{out} = c_{dest}\left(1-\alpha_{src}\right)+c_{src}\alpha_{src}$$

- 2 reads + 1 write
  - 2 adds + 2 muls
- Preprocessing

Filtering

wikipedia

- 4 reads + 1 write
- Motion blur
- Prescaling

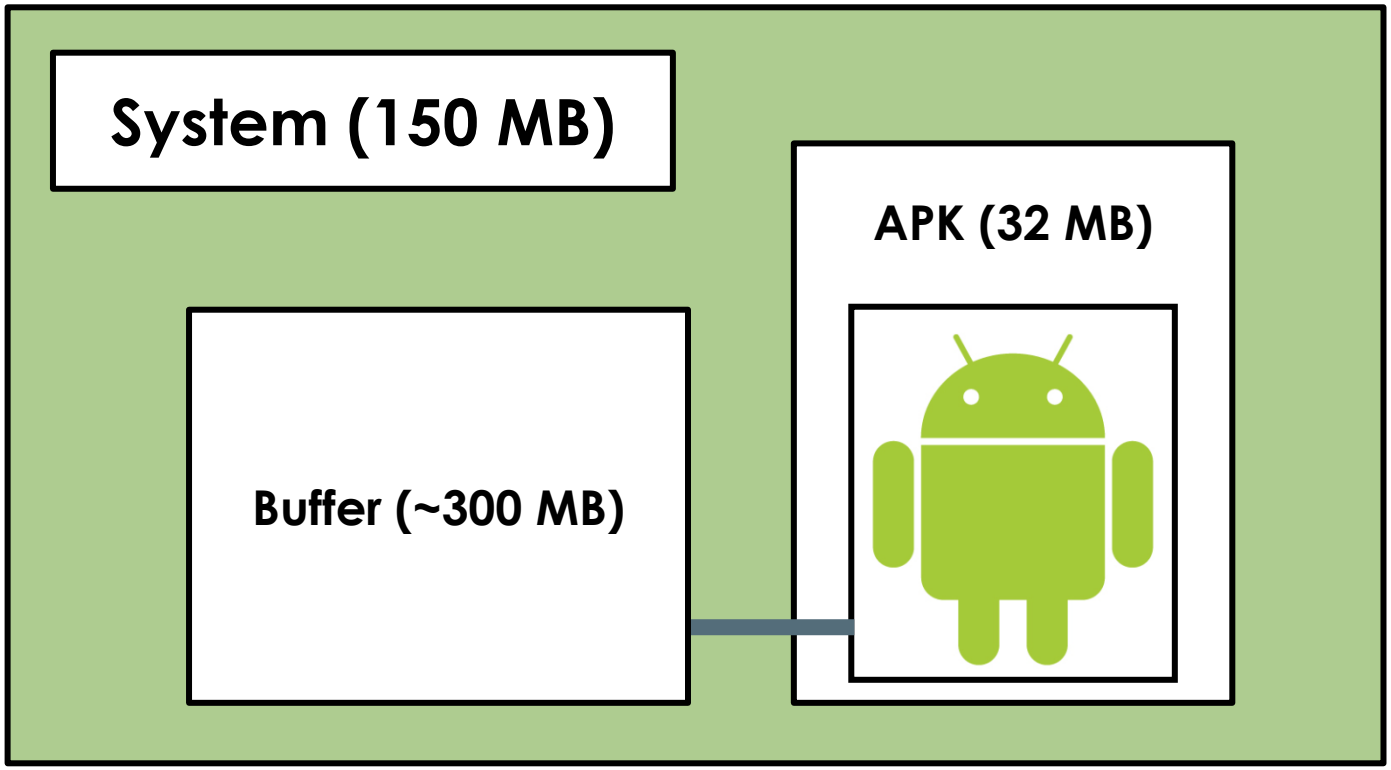
Memory

$$32MB/800/480/4bpp = 20$$

- Not only graphics
- Preprocessing
  - Scaling
  - Shadows
  - Blending
- Cache or on-the-fly?

