Freedreno on Android

Lucas Fryzek

Oct 19th, 2023

XDC 2023, A Coruña

Introduction

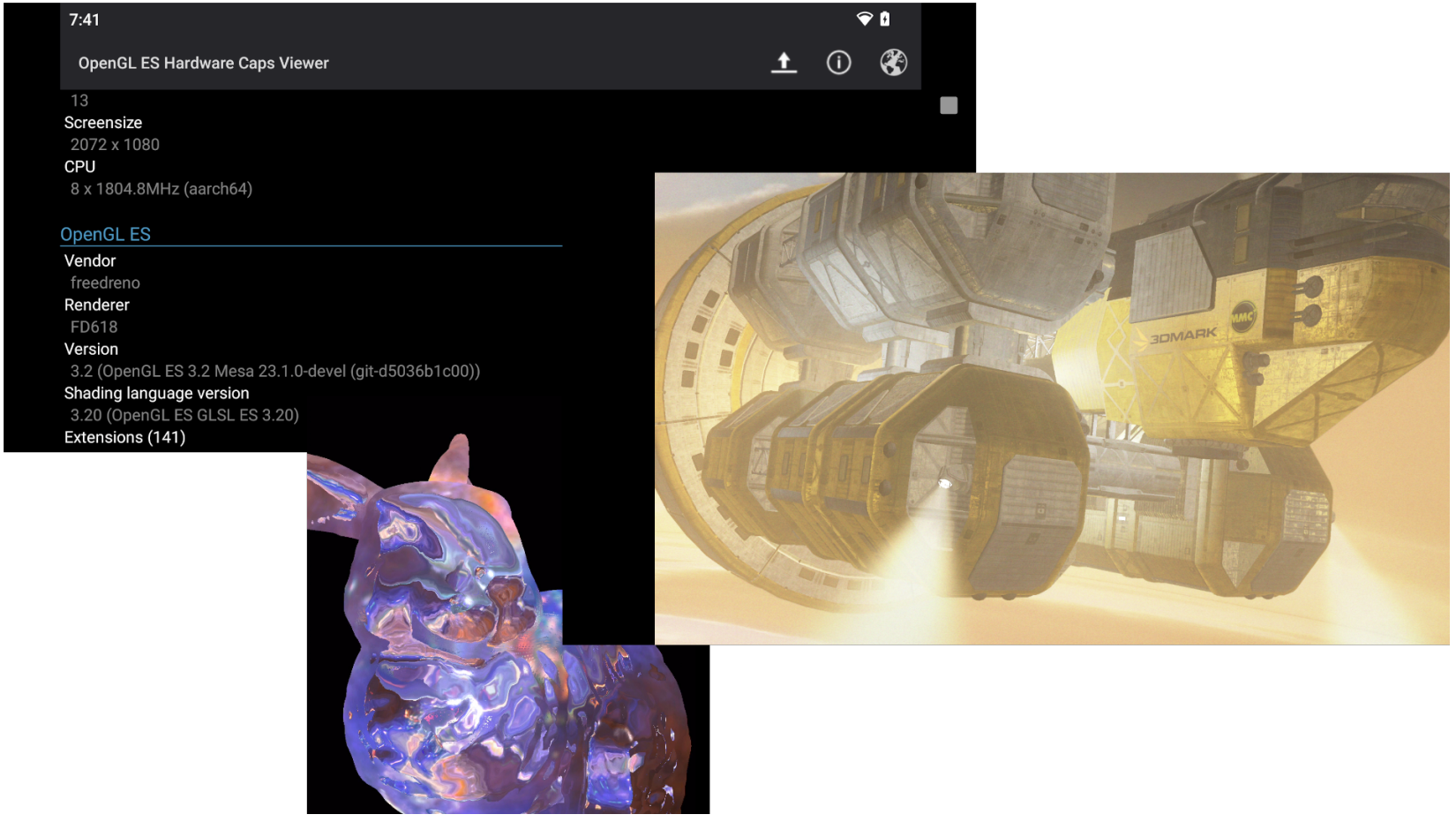
- State of drivers with Adreno GPUs
- Why would you want to do this?
- How to build and run Mesa on Android
- Driver changes necessary to run on Android
- Areas of needed improvement for Android support
- Shortcomings of my work

What does it look like

During development:

igalia

After:


7:41
OpenGL ES Hardware Caps Viewer
13
Screensize
2072 x 1080
CPU
8 x 1804.8MHz (aarch64)
OpenGL ES
Vendor
freedreno
Renderer
FD618
Version
3.2 (OpenGL ES 3.2 Mesa 23.1.0-devel (git-d5036b1c00))
Shading language version
3.20 (OpenGL ES GLSL ES 3.20)
Extensions (141)
3DMARK

A tale of two drivers

- MSM
  - Your friendly neighborhood DRM-compliant upstream kernel-mode driver.
- **KGSL**
  - Qualcomm's kernel mode driver supported by their proprietary userspace driver

Why KGSL?

