Driving Change

Vulkanising Mad Max

Who are we?

Feral Interactive - Mac/Linux/Mobile Games Publisher and Porter

Marc Di Luzio - Linux Developer and Linux Group Lead

Alex Smith - Linux Developer, currently Lead Vulkan Developer

Why Vulkan?

- Cross platform (ish)
- In need of lower level control
- Explicit APIs are the future for our low level work (DX12)
- Vulkan was the only option, but a fitting one
    - Shared GLSL frontend (HLSL supported too)
    - Very good documentation
    - Validation layers + conformance tests

How to transition to Vulkan in 5 easy steps

1. Research and discuss with other developers
2. Pick a game with a D3D11 and GL4 reference - MAD MAX on Linux
3. Develop a Vulkan backend
4. Ship a proof of concept beta
5. Try not to drown in feedback

Example - Tyrant’s Lash Camp

- An area in the game particularly problematic on GL for performance
- ~4800 draws per frame (high settings) & CPU-limited in most cases on GL
- A focus for testing performance improvements on Vulkan

Getting Started

- Vulkan is significantly different to D3D11/GL4
- Porting an existing engine can be done in stages
  - Initial implementation without major engine changes
  - Later on, engine changes to make better use of Vulkan concepts
- Initial implementation may not follow all the Vulkan “best practices”

- Some problem areas:
  - Resource binding
  - Hazard tracking

Resource Binding

- GL/D3D11: Bind resources to global/per-stage slots, change them arbitrarily as needed
- Vulkan: Descriptor sets containing groups of resource bindings
  - Recorded separately to GPU commands
  - Can’t write to descriptors while the GPU might still be using them
- Ideal usage:
  - Organise resources into sets by the frequency at which they change (e.g. per-object, per-material)
  - Bake descriptor sets up-front to reduce draw-time overhead

Resource Binding

- Difficult to map an engine based on the D3D11 model to Vulkan
- We ended up initially just rewriting descriptors per-draw
- Expected this to be terrible for performance, in fact it isn't too bad
  - CPU time spent allocating/writing/binding descriptor sets in Tyrant's Lash:
    - NVIDIA: ~6.8%
    - AMD: ~5.5%
  - Current implementation is faster than GL!
- Still want to revisit later on to see if we can do better
  - VK_KHR_push_descriptor may help
  - Look for more opportunities to reuse descriptor sets

Hazard Tracking

- Ensuring correct ordering of operations and visibility of memory writes
- D3D11 and GL (mostly) handle this for you
  - GL to a slightly lesser extent - explicit glMemoryBarrier for image load/store and SSBOs
- Vulkan guarantees very little about this
  - Up to the app to enforce ordering/visibility with synchronisation primitives (barriers, etc.)
- Also have to deal with image layouts
  - Images must be in an appropriate layout for any given usage
  - Transitions between layouts are done explicitly by the app
- Probably one of the trickiest parts of Vulkan
  - For us, has been the source of lots of subtle rendering issues!

Hazard Tracking

- Again, difficult to handle this in an existing D3D11-oriented engine
- Our approach was to implement automatic tracking to add barriers + layout transitions where needed
- We try to be smart about this
  - Batch barriers together as much as possible!
  - Individual barriers per-resource can cause a lot of unnecessary flushing/waiting
- Advantages:
  - We can re-use this system across multiple games
  - Automatic barrier batching improves efficiency while cutting down development effort
- Disadvantages:
  - Has some overhead (~3% CPU time in Tyrant's Lash)
  - Difficult to get it right! Had lots of obscure edge cases

Getting Started cont.

- We still see improvements over GL just from this basic port approach
- Lots of room for further improvement!
- Exposing Vulkan concepts directly in your graphics API abstraction layer can have benefits
  - Means more work can be moved away from draw-time
  - Compute things ahead of time = reduced CPU overhead

Best Practices

- Things that you should do
  - Can be done even within an engine targeting older APIs!
- Examples:
  - Memory management
  - Vendor-specific differences

Memory Management

- Vulkan decouples memory allocation from resource creation
- Managing memory efficiently is now your responsibility
- Do not just allocate and free memory directly from the driver per-resource
  - Slow!
  - Some platforms have low limits on the maximum number of allocations
- Implement a proper memory management system early on
  - Allocate memory in large chunks and manage it yourself
  - Persistently map host-visible memory allocations
  - Goal is to avoid overhead of driver/OS as much as possible
- Some of our largest performance improvements have been from memory management improvements

Memory Management

- Our solution: general memory manager + fast path for dynamic buffers
- Fast path for dynamic buffers:
  - Each buffer maintains its own internal ring buffer, cycle between slots on a map/discard
  - Use a per-frame fence to determine when slots can be reused
  - If there's no free slots, reallocate with more slots
  - ~15% frametime reduction in Tyrant's Lash over just using the general memory manager

- Vulkan gives you the flexibility to tailor things to your application
  - Both to increase performance and reduce VRAM usage (e.g. memory aliasing)

Vendor-Specific Differences

- Make sure you handle and take advantage of differences between vendors
- Test on different hardware/drivers early on
- Not necessarily vastly different code paths!