More memory

JNI gives access to entire memory



More memory 2

```
JNIEXPORT jint JNICALL setBitmapData(JNIEnv *env,jobject obj,
jobject bitmap){
    AndroidBitmapInfo info;
    void *pixels;

    AndroidBitmap_getInfo(env, bitmap, &info));
    AndroidBitmap_lockPixels(env, bitmap, &pixels));

    int length = info.stride*info.height;
    void *data = malloc(length*sizeof(int));
    memcpy(data,pixels,length);

    AndroidBitmap_unlockPixels(env, bitmap);

    return data;
}
```

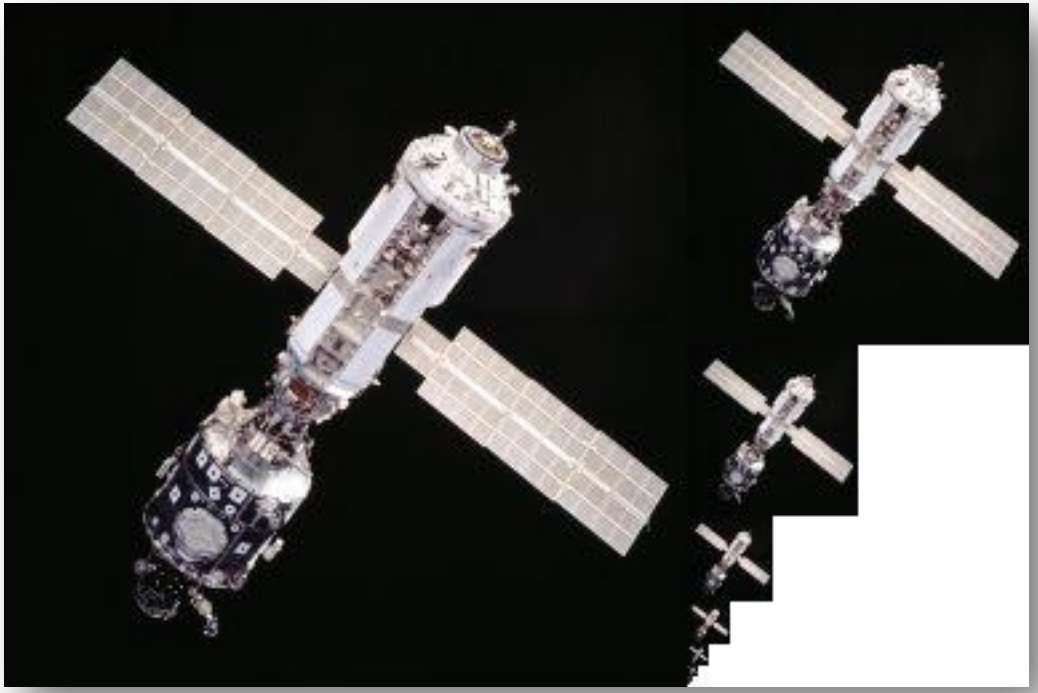
More memory 3

```
Bitmap bitmap =
BitmapFactory.decodeResource(R.drawable.nice_pic);
int pointer = NativeBuffer.setBitmapData(bitmap);
bitmap.recycle();
```

```
public void draw(Canvas canvas){
    if(bitmap.isRecycled()){
        NativeBuffer.getBitmapData(bitmap,pointer);
    }
    setImageBitmap(bitmap);
    super.draw(canvas);
}
```

Google code: nativebuffer

Mipmapping



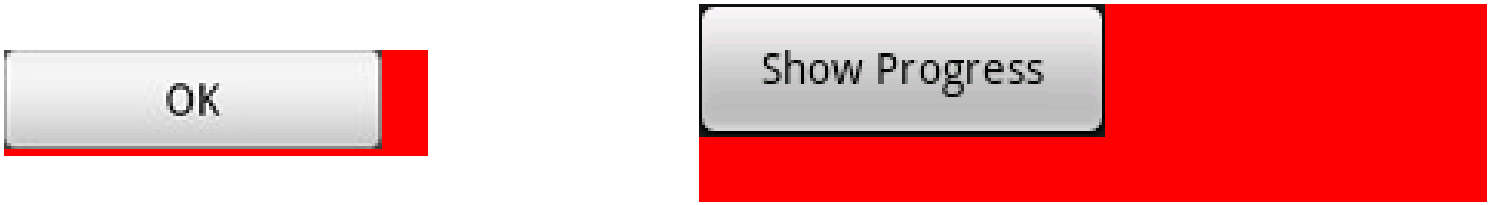
- Faster loading
- Smaller fillrate footprint

GPU and UI

- Dynamic views
- OpenGL libraries
- Unsupported clipping

GPU and UI 2

- Power of two textures



Can we do more?

- Don't draw occludees
- Thread priority
  - 90% for drawing thread on iOS
- Object reusing
  - Views
  - Strings
  - Arrays of objects (DOD)

Optimise. Or optimise not.

Back to real life

- Graphics is not that slow
  - Text and XML
  - Files and databases
  - Media decoding
  - Networking
- Business approach is not working


That's all Folks!