How to write a Vulkan driver in 2022

Faith Ekstrand
XDC 2022

About me

- Jason Ekstrand (jekstrand)
- First freedesktop.org commit: wayland/31511d0e, Jan 11, 2013
- Worked at Intel from June 2014 to December 2022
  - NIR, Intel (ANV) Vulkan driver, SPIR-V → NIR, ISL, other Intel bits
- Started at Collabora in January 2022
  - Work across the upstream Linux graphics stack, wherever needed
  - So far, mostly focused on Vulkan runtime code

COLLABORA

History of Vulkan in Mesa

History of Vulkan in Mesa

- Intel (ANV) Vulkan driver merged on April 15, 2016
  - Refactored Intel OpenGL driver code for sharing w/ Vulkan
    - Moved into a new src/intel folder
  - Added a new SPIR-V front-end for NIR
- RADV was merged on October 7, 2016
  - Started as a **copy+paste** from ANV
- Other vulkan drivers either derive from ANV or RADV

History of Vulkan runtime in Mesa

- First significant common code was WSI
  - Shared between ANV and RADV
- Common base object for VK_EXT_private_data, May 2020
- Common entrypoint table generator, February 2021
- Common render pass implementation, March 2022
- Common graphics state tracking, July 2022
- Common Vulkan meta (copy/blit/clear), Coming soon!

COLLABORA

Writing a Vulkan driver in 2022

Directory structure

```
src/<hardware>/:
 |- meson.build
 |- compiler
 |    |- meson.build
 |    |  ...
 |- vulkan:
 |    |- meson.build
 |    |- drv_private.h
 |    |- drv_device.c
 |    |  ...
 |   ...
```

Common base objects

- Every Vulkan object should derive from `vk_object_base`
- You can also derive from one of the other `vk_foo` base structs
  - `vk_device`, `vk_image`, `vk_queue`, etc.
- Use `VK_DEFINE_HANDLE_CASTS()` to declare handle cast helpers
- `VK_EXT_private_data` is implemented for you

Common dispatch framework

- Driver implementations of core objects derive from vk_foo
  - vk_instance, vk_physical_device, vk_device
- Everything else drives from vk_base_object or other vk_foo
- Use vk_entrypoints_gen.py to generate driver-prefixed tables
- vkGet*ProcAddr() are implemented in common code:
  - vk_instance_get_proc_addr()
  - vk_common_GetDeviceProcAddr()

Common vkFoo2() wrappers

- If both vkFoo() and vkFoo2() exist, only implement vkFoo2()
- Common code implements vkFoo() in terms of vkFoo2()

```
VKAPI_ATTR VkResult VKAPI_CALL
vk_common_BindImageMemory(VkDevice _device,
                          VkImage image,
                          VkDeviceMemory memory,
                          VkDeviceSize memoryOffset)
{
   VK_FROM_HANDLE(vk_device, device, _device);

   VkBindImageMemoryInfo bind = {
      .sType        = VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_INFO,
      .image        = image,
      .memory       = memory,
      .memoryOffset = memoryOffset,
   };

   return device->dispatch_table.BindImageMemory2(_device, 1, &bind);
}
```

Common vkFoo2() wrappers

- If both `vkFoo()` and `vkFoo2()` exist, only implement `vkFoo2()`
- Common code implements `vkFoo()` in terms of `vkFoo2()`
- You don't need implement VK_EXT_foo2 first
  - The framework doesn't care if the extension is enabled or even supported
- This includes VK_KHR_synchronization2!

Logging

- Common logging helpers

```
vk_logd(VK_LOG_OBJS(device), "vkDeviceWaitIdle() took %u us", wait_time);
```

- Take a list of objects or instance as the first parameter
- Reports log messages via
  - stderr
  - VK_KHR_debug_utils
  - VK_EXT_debug_report

Error reporting

- Generic error reporting
  - `return vk_errorf(obj, VK_ERROR_FOO, “Message: %u”, i)`
- Command buffer error recording
  - `return vk_command_buffer_set_error(&cmd_buffer→vk, VK_ERROR_FOO)`
- Device loss reporting
  - `return vk_device_set_lost(device, “Lost device message: %u”, i)`
  - `return vk_queue_set_lost(queue, “Lost queue message: %u”, i)`

Synchronization and queue submit

- Do not implement VkFence or VkSemaphore directly
  - Especially not timeline semaphores!
- Single common vk_sync primitive
  - Supports binary and timeline
  - Supports GPU and CPU waits
  - Supports various import/export
- VkFence or VkSemaphore implemented in terms of vk_sync

Synchronization and queue submit

- Common synchronization requires common queue submit
- Driver implements `vk_queue::driver_submit`
- Automatically spawns a thread when needed
  - To handle cross process submit re-ordering for timeline semaphores
  - To handle CPU work in userspace which needs to block
    - Be careful here! This is incompatible with sync file export
- Also handles `vkDevice/QueueWaitIdle()`

