A little Windows with your Mesa?

Faith Ekstrand

XDC 2024

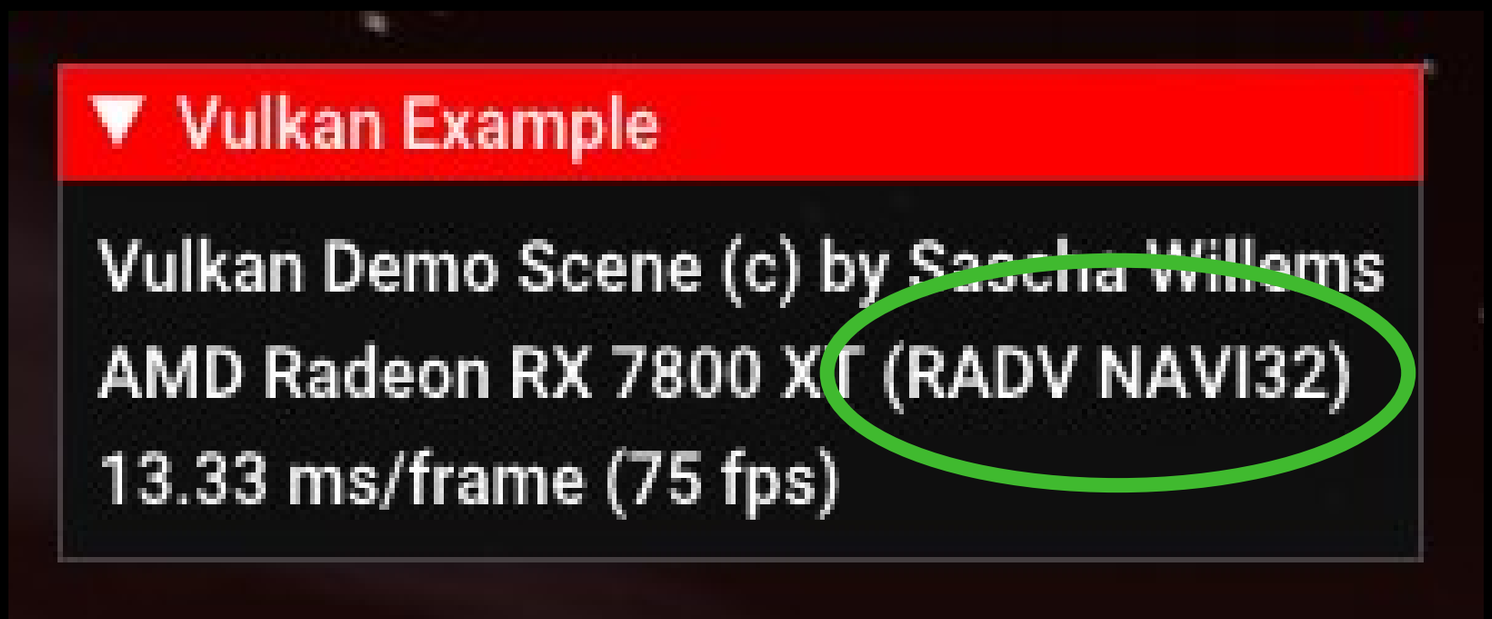
+

A few months ago on a desktop far,
far away. . . .

(It’s my desktop. It’s in my house, 1000s of miles from Montreal.)


Vulkan Example
Vulkan Demo Scene (c) by Sascha Willems
AMD Radeon RX 7800 XT (RADV NAVI32)
23.26 ms/frame (43 fps)
Vulkan
TM
STRY FORGED

Yes, that’s RADV running on Windows 11



Let’s talk about this…

- How did I do it?
- Why did I do it?
- What does this mean for the future of Mesa?

About me

- Faith Ekstrand
  - @gfxstrand@mastodon.gamedev.place
- Been around freedesktop.org since 2013
  - First commit: wayland/31511d0e, Jan 11, 2013
- At Intel from June 2014 to December 2022
  - NIR, Intel (ANV) Vulkan driver, SPIR-V → NIR, ISL, other Intel bits
- At Collabora since January 2022
  - Work across the upstream Linux graphics stack, wherever needed
  - Currently the lead developer / maintainer of NVK

First, let’s talk about WDDM2

What is WDDM2?

- **W**indows **D**river **D**isplay **M**odel, version 2
- WDDM 1 was introduced in Windows Vista
  - Better separation between userspace and kernel driver
  - Composited desktop
  - GPU Virtual memory support for process separation
  - GPU work scheduling
  - GPU reset handling
- WDDM 2 was introduced in Windows 10
  - Explicit virtual memory management (sparse memory)
  - Improved synchronization model (timeline semaphores)

What is WDDM2?

