Zink: OpenGL on Vulkan

Erik Faye-Lund, Collabora
July 2019
What is Zink?
What is Mesa/Gallium?

- Mesa is an Open Source implementation of several graphics-related APIs
- Gallium is a hardware-abstraction layer in Mesa
  - State-tracker / driver architecture
- Mesa has a Gallium state-tracker for OpenGL
  - This is our primary target
- But other APIs exist as well:
  - DRI, GLX, WGL
  - OpenCL
  - Direct3D 9, OpenMX, VDPAU, XVMC
- Mesa contains a GLSL compiler with an SSA IR (NIR)
What is Zink?

- Zink is a Gallium driver in Mesa, that output Vulkan commands
  - Includes a NIR to SPIR-V compiler
  - This means we can get a full OpenGL to Vulkan translation layer
- It’s in early “working prototype”-state
  - Currently expose OpenGL 2.1 on top of Intel’s ANV Vulkan driver
    - Experimental patches exist to support RADV for AMD GPUs
    - Experimental patches exist to expose OpenGL 3.0 support
    - Not yet ready for prime-time use
- Currently out-of-tree in Mesa
Why?
Original by Randall Munroe: https://xkcd.com/927/
But seriously, why?
Why OpenGL on Vulkan?

1. Vulkan is the *future* of open graphics standards
2. OpenGL is a *requirement* for desktop
3. Enable more *use-cases*
Vulkan: the future

- Vulkan has been around for a bit more than 3 years
  - It has proven itself by now
  - Vulkan 1.1 released last year
  - Moving fast
- OpenGL is 27 years old by now
  - Latest OpenGL specification (4.6) was released in 2016
    - Mostly covers Vulkan interopability
  - No sign of OpenGL 4.7 yet
  - Not dead, but not exactly young and vigorous any longer
- Vulkan is likely to be the leading “high-end” API going forward
OpenGL: a requirement for desktop

- OpenGL is *very widely adopted*
  - We can’t just ignore OpenGL, and move forward to a Vulkan-only future
    - Some applications have large OpenGL-dependent code-bases that would take a *significant effort* to port over
    - Some applications *don’t even need* the CPU overhead improvements of Vulkan
  - *Already shipped* applications needs to keep working
    - If the developer pivoted or dissapeared, porting might *not even be an option*
- It’s better for the community if we can focus on one API
- We need to *keep supporting OpenGL* somehow
More use-cases

- Better GPU virtualization
  - Virtualizing both OpenGL and Vulkan is twice the amount of work
    - This code typically runs at a very high privilege level
- Full OpenGL everywhere
  - Many new systems doesn’t support OpenGL, or only supports OpenGL ES
    - Imagine Autodesk Maya on Android?
- Enable hybrid applications
  - Why not use both OpenGL and Vulkan?
Why use Mesa

- See XKCD slide 😊
  - Implementing a complete desktop OpenGL implementation is a huge amount of work.
  - Mesa has done a lot of the hard work for us, for example:
    - Fixed function pipeline emulation
    - GLSL compiler
- Pre-existing solutions only solves parts of the problem
  - GLOVE / ANGLE only do OpenGL ES variants
  - Enough for getting rid of OpenGL drivers on mobile
  - VKGL only does 3.2 Core Profile
What works?
OpenGL 2.1

- Most OpenGL 2.1 functionality seems to work
- Things that *don’t work* (yet):
  - `glPolygonMode` with differing front and back states
  - Arbitrary texture-border colors
  - Flat shading
  - Edge flags
OpenGL 3.0

- Some experimental patches for OpenGL 3.0 support exist
  - Requires:
    - VK_EXT_transform_feedback
    - VK_EXT_conditional_rendering
- Challenges:
  - Arbitrary primitive-restart index
    - Solution: Rewrite index buffer?
OpenGL 3.1-3.3

- Might require `VK_EXT_vertex_attribute_divisor`?

- Challenges:
  - Geometry shaders
    - Will create some trouble for our planned provoking vertex fix
  - UBO support
    - Might require rethinking uniforms → UBO lowering
    - Perhaps use push-constants for regular uniforms instead
OpenGL 4.0-4.6

- Might require `VK_KHR_sampler_mirror_clamp_to_edge`
  - Can also emulate in shader, but that’s not ideal

- Challenges:
  - Too many to list here!
  - But nothing seems to be a real show-stopper
    - Let me know if you know of something!
Demo time!
Emulating features
Provoking vertex

- OpenGL support first or last vertex as the provoking vertex, Vulkan only support first
- Currently just ignore this and always use first
  - This cause rendering issues with flat-shading
- Emulation options:
  - Reorder index buffer
    - Needs extra work for \texttt{gl\_VertexID}
    - Won't work with geometry shaders
  - Inject geometry-shader code that rotates the vertex order to match
    - Might have challenges when combined with stream-out
Polygon mode

- OpenGL allows different polygon mode for front and back-faces, Vulkan does not
  - We currently print a warning and use the front-face state for both
- Emulation options:
  - Draw all back-faces, then all front faces?
    - Doesn’t give the right draw-order, but it’s cheap and better than nothing
  - Write primitives out to a buffer, and create triangles out of lines and points?
    - Might need to always do this to avoid rasterization-variance...
- Is this even tested by conformance tests?¯\_(ツ)_/¯
Texture border-colors

- OpenGL allows arbitrary texture-border colors, Vulkan supports either:
  - Transparent black
  - Opaque black
  - Opaque white
- Currently hard-coded as transparent black
- Emulation option: Inject shader-code
  - Not as bad as it sounds
  - Prototype: https://www.shadertoy.com/view/XtdfW8
- This *might* warrant a Vulkan extension for exposing support on existing hardware
The future
Reliance on extensions

- Will e.g. VK_EXT_transform_feedback be available everywhere forever?
  - My guess: No.
  - Probably already not supported on most mobile GPUs...
- We need alternative fall-back plans for these in the long term
  - Perhaps we can write to an SSBO with the vertex-ID instead?
    - Might be tricky when combined with geometry or tessellation shaders
  - Emulate non-fragment stages using Compute in the longer term?
  - There’s similar concerns for other extensions
Get involved!
GitLab!

- Project currently lives in the freedesktop.org GitLab instance under my user:
  - https://gitlab.freedesktop.org/kusma/mesa/
- Code:
  - Current version: https://gitlab.freedesktop.org/kusma/mesa/tree/zink
  - Older + GL3 patches: https://gitlab.freedesktop.org/kusma/mesa/tree/zink-old-gl3
- Issue board: https://gitlab.freedesktop.org/kusma/mesa/--/boards
- Wiki: https://gitlab.freedesktop.org/kusma/mesa/wikis/zink
- All of this will move once we upstream
Contact

- Right now most communication happens through the GitLab project
- But you can also e-mail me directly at kusmabite@gmail.com
  or erik.faye-lund@collabora.com
- Alternatively, I’m on IRC, with the username kusma in #dri-devel
  at FreeNode
- ...or you can ping me on Twitter: @kusmabite
Acknowledgments

- I didn’t do this work alone
  - Dave Airlie of Red Hat contributed a lot of great features, including the current window-system integration code and most of the GL3 patches
  - Jason Ekstrand of Intel helped me with a lot of compiler questions
  - As well as many others from the Mesa 3D project
- Also, big thanks to Collabora for sponsoring this work!
- Thanks to Khronos for letting me do this talk!
Thank you!

https://gitlab.freedesktop.org/kusma/mesa/
erik.faye-lund@collabora.com