Descriptors are hard

Faith Ekstrand

XDC 2025, Vienna Austria

About Me

- Faith Ekstrand
  - @gfxstrand@treehouse.systems
- 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

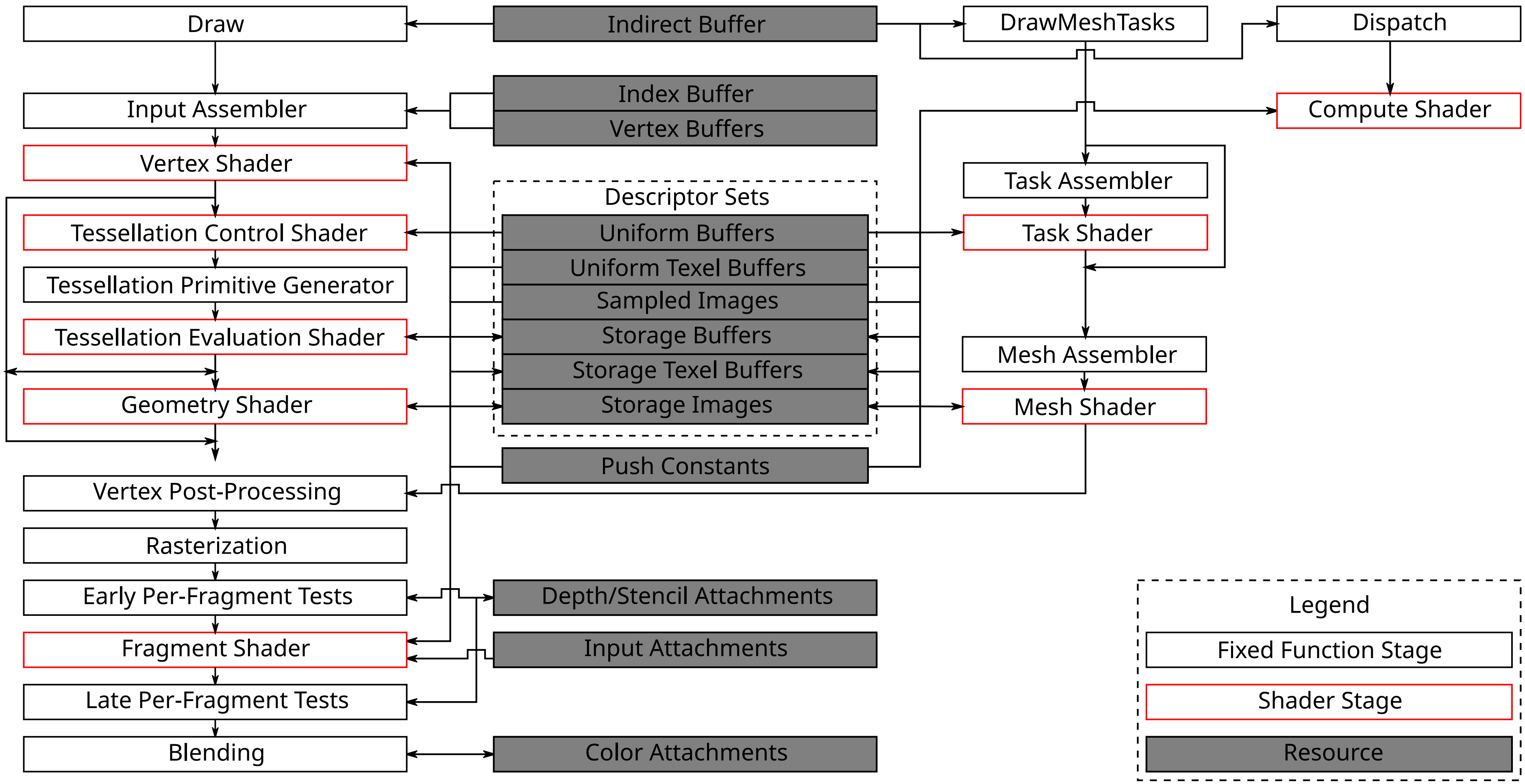
Descriptors are hard

What are descriptors?

- Modern GPUs are a combination of programmable shader cores and fixed-function hardware

What are descriptors?

- Modern GPUs are a combination of programmable shader cores and fixed-function hardware
- The fixed-function hardware comes in two forms:
  - Hardware to feed the shader cores (input assembler, rasterizer, dispatcher)
  - Hardware to accelerate resource access (texture sampling, image load/store, etc.)