- WDDM is an interface provided by Microsoft
- Provides entrypoints (think ioctls) for operations:
  - Enumeration of adapters (think VkPhysicalDevice)
  - Creation of devices and queues
  - Memory allocation and mapping
  - Virtual address assignment (vkQueueBindSparse)
  - Work submission (vkQueueSubmit)
  - Synchronization
  - Presentation (vkQueuePresent)
- All of this is standardized! (Well, sort of...)

So it’s standardized?

- Well, sort-of…
- Many entrypoints have a pPrivateData
  - Represented as a void pointer and size
- Passed verbatim between UMD and KMD
  - May contain whatever, Windows doesn’t care
- Sometimes pPrivateData contains vital details
  - Sizes and types of memory allocations
  - Type of queue (graphics, compute, etc.)
  - Everything about command submission

Is the WDDM2 API stable?

- Yes, sort-of…
- The WDDM 2 API is stable
  - There was a break between WDDM 1 and 2
  - Otherwise, the API only moves forwards
- The pPrivateData fields are not stable
  - The IHV is free to change their layout/meaning at will
- IHVs ship userspace and kernel together
  - They are always updated together
  - The only requirement is that the shipped KMD/UMD pair can talk to each other.

Is the WDDM2 API documented?

- Technically, yes
  - https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/d3dkmthk/
- The docs aren’t great and there are no examples
  - Well, okay, there are now that my RADV branch exists 😅
- Nothing in pPrivateData is documented
  - And good luck getting the IHV to let you look at their Perforce repo!

Reverse-engineering AMD’s pPrivateData

Reverse-engineering pPrivateData

- First, you'll need a D3D12 driver...
  - I used D3D12 because WDDM is designed for D3D
  - Available inside the WSL2 container since 2020
  - WSL2 makes everything easier 😉
- And some carefully targeted tests
  - https://gitlab.freedesktop.org/gfxstrand/wddm2-pdd-re
- And a way to scrape pPrivateData from the driver
  - https://github.com/gfxstrand/libdxg/tree/pdd-re
- Then you poke at D3D12 and see what comes out!

Reverse-engineering pPrivateData

```
auto dev = get_device(get_adapter());

std::vector<D3D12_COMMAND_LIST_TYPE> clist_types;
clist_types.push_back(D3D12_COMMAND_LIST_TYPE_DIRECT);
clist_types.push_back(D3D12_COMMAND_LIST_TYPE_COMPUTE);
clist_types.push_back(D3D12_COMMAND_LIST_TYPE_COPY);

for (auto clist_type : clist_types) {
    char fname[25];
    snprintf(fname, sizeof(fname), "out/queue_%s",
             COMMAND_LIST_TYPE_to_str(clist_type));
    PDDLogToFile logger(fname);

    ComPtr<ID3D12CommandQueue> spQueue;
    D3D12_COMMAND_QUEUE_DESC CQDesc = { D3D12_COMMAND_LIST_TYPE_COMPUTE };
    CHECK(dev->CreateCommandQueue(&CQDesc, IID_PPV_ARGS(&spQueue)));
}
```

What I know about AMD's PPDs

- Adapter info: maybe 2%
  - I know where they put the PCI ID
  - I know very little else
  - I can't change the adapater info so it's hard to R/E

What I know about AMD's PPDs

- Adapter info: maybe 2%
- Memory allocation: 20%, enough for now
  - Client requested size
  - Aligned size
  - A few bits that control placement, mapping, etc.
  - Size is duplicated ~10x and I don't know why
  - Nothing about shared images (It's probably AddrLib? 🤷‍♀️)

What I know about AMD's PPDs

- Adapter info: maybe 2%
- Memory allocation: 20%, enough for now
- Queue creation: 20%, enough to create a queue
  - There is a pPrivateData but nothing goes in it
  - There are NodeOrdinal and EngineAffinity fields in the WDDM2 API
  - I don't really know what either of them mean
  - I know (0, 1) gives me graphics and (2, 1) gives me compute
  - I don't know how any of it actually maps to hardware

What I know about AMD's PPDs

- Adapter info: maybe 2%
- Memory allocation: 20%, enough for now
- Queue creation: 20%, enough to create a queue
- Queue submit: 80%
  - I pretty much know how queue submission works
  - Including prelude and postlude command buffers
  - I don't know how HW contexts work

What I know about AMD's PPDs

- Adapter info: maybe 2%
- Memory allocation: 20%, enough for now
- Queue creation: 20%, enough to create a queue
- Queue submit: 80%

Overall, it's working but lots of stuff doesn't work yet

A full CTS run has enough bugs to take down Windows in 5 minutes

Why use WSL?

Graphics drivers in WSL

- Microsoft added graphics to WSL in 2020
- Paravirtualized WDDM2 is exposed to Linux
  - dxgkrnl kernel driver maps WDDM2 in the host to ioctls in the guest
  - libdxg.so maps those ioctls back to the WDDM2 interface
  - The only real difference is WCHAR on Linux vs. Windows
- IHVs provide a Linux build of their D3D12 driver
- Other APIs like OpenGL are layered on top of D3D12
  - They’re just Mesa drivers that run on D3D12
  - There are a couple exceptions for things like CUDA

Why use WSL?

