Building X 2D rendering acceleration with OpenGL

Eric Anholt
Intel Open Source Technology Center

How 2D has worked

- X has always implemented graphics acceleration in a hardware specific driver
- Acceleration architectures (XAA, EXA, UXA, SNA) provided helpers for translating X primitives to GPU operations.
- Every new chipset, every year, you get to write new 2D code

GL

- Used to be an optional thing on the side.
- You hoped it worked, sometimes you got lucky
- Used to be totally private memory management

What changed?

- GEM, KMS, and DMABUF gave us interfaces for memory management across processes.
- Wayland development led to GBM for interfacing between Linux's KMS and EGL/GL, which is also used on Android and Mir.
  - the “generic/graphics buffer manager”.
  - totally ad-hoc API, no specification, but it works.

Do we need device-specific X 2D?

- Intel, AMD, and Nvidia have open source GL drivers for all currently released chips.
- Rob Clark presented about progress on embedded chips at XDC: basically, there's a project for every GPU, and they're all building GL acceleration first.

Introducing glamor

- Started development in 2008 using Xephyr as a testbed.
  - Got Xephyr (X on top of X) basically functional.
  - Got stuck trying to figure out how to make DRI2 work.
- Picked up by Zhigang Gong in 2011
  - moved to external tree
  - DRI2 support added.
  - large pixmap support
  - Performance fixes.

Glamor uptake

- Only 2D acceleration supported on AMD as of Southern Islands chipsets. (2012)
- ./configure option on Intel, driven by Intel UXA code.
- Not yet enabled for nouveau.

Hard to hack on X

- Writing cross-API GL code is hard
- Testing X rendering code is hard

Cross-API GL is hard

- GL development occurs through extensions
  - One or more vendors write a spec, implement it.
  - Vendor specs get functions and enumerants under a vendor-specific namespace.
- GLES removes functionality from desktop GL to build its spec, vendor extensions re-add it.
- GL doesn't rely on dlsym(), and instead each window system defines its own dlsym()-like API and what subset of functions should be accessed by it.

libepoxy: hiding GL API badness

- Single GL ABI for all of OpenGL 1.2-4.4, GLES 1, 2-3, EGL, WGL, GLX.
- Built from Khronos's gl.xml, egl.xml, glx.xml
- Uses function pointers with dynamic resolution.
- Drop in to your application in place of:
  - `#include GL/gl.h`
  - `-lGL`
- https://github.com/anholt/libepoxy

Testing X rendering is hard

- Ask any X developer: “Just run XTS5”
- Actual testing: Span some terminals, drag windows around in metacity, ship it.
- XTS5:
  - Test suite from 1981
  - Build system improved by Dan Nicholson, Peter Hutterer, Peter Harris, and others.
  - Still impossible to run.

Glamor problems

- GLES2 support.
  - Have to use arbitrarily different functions to get the same job done.
    - Wrote libepoxy to hide this GL function pointer management from glamor (and other apps).
    - https://github.com/anholt/libepoxy (please use it in your GL applications!).
- Core GL support.
  - Need to use GL vertex array objects.
- Performance opportunities missed.
  - GL_ARB_vertex_attrib_binding.
  - Fast-path shaders for not needing GLES2 workarounds.
  - GL_ARB_texture_view for Render extension Picture formats.
  - Integer textures for fb operations.
- Xorg DDX dependency.

Can we do better?

- Each X 2D driver has a copy of drmmode_display.c
- Each X 2D driver should have a copy of glamor initialization
  - take fd from DRM, pass it to glamor's set-up-EGL-from-DRM-fd, done.
- Each X 2D driver should have a copy of DRI2/Present support.
  - Get handle from pixmap, wrap it in an FB, tell the kernel to pageflip to the FB, get events for vblanks and pageflip completes and pass them back.
- Each X 2D driver should have a copy of DRI3 support
  - turn a DMABUF fd into an X pixmap, or get the DMABUF fd for an X pixmap.
  - Take your current DRM fd, and make another one like it that's authenticated, and pass it over the wire to the client.
- Each X 2D driver also has reams of XF86 initialization because it's a super crufty API.

xf86-video-modesetting

- Generic 2D driver built on KMS.
  - Initially written by Tungsten Graphics.
  - Used as fallback driver for KMS-supported chipsets without a native 2D driver.
  - Opens DRM device nodes on the system until it finds one matching the PCI ID it's supposed to probe
  - Attaches to it, gets KMS output configuration information, tells X about it.
  - Same drmmode_display.c as everyone else.

