Android WebKit Drawing Pipeline and Instrumentation

Wei Dong

April 20, 2012

1 WebView and WebViewCore

1.1 Construction

WebView is a widget provided by the android API.

```
/* frameworks/base/core/java/android/webkit */

public class WebView ... {
    private WebViewCore mWebViewCore;

    public WebView(Context context);
    public WebView(Context context, AttributeSet attrs);
    public WebView(Context context, AttributeSet attrs, int defStyle);

    protected WebView (Context context, AttributeSet attrs, int ↩
        defStyle, Map<String, Object> javascriptInterfaces) {
        ...
        mCallbackProxy = new CallbackProxy(context, this);
        ...
        mWebViewCore = new WebViewCore(context, this, mCallbackProxy, ↩
            javascriptInterfaces);
        ...
    }
};
```

The WebView widget use WebViewCore to implement a series of core functionalities, for example

```
/* frameworks/base/core/java/android/webkit/WebView.java */

public class WebView ... {

    public void loadUrl(String url, Map<String, String> extraHeaders) ↩
        {
        switchOutDrawHistory();
        WebViewCore.GetUrlData arg = new WebViewCore.GetUrlData();
        arg.mUrl = url;
        arg.mExtraHeaders = extraHeaders;
        mWebViewCore.sendMessage(EventHub.LOAD_URL, arg);
        clearTextEntry(false);
```

```
    }
}
```

We see that the parameters are packed into a message, and the method named mWebViewCore.sendMessage is invoked to send the message. The message sending mechanism is used because the actual workload of the methods are carried out in a separate thread created within the WebViewCore object, and the message is sent to that thread.

```
/* frameworks/base/core/java/android/webkit/WebViewCore.java */

final class WebViewCore {

    public WebViewCore(Context context, WebView w, CallbackProxy proxy↩
        , Map<String, Object> javascriptInterfaces) {
        ...
        synchronized (WebViewCore.class) {
            ...
            Thread t = new Thread(new WebCoreThread());
            ...
            t.start();
            ...
            WebViewCore.class.wait();     // wait for the thread to ↩
                start
        }
        ...

        mEventHub = new EventHub();       // explained later
        mSettings = new WebSettings(mContext, mWebView); // explained ↩
            later
        ...

    }

    // the actual implementation of the thread
    private static class WebCoreThread implements Runnable {
        public void run() {
            ...
            // initialize
            ...
            synchronized (WebViewCore.class) {
                ...
                // initialization done, thread will start looping
                WebViewCore.class.notify();
            }
            // The WebViewCore constructor will continue at this point
            Looper.loop();  // Run message queue in the loop.
        }
    }
}
```

1.2 Message Passing Mechanisim

The message passing is implemented with static methods of the android.os.Looper class. When the method Looper.loop() is invoked in WebCoreThread.run, the threads starts a event loop, and invokes mHandler to process messages.

```
```

```
/* frameworks/base/core/java/android/webkit/WebViewCore.java */

final class WebViewCore {

    // the two objects below are constructed in the constructor after
    // the WebCoreThread is created.
    private Handler mHandler;
    private ArrayList<Message> mMessages = new ArrayList<Message>();


    private synchronized void sendMessage(Message msg) {
        ...
        if (mMessages != null) {
            mMessages.add(msg);
        } else {
            mHandler.sendMessage(msg);
        }
    }

    ...
    class EventHub {

        private EventHub() {}

        private void transferMessages() {
            mHandler = new Handler() {
                ...
                switch (msg.what) {
                    case WEBKIT_DRAW:
                        webkitDraw();
                        break;
                    ...
                    case LOAD_URL:
                        GetUrlData param = (GetUrlData) msg.obj;
                        loadUrl(param.mUrl, param.mExtraHeaders);
                        break;
                    ...
                }
            }

            synchronized (this) {
                int size = mMessages.size();
                for (int i = 0; i < size; i++) {
                    mHandler.sendMessage(mMessages.get(i));
                }
                mMessages = null;
            }
        }
    }
    ...

    private void initialize() {
        ...
        mEventHub.transferMessages();
        ...
    }
}
```