- libdxg.so provides the same interface as Gdi32.dll
- It lets me use a Linux build of Mesa
  - No need to sort out all the Windows build issues
  - I can rsync builds between my laptop and my Windows box
- I can just use LD_PRELOAD to replace libdxg.so
  - I don’t know how to hook a DLL but I do know how to use LD_PRELOAD
- WSL exposes a Wayland compositor with wl_shm
  - I didn’t have rewrite the Vulkan WSI before I can see something


Vulkan Example
Vulkan Demo Scene (c) by Sascha Willems
AMD Radeon RX 7800 XT (RADV NAVI32)
23.26 ms/frame (43 fps)
Vulkan
TM
STRY FORGED

But why?!?

Why not? 

Why?

- I’ve talked about Mesa on Windows for years
  - Why not? Why should Windows drivers be closed?
- IHV Windows people like to say it won’t work
  - “But there’s all this IP involved in Windows drivers!”
  - It’s all nonsense…
  - I want to prove them wrong
- RADV is better than AMD’s Vulkan driver
  - Faster, more features, better development model…
- Game developers are tired of closed-source drivers
  - They’re undebuggable without IHV help

Mesa: The GCC of graphics?

- Once upon a time, everyone wrote C compilers
  - MSVC on Windows
  - Sun, HP, SGI all had their own C compilers
  - Intel had ICC
  - C was the standard

Mesa: The GCC of graphics?

- Once upon a time, everyone wrote C compilers
- There were oddly specific compilers
  - Want a C compiler that targets 2nd gen AMD Opteron?
  - Auto-vectorizing FORTRAN compiler? There's a VC startup for that

Mesa: The GCC of graphics?

- Once upon a time, everyone wrote C compilers
- There were oddly specific compilers
- Now there are 3: GCC, LLVM/clang, and MSVC

Mesa: The GCC of graphics?

- Once upon a time, everyone wrote C compilers
- There were oddly specific compilers
- Now there are 3: GCC, LLVM/clang, and MSVC
- Why shouldn’t Mesa be the GCC of graphics?
  - NVIDIA will always write their own
  - But what about everyone else?

Mesa: The GCC of graphics?

- Once upon a time, everyone wrote C compilers
- There were oddly specific compilers
- Now there are 3: GCC, LLVM/clang, and MSVC
- Why shouldn’t Mesa be the GCC of graphics?
- If Mesa is going to be the de-facto Vulkan implementation, we need to support Windows


Here's to the
future ruler of...
THE WORLD!

But can we actually ship it?

But can we actually ship it?

- Not if AMD interferes
  - Ideally, we would even get their help
- The pPrivateData structs aren't stable
  - AMD can re-arrange them at will and break Mesa
- The pPrivateData structs aren't not stable
  - IHVs don't change those structs on a whim
  - Even for internal development, updating your KMD is a pain
  - UMDs working on a 6 month old KMD is pretty common
  - I was able to run a 2 year old Mesa branch on a new AMD driver

But can we actually ship it?

- Ideally, the IHV would provide a LibAMDPPD.dll
  - Updated along with the KMD and D3D12
  - Would have a stable getter/setter API
- Or, the IHV can just ship Mesa
  - Upstream Mesa would always track the latest KMD release
  - We hope there's not too much churn
  - The IHV ships a Mesa alongside D3D12 and the KMD and verifies the combination as part of their release process
  - This option isn't great because you can't bisect old bugs
- There's work to do here but it should be possible

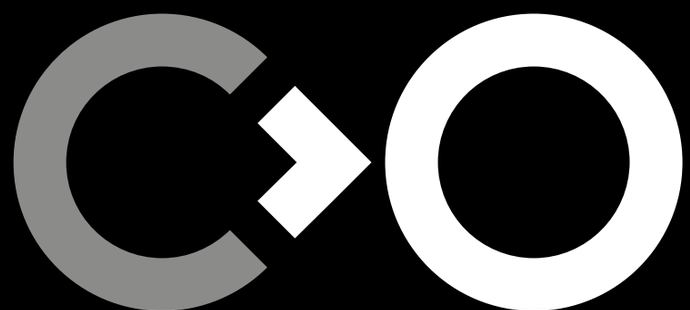
What is the impact on Mesa?

What is the impact on Mesa?

- Vulkan runtime support for WDDM2
  - https://gitlab.freedesktop.org/mesa/mesa/-/merge_requests/29945
- Mesa will need to build on Windows
  - We already do for Dozen, GLOn12, and lavapipe
- We need to improve the Win32 WSI code
  - Like rest of WSI, most developers won't have to care
- We have to sometimes debug on Windows
  - Linux with DXVK/VKD3D have way more titles than native Vulkan
- We might get some new developers!

Is this actually a good idea?

I’ll let you think about that



Thank you!