Open Source OpenGL on the Raspbery Pi

Eric Anholt
Broadcom

Outline

- Raspberry Pi architecture
- Previous SW architecture
- New SW architecture
- Raspberry Pi challenges

Raspberry Pi HW architecture

- ARM CPU (700Mhz ARMv6)
- VPU
  - loads a small OS from SD card, executes it to run code to turn on the ARM and send it into its bootloader
- QPU
  - GLES2 3D engine
  - Tiled renderer

Raspberry Pi SW architecture

- VPU GLES2 driver side
  - custom vendor driver
  - closed source
  - Generates shaders and command stream for the QPU
- ARM GLES2 driver side
  - ships GL command stream to VPU
  - 3-clause BSD code dump
  - Not useful to open source developers

How I got here

- Intel graphics developer for 8 years.
- Looking for a chance to fix Android graphics.
  - I wish my phone would stop crashing
- Broadcom released specs and source February 28, 2014
  - VPU driver stack ported to ARM for a cell phone chip
  - 3-clause BSD license
  - Android-only
  - No GLX or non-Android EGL components.
- Joined Broadcom June 16

A new driver project

- Free software Mesa driver (MIT licensed) running on the ARM
  - OpenGL, GLESv2 support
  - GLX, EGL support.
- Free software DRM kernel driver
  - Target upstream merging
- xf86-video-modesetting 2D driver for X11

Development under simulation

- simpenrose is the closed source HW simulator
- small C library with about 4 entrypoints
- Built an “i965” driver on my x86 system.
  - Allocate GEM buffers from i915 kernel
  - Talk DRI3 to the native 2D driver
  - generate vc4 code, execute in simulator, copy result to GEM buffers.
- I can print registers!
- I can gdb when the “GPU” crashes!
- I can valgrind!
- (I can sometimes forget to test on the real hardware before pushing code)

Simple hardware makes it easy

- 110 page hardware spec
  - compared to 1727 for the first hardware I worked on at Intel
- 9 state packets for GL state
- 6 state packets for GL draw call setup
- 8 state packets for binner setup
- demo code from Scott Mansell in 340 lines

Simple hardware means more work

- Many areas of OpenGL handled in shaders
  - vertex fetch format conversion
  - user clipping
  - shadow mapping
  - blending
  - logic ops
  - color masking
  - point sprites
  - alpha test
  - two-sided color
  - texture rectangles
  - some wrap modes

Desktop OpenGL on a GLES2 part

- GL_QUADS turn into GL_TRIANGLES with an index buffer.
- 32-bit index buffers trimmed to 16 bit
- Turn GL_CLAMP to clamping texture coordinates to [0,1]
- 0 occlusion query counter bits
- shadow map texturing
- Not done:
  - Polygon/line stipple
  - Polygon fill modes / edge flag
  - 3D textures
  - derivatives in shaders
  - LOD clamping

Funny QPU architecture

- Each instruction contains 2 operations
  - 1 ADD, 1 MUL
- Each operation has 2 arguments
- Only 1 address into each register file (A/B) available
  - except arbitrary access to accumulators r0-r3
- No ability to spill registers

```
fadd ra0, r3, ra0 ; fmul r3, rb2, rb2
```

Register allocation solution

- Standard (Runeson/Nyström) graph-coloring register allocator
  - register file A/A and B/B conflicts handled by reserving one register each from A and B and spilling into them
  - Most nodes in the normal register class, unpacks (pick 8 bit unorm channel, expand to float) are an A-only register class
- Generate stream of single-operation instructions
- Instruction scheduler attempts to pair up operations
  - Converts ADD-based MOVs into MUL-based MOVs to fit
  - Convert some regfile A references into regfile B references

Register allocation future plans

- Extend the current allocator to give the driver a chance to choose a preferred register during Select.
- Try a pre-pass splitting registers into A or B with MOVs in between, then try register coalescing during allocation?
- Try a bottom up, linear scan allocator.
- Possibly an entirely different SSA allocator`

SSA?