And there is a one-shot message passing carried out during the initialization process with the member "sWebCoreHandler".

```
/* frameworks/base/core/java/android/webkit/WebViewCore.java */

final class WebViewCore {
```

```
    private static Handler sWebCoreHandler;

    private static class WebCoreThread implements Runnable {

        public void run() {
            ...
            synchronized (WebViewCore.class) {
                sWebCoreHandler = new Handler() {
                    @Override
                    public void handleMessage(Message msg) {
                        switch (msg.what) {
                            case INITIALIZE:
                                WebViewCore core = (WebViewCore) msg.↩
                                    obj;
                                core.initialize();
                                break;
                ...
                WebViewCore.class.notify();
            }
    }

    public WebViewCore(Context context, WebView w, CallbackProxy proxy↩
        , Map<String, Object> javascriptInterfaces) {
        ...
        synchronized (WebViewCore.class) {
            ...
            Thread t = new Thread(new WebCoreThread());
            ...
            t.start();
            ...
            WebViewCore.class.wait();     // wait for the thread to ↩
                start
        }
        ...
        ...
       Message init = sWebCoreHandler.obtainMessage(
                WebCoreThread.INITIALIZE, this);
       sWebCoreHandler.sendMessage(init);
    }
}
```

1.3 Wrap up

So the initialization process is as follows

1. WebView constructor begins.
2. WebViewCore constructor begins.
3. WebCoreThread created and started;
4. WebViewCore constructor sends the "INITIALIZE" message to the WebCoreThread.
5. Upon receiving the message, WebCoreThread invokes the "initialize" message of the WebCore object.
6. The initialize method finishes the start-up process by creating a mHandler object and use it to process all the messages accumulated in the mMessages queue.

7. After this, mMessage queue is deleted, and all forth-coming messages are directly sent via mHandler, using the android.os.Looper mechanism.

2 Bridging the Java WebViewCore and C++ WebCore::ScrollView

2.1 The Java Part

We already saw that the webkitDraw method is invoked upon the WEBKIT_DRAW message.

```
/* frameworks/base/core/java/android/webkit/WebViewCore.java */

final class WebViewCore {

    private void webkitDraw() {
        DrawData draw = new DrawData();
        if (DebugFlags.WEB_VIEW_CORE) Log.v(LOGTAG, "webkitDraw start"↩
            );
        if (nativeRecordContent(draw.mInvalRegion, draw.mWidthHeight)
                == false) {
            if (DebugFlags.WEB_VIEW_CORE) Log.v(LOGTAG, "webkitDraw ↩
                abort");
            return;
        }
        if (mWebView != null) {
            ...
            Message.obtain(mWebView.mPrivateHandler,
                WebView.NEW_PICTURE_MSG_ID, draw).sendToTarget();
            // if the picture is scrolled, sync to the scroll point
        }
    }
}

/* frameworks/base/core/java/android/webkit/WebView.java */

public class WebView {

    class PrivateHandler extends Handler {
        public void handleMessage(Message msg) {
            ...
            switch (msg.what) {
                ...
                case NEW_PICTURE_MSG_ID: {
                    ...
                    // drawing already happened.
                    // information like the drawn region size is ↩
                       contained
                    // in the DrawData object.  Use that to
                    // update states like zoom range.
                    ...
                    break;
                }
            }
        }
    }
}
```

So we see that NativeRecordContent is invoked to carry out the actual drawing, and after that is done, some information, like the drawn webpage size, is sent back to WebView, so that the zoom range is adjusted.