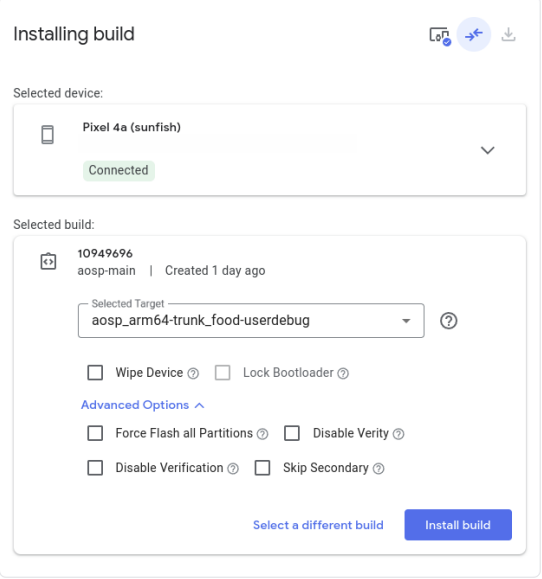
- Not every SOC is upstreamed
- Would like to use an open source driver without requiring lots of merging upstream & downstream code
- Provides the ability to run Qualcomm's proprietary userspace driver and Mesa on the same device with the same KMD
  - Run different driver in a chroot from host OS

- Turnip already runs on top of KGSL
  - Basis for all my work on Freedreno & EGL
- Freedreno developers have already started abstracting KMD interface in `src/freedreno/drm`

Getting Started

- Work done Pixel 4a
  - Adreno 618 GPU - Already supported by Freedreno
- NDK version 25 and version 13
  - Will talk about why two versions later
- Debug rom from **https://flash.android.com**
  - Perk of using google supported devices

igalia



- Make sure you use a "userdebug" rom
- If device is not supported by flashing tool you'll need to build a ROM from scratch with system partition unlocked

igalia

How to build Mesa on Android?

- If we are targeting Android NDK (I'll get back to this later):
  - You can make use of Meson cross files to build Mesa
  - Example of how to do that here
    - **https://docs.mesa3d.org/android.html**

igalia

Important meson flags for freedreno

- `-Dplatforms=android`
- `-Dplatform-sdk-version=25`
- `-Dandroid-stub=true`
- `-Dgallium-drivers=freedreno`
- `-Dfreedreno_kmds=kgsl`

How do you even deploy system libraries on an OS such as Android?

- Existing Mesa documentation for Turnip talks about replacing libraries in `/vendor/lib64/`, likely need to do the same thing for OpenGL
- Also need to unlock system partition

```
adb disable-verity
adb reboot
adb remount -R
```

- Thankfully Android is open source, we can read **Android's EGL loader** source code
- Replace following libraries in `/vendor/lib64/egl`
  - `libEGL_adreno.so`
  - `libGLESv1_CM_adreno.so`
  - `libGLESv2.so`

igalia

- Trying to run GL apps now, we run into another problem
  - Apps don't use new driver
- If we restart the device it will start using the new driver
  - Not the best dev environment
- Reading **Android documentation** there is config property to preload GL driver `ro.zygote.disable_gl_preload`
  - Need to override the default and set it to true

- Environment variables need to be set to force the mesa loader to work properly on Android
  - Android has a "prop" system that mesa already abstracts for environment variable access
  - `adb setprop "mesa.loader.driver.override" "kgsl"`

Testing

- You can run regular Android APKs to test the driver
  - CTS can be built as an APK
  - Not the most convenient development environment

You can actually run command line apps on Android!

- **Freedreno reverse engineering tools repo** has a build environment for this
  - Build OpenGL apps using offscreen EGL Pbuffers
  - Can run the apps from `adb shell`

- Repo has build scripts setup already
- It expects NDK version 13
- NDK version 13 has some other goodies that are useful for debugging applications

- NDK 13 was the last shipped version with gdbserver
- Can copy binary from NDK to device
- Makes debugging a lot easier
  - Can preform debugging on `adb shell` launched apps
  - Can just use Android studio debugger to debug NDK code in APKs

We can take it further!

