Vulkan, OpenGL, OpenGL ES

SIGGRAPH 2016
## Agenda

<table>
<thead>
<tr>
<th>Time</th>
<th>Khronos 3D Graphics BoF</th>
<th>Speakers</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:30</td>
<td>Vulkan and OpenGL Status Updates</td>
<td>Neil Trevett, NVIDIA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tobias Hector, Imagination Tech</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tom Olson, ARM</td>
</tr>
<tr>
<td>3:00</td>
<td>ISV Experience: Porting Unreal Engine 4 to Vulkan</td>
<td>Rolando Caloca Olivares, Epic Games</td>
</tr>
<tr>
<td>3:30</td>
<td>ISV Experience: Porting DOOM to Vulkan</td>
<td>Axel Gneiting, id Software</td>
</tr>
<tr>
<td>4:00</td>
<td>Panel: Best practices for Programming to the Vulkan API</td>
<td>Chris Hebert, NVIDIA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tobias Hector, Imagination Tech</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dan Archard, Qualcomm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rolando Caloca Olivares, Epic Games</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Axel Gneiting, id Software</td>
</tr>
<tr>
<td>5:00</td>
<td>Panel: Tools for the Vulkan Ecosystem</td>
<td>Bill Hollings, The Brenwill Workshop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Kyle Spagnoli, NVIDIA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Karl Schultz, LunarG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Andrew Woloszyn, Google</td>
</tr>
<tr>
<td>6:00</td>
<td>Party Time!</td>
<td></td>
</tr>
</tbody>
</table>
NEW ARB_gl_spirv OpenGL Extension

- Enables OpenGL driver to ingest compiled SPIR-V code
  - Specification released here at SIGGRAPH
  - Available today in developer release drivers from NVIDIA
- Accepts SPIR-V output from open source Glslang Khronos Reference compiler
  - https://github.com/KhronosGroup/glslang

Enables OpenGL to participate in SPIR-V-based toolchain innovations
SPIR-V Ecosystem

SPIR-V
- Khronos defined and controlled cross-API intermediate language
- Native support for graphics and parallel constructs
  - 32-bit Word Stream
- Extensible and easily parsed
- Retains data object and control flow information for effective code generation and translation

Khronos has open sourced these tools and translators

Khronos plans to open source these tools soon

https://github.com/KhronosGroup/SPIRV-Tools

Third party kernel and shader Languages

GLSL

HLSL

OpenCL C

OpenCL C++

SPIR-V (Dis)Assembler

SPIR-V Validator

Other Intermediate Forms

IHV Driver Runtimes

SPIR-V Magic #: 0x07230203
SPIR-V Version 99
Builder’s Magic #: 0x051a00BB
<id>-bound is 50

OpMemoryModel
Logical
GLSL40

OpEntryPoint
Fragment shader
function <id> 4

OpTypeVoid
<id> is 2

OpTypeFunction
<id> is 3
return type <id> is 2

OpFunction
Result Type <id> is 2
Result <id> is 4

Function Type <id> is 3

LLVM to SPIR-V Bi-directional Translator

LLVM

New with OpenCL 2.2 And SPIR-V 1.1

New with ARB_gl_spirv

© Copyright Khronos Group 2016 - Page 5
OpenGL Driver Support Update

• ARB extension support increased across the board
• Mesa 12.1 released yesterday reaches OpenGL 4.5!
• GLEW 2.0 released today!
  - Forward-compatible contexts, adds new extensions, OSMesa and EGL support
    - [https://github.com/nigels-com/glew.git](https://github.com/nigels-com/glew.git)

Khronos significantly improving OpenGL 4.5 conformance tests
- Release in April
- Working to release as many tests in open source as possible

[http://www.g-truc.net/doc/OpenGL%204%20Hardware%20Matrix.pdf](http://www.g-truc.net/doc/OpenGL%204%20Hardware%20Matrix.pdf)
More OpenGL News

9th Edition of the OpenGL Programming Guide released - includes OpenGL 4.5 with SPIR-V support

Doom4 primary API is OpenGL
Safety Critical 3D

OpenGL SC 1.0 - 2005
Fixed function graphics subset

OpenGL ES 1.0 - 2003
Fixed function graphics

OpenGL ES 2.0 - 2007
Shader programmable pipeline subset

OpenGL SC 2.0 - April 2016
Shader programmable pipeline subset

New Generation APIs for safety certifiable vision, graphics and compute
e.g. ISO 26262 and DO-178B/C

Experience and Guidelines

Safety Critical Advisory Panel Announced Today!
Generating API design guidelines to enable system certifications
https://www.khronos.org/openglsc/

Khronos Launches Safety Critical Advisory Panel - and Invites Industry Experts to Participate

Experienced practitioners in the field of safety critical system design are invited to apply for Advisory Panel membership simply by sending an email to Khronos_scap_apply@khronos.org. Please include your contact information, a short history of your experience along with why you feel you could help us set the future direction of safety critical APIs.