2.2 The C++ Part

The method NativeRecordContent is a JNI method implemented with C++.

```
/* external/webkit/WebKit/android/jni/WebViewCore.cpp */

#define GET_NATIVE_VIEW(env, obj) ((WebViewCore*)env->GetIntField(obj,↩
     gWebViewCoreFields.m_nativeClass))

static bool RecordContent(JNIEnv *env, jobject obj, jobject region, ↩
    jobject pt)
{
    WebViewCore* viewImpl = GET_NATIVE_VIEW(env, obj);
    SkRegion* nativeRegion = GraphicsJNI::getNativeRegion(env, region)↩
        ;
    SkIPoint nativePt;
    bool result = viewImpl->recordContent(nativeRegion, &nativePt);
    GraphicsJNI::ipoint_to_jpoint(nativePt, env, pt);
    return result;
}
```

The method WebViewCore::recordContent is invoked to do the rendering. The following methods are involved in the process

- WebViewCore::updateFrameCache
- WebViewCore::recordPicture
- WebViewCore::recordPictureSet
- WebViewCore::rebuildPicture
- WebViewCore::rebuildPicture

Finally, all paths of invocation points to the method

```
/* external/webkit/WebKit/android/jni/WebViewCore.{h,cpp} */

... within the context of WebViewCore class ...
    m_mainFrame->view()->platformWidget()->draw(&gc, ...)
...
// *m_mainFrame is of type WebCore::Frame
// m_mainFrame->view() returns a pointer to WebCore::FrameView
```

The WebCore::FrameView::platformWidget() returns a pointer to WebCore-ViewBridge. This is via the following inheritance hierarchy.

```
/* external/webkit/WebCore/platform/Widget.h */
```

```
#if PLATFORM(ANDROID)
class WebCoreViewBridge;
typedef WebCoreViewBridge* PlatformWidget;
#endif

class Widget :... {

    PlatformWidget platformWidget() const { return m_widget; }
private:
    PlatformWidget m_widget;

};


/* external/webkit/WebCore/platform/ScrollView.h */

class ScrollView : public Widget, public ScrollbarClient {
   ...
};

/* external/webkit/WebCore/platform/FrameView.h */

class FrameView : public ScrollView {
    ...

};
```

And WebCoreViewBridge::draw actually invokes WebCore::FrameView::paintConents.

```
/* external/webkit/WebKit/android/jni/WebCoreViewBridge.h */

class WebCoreViewBridge: ... {
    virtual void draw(WebCore::GraphicsContext* ctx,
        const WebCore::IntRect& rect) = 0;

};


/* external/webkit/WebKit/android/jni/WebFrameView.{h,cpp} */

class WebFrameView:: public WebCoreViewBridge {

    WebCore::FrameView* mFrameView;

    virtual void draw(WebCore::GraphicsContext* ctx, const WebCore::↩
        IntRect& rect) {
        ...
        mFrameView->paintContents(ctx, transRect);
        ...
    }
};
```

3 The WebKit Pipeline

```
/* external/webkit/WebKit/android/jni/WebViewCore.{h,cpp} */
android::WebFrameView::draw(WebCore::GraphicsContext* ctx, const ↩
    WebCore::IntRect& rect) {
    ...
```

```
    mFrameView->paintContents(ctx, ...);
    ...
}

android::FrameView::paintContents (WebCore::GraphicsContext* ctx, ↩
    const IntRect& rect) {
    RenderView* contentRenderer = frame()->contentRenderer();
    contentRenderer->layer()->paint(p, rect, paintBehavior, ↩
        eltRenderer);
}

/* external/webkit/WebKit/WebCore/rendering/RenderLayer.cpp */
void RenderLayer::paint (GraphicsContext* p, const IntRect& damageRect↩
    , PaintBehavior paintBehavior, RenderObject *paintingRoot)
{
    renderer()->paint(paintInfo, tx, ty);
    // *renderer() is RenderBoxModeContent
}

/* external/webkit/WebKit/WebCore/rendering/RenderImage.cpp */
void RenderImage::paint(PaintInfo& paintInfo, int tx, int ty) {
    ...
}
```

