Help save net neutrality! A free, open internet is once again at stake—and we need your help. **Learn more**

google / **grafika**

Grafika test app

60 commits **1** branch **1** release **1** contributor Apache-2.0

Branch: **master** **New pull request** **Create new file** **Upload files** **Find file** **Clone or download**

fadden Fix scheduled-swap crash … Latest commit `f3c8c3d` on 23 May 2014

.settings	Initial checkin of project.	4 years ago
libs	Initial checkin of project.	4 years ago
res	Minor tweaks	4 years ago
src/com/android/grafika	Fix scheduled-swap crash	4 years ago
.classpath	Initial checkin of project.	4 years ago
.gitignore	Many updates	4 years ago
.project	Initial checkin of project.	4 years ago
AndroidManifest.xml	Fix video player crash	4 years ago
LICENSE	Initial checkin of project.	4 years ago
README.md	Added link to arch doc	4 years ago
proguard-project.txt	Initial checkin of project.	4 years ago
project.properties	Initial checkin of project.	4 years ago

README.md

Grafika

Welcome to Grafika, a dumping ground for Android graphics & media hacks.

Grafika is:

- A collection of hacks exercising graphics features.
- An SDK app, developed for API 18 (Android 4.3). While some of the code may work with older versions of Android, no effort will be made to support them.
- Open source (Apache 2 license), copyright by Google. So you can use the code according to the terms of the license (see "LICENSE").
- A perpetual work-in-progress. It's updated whenever the need arises.

However:

- It's not stable.
- It's not polished or well tested. Expect the UI to be ugly and awkward.
- It's not intended as a demonstration of the proper way to do things. The code may handle edge cases poorly or not at all. Logging is often left enabled at a moderately verbose level.
- It's barely documented.
- It's not part of the Android Open Source Project. We cannot accept contributions to Grafika, even if you have an AOSP

CLA on file.

- It's NOT AN OFFICIAL GOOGLE PRODUCT. It's just a bunch of stuff that got thrown together on company time and equipment.
- It's generally just not supported.

To some extent, Grafika can be treated as a companion to the Android System-Level Graphics Architecture document. The doc explains the technology that the examples rely on, and uses some of Grafika's activities as examples. If you want to understand how the code here works, start by reading that.

There is some overlap with the code on http://www.bigflake.com/mediacodec/. The code there largely consists of "headless" CTS tests, which are designed to be robust, self-contained, and largely independent of the usual app lifecycle issues. Grafika is a conventional app, and makes an effort to handle app issues correctly (like not doing lots of work on the UI thread).

Features are added to Grafika as the need arises, often in response to developer complaints about correctness or performance problems in the platform (either to confirm that the problems exist, or demonstrate an approach that works).

There are two areas where some amount of care is taken:

- Thread safety. It's really easy to get threads crossed in subtly dangerous ways when working with the media classes. (Read the Android SMP Primer for a detailed introduction to the problem.) GL/EGL's reliance on thread-local storage doesn't help. Threading issues are frequently called out in comments in the source code.
- Garbage collection. GC pauses cause jank. Ideally, none of the activities will do any allocations while in a "steady state". Allocations may occur while changing modes, e.g. starting or stopping recording.

All code is written in the Java programming language -- the NDK is not used.

The first time Grafika starts, two videos are generated (gen-eight-rects, gen-sliders). If you want to experiment with the generation code, you can cause them to be re-generated from the main activity menu ("Regenerate content").

Some features of Grafika may not work on an emulator, notably anything that involves encoding video. The problem is that the software AVC encoder doesn't support `createInputSurface()` , and all of Grafika's video encoding features currently make use of that. Very little works in the AOSP emulator, even with `-gpu on` .

Current features

* Play video (TextureView). Plays the video track from an MP4 file.