Read the press release
OpenGL ES Update

Tobias Hector | OpenGL ES Chair
Lead Software Design Engineer, Imagination
Introduction

• You might have noticed...
  - I’m not Tom!
  - Really, I’m not just Tom wearing a beard.

• I took the helm in May
  - Have been steering this ship ever since
Introduction

• You might have noticed...
  - I’m not Tom!
  - Really, I’m not just Tom wearing a beard.

• I took the helm in May
  - Have been steering this ship ever since

• Tom was an excellent chair for nearly 10 years
  - Comfortable
  - Sturdy
  - Easy to clean
  - And saw through 4 OpenGL ES releases!
OpenGL ES Status

• Little demand for a new OpenGL ES at present
  - So not announcing one this year
  - Keeping an eye on the market for changes

• High demand for making OpenGL ES more robust
  - Particularly with regards to WebGL

• Focus on fixes and enhancements
  - 3.2 API spec updated last month
  - More fixes on the way (including for 3.0 and 3.1 specifications, and ESSL)
ES 3.2 Conformance

• OpenGL ES 3.2 CTS Released!
  - Integration of ES tests from AOSP
  - Many ES 3.2 tests

• New OpenGL ES CTS Lead
  - Alexander Galazin (ARM)
  - Elected in May - doing a great job!

• Many companies now conformant
  - Nvidia
  - ARM
  - Verisilicon
  - Other submissions pending
Vulkan Update
SIGGRAPH 2016

Tom Olson, ARM | Vulkan Working Group chair
Status

• Vulkan 1.0 launched in February
  - Only two months late...

• A complete package
  - Specs (API, SPIR-V, Data Formats, extensions)
  - GLSL to SPIR-V compiler (glslang)
  - Standard loader and validation layers
  - Conformance test suite
  - Drivers and SDKs

• All Khronos resources in open source
  - Software under Apache 2.0
  - Specification license on the way
  - https://github.com/KhronosGroup/
Adoption and Availability - Hardware

- Conformant GPUs
  - AMD GCN (production)
  - Intel Skylake and Broadwell (beta, production coming soon)
  - NVIDIA Kepler, Maxwell, Pascal (production)

- Desktop hardware
  - AMD GCN (production)
  - Intel Skylake and Broadwell (beta, production coming soon)
  - NVIDIA Kepler, Maxwell, Pascal (production)

- Mobile hardware
  - Samsung Galaxy S7
  - NVIDIA Shield / Shield TV
  - Google Nexus 5X, 6P, Player, Pixel C (Android N Developer Preview)
  - *Lots* more on the way!
Adoption - Platforms

- Windows
  - Windows 7
  - Windows 8
  - Windows 10
- Linux
  - SteamOS
  - ubuntu
  - redhat
  - Tizen
- iOS / MacOS
  - MoltenVK
Adoption - Games and Engines

- ‘ProtoStar’ demo on Vulkan port of Unreal Engine 4
- DOOM on Vulkan port of id Tech 6
- DotA 2 on Vulkan port of Source 2
- Talos Principle on Vulkan port of Serious Engine
Community and Ecosystem

A huge amount of activity on GitHub!

Ports

Tools

Tutorials
Community and Ecosystem: What’s New

• Vulkan Conformance Test 1.0.1 nearing release
  - 107k total test cases (34% increase vs 1.0.0)
  - Substantial coverage improvement
  - Thanks Samsung, Intel, Google!