4 Skia in the Bottom

```
/* external/webkit/WebKit/WebCore/rendering/RenderImage.cpp */
void RenderImage::paintIntoRect(GraphicsContext* context, const ↩
    IntRect& rect)
{
    context->drawImage(image(rect.width(), rect.height()), style()->↩
        colorSpace(), rect, compositeOperator, useLowQualityScaling);
}

/* external/webkit/WebCore/platform/graphics/GraphicsContext.h */

#if PLATFORM(ANDROID)
namespace WebCore {
class PlatformGraphicsContext;
}
class SkPaint;
struct SkPoint;
#else
#if PLATFORM(ANDROID)
namespace WebCore {
class PlatformGraphicsContext;
}
class SkPaint;
struct SkPoint;
#else

class GraphicsContext : public Noncopyable {
public:
    GraphicsContext(PlatformGraphicsContext*);
private:
    GraphicsContextPrivate* m_common;
    GraphicsContextPlatformPrivate* m_data;
};


/* external/webkit/WebCore/platform/graphics/android/↩
    GraphicsContextAndroid.cpp */
```

```
GraphicsContext::GraphicsContext(PlatformGraphicsContext *gc)
    : m_common(createGraphicsContextPrivate())
    , m_data(new GraphicsContextPlatformPrivate(this, gc))
{
    setPaintingDisabled(NULL == gc || NULL == gc->mCanvas);
}

class GraphicsContextPlatformPrivate {
public:
    // All skia stuff.
}
```

5 Coordinate Transformation

WebKit rendering is carried out in blocks called Pictures.

```
/* external/webkit/WebKit/android/jni/WebViewCore.cpp */

// translation is done such that
// the picture contains the "inval" region from offset (0,0).

SkPicture* WebViewCore::rebuildPicture(const SkIRect& inval)
{
    ...
    SkPicture* picture = new SkPicture();
    ...
    WebCore::GraphicsContext gc(&pgc);  // canvas backed up with the ↩
        picture object
    ...
    recordingCanvas->translate(-inval.fLeft, -inval.fTop);
    recordingCanvas->save();
    view->platformWidget()->draw(&gc, WebCore::IntRect(inval.fLeft,
        inval.fTop, inval.width(), inval.height()));
    ...
}


/* external/webkit/WebKit/android/jni/PictureSet.cpp */

// when the picture is actually painted into the view canvas,
// the picture is translated back to it's proper offset

bool PictureSet::draw(SkCanvas* canvas)
{
    ...
    canvas->translate(pathBounds.fLeft, pathBounds.fTop);
    canvas->save();
    ...
    canvas->drawPicture(*working->mPicture);
    ...
}

}
```

WebKit stores all the rendered Pictures in the member WebViewCore::m_content of the type PictureSet. The following function is to actually draw the rendering results to the widget view.

```
/* external/webkit/WebKit/android/jni/WebViewCore.cpp */
```

```
bool WebViewCore::drawContent(SkCanvas* canvas, SkColor color)
{
    ...
    m_contentMutex.lock();
    PictureSet copyContent = PictureSet(m_content);
    m_contentMutex.unlock();
    ...
    bool tookTooLong = copyContent.draw(canvas);
    ...
}
```

A copy of the content is used for drawing, so that the original content can be updated at the same time when drawing happens.

6 When Drawing Actually Happens

```
/* frameworks/base/core/java/android/webkit/WebView.java */
class WebView {

    protected void draw (Canvas canvas) {
        ...
        int sx = getScrollX();
        int sy = getScrollY() - hiddenHeightOfTitleBar();
        ...
        canvas.translate(sx, sy);
        ...
        mProxy.onDraw(canvas);
        ...
    }

    protected void onDraw(Canvas canvas) {
        ...
        drawContent(canvas);
        ...
    }

    private void drawContent (Canvas canvas) {
        ...
        drawCoreAndCursorRing(canvas, mBackgroundColor, ↩
            mDrawCursorRing);
    }

    private void drawCoreAndCursorRing(Canvas canvas, int color, ↩
        boolean drawCursorRing) {
        ...
        canvas.scale(mActualScale, mActualScale);
        ...
        mWebViewCore.drawContentPicture(canvas, color, ..., ...);
        ...
    }
}


/* external/webkit/WebKit/android/jni/WebViewCore.cpp */

class WebViewCore {
    void drawContentPicture(Canvas canvas, int color,
                                        boolean animatingZoom,
                                        boolean animatingScroll) {
        ...
        boolean tookTooLong = nativeDrawContent(canvas, color);
```

```
        ...
    }
}
```

7 Security Issue

Android has very strict security rules. That is, Java apps are not allowed to write to most places in the filesystem. However, this does not apply to C++ code used to implement JNI methods. That's why we need to implement a JNI method WebViewCore.nativeInstrumentDisplay.

