Anatomy of a Screenshot

Rodrigo Chiossi
**Background** (Or where did this talk came from!)

- **Project requirements:**
  - Implement a new SystemUI feature
  - Take screenshot, process, present the result to the user.

- **Constraints**
  - Results must be generated within 500ms

- **Target**
  - Android 4.4.4
Objectives

- Starting from the event trigger (button event detected) get to OpenGL calls.
- Analyse filtering performance.
- Emulator problems and fixes.
- Improvements in Android 5.0+

PS: All code snippets in this presentation are from android tag android-4.4.4_r1
Screenshot: Who is involved

Considering the usual method of taking a screenshot in an Android Phone, three processes take part:

- `system_server`
- `screenshot (SystemUI)`
  - `android:process=":screenshot"`
- `surface_flinger`

PS: Usual Method - Power Button + Volume Down Button
Process Stack

- system_server
- screenshot (SystemUI)
- surface_flinger
Source File Stack

PhoneWindowManager.java
TakeScreenshotService.java
GlobalScreenshot.java
SurfaceControl.java
android_view_SurfaceControl.cpp
SurfaceComposerClient.cpp/.h
ISurfaceComposer.cpp/.h
SurfaceFlinger.cpp/.h
Layer.cpp/.h
GLES{##}RenderEngine.cpp/.h
First: frameworks/base
PhoneWindowManager.java

- Intercept ScreenshotChord()
  - Button trigger event is identified
  - mScreenshotRunnable posted
    - Start and bind to TakeScreenshot service

```java
ComponentName cn = new ComponentName("com.android.systemui",
  "com.android.systemui.screenshot.TakeScreenshotService");
Intent intent = new Intent();
...
if (mContext.bindServiceAsUser (intent, conn, Context.BIND_AUTO_CREATE, UserHandle.CURRENT)) {
```

...
TakeScreenshotService.java

- Instantiate GlobalScreenshot
- call takeScreenshot()

```java
if (mScreenshot == null) {
    mScreenshot = new GlobalScreenshot (TakeScreenshotService.this);
}

mScreenshot.takeScreenshot (new Runnable () {
```
GlobalScreenshot.java

- Check display metrics and rotation
- Take the screenshot
- Generate the final bitmap
  - Apply rotation if required
- Show Screenshot animation

```java
mDisplay.getRealMetrics(mDisplayMetrics);
float[] dims = {mDisplayMetrics.widthPixels, mDisplayMetrics.heightPixels};
...
// Take the screenshot
mScreenBitmap = SurfaceControl.screenshot((int) dims[0], (int) dims[1]);
```
SurfaceControl.java

- Set the layers to be included in the screenshot.
  - In this case, all layers are included.
- Move to native code.

```java
public static Bitmap screenshot(int width, int height) {
    ...
    IBinder displayToken = SurfaceControl.getBuiltInDisplay(
        SurfaceControl.BUILT_IN_DISPLAY_ID_MAIN);
    return nativeScreenshot(displayToken, width, height, 0, 0, true);
}
```
PS: Layers

- Defines the type of the Window and it’s Z order.
- Defined in WindowManager.java
- Managed by PhoneWindowManager.java
  - Assign an integer value to each layer
  - windowTypeToLayer()
- Calculated per WindowState in WindowManagerService
  - assignLayersLocked()
android_view_SurfaceControl.cpp

- Map java methods to native methods.
  - `nativeScreenshot()` → `nativeScreenshotBitmap()`
- Creates a ScreenshotClient instance.
- Take the screenshot.
  - Call `ScreenshotClient->update()`
- Creates a bitmap for the screenshot.
  - Set pixel format.

```cpp
126  ScreenshotClient * screenshot = new ScreenshotClient();
127  status_t res = (width > 0 && height > 0)
128      ? (allLayers
129        ? screenshot->update(displayToken, width, height)
130          : screenshot->update(displayToken, width, height, minLayer, maxLayer))
131      : screenshot->update(displayToken);
```
Second: frameworks/native
SurfaceComposerClient.cpp/.h

- Defines ScreenshotClient.
- Gets the Interface to SurfaceComposer.
- Capture the screen.
  - ISurfaceComposer->captureScreen()

```cpp
654  sp<ISurfaceComposer> s(ComposerService::getComposerService());
...
664  status_t err = s->captureScreen(display, mBufferQueue,
       reqWidth, reqHeight, minLayerZ, maxLayerZ);
```
Defines BnSurfaceComposer class.
  ○ “Implements” ISurfaceComposer interface.
Send CAPTURE_SCREEN message.
Call captureScreen() when message is received on target.

```cpp
118     remote()->transact(BnSurfaceComposer::CAPTURE_SCREEN, data, &reply);
...
279     case CAPTURE_SCREEN: {
...
288     status_t res = captureScreen(display, producer,
289           reqWidth, reqHeight, minLayerZ, maxLayerZ);
```
SurfaceFlinger.cpp/.h

- Inherits BnSurfaceComposer.
- Creates MessageCaptureScreen.
  - postMessageAsync().

```cpp
class MessageCaptureScreen : public MessageBase {
...

  virtual bool handler() {
...

    result = flinger->captureScreenImplLocked(hw,
    producer, reqWidth, reqHeight, minLayerZ, maxLayerZ);
...

    sp<MessageBase> msg = new MessageCaptureScreen(this,
    display, IGraphicBufferProducer::asInterface(wrapper),
    reqWidth, reqHeight, minLayerZ, maxLayerZ);
...

    status_t res = postMessageAsync(msg);
```
SurfaceFlinger.cpp/.h