Draw
Indirect Buffer
DrawMeshTasks
Dispatch
Input Assembler
Index Buffer
Vertex Buffers
Compute Shader
Vertex Shader
Task Assembler
Descriptor Sets
Tessellation Control Shader
Uniform Buffers
Task Shader
Uniform Texel Buffers
Tessellation Primitive Generator
Sampled Images
Tessellation Evaluation Shader
Storage Buffers
Mesh Assembler
Storage Texel Buffers
Geometry Shader
Storage Images
Mesh Shader
Push Constants
Vertex Post-Processing
Rasterization
Early Per-Fragment Tests
Depth/Stencil Attachments
Legend
Fragment Shader
Input Attachments
Fixed Function Stage
Late Per-Fragment Tests
Shader Stage
Blending
Color Attachments
Resource

What are descriptors?

- Modern GPUs are a combination of programmable shader cores and fixed-function hardware
- The fixed-function hardware comes in two forms:
- Descriptors are the HW description of a resource
  - Textures and samplers
  - Storage images
  - Texel buffers
  - UBOs and SSBOs
  - Acceleration structures

Why are descriptors hard?

Descriptors are expensive

Descriptors are expensive

- For simple things like a UBO or SSBO, descriptors are easy
  - A simple base address + size is all you need
  - Acceleration structures are just a pointer
  - Typically 64 or 128 bits
  - NVIDIA can pack a whole UBO into 64 bits

Descriptors are expensive

- For simple things like a UBO or SSBO, descriptors are easy
- For images, descriptors can get quite large
  - Needs to describe the complete image layout
    - Base address, Mip layout, tiling, etc.
  - On AMD, an image descriptor is 32 bytes and a sampler is 16 bytes
  - On NVIDIA, both are 32 bytes
  - On Intel, an image descriptor is 64 bytes and a sampler is 16 bytesrto

Descriptors are expensive

- For simple things like a UBO or SSBO, descriptors are easy
- For images, descriptors can get quite large (up to 64B)
- GPU shaders execute in subgroups of up to 128 invocations
  - On AMD, they use either 64 or 32-wide subgroups
  - A texture instruction can have up to 2 vec4s (32B) of client data (coords, etc.)
  - Combined with an image+sampler, that makes 80B per-invocation on AMD
  - 64 lanes x 80B = 5120B of data per per instruction (that’s more than a CPU page!)

That’s a LOT of data

How do we reduce this cost?

How do we reduce this cost?

- On AMD, they use SGPRs
  - They have fast scalar load instructions capable of pulling an entire descriptor into SGPRs on a single instruction
  - Descriptors are sent in SGPRs and the client data is sent in VGPRs
  - SGPRs are only sent once for the entire subgroup so they're basically free
    - One SGPR costs the same as 32 or 64 UGPRs
  - If the descriptor is non-uniform, they have to loop

How do we reduce this cost?

- On AMD, they use SGPRs

```
while (true) {
    bool tex_eq_first = readFirstInvocationARB(texture) == texture;
    bool smp_eq_first = readFirstInvocationARB(sampler) == sampler;
    if (tex_eq_first && smp_eq_first) {
        res = texture(texture, sampler, ...);
        break;
    }
}
```

How do we reduce this cost?

- On AMD, they use SGPRs
- On NVIDIA, everything goes in a big table
  - Actually, two tables: One for images and one for samplers
  - The tables are bound to the context and very expensive to switch
  - The hardware caches these table like crazy
  - In the shader, a single 32-bit handle is passed to the sampler unit
    - 12 bits of sampler index, 20 bits of texture index
  - Non-uniform texture access is “free” on Turing+

Both of these designs are bindless

How do we reduce this cost?

- On AMD, they use SGPRs
- On NVIDIA, everything goes in a big table
- Intel is pretty similar to NVIDIA, but different
  - They have bindless surface/sampler tables
  - Hardware instructions pass table indices, except they're uniform
  - They also have HW binding tables which provide an extra indirection
    - Used for render targets and "bound" resources
  - Again, they cache everything like crazy

How do we reduce this cost?