• SDK and Validation Layers progress
  - 8 SDK releases over last six months
  - All areas of spec have some coverage - growing every day
  - 1450+ commits; 222+ GitHub and 180+ LunarXchange issues resolved since launch

• Glslang compiler has partial HLSL support
  - See GitHub glslang issue #362 Complete basic HLSL parser

• New tools
  - SPIRV-Cross cross-compiler / reflection tool (Hans-Kristian Arntzen, ARM)
  - Vulkan-hpp (Markus Tavenrath / Andreas Süßenbach, NVIDIA)
What we’re working on: Vulkan 1.0

• Vulkan 1.0 spec maintenance
  - Bug fixes
  - Clarifications
  - Reference page extraction
  - Extensions to fill gaps

• BTW: Putting specs on GitHub was a GREAT idea!
  - Fantastic input from community
  - Typo and error reports
  - Requests for clarification
  - Notes on undefined corner cases

• Spending 50% of meeting time on GitHub issues
  - Weekly spec update (most weeks)
What else we’re working on: Vulkan Next

• Vulkan Next is in active development
  - Core spec in definition
  - Some features may come out as extensions
  - Schedule TBD

• Top priorities
  - Better multi-GPU support
  - VR support (e.g. efficient multi-view rendering, direct screen access)
  - Cross-API and cross-process sharing
  - Subgroup instructions (e.g. shader ballot)
  - Generalized renderpass / subpass dependencies
  - Rigorous memory model
We need your help!

• Use Vulkan
  - At least experimentally
  - ...and give us feedback

• Contribute to the ecosystem
  - All Khronos Vulkan code projects are Apache 2.0
  - We need examples, tutorials, demos, tools...
  - *Note* - watch for RFQs forthcoming at www.khronos.org

• Help us promote the API
  - Got a cool Vulkan-generated video? Let us host and promote it!
  - Send mail to ‘marketing' at khronos.org
Porting UE4 to Vulkan

Lessons learned during Protostar demo (and beyond!)

Rolando Caloca O.
Epic Games
Intro

• UE4 RHI Architecture in a hurry
• Protostar & Initial RHI
• Optimizations for Protostar
• How the RHI works
• Future plans & challenges
UE4 RHI Architecture in a hurry

• RHI = Render Hardware Interface
  – aka our cross-platform way to talk to each Gfx API
UE4 RHI Architecture in a hurry

• Original architecture
  – Game Thread enqueues rendering commands
  – Rendering Thread generates Vulkan Cmd Buffers
UE4 RHI Architecture in a hurry

- Improved architecture
  - Game Thread enqueues rendering commands
  - Rendering Thread generates RHI command list
  - RHI Thread translates into Vulkan Cmd Buffers
UE4 RHI Architecture in a hurry

- Finally, multithreaded: N Render threads with M RHI threads
UE4 RHI Architecture in a hurry

• Why use the RHI command list/thread and not directly generate Vulkan commands?
  – Easier to bring up new RHIs!
  – Allows us to decouple frontend/backend which makes multithreading easier
  – We got a CPU improvement ~5 - 10% due to cache locality (both instruction & data)
Vulkan

• Why?
  – Cross-platform, high-performance API
  – Predictability
    • eg Driver doesn’t mysteriously take different time during the same draw calls on different runs
  – Control over memory allocations, aliasing
  – Control over GPU performance
    • Flushing caches, etc
  – Very similar to D3D12 and Metal
Protostar

- Collaboration between Epic, Samsung, Qualcomm and Confetti
- Tech Demo showcasing the Samsung S7 phone and the Vulkan API on mobile
  - Help push the industry adoption of Vulkan!
Vulkan RHI 0.1

- One big pool for DescriptorSets
  - 32k entries
  - Would run out after a while, plus had some sync issues
- All updates to buffers/textures doing in-place map/unmap
  - Didn’t work on some drivers as they don’t allow linear textures on host visible memory
- Immediately after every unmap, submit CmdBuffer and wait
  - GPU stalling the CPU during load!
Vulkan RHI 0.1

• Crazy hitching during PSO creation
  – We’ll talk about that more later...

• No RHI thread
  – Rendering Thread directly generating Vulkan commands

• Barely hitting 20 fps on CPU
Vulkan RHI 0.2

• Optimization time!
  – Profile CPU using hierarchical counter and address each bottleneck
    • eg DescriptorSet writes were generated every update, so cache them!
    • eg Split DescriptorSets into one for Vertex and one for Pixel
    • eg Remove tons of dynamic object allocations
  – Rinse & repeat!