8 Instrumentation Implementation

```
project external/webkit/
diff --git a/WebCore/rendering/RenderImage.cpp b/WebCore/rendering/↩
    RenderImage.cpp
index 881d0b4..af5a08c 100644
--- a/WebCore/rendering/RenderImage.cpp
+++ b/WebCore/rendering/RenderImage.cpp
@@ -23,6 +23,8 @@
  *
  */

+#include <stdio.h>
+
 #include "config.h"
 #include "RenderImage.h"

@@ -51,6 +53,8 @@
 #include "WMLNames.h"
 #endif

+#include "CString.h"
+
 using namespace std;

 namespace WebCore {
@@ -497,6 +501,16 @@ void RenderImage::paintIntoRect(GraphicsContext* ↩
    context, const IntRect& rect)
     HTMLImageElement* imageElt = (node() && node()->hasTagName(imgTag↩
         )) ? static_cast<HTMLImageElement*>(node()) : 0;
     CompositeOperator compositeOperator = imageElt ? imageElt->↩
         compositeOperator() : CompositeSourceOver;
     bool useLowQualityScaling = RenderImageScaleObserver::↩
         shouldImagePaintAtLowQuality(this, rect.size());
+    do {
+
+   CachedImage *ci = cachedImage();
+        if (ci == NULL) break;
+   String url = ci->url();
+   FILE *os = fopen("/sdcard/log/webkit.log", "a");
+   fprintf(os, "IMAGE %d %d %d %d %s\n",
+           rect.x(), rect.y(), rect.width(), rect.height(), url.utf8↩
    ().data());
+   fclose(os);
+    } while(0);
```

```
     context->drawImage(image(rect.width(), rect.height()), style()->↩
         colorSpace(), rect, compositeOperator, useLowQualityScaling);
 }

diff --git a/WebKit/android/jni/WebViewCore.cpp b/WebKit/android/jni/↩
    WebViewCore.cpp
index 70e96cd..e3cd84e 100644
--- a/WebKit/android/jni/WebViewCore.cpp
+++ b/WebKit/android/jni/WebViewCore.cpp
@@ -354,6 +354,7 @@ WebViewCore::WebViewCore(JNIEnv* env, jobject ↩
    javaWebViewCore, WebCore::Frame* m
     reset(true);

     WebViewCore::addInstance(this);
+
 }

 WebViewCore::~WebViewCore()
@@ -373,6 +374,13 @@ WebViewCore::~WebViewCore()
     delete m_navPictureKit;
 }

+void WebViewCore::instrument_display (int w, int h, int x, int y, ↩
    float s)
+{
+    FILE *fout = fopen("/sdcard/log/webkit.log", "a");
+    fprintf(fout, "DRAW %d %d %d %d %g\n", w, h, x, y, s);
+    fclose(fout);
+}
+
 WebViewCore* WebViewCore::getWebViewCore(const WebCore::FrameView* ↩
     view)
 {
     return getWebViewCore(static_cast<const WebCore::ScrollView*>(↩
         view));
@@ -2567,6 +2575,12 @@ static void UpdateFrameCacheIfLoading(JNIEnv *↩
    env, jobject obj)
     GET_NATIVE_VIEW(env, obj)->updateFrameCacheIfLoading();
 }

+static void instrument_display (JNIEnv *env, jobject obj, jint w, ↩
    jint h, jint x, jint y, jfloat s)
+{
+    WebViewCore* viewImpl = GET_NATIVE_VIEW(env, obj);
+    viewImpl->instrument_display(w, h, x, y, s);
+}
+
 static void SetSize(JNIEnv *env, jobject obj, jint width, jint height↩
    ,
         jint screenWidth, jfloat scale, jint realScreenWidth, jint ↩
             screenHeight,
         jint anchorX, jint anchorY, jboolean ignoreHeight)
@@ -3177,6 +3191,8 @@ static JNINativeMethod gJavaWebViewCoreMethods[]↩
    = {
         (void*) SendListBoxChoice },
     { "nativeSetSize", "(IIIFIIIIZ)V",
         (void*) SetSize },
+    { "nativeInstrumentDisplay", "(IIIIF)V",
+   (void*) instrument_display},
     { "nativeSetScrollOffset", "(III)V",
         (void*) SetScrollOffset },
     { "nativeSetGlobalBounds", "(IIII)V",
diff --git a/WebKit/android/jni/WebViewCore.h b/WebKit/android/jni/↩
    WebViewCore.h
index 056dba1..e9e5c89 100644
--- a/WebKit/android/jni/WebViewCore.h
+++ b/WebKit/android/jni/WebViewCore.h