- On AMD, they use SGPRs
- On NVIDIA, everything goes in a big table
- Intel is pretty similar to NVIDIA, but different
- Arm (v9+) has VK_EXT_descriptor_buffer in hardware
  - They have 32 descriptor set bindings
  - Each binding points to a buffer full of descriptors
  - Texture instructions reference the set + index (8:24 bits)
  - Unlike NVIDIA and Intel, these set bindings are fully pipelined

How do we reduce this cost?

- On AMD, they use SGPRs
- On NVIDIA, everything goes in a big table
- Intel is pretty similar to NVIDIA, but different
- Arm (v9+) has VK_EXT_descriptor_buffer in hardware
- Arm (v7-) has a table per-stage
  - This table contains everything: Vertex buffers, images, textures, UBOs, etc.
  - Shaders pass indices into this table to the sampler
  - SSBOs are just an address + size living in a UBO somewhere

Let’s look at the big picture

Types of descriptors

Roughly, descriptors come in a few types:

- Direct access (D)
  - The shader passes the entire descriptor to the memory unit directly
  - It could come from a buffer or be baked into the shader
  - The global addresses for SSBOs are a form of direct descriptor

Types of descriptors

Roughly, descriptors come in a few types:

- Direct access (D)
- Descriptor Buffers (B)
  - Some set of buffers are bound as pipelined state
  - Shaders pass a descriptor buffer index + offset/index to the memory unit
  - Unlike direct descriptors, you *must* indirect through one of the bound buffers

Types of descriptors

Roughly, descriptors come in a few types:

- Direct access (D)
- Descriptor Buffers (B)
- Descriptor Heaps (H)
  - Heaps are bound to the context and expensive to change
  - Shaders pass a heap index to the memory unit
  - Saves a lot of internally wiring because the heap addresses are global

Types of descriptors

Roughly, descriptors come four types:

- Direct access (D)
- Descriptor Buffers (B)
- Descriptor Heaps (H)
- Fixed HW bindings (F)
  - Everything else: HW binding tables, MMIO regs, etc.
  - Generally pipelined, but very restrictive

Types of descriptors

	Textures	Images	Samplers	Border Colors	Typed buffers	UBOs	SSBOs
NVIDIA	H	H	H		H	D/F	D
AMD	D	D	D	H	D	D	D
Intel (gfx9+)	H/F	H/F	H		H/F	H/D/F	H/D/F
Intel (gfx8-)	F	F	F		F	D/F	D/F
Arm (v9+)	B	B	B		B	B/D/F	B/D
Arm (v7-)	F	F	F		F	D/F	D
Qualcomm (a5xx+)	B	B	B		B	B	B
Broadcom (vc5)	D	D	D		D	D	D
Apple	B/F	B/F	H	🤣	N/A	D	D

How do we model this in the API?

OpenGL [ES]

Descriptors in OpenGL [ES]

- Resources in OpenGL [ES] are bound to slots

Descriptors in OpenGL [ES]

- Resources in OpenGL [ES] are bound to slots
- There is a fixed number per type of resource
  - Gallium supports 32 samplers, for instance

Descriptors in OpenGL [ES]

- Resources in OpenGL [ES] are bound to slots
- There is a fixed number per type of resource
- The slots are shared across all shader stages
  - No per-stage bindings
  - No separation between 3D and compute

Descriptors in OpenGL [ES]

- Resources in OpenGL [ES] are bound to slots
- There is a fixed number per type of resource
- The slots are shared across all shader stages
- Drivers translate this to whatever they want
  - Bindless + a UBO of handles on NVIDIA
  - A descriptor buffer on AMD, Arm, and Qualcomm
  - Push constants on Broadcom

Descriptors in OpenGL [ES]

- Resources in OpenGL [ES] are bound to slots
- There is a fixed number per type of resource
- The slots are shared across all shader stages
- Drivers translate this to whatever they want
- This works pretty well for fixed HW descriptors

Bindless texturing in OpenGL

- ARB_bindless_texture added bindless texturing to OpenGL

Bindless texturing in OpenGL

- ARB_bindless_texture added bindless texturing to OpenGL
- The client calls `glGetImageHandleARB()` to get a 64-bit "handle" to the texture/sampler

Bindless texturing in OpenGL

- ARB_bindless_texture added bindless texturing to OpenGL
- The client calls `glGetImageHandleARB()` to get a 64-bit “handle” to the texture/sampler
- The client also has to manage texture/image residency
  - `glMakeTextureHandleResidentARB()`
  - `glMakeTextureHandleNonResidentARB()`

Bindless texturing in OpenGL

- ARB_bindless_texture added bindless texturing to OpenGL
- The client calls `glGetImageHandleARB()` to get a 64-bit "handle" to the texture/sampler
- The client also has to manage texture/image residency
- In the shader, the client can texture using that handle instead of a bound texture object.