- captureScreenImplLocked().
  - Calls renderScreenImplLocked().
- renderScreenImplLocked().
  - Set Viewport and Projection.
  - Iterate layers that will be drawn.

```cpp
renderScreenImplLocked(hw, reqWidth, reqHeight,
                        minLayerZ, maxLayerZ, true);
```
```cpp
engine.setViewportAndProjection(reqWidth, reqHeight, hw_w, hw_h, yswap);
```
```cpp
for (size_t i=0 ; i<count ; ++i) {
    layer->draw(hw);
```
Layer.cpp/.h

- `draw()` → `onDraw()` → `drawWithOpenGL()`
- Get RenderEngine
  - OpenGLES 1.0, 1.1 or 2.0
- `drawMesh()`

```
598   RenderEngine & engine(mFlinger->getRenderEngine());
...
600   engine.drawMesh(mMesh);
```
GLES{##}RenderEngine.cpp/.h

- Different versions for OpenGLES 1.0, 1.1 and 2.0.
- Implements the actual drawing.

```cpp
186  void GLES20RenderEngine::drawMesh(const Mesh& mesh) {
205    glDrawArrays(mesh.getPrimitive(), 0, mesh.getVertexCount());
```
Timing

- Time to take a screenshot and show the result screen inside an ImageView
  - \( \approx 600 \text{ms} \)
- Too high for smooth user experience
- Could it be optimized?
void SurfaceFlinger::renderScreenImplLocked(...) {
    // get screen geometry
    const uint32_t hw_w = hw->getWidth();
    const uint32_t hw_h = hw->getHeight();
    const bool filtering = reqWidth != hw_w || reqWidth != hw_h;

    const LayerVector& layers( mDrawingState.layersSortedByZ);
    const size_t count = layers.size();
    for (size_t i=0 ; i<count ; ++i) {
        const sp<Layer>& layer (layers[i]);
        const Layer::State& state (layer->getDrawingState());
        if (state.layerStack == hw->getLayerStack()) {
            if (state.z >= minLayerZ && state.z <= maxLayerZ) {
                if (layer->isVisible()) {
                    if (filtering) layer->setFiltering(true);
                    layer->draw(hw);
                    if (filtering) layer->setFiltering(false);
                }
            }
        }
    }
}
void SurfaceFlinger::renderScreenImplLocked(...) {
    // get screen geometry
    const uint32_t hw_w = hw->getWidth();
    const uint32_t hw_h = hw->getHeight();
    const bool filtering = reqWidth != hw_w || reqWidth != hw_h;
    const bool filtering = reqWidth != hw_w || reqHeight != hw_h;
    const LayerVector& layers( mDrawingState. layersSortedByZ );
    const size_t count = layers.size();
    for (size_t i=0 ; i<count ; ++i) {
        const sp<Layer>& layer(layers[i]);
        const Layer::State& state( layer->getDrawingState() );
        if (state.layerStack == hw->getLayerStack()) {
            if (state.z >= minLayerZ && state.z <= maxLayerZ) {
                if (layer->isVisible()) {
                    if (filtering) layer->setFiltering( true );
                    layer->draw(hw);
                    if (filtering) layer->setFiltering( false );
                }
            }
        }
    }
}
Timing Fixed

- Time to take a screenshot and show the result screen inside an ImageView.
  - \( \approx 90 \text{ms.} \)
- Fast enough for smooth user experience.
- Bug still present in Android 5.1.
- Patch:
  - [filtering.patch](http://androidxref.com)
  - My fault for not submitting for AOSP!
Emulator

● Unable to take screenshots
● Emulator uses OpenGLES 1.0 when GPU acceleration is disabled.
● OpenGLES 1.0 does not have support for framebuffers, which are required for taking screenshots
● Legacy screenshot code does not use framebuffers
● Workaround Patch:
  ○ http://androidxref.com/goldfish_screenshot.patch
Emulator

- When GPU acceleration is enabled (-gpu on), OpenGLES 1.1 is used.
- Workaround patch does not work for 1.1, but will generate a black screenshot instead of failing the operation.
Memory Consumption

- Capturing just a subregion of the screen requires two Bitmaps to be created.
  - Viewport/Projection is fixed.
  - Create a Bitmap of the entire screen then crop

- Easy to hit maximum heap size.
  - Bitmaps are allocated in the app heap to make it easier to debug.
  - android:largeHeap="true" helps but may not be enough
Android 5.0

• Support for custom projection and viewport
  ○ Allows to capture only a subregion of the screen.
  ○ Reduces the amount of memory required in the operation (Only one bitmap required, instead of two).

```java
private static void screenshot(IBinder display, Surface consumer, Rect sourceCrop,
    int width, int height, int minLayer, int maxLayer, boolean allLayers,
    boolean useIdentityTransform) {

void GLES20RenderEngine::setViewportAndProjection(
    size_t vpw, size_t vph, Rect sourceCrop, size_t hwh, bool yswap,
    Transform::orientation_flags rotation) {
```
Rodrigo Chiossi
r.chiossi@androidxref.com
@rchiossi