- Only sees files in `/data/data/com.android.grafika/files/` . All of the activities that create video leave their files there. You'll also find two automatically-generated videos (gen-eight-rects.mp4 and gen-slides.mp4).
- By default the video is played once, at the same rate it was recorded. You can use the checkboxes to loop playback and/or play the frames at a fixed rate of 60 FPS.
- Uses a `TextureView` for output.
- Name starts with an asterisk so it's at the top of the list of activities.

Continuous capture. Stores video in a circular buffer, saving it when you hit the "capture" button. (Formerly "Constant capture".)

- Currently hard-wired to try to capture 7 seconds of video from the camera at 6MB/sec, preferrably 15fps 720p. That requires a buffer size of about 5MB.
- The time span of frames currently held in the buffer is displayed. The actual time span saved when you hit "capture" will be slightly less than what is shown because we have to start the output on a sync frame, which are configured to appear once per second.
- Output is a video-only MP4 file ("constant-capture.mp4"). Video is always 1280x720, which usually matches what the camera provides; if it doesn't, the recorded video will have the wrong aspect ratio.

Double decode. Decodes two video streams side-by-side to a pair of `TextureViews` .

- Plays the two auto-generated videos. Note they play at different rates.

- The video decoders don't stop when the screen is rotated. We retain the `SurfaceTexture` and just attach it to the new `TextureView`. Useful for avoiding expensive codec reconfigures. The decoders *do* stop if you leave the activity, so we don't tie up hardware codec resources indefinitely. (It also doesn't stop if you turn the screen off with the power button, which isn't good for the battery, but might be handy if you're feeding an external display or your player also handles audio.)
- Unlike most activities in Grafika, this provides different layouts for portrait and landscape. The videos are scaled to fit.

Hardware scaler exerciser. Shows GL rendering with on-the-fly surface size changes.

- The motivation behind the feature this explores is described in a developer blog post: http://android-developers.blogspot.com/2013/09/using-hardware-scaler-for-performance.html
- You will see one frame rendered incorrectly when changing sizes. This is because the render size is adjusted in the "surface changed" callback, but the surface's size doesn't actually change until we latch the next buffer. This is straightforward to fix (left as an exercise for the reader).

Live camera (TextureView). Directs the camera preview to a `TextureView`.

- This comes more or less verbatim from the TextureView documentation.
- Uses the default (rear-facing) camera. If the device has no default camera (e.g. Nexus 7 (2012)), the Activity will crash.

Multi-surface test. Simple activity with three overlapping SurfaceViews, one marked secure.

- Useful for examining HWC behavior with multiple static layers, and screencap / screenrecord behavior with a secure surface. (If you record the screen one of the circles should be missing, and capturing the screen should just show black.)
- If you tap the "bounce" button, the circle on the non-secure layer will animate. It will update as quickly as possible, which may be slower than the display refresh rate because the circle is rendered in software. The frame rate will be reported in logcat.

Play video (SurfaceView). Plays the video track from an MP4 file.

- Works very much like "Play video (TextureView)", though not all features are present. See the class comment for a list of advantages to using SurfaceView.

Record GL app. Simultaneously draws to the display and to a video encoder with OpenGL ES, using framebuffer objects to avoid re-rendering.

- It can write to the video encoder three different ways: (1) draw twice; (2) draw offscreen and blit twice; (3) draw onscreen and blit framebuffer. #3 doesn't work yet.
- The renderer is trigged by Choreographer to update every vsync. If we get too far behind, we will skip frames. This is noted by an on-screen drop counter and a border flash. You generally won't see any stutter in the animation, because we don't skip the object movement, just the render.
- The encoder is fed every-other frame, so the recorded output will be ~30fps rather than ~60fps on a typical device.
- The recording is letter- or pillar-boxed to maintain an aspect ratio that matches the display, so you'll get different results from recording in landscape vs. portrait.
- The output is a video-only MP4 file ("fbo-gl-recording.mp4").

Scheduled swap. Exercises a SurfaceFlinger feature that allows you to submit buffers to be displayed at a specific time.