Does this sound familiar?

Yeah, it’s the NVIDIA model…

NV_bindless_texture should have been a hint. 😅

Vulkan descriptor sets

Vulkan descriptor sets

- Vulkan descriptor sets are a compromise

Vulkan descriptor sets

- Vulkan descriptor sets are a compromise
- They can be backed by buffers of descriptors
  - In which case the client manages memory and lifetimes for you

Vulkan descriptor sets

- Vulkan descriptor sets are a compromise
- They can be backed by buffers of descriptors
- They are also CPU-inspectable so you can use HW bindings
  - Static use rules let the driver know what descriptors are used by a shader
  - The driver scrapes bindings out of the set at draw time and maps them to HW
  - Old Intel and Mali both need this, others use it as an optimization

Vulkan descriptor sets

- Vulkan descriptor sets are a compromise
- They can be backed by buffers of descriptors
- They are also CPU-inspectable so you can use HW bindings
- With VK_EXT_descriptor_indexing, you can do bindless
  - Large descriptor sets (way bigger than typical fixed HW limits)
  - Non-uniform indexing of descriptor arrays
  - Update-after-bind (not CPU-inspectable)

VK_EXT_descriptor_buffer

- VK_EXT_descriptor_buffer gives the client more control

VK_EXT_descriptor_buffer

- VK_EXT_descriptor_buffer gives the client more control
- Descriptor set layouts are still determined by the driver

VK_EXT_descriptor_buffer

- VK_EXT_descriptor_buffer gives the client more control
- Descriptor set layouts are still determined by the driver
- The client creates a buffer backed by client memory

VK_EXT_descriptor_buffer

- VK_EXT_descriptor_buffer gives the client more control
- Descriptor set layouts are still determined by the driver
- The client creates a buffer backed by client memory
- The client gets descriptors from the driver and writes them into the buffer

EDB sucks on heap-based HW

VK_EXT_descriptor_buffer on heaps

- NVK, NVIDIA, and Intel all implement it

VK_EXT_descriptor_buffer on heaps

- NVK, NVIDIA, and Intel all implement it
- Only Intel implements it “properly”

VK_EXT_descriptor_buffer on heaps

- NVK, NVIDIA, and Intel all implement it
- Only Intel implements it “properly”
- On NVIDIA, you end up with indices in the buffer
  - Actual descriptors are still managed by `VkImageView`
  - Adds an extra indirection

VK_EXT_descriptor_buffer on heaps

- NVK, NVIDIA, and Intel all implement it
- Only Intel implements it “properly”
- On NVIDIA, you end up with indices in the buffer
- Texel buffers also have to be emulated
  - `VkBufferView` is gone so there’s no place to manage the descriptor lifetime
  - We allocate ~10k buffer views at device create and do shader shenanigans

VK_EXT_descriptor_buffer on heaps

- NVK, NVIDIA, and Intel all implement it
- Only Intel implements it “properly”
- On NVIDIA, you end up with indices in the buffer
- Texel buffers also have to be emulated
- When combined with VKD3D-Proton, it’s a mess
  - As many as 5 indirections just to do a texture fetch
  - Breaks our cbuf textures optimization

VK_EXT_descriptor_buffer on heaps

- NVK, NVIDIA, and Intel all implement it
- Only Intel implements it “properly”
- On NVIDIA, you end up with indices in the buffer
- Texel buffers also have to be emulated
- When combined with VKD3D-Proton, it’s a mess
- This is why VKD3D-Proton perf sucks on NVIDIA

D3D12 descriptor heaps

D3D12 descriptor heaps

- `ID3D12DescriptorHeap` provides a heap object
  - A big array of descriptors
  - Mappable (sort of)
  - Clients write descriptor straight into the heap

D3D12 descriptor heaps

- `ID3D12DescriptorHeap` provides a heap object
- One heap for samplers and one for everything else
  - Texture views, UAVs, buffers, etc. all go in the heap

D3D12 descriptor heaps

- `ID3D12DescriptorHeap` provides a heap object
- One heap for samplers and one for everything else
- Heaps get bound as command buffer state
  - They’re assumed to be very expensive to change