@@ -100,6 +100,7 @@ namespace android {
```

```
         WebViewCore(JNIEnv* env, jobject javaView, WebCore::Frame* ↩
             mainframe);
         ~WebViewCore();

+    void instrument_display (int w, int h, int x, int y, float s);
         // helper function
         static WebViewCore* getWebViewCore(const WebCore::FrameView* ↩
             view);
         static WebViewCore* getWebViewCore(const WebCore::ScrollView*↩
              view);

project frameworks/base/
diff --git a/core/java/android/webkit/WebView.java b/core/java/android↩
    /webkit/WebView.java
index 921d0f5..6c2a0b5 100644
--- a/core/java/android/webkit/WebView.java
+++ b/core/java/android/webkit/WebView.java
@@ -3233,6 +3233,13 @@ public class WebView extends AbsoluteLayout

     @Override
     protected void onDraw(Canvas canvas) {
+    {
+        int width = getViewWidth();
+        int height = getViewHeight();
+        int sx = getScrollX();
+        int sy = getScrollY() - getTitleHeight();
+        mWebViewCore.nativeInstrumentDisplay(width, height, sx, sy, ↩
    mActualScale);
+    }
         // if mNativeClass is 0, the WebView has been destroyed. Do ↩
             nothing.
         if (mNativeClass == 0) {
             return;
diff --git a/core/java/android/webkit/WebViewCore.java b/core/java/↩
    android/webkit/WebViewCore.java
index 4118119..08e5697 100644
--- a/core/java/android/webkit/WebViewCore.java
+++ b/core/java/android/webkit/WebViewCore.java
@@ -156,7 +156,8 @@ final class WebViewCore {
                     Log.e(LOGTAG, Log.getStackTraceString(e));
                 }
             }
-        }
+    }
+
         // Create an EventHub to handle messages before and after the↩
             thread is
         // ready.
         mEventHub = new EventHub();
@@ -488,6 +489,8 @@ final class WebViewCore {
             float scale, int realScreenWidth, int screenHeight, int ↩
                 anchorX,
             int anchorY, boolean ignoreHeight);

+    public native void nativeInstrumentDisplay(int w, int h, int x, ↩
    int y, float s);
+
     private native int nativeGetContentMinPrefWidth();

     // Start: functions that deal with text editing

project packages/apps/Browser/
diff --git a/AndroidManifest.xml b/AndroidManifest.xml
index 36e2820..a08f449 100644
--- a/AndroidManifest.xml
+++ b/AndroidManifest.xml
@@ -34,6 +34,8 @@
```

```
     <uses-permission android:name="com.android.browser.permission.↩
         READ_HISTORY_BOOKMARKS"/>
     <uses-permission android:name="com.android.browser.permission.↩
         WRITE_HISTORY_BOOKMARKS"/>
     <uses-permission android:name="android.permission.↩
         SEND_DOWNLOAD_COMPLETED_INTENTS" />
+    <uses-permission android:name="android.permission.↩
    WRITE_EXTERNAL_STORAGE" />
+

     <application   android:name="Browser"
                    android:label="@string/application_name"
diff --git a/src/com/android/browser/TabControl.java b/src/com/android↩
    /browser/TabControl.java
index 7cd2ccb..8eecc81 100644
--- a/src/com/android/browser/TabControl.java
+++ b/src/com/android/browser/TabControl.java
@@ -538,6 +538,10 @@ class TabControl {
      * Creates a new WebView and registers it with the global ↩
          settings.
      */
     private WebView createNewWebView() {
+    try {
+    Runtime.getRuntime().exec("echo hello > /sdcard/log/hello").↩
    waitFor();
+    } catch (Exception e) {}
+
         // Create a new WebView
         WebView w = new WebView(mActivity);
         w.setScrollbarFadingEnabled(true);
```

