Zink: OpenGL on Vulkan

Simplifying the future of the graphics stack?

Erik Faye-Lund
Why OpenGL on Vulkan

• OpenGL is a requirement for desktop
  - Some modern use-cases are outside of what OpenGL was designed for

• Vulkan is here to stay
  - Likely to be the leading “high-end” API going forward

• Requiring two implementations through all components of the stack is a massive support-burden
  - Can we reduce the requirement to one?
Existing solutions

• GLOVE, ANGLE, VKGL
  - Only implements OpenGL ES 2/3 or GL Core 3.2
  - Adding Full OpenGL support is a big undertaking
    • Legacy GL and compatibility contexts

• GLO (G-Truc)
  - Vapourware

• Some other non-public solutions exist
  - Impossible to reason about those
Zink: Gallium to Vulkan

- Translates Gallium API calls to Vulkan
  - A very rough proof-of-concept exist
    - Can render glxgears, some of glmark2 and maybe some other basic things
    - Currently runs in lock-step with the CPU

- Written by me as a side-project in a couple of week
  - A result of some architectural issues with Virgil 3D

- Not in any way a proven idea
  - More work is needed, for sure.
Challenges

- NIR → SPIR-V
- WSI
- Pipeline caching
- Transform feedback
- Image layout management
Demo time!

Get the code at:

https://gitlab.freedesktop.org/kusma/mesa/tree/zink
The future

• Lots of work to be done, most importantly:
  - Pipeline caching
  - Making the compiler less terrible

• There’s a good chance we’re going forward with this approach
  - After that, implementing more modern OpenGL features
Help wanted!

There’s a lot of things to do, so please help out if you’re interested!

- Interesting compiler work
  - “type system”: Add support bool, int, int64, double types?
  - Conditional code and loops
  - More texturing instructions

- Improve the execution model
  - How do we deal with image barriers in an efficient way?

- Lots more that I can’t fit here

- Patches welcome!
Thank you!