D3D12 descriptor heaps

- `ID3D12DescriptorHeap` provides a heap object
- One heap for samplers and one for everything else
- Heaps get bound as command buffer state
- Shaders reference heap entries by index
  - With D3D12 Bindless, it's an actual index in the shader
  - Most HLSL shaders use a dynamic mapping mechanism

D3D12 descriptor heaps

- `ID3D12DescriptorHeap` provides a heap object
- One heap for samplers and one for everything else
- Heaps get bound as command buffer state
- Shaders reference heap entries by index
- There are also root constants and root descriptors
  - Only buffers can go in root descriptors
  - Root descriptors go directly in the root table, not in the heap

D3D12 descriptor heaps

- `ID3D12DescriptorHeap` provides a heap object
- One heap for samplers and one for everything else
- Heaps get bound as command buffer state
- Shaders reference heap entries by index
- There are also root constants and root descriptors
- Developers really like the D3D12 model

D3D12 + VKD3D + NVIDIA = 🐌

VKD3D heap emulation

- VKD3D-Proton emulates heaps as one big descriptor set
  - One giant array per descriptor type in the shader that cover the whole heap
  - It can also use VK_EXT_descriptor_buffer

VKD3D heap emulation

- VKD3D-Proton emulates heaps as one big descriptor set
- Accessing a texture is a multi-step process
  - Look up the client index in the root table
    - Root tables are too big for push so this is a UBO in a descriptor set
    - Fetch set address, fetch UBO descriptor, fetch value from UBO (3 fetches)
  - Calculate the heap index (this is just math)
  - Texture from `tex[idx]`