Code to render the trace (with Skia).

```
// modified from skimage_main.cpp
//
#include <boost/lexical_cast.hpp>
#include <string>
#include <vector>
#include "SkBitmap.h"
#include "SkGraphics.h"
#include "SkImageDecoder.h"
#include "SkImageEncoder.h"
#include "SkStream.h"
#include "SkTemplates.h"
#include "SkCanvas.h"

using namespace std;
using namespace boost;

static bool decodeFile(SkBitmap* bitmap, const char srcPath[]) {
    SkFILEStream stream(srcPath);
    if (!stream.isValid()) {
        SkDebugf("ERROR: bad filename <%s>\n", srcPath);
        return false;
    }

    SkImageDecoder* codec = SkImageDecoder::Factory(&stream);
    if (NULL == codec) {
        SkDebugf("ERROR: no codec found for <%s>\n", srcPath);
        return false;
    }

    SkAutoTDelete<SkImageDecoder> ad(codec);

    stream.rewind();
    if (!codec->decode(&stream, bitmap, SkBitmap::kARGB_8888_Config,
```

```
                        SkImageDecoder::kDecodePixels_Mode)) {
        SkDebugf("ERROR: codec failed for <%s>\n", srcPath);
        return false;
    }
    return true;
}

struct Meta {
    int w, h, x, y;
    float s;
};

struct Image {
    int x, y, w, h;
    string path;
};

void render (Meta const &meta, vector<Image> const &images, string ↩
    const &path) {
    SkBitmap bitmap;;
    bitmap.setConfig(SkBitmap::kARGB_8888_Config, meta.w, meta.h);
    bitmap.allocPixels();
    SkCanvas canvas(bitmap);
    canvas.drawColor(SK_ColorWHITE);
    canvas.translate(-meta.x, -meta.y);
    canvas.scale(meta.s, meta.s);

    for (unsigned i = 0; i < images.size(); ++i) {
        const Image &image = images[i];
        string cmd = "convert " + image.path + " -resize " + ↩
            lexical_cast<string>(image.w) + "x" + lexical_cast<string↩
            >(image.h) + " tmp.png >& /dev/null";
        system(cmd.c_str());
        SkBitmap bm;
        decodeFile(&bm, "tmp.png");
        canvas.drawBitmap(bm, image.x, image.y);
    }

    SkImageEncoder::EncodeFile(path.c_str(), bitmap, SkImageEncoder::↩
        kPNG_Type, 100);
}

int main (int argc, char * const argv[]) {
    int count=0;
    string method;
    Meta meta;
    vector<Image> images;
    bool first = true;
    while (cin >> method) {
        if (method == "DRAW") {
            if (images.size()) {    // the last draw command
                render(meta, images, "web/" + lexical_cast<string>(↩
                    count++) + ".png");
            }
            cin >> meta.w >> meta.h >> meta.x >> meta.y >> meta.s;
            first = true;
        }
        else {
            Image image;
            cin >> image.x >> image.y >> image.w >> image.h >> image.↩
                path;
            if (first) {
                first = false;
                images.clear();
            }
            images.push_back(image);
        }
    }
```

```
    if (!first) {
        render(meta, images, "web/" + lexical_cast<string>(count++) + ↩
            ".png");
    }

    return 0;
}
```

Figure 1: Emulator snapshot (a) and rendered trace (b).