- Requires API 19 (Android 4.4 "KitKat") to do what it's supposed to. The current implementation doesn't really look any different on API 18 to the naked eye.
- You can configure the frame delivery timing (e.g. 24fps uses a 3-2 pattern) and how far in advance frames are scheduled. Selecting "ASAP" disables scheduling.
- Use systrace with tags `sched gfx view --app=com.android.grafika` to observe the effects.
- The moving square changes colors when the app is unhappy about timing.

Show + capture camera. Attempts to record at 720p from the front-facing camera, displaying the preview and recording it simultaneously.

- Use the record button to toggle recording on and off.
- Recording continues until stopped. If you back out and return, recording will start again, with a real-time gap. If you try to play the movie while it's recording, you will see an incomplete file (and probably cause the play movie activity to crash).
- The recorded video is scaled to 640x480, so it will probably look squished. A real app would either set the recording size equal to the camera input size, or correct the aspect ratio by letter- or pillar-boxing the frames as they are rendered to the encoder.
- You can select a filter to apply to the preview. It does not get applied to the recording. The shader used for the filters is not optimized, but seems to perform well on most devices (the original Nexus 7 (2012) being a notable exception). Demo here: http://www.youtube.com/watch?v=kH9kCP2T5Gg
- The output is a video-only MP4 file ("camera-test.mp4").

Simple Canvas in TextureView. Exercises software rendering to a `TextureView` with a `Canvas` .

- Renders as quickly as possible. Because it's using software rendering, this will likely run more slowly than the "Simple GL in TextureView" activity.
- Toggles the use of a dirty rect every 64 frames. When enabled, the dirty rect extends horizontally across the screen.

Simple GL in TextureView. Demonstates simple use of GLES in a `TextureView` , rather than a `GLSurfaceView` .

- Renders as quickly as possible. On most devices it will exceed 60fps and flicker wildly, but in 4.4 ("KitKat") a bug prevents the system from dropping frames.

Texture from Camera. Renders Camera preview output with a GLES texture.

- Adjust the sliders to set the size, rotation, and zoom. Touch anywhere else to center the rect at the point of the touch.

Color bars. Displays RGB color bars.

OpenGL ES Info. Dumps version info and extension lists.

- The "Save" button writes a copy of the output to the app's file area.

glTexImage2D speed test. Simple, unscientific measurement of the time required to upload a 512x512 RGBA texture with `glTexImage2D()` .

glReadPixels speed test. Simple, unscientific measurement of the time required for `glReadPixels()` to read a 720p frame.

Known issues

- Nexus 4 running Android 4.3 (JWR67E): "Show + capture camera" crashes if you select one of the filtered modes. Appears to be a driver bug (Adreno "Internal compiler error").

Feature & fix ideas

In no particular order.

- Stop using AsyncTask for anything where performance or latency matters.
- Add a "fat bits" viewer for camera (single SurfaceView; left half has live camera feed and a pan rect, right half has 8x pixels)
- Change the "Simple GL in TextureView" animation. Or add an epilepsy warning.
- Cross-fade from one video to another, recording the result. Allow specification of the resolution (maybe QVGA, 720p, 1080p) and generate appropriately.
- Add features to the video player, like a slider for random access, and buttons for single-frame advance / rewind (requires seeking to nearest sync frame and decoding frames until target is reached).
- Convert a series of PNG images to video.
- Use virtual displays to record app activity.
- Play continuous video from a series of MP4 files with different characteristics. Will probably require "preloading" the next

movie to keep playback seamless.
- Experiment with alternatives to glReadPixels(). Add a PBO speed test. (Doesn't seem to be a way to play with eglCreateImageKHR from Java.)
- Do something with ImageReader class (req API 19).
- Figure out why "double decode" playback is sometimes janky.
- Add fps indicator to "Simple GL in TextureView".
- Capture audio from microphone, record + mux it.
- Enable preview on front/back cameras simultaneously, display them side-by-side. (This appears to be impossible except on specific devices.)
- Add a test that renders to two different TextureViews using different EGLContexts from a single renderer thread.