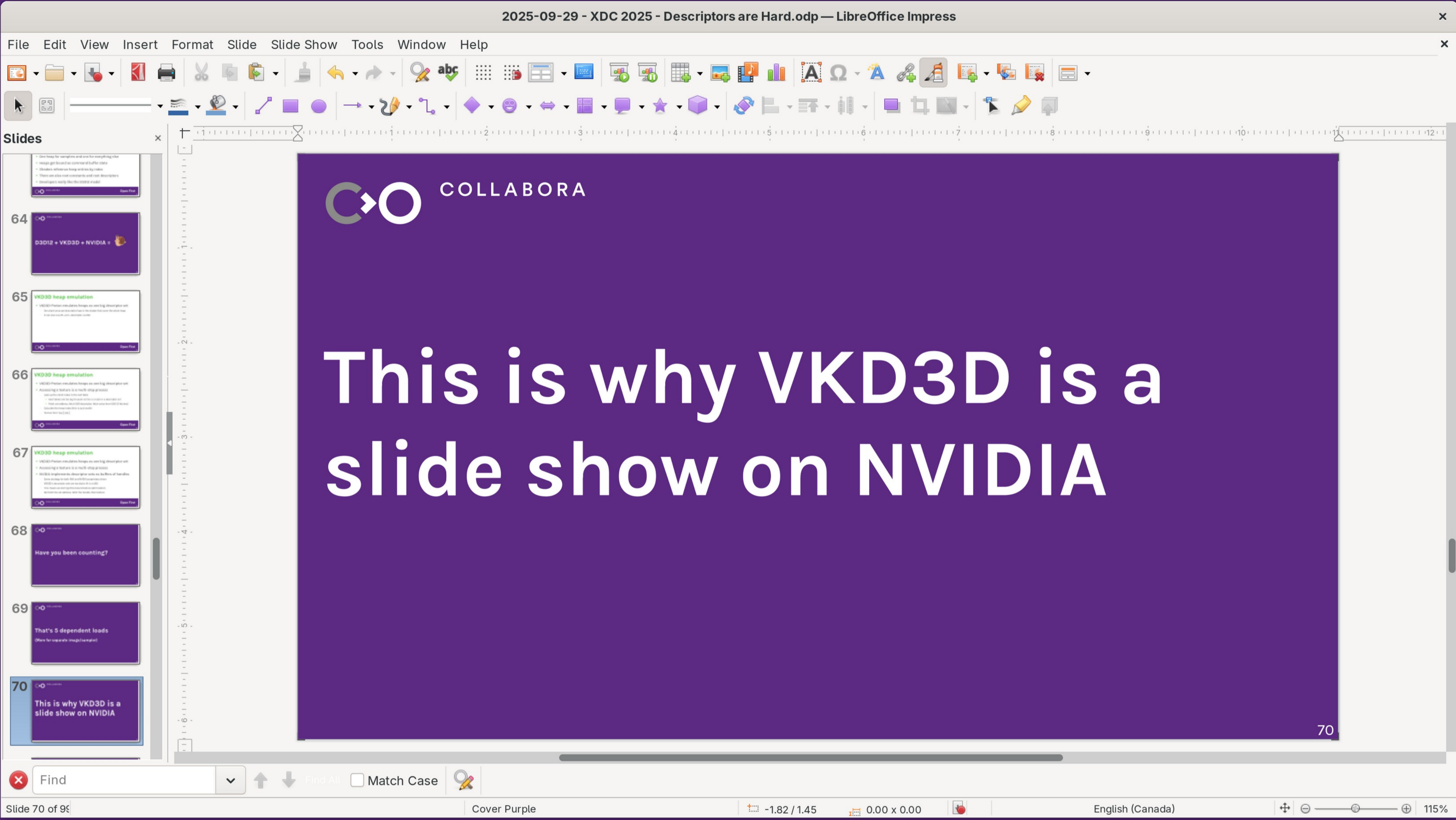
VKD3D heap emulation

- VKD3D-Proton emulates heaps as one big descriptor set
- Accessing a texture is a multi-step process
- NVIDIA implements descriptor sets as buffers of handles
  - Same strategy for both NVK and NVIDIA proprietary driver
  - VKD3D's descriptor sets are too big to fit in a UBO
  - This means we don't get the bound texture optimization
  - We fetch the set address, fetch the handle, then texture

Have you been counting?

That’s 5 dependent loads

(More for separate image/sampler)



2025-09-29 - XDC 2025 - Descriptors are Hard.odp — LibreOffice Impress

COLLABORA

This is why VKD3D is a slide show on NVIDIA

70

So what are we doing about it?

The Future of descriptors in Vulkan

The Future of descriptors in Vulkan

- We’ve been listening to the voices of developers

The Future of descriptors in Vulkan

- We’ve been listening to the voices of developers
- We’re working on a new descriptor model for Vulkan

The Future of descriptors in Vulkan

- We’ve been listening to the voices of developers
- We’re working on a new descriptor model for Vulkan
- Based on heaps, but better!
  - Heaps are just buffers, not objects
  - Clients can CPU map them directly
  - Clients can even DMA to them or write them from a shader

The Future of descriptors in Vulkan

- We’ve been listening to the voices of developers
- We’re working on a new descriptor model for Vulkan
- Based on heaps, but better!
- Clients control the in-memory layout
  - Implementation advertises descriptor sizes and alignments
  - Clients place descriptors in memory

The Future of descriptors in Vulkan

- We’ve been listening to the voices of developers
- We’re working on a new descriptor model for Vulkan
- Based on heaps, but better!
- Clients control the in-memory layout
- Embedded samplers replace immutable samplers
    - Required for YcbCr conversion
    - Managed by the driver, not backed by a VkSampler

The Future of descriptors in Vulkan

- We’ve been listening to the voices of developers
- We’re working on a new descriptor model for Vulkan
- Based on heaps, but better!
- Clients control the in-memory layout
- Embedded samplers replace immutable samplers
- Compatible with D3D12
  - Designed for both app developers and translation layers
  - New SPIR-V extension for direct descriptor access
  - Provides convenient mappings from set/binding to heaps

The Future of descriptors in Vulkan

- We’ve been listening to the voices of developers
- We’re working on a new descriptor model for Vulkan
- Based on heaps, but better!
- Clients control the in-memory layout
- Embedded samplers replace immutable samplers
- Compatible with D3D12
- Coming soon(ish)

What does this mean for Mesa?