Render passes

- Render passes are now optional for drivers which
  - Support VK_KHR_dynamic_rendering
  - Support the `VkRenderingAttachmentInitialLayoutMESA` pseudo-extension struct
  - Lower input attachments via `nir_lower_input_attachments()`
  - Support VK_EXT_attachment_feedback_loop_layout
- Implement `vkCmdBegin/EndRendering()` and the rest is magic!
- Drivers can still implement render passes directly

Graphics state tracking

- The new `vk_graphics_pipeline_state` struct tracks all state that can be embedded in a graphics pipeline
  - Automatically handles possibly-garbage pointers
    - Everything is either NULL or valid
  - Handles pipeline libraries state accumulation
  - Avoids chasing pointers for dynamic-only state

Graphics state tracking

- The new `vk_dynamic_graphics_state` tracks all dynamic 3D state
- Helpers are provided for
  - Populating from a `vk_graphics_pipeline_state`
  - Copying to another `vk_dynamic_graphics_state`
  - Dirty tracking of dynamic state
- All `vkCmdSet*()` are implemented in common code

Meta ops (copy/blit via shaders)

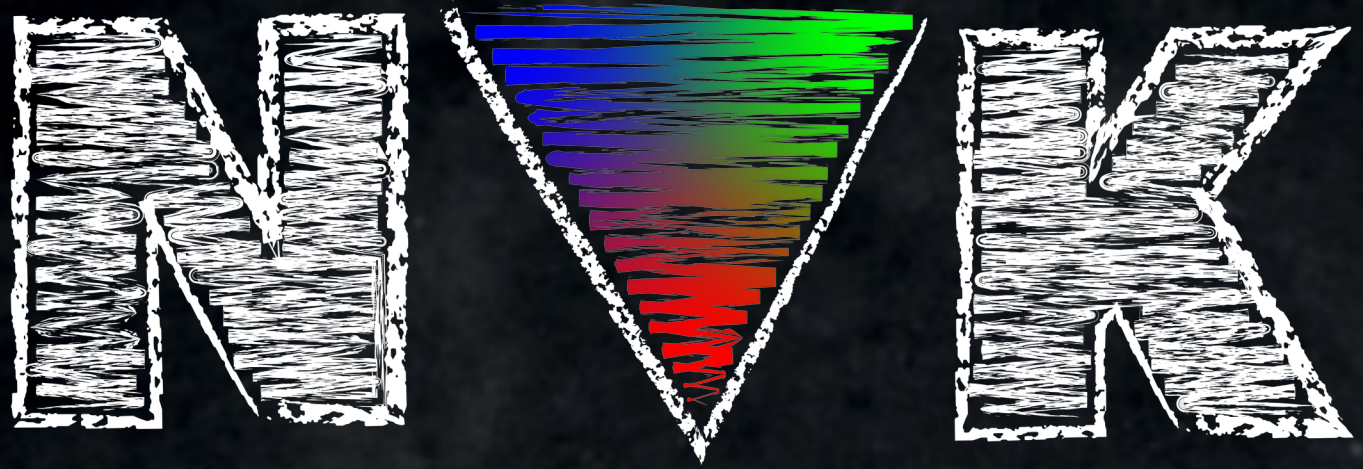
- The goal is to provide helpers for all transfer ops
  - Clears (render pass, image, and attachment)
  - Blits and resolves
  - Copies (image, buffer, buffer image, buffer fill)
- Pipelines and persistent objects created once and cached
- Transient objects (image views, etc.) tracked by the command buffer
- Drivers are responsible for state save/restore

Meta ops (copy/blit via shaders)

- Current status:
  - Clears: (may need some re-shuffling long-term)
  - Blits:
  - Resolves: (probably next, NVK needs them)
  - Copies:
- Currently tested in NVK, scheming with Alyssa to convert panvk
- Hopefully the framework will also work for driver custom meta

COLLABORA

So, I did write a Vulkan driver in 2022…


NVK

What is NVK?

- Brand new Vulkan driver for NVIDIA hardware
  - 100% from scratch (very little copy+paste from nouveau)
  - Uses the newly released official NVIDIA headers
- Written by Jason Ekstrand, Karol Herbst, and Dave Airlie
- Intended to be the new “reference” driver in Mesa
  - Clean well-organized code-base
  - Takes full advantage of runtime code

Status of NVK

- Currently supports Turing+
  - Karol has partial enabling patches for Kelper+, final HW support TBD
- Needs a new kernel uAPI which doesn't exist yet
  - This is going to mean major nouveau.ko surgery
  - Merging to mesa/main blocking on kernel uAPI
- Current CTS pass rate:
  - Pass: 193734, Fail: 1064, Crash: 1286, Warn: 4, Skip: 1364208, Flake: 265

How to test/contribute

- Currently lives in the nvk/main branch in nouveau/mesa
  - https://gitlab.freedesktop.org/nouveau/mesa
- Feel free to submit MRs!
  - https://gitlab.freedesktop.org/nouveau/mesa/-/merge_requests
- Please be kind with the issue tracker
  - No features requests yet, we know there's a lot missing
  - No game bugs yet, there are a lot of missing features

More details coming in a blog post

https://www.collabora.com/news-and-blog/

@Collabora

@jekstrand_

Thank you!

We are hiring

col.la/careers