- GLSL IR->TGSI->QIR->QPU is the current compiler architecture.
- QIR is SSA, with no control flow
- Need control flow for ES conformance
  - GLSL IR loop unroller is not so hot
- NIR landed this morning
- GLSL IR->TGSI->NIR->TGSI->QIR->QPU works
- GLSL IR->TGSI->NIR->QIR->QPU is almost working
- Pie-in-the-sky future of GLSL IR->NIR->QIR->QPU.

No MMU under the GPU

- GPU has direct access to system memory
- Requires contiguous memory allocations
  - CMA support in the kernel helps a lot
- Huge security hole
  - Ask the vertex fetcher to fetch arbitrary memory
  - Ask the texture unit to fetch arbitrary memory
  - Ask the tile buffer to store to arbitrary memory!

MMU solution

- Not handled in the closed stack
- vc4 DRM driver does validation
  - Parse shaders, decide which uniforms read from textures
    - Make sure read addresses are clamped!
  - Parse uniforms, make sure they reference valid textures
  - Parse command stream
    - decide whether vertex reads are from valid memory
    - decide whether the tile buffer is loaded/stored to valid memory
- Costs about 5% of ARM CPU time
- Scariest code I've ever written

Other kernel execution details

- drm_gem_cma_helper.c based BO allocation
  - thin VC4 wrapper around them to track the BO's presence in the GPU command queue and in the BO cache
- in-kernel BO cache
  - binner needs arbitrary amounts of memory at runtime, triggered by GPU interrupts
- 3 ioctls
  - SUBMIT_CL
  - WAIT_SEQNO
  - WAIT_BO
  - (oh wait, and CREATE_DUMB and MAP_DUMB)

Kernel details: KMS

- Currently abusing the VPU firmware's modesetting for bringup
  - Ask it to set up a framebuffer for us with 1680x1050
  - Smash the HVS display list to scan out of our GEM BO instead
    - Oh, and assume ARGB8888 and untiled
- Need something better

KMS plans

- Most hardware has a few scanout planes (display, overlay, cursor)
- VC4 has the HVS display list
  - series of rect, format, address
  - At each scanline, hardware reads the list, finds intersection with rects, reads lines from src, blends/replaces as appropriate
  - Number of planes limited only by memory bandwidth and number of rects that can be stored
- Expose this as a steaming pile of KMS planes, and atomic modeset that sometimes says “no.”

X11 plans

- With Present, X now asks the driver to set a CRTC's scanout at a specific vblank.
- What if X instead asked to set a CRTC's scanout to a set of planes?
- driver could ask KMS to set the planes, and if KMS says “no”, X could manually composite some of the planes down
- Initially fallback using GL, but the HVS has some magic
- X could implement any CopyArea to the screen as an overlay
    - Well, unless other userspace might have another reference to that buffer.

Merging kernel upstream

- Raspberry Pi maintains a vendor kernel tree
  - non-devicetree-based
  - 3.16 in rasbpian
  - Huge squash commits of rebased code
- My tree is based on a Raspberry Pi tree
  - kernel 3.15
  - couple of hacks to core DRM
  - 59 other commits to build up the driver
- Upstream has limited support for the 2835.
  - USB (for networking) support may now be landing
  - No mailbox to the VPU
  - No CPU clock control
  - No sound

Kernel upstreaming blockers

- Need bootable upstream RPi kernel
- Need mailbox driver for upstream RPi kernel
- Need to fix critical vc4 ABI issues
  - Introduce our own create/map ioctls (Hi Dave!)
  - New single-GEM handle CL packet?
  - Avoid GEM handle CL packets in some other packet typess?
  - Redo relocations entirely?
- Need review on shader validation

Status

- 14530 lines 3D driver code
- 4971 lines kernel code
  - 1/3 is shader/command stream validation!
- 98.7% passrate on ES2 conformance tests (simulation)
- 92.5% passrate on piglit GPU tests (simulation)
- Hacked-up KMS works on my monitor, but not yours

Links

- TODO list and build instructions for free software driver:
  - http://dri.freedesktop.org/wiki/VC4/
- Hardware specification:
  - http://www.broadcom.com/docs/support/videocore/VideoCoreIV-AG100-R.pdf
- Broadcom sample implementation:
  - https://github.com/simonjhall/challenge