Hacking up xf86-video-modesetting

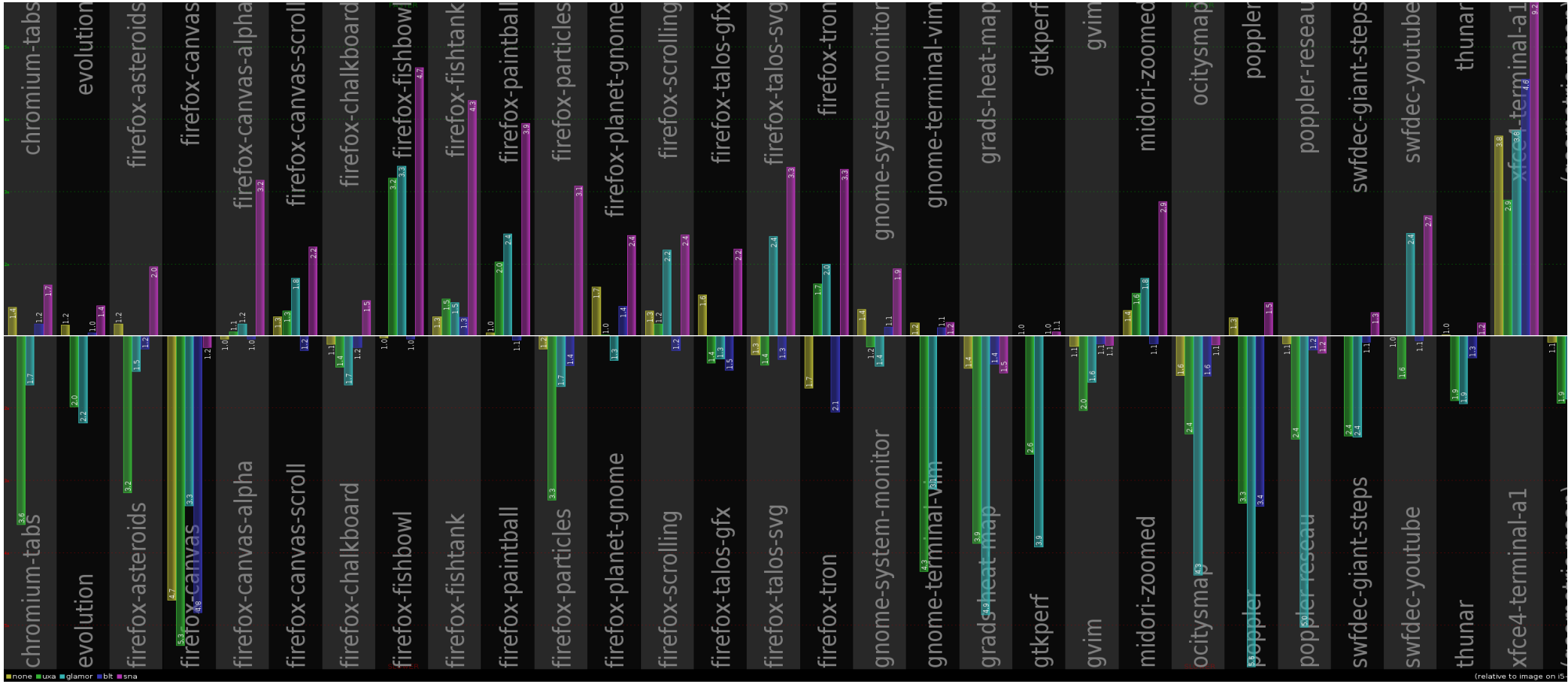
- Pass the FD from opening the KMS driver to glamor_egl to set up 2D acceleration.
- Make glamor_egl add support for DRI3 using that fd.
- Add DRI2 support so that AIGLX can initialize
  - One of two pieces of device-specific code: A table of PCI IDs to tell the client (AIGLX, Mesa) what filename to dlopen() to get a 3D driver for the device.
    - What a ridiculous layering violation. DRI3 and EGL know how to load drivers on their own. Let's fix that in Mesa's DRI2 and AIGLX.
- Add Present support so that DRI3 vblank synchronization works.
  - Second piece of device specific code: there is a second type of CRTC identifier on Intel for the kernel's pageflip API, and you need a device-specific ioctl to get it from your CRTC.

Can GL perform like native?

- 3D games do fine.
- 2D pushes CPU overhead harder than 3D games do.

glamor vs uxa (vs more) on Intel

- baseline is CPU, green is old UXA code, teal is glamor.



Performance results

- glamor is faster than a limited couple-of-months-per-chipset 2D acceleration.
- glamor is not as fast as a year-round 2D acceleration tuning effort.
- glamor is still demo code, though.

glamor performance projects

- Use GL_ARB_buffer_storage to avoid buffer mapping overhead.
- Use GL_ARB_vertex_array_object and GL_ARB_vertex_attrib_binding to avoid vertex attribute overhead.
- Fast-path shaders for skipping repeat workaroudns.
- Native trapezoid shaders using GLSL 1.30.
- Lots of fb codepaths to rewrite in GL.

core text rendering in glamor

- 36.5k glyphs/sec
  - Make a temporary bitmap of each character, then read that bitmap back out, compute RLE spans of pixels to draw, request GL to draw those spans one at a time
- 193k glyphs/sec:
  - Make a temporary bitmap of each character, read that bimap back out, compute RLE spans of pixels to draw, request GL to draw all those spans all at once.
- 290k glyphs/sec:
  - Make a temporary bitmap of each character, read that bitmap back out, request GL to draw each pixel present in the bitmap.
- Still can do better.

Other glamor projects

- Finish DRI2/DRI3 pageflipping
- Fix DRI3 API for PRIME support.
- Add overlay XV support.
- de-xf86 XV and EGL code.
- Finish GLES2 port.
- Automatic pageflipping no-tearing X compositor.
-