WIP Implementations

- We’ve been working on implementations

WIP Implementations

- We’ve been working on implementations
- WIP implementations in multiple Mesa drivers
  - NVK (Nvidia), RADV (AMD), and ANV (Intel)
  - Currently the code still under the Khronos NDA
  - Available to anyone who is a Khronos member

WIP Implementations

- We’ve been working on implementations
- WIP implementations in multiple Mesa drivers
- Most of the compiler work is done in NIR and the runtime
  - Common lowering pass for set/binding → heap mappings
  - SPIR-V parser support for new heap intrinsics
  - Drivers just see heap offsets

WIP Implementations

- We’ve been working on implementations
- WIP implementations in multiple Mesa drivers
- Most of the compiler work is done in NIR and the runtime
- New Meta paths which use heaps
  - Heaps raise extra issues for meta commands (copy, blit, MSAA resolve)
  - New Meta interfaces being added to allow it to work with heaps
  - Drivers can still use descriptor set paths if they prefer

WIP Implementations

- We’ve been working on implementations
- WIP implementations in multiple Mesa drivers
- Most of the compiler work is done in NIR and the runtime
- New Meta paths which use heaps
- WIP support in DXVK and VKD3D-Proton
  - Still pretty WIP but we will hopefully be able to start analyzing perf soon

What does this mean for your driver?

Driver changes to support heaps

- Need to be able to bind heaps
  - For Intel and NVIDIA, we bind the client heap as the HW heap
    - There’s a bunch of work to avoid stalls
    - On Nvidia HW, we also have to deal with internal descriptor ranges
  - For HW with descriptor buffers (including AMD), there are 3 buffers:
    - Client Sampler heap
    - Client Resource heap
    - Embedded sampler heap

Driver changes to support heaps

- Need to be able to bind heaps
- Need to manage the embedded sampler heap
  - If you’re AMD, samplers can go straight in the shader binary
  - The rest of us need a hash+cache heap
  - API limits are in terms of unique samplers used
    - There is no `VkSampler` object for these

Driver changes to support heaps

- Need to be able to bind heaps
- Need to manage the embedded sampler heap
- Need to implement descriptor queries
  - Might involve a little refactoring of image/buffer view code
  - `VkImage/BufferView` are gone, they just take `p*CreateInfo`
  - UBO/SSBO are address + size

Driver changes to support heaps

- Need to be able to bind heaps
- Need to manage the embedded sampler heap
- Need to implement descriptor queries
- Need to sort out meta command descriptors
  - If you're a descriptor buffer driver, you can still use sets at no perf cost
  - If you're Intel, you can still use binding tables at least for now
  - It's a mess for NVK. 🙈

Driver changes to support heaps

- Need to be able to bind heaps
- Need to manage the embedded sampler heap
- Need to implement descriptor queries
- Need to sort out meta command descriptors
- Need to add heap lowering code

Driver changes to support heaps

- Need to be able to bind heaps
- Need to manage the embedded sampler heap
- Need to implement descriptor queries
- Need to sort out meta command descriptors
- Need to add heap lowering code
- And that’s it!

NIR changes for heaps

- Image/texture support for heap offsets
  - New `image_heap` intrinsics
  - New `nir_tex_src_texture/sampler_heap_offset`

NIR changes for heaps

- Image/texture support for heap offsets
- Embedded sampler support
  - Currently being scraped out and passed side-band
  - New `nir_tex_instr` bits to select an embedded sampler
  - May get embedded directly in NIR with a pass to scrape them out
    - This is annoying because NIR would have to reference `vk_sampler_state`
  - Details still WIP

NIR changes for heaps

- Image/texture support for heap offsets
- Embedded sampler support
- New `load_buffer_ptr` intrinsic
  - Replaces `load_vulkan_resource_descriptor`
  - Also works with descriptor sets
  - Draft MR: https://gitlab.freedesktop.org/mesa/mesa/-/merge_requests/37286

NIR changes for heaps

- Image/texture support for heap offsets
- Embedded sampler support
- New `load_buffer_ptr` intrinsic
- New `load_descriptor_heap[_data]` intrinsics
  - Loads a buffer or acceleration structure descriptor
  - Takes a descriptor type enum so the driver knows what to load
  - `load_descriptor_heap_data` loads raw data with no conversion

NIR changes for heaps

- Image/texture support for heap offsets
- Embedded sampler support
- New `load_buffer_ptr` intrinsic
- New `load_descriptor_heap[_data]` intrinsics
- New `global_addr_to_descriptor` intrinsic
  - Converts a 64-bit global address to a buffer descriptor

NIR changes for heaps

- Image/texture support for heap offsets
- Embedded sampler support
- New `load_buffer_ptr` intrinsic
- New `load_descriptor_heap[_data]` intrinsics
- New `global_addr_to_descriptor` intrinsic
- NVK lowering for all this is 248 LOC

We are hiring

col.la/careers