- Memory management:
  - Different memory configurations (heaps/types) on different vendors/drivers
  - What works for one vendor might not work/be optimal on another
  - E.g. AMD has device-local *and* host-visible memory types, NV does not
    - Write to/read from VRAM directly
    - Use for dynamic buffers and GPU-to-CPU read-back

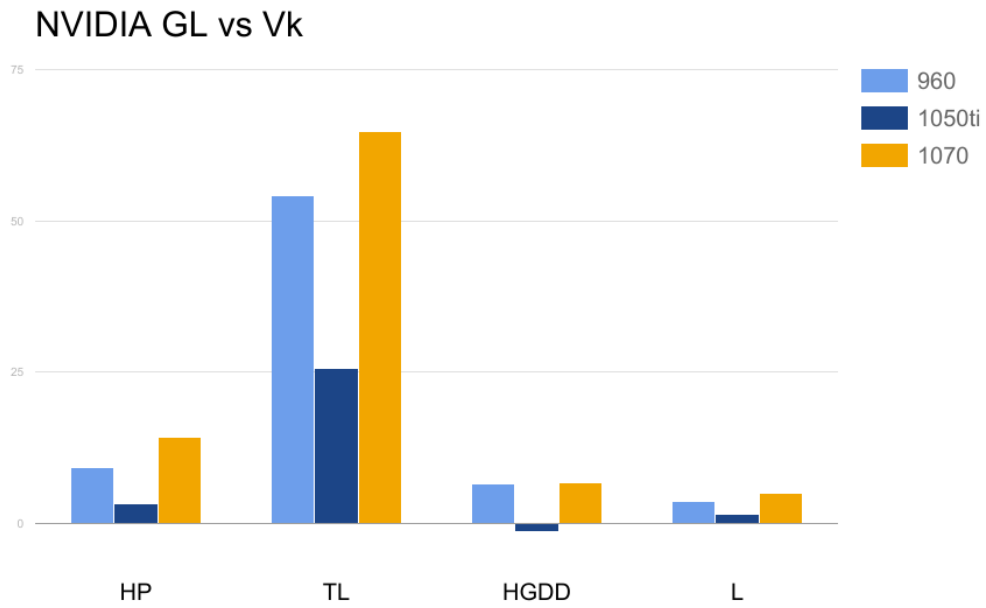
Vendor-Specific Differences

- Vendor-specific extensions:
  - E.g. VK_NV_dedicated_allocation
    - Use for render targets + large resources to enable hardware optimisations
    - GPU-limited case saw ~15% memory bandwidth reduction -> ~7% frametime reduction
  - Plus others to expose hardware features

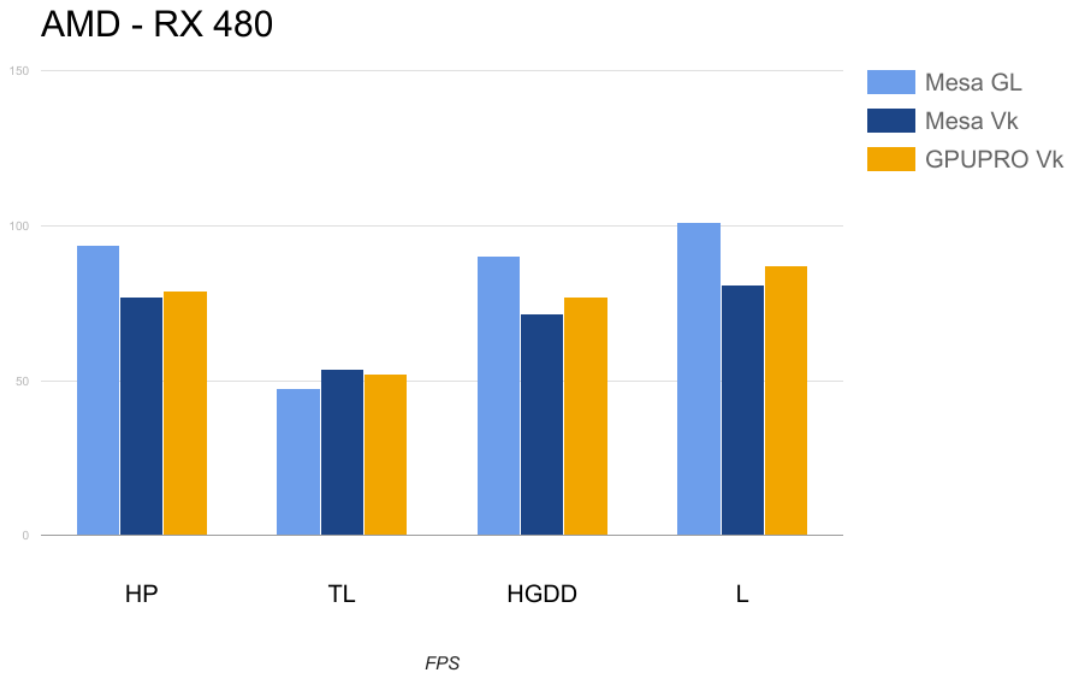
Tyrant’s Lash revisited

- Faster than GL with even a rudimentary approach
  - Workload saved on the CPU by taking driver’s work
  - Keeping the GPU busy

NVIDIA



AMD



Summary

- Wide support on Linux (NVIDIA/AMD/Mesa/Intel)
  - Only had to drop older Intel chips (Haswell era)
- Open source driver support coming along (RADV/ANV)
  - RADV competitive even before full certification
- Vulkan further along than expected internally and externally
  - Drivers as stable as GL already on Linux
- Still plenty of avenues for improvement

Going forward

FERAL

And Lastly

- Get in touch if
  - You work with an IHV
  - You’re a driver programmer
  - You know LLVM like the back of your hands
- We’ll have the game running at lunch
- Any Questions?