- With a few tweaks CTS Android platform can be built as a standalone program just like on Linux
- deqp-runner can even be built through **cargo-ndk**

igalia

Source code changes

- Only about ~900 lines of code to add support
- Majority of changes in new `kgsl` backend in `src/freedreno/drm`
- There are also significant changes in `egl/drivers/dri2/platform_android.c`

`kgsl` backend

- Handles BO allocation and mapping
- Querying properties from kernel mode driver
- Submitting command queues to hardware
- Handling synchronization

- Backend code mostly came from Turnip
- Lots of copy pasting
- Could have more common code added to `src/freedreno` for both drivers

Interesting quirks

- No referencing count on buffers
  - Need to ensure buffers are done being used by GPU before freeing
- KGSL ignores offset argument in `kgsl_command_object`
  - Need to ensure that the GPU address contains the offset

Some backend changes necessary to accommodate KGSL quirks:

- Framebuffers allocated by Android need to mapped in a different way
  - Added per backend implementation of mapping function
  - Added per backend implementation of importing dmabufs

`platform_android.c` changes were not as nice....

igalia

Dealing with gralloc

- Graphics allocator on Android
- Framebuffers for APK apps are allocated by gralloc and passed to us
- Gralloc implementation is driver specific
- Turnip already has code to interface with Qualcomm's version of gralloc

Turnip's code...

```
uint32_t gmsm = ('g' << 24) | ('m' << 16) | ('s' << 8) | 'm';
if (handle_data[0] != gmsm) {
 return vk_errorf(device, VK_ERROR_INVALID_EXTERNAL_HANDLE,
                  "private_handle_t::magic is %x, expected %x",
                  handle_data[0], gmsm);
}

ubwc = handle_data[1] & 0x08000000;

*dma_buf = handle_fds[0];
```


igalia

- No external API to access internal data
- Data is interpreted based on **gralloc implementation source code**
- But it works!

igalia

- Implemented same interface in `egl/drivers/dri2/platform_android.c`
- Existing implementation that work with upstream DRM drivers

- Other Mesa android devs have been working on a newer "Gralloc 4" interface in `platform_android_mapper.cpp`
  - Uses newer standard API to interface with allocator
  - Cannot be used with Android NDK
    - c++ namespace between NDK and android tree build are different
  - Building Mesa in tree with Android is **likely to be deprecated**
    - This is the alternative way to currently build Mesa without the NDK

Bugs encountered

- Main issue is with surface allocated by gralloc for the framebuffer
  - Does not necessarily allocate surfacing matching hardware limitations
  - For example, the blitting engine on A6xx GPUs performs copies on 16x4 pixel chunks
  - Causing IOMMU faults when the GPU accesses memory beyond the framebuffer

- Biggest issue in preventing KGSL changes from being merged
- Problem triggered by Android UI elements
  - Android UI will flicker whenever GPU iommu fault is triggered
  - Normal APKs seem to always allocate framebuffer equal to display size
- Qualcomm blob driver is getting the same surfaces but somehow avoids this issue
- This is where things got kind of stuck...

Notes on Freedreno RE tools

- Freedreno has a diverse set of tools for inspecting what the Qualcomm driver and Freedreno driver are doing
- Most of these tools are designed to work apps launched from command line
  - Doesn't help a lot of problem only happens with Android APKs

One key tool is libwrap

- Library that allows you to trace command streams on Qualcomm hardware
  - Works with both Freedreno and Qualcomm propeitary driver
- Uses `LD_PRELOAD` to load library and override system functions

How do you run this with APKs?

- Made some changes to libwrap
- How can you `LD_PRELOAD` on Android?
  - use prop `wrap.<app-name>` to override environment variables in APK processes
- Fix issues associated with tracing Android APKs
  - APKs had multiple threads accessing the `kgsl` FD

Conclusions

- A lot of code in mesa already exists to make running on android easy
- Proper DRM drivers will likely just work
- If you want to use a downstream kernel mode driver (and gralloc implementation) some more work is necessary

- Development is faster when you do more of your work from `adb shell`
  - Setting up a good development environment pays dividends

- One of the biggest problem area in Mesa's android support is window system integration
  - Gralloc appears to be the standard for this in Android
  - Not clear how newer versions of the API can be used in Mesa
  - Not easy to get information/documentation on Gralloc without digging through source code
  - `platform_android.cpp` currently needs to be hacked to work with non-drm drivers

Questions?

We're hiring!

igalia.com/jobs/

igalia

igalia