• After a couple of weeks doing optimization work, got to 30 fps on both CPU & GPU
Vulkan RHI 0.2

• However lots of validation issues...
Vulkan RHI 0.2

- However lots of validation issues...
Vulkan RHI 0.2

• However lots of validation issues...
• Ship it!
Vulkan RHI 1.0

• Demo out of the door!
• Now figure out what is needed to make this usable for full titles!
  – Just come up with a list...
Vulkan RHI 1.0

- Demo out of the door!
- Now figure out what is needed to make this usable for full titles!
  - Just come up with a list...
Vulkan RHI 1.0 Task List

- Cleanup
  - Remove all TODOs & hacks
Vulkan RHI 1.0 Task List

• Cleanup
  – Remove all TODOs & hacks

A LANNISTER ALWAYS PAYS
HER TECHNICAL DEBTS
Vulkan RHI 1.0 Task List

• Robust & fault tolerant
• Support separate RHI thread
  – Then support parallel RHI threads!
• Pass all validation layer warnings!
  – Some perf warnings *might* be acceptable...
    • eg Pixel shader outputs to disabled attachment
Vulkan RHI 1.0 Task List

• Feature parity with D3D12 & Metal
Vulkan RHI Task list

- Run Kite!
Vulkan RHI Task list

• Run Paragon!
  – Same or better than D3D11!
And Beyond!

- Get the full Editor running...
Today’s Vulkan RHI

• Today’s state:
  – Separate RHI Thread translating commands
  – Mobile renderer working
  – Decent perf
    • Missing optimized Descriptor Set Layouts
  – Passing most validation
    • Mostly missing image layouts
  – Starting to get SM4/Deferred up & running
Today’s Vulkan RHI

- Command Buffers
- Resource Management
- Back Buffer/Swapchains
- Rendering
- Render Passes
- Shaders
- PSOs
- Tools
Vulkan RHI: Command Buffers

• Every RHI thread/Context has a CmdBuffer Manager
• CmdBuffer Manager has a list of persistent CmdBuffers
  – Also has an Active and Upload CmdBuffer
    • Upload needed as you can’t copy data in the middle of a RenderPass
• Every CmdBuffer:
  – Has a Fence and a Counter
    • Tracks how many times the Fence has been signaled (Periodically queried, then reset to unsignaled)
  – Knows its state (ReadyForBegin, Inside/OutsideRenderPass, Ended, Submitted)
Vulkan RHI: Command Buffers

- State Flow

- Ready For Begin
  - Begin
  - Begin Render Pass
  - Inside Render Pass
    - End Render Pass
    - Fence Signaled
  - Inside Begin
    - End
  - Submitted
    - Submit
  - Ended
Vulkan RHI: Resources

- Buffers, Images, Fences and Semaphores
- Allocating a Resource means acquiring one from its pool
  - Could be a reused one
  - Could be a brand new one
- Releasing a Resource means not used by the application
- Destroying a Resource means calling vkDestroy*()}
Vulkan RHI: Resource Managers

• General Pattern for Managers:
  – Has a UsedList, PendingFreeList and FreeList
  – Alloc resource
    • Is there a matching one in the FreeList? If so return one from there and move to the UsedList, otherwise make a new one and put in UsedList
  – Release resource
    • Move from UsedList to PendingFreeList, and store Fence Count
  – Periodically (eg once per frame, every CmdBuffer submit)
    • Go through FreeList and anything not used for N frames, Destroy
    • Go through PendingFreeList, and if the Cmd Buffer’s Fence counter > Released Fence counter, move to FreeList
Vulkan RHI: Other Managers/Utils

- **Buffer SubAllocations**
  - Manages sub-ranges so we don’t constantly have to create VkBuffers
- **Fence Manager**
- **TempFrameAllocator**
  - Tape/linear buffer sub allocations, resets every frame (after Fence signaled)
- **Deferred Deletion Queue**
  - High level releases a ref count ptr of a texture or buffer, which gets added to this Queue
  - This checks Fences and directs it to its appropriate Resource Manager
Vulkan RHI: BackBuffer/Swapchain

- RHI::GetBackBuffer()
  - That would be ideal place for calling `vkAcquireNextImageKHR()`.
  - But that’s called both inside and outside RHI::BeginViewport() and potentially multiple times, both on Render and RHI threads.
  - RHI Thread would have to sync back with Rendering Thread.
  - One solution would be to have 2 BackBuffers:
    - One for Rendering Thread
    - One for RHI Thread
  - Makes sync with Queues & Presentation hard!
Vulkan RHI: BackBuffer/Swapchain

• Instead: Dummy BackBuffer texture
  – Rendering Thread creates new dummy texture if it doesn’t have one
    • And inserts a command for the RHI thread to call `vkAcquireNextImage()`
  – Now Renderer can set the Dummy BB to `nullptr` when needed

Render Thread:

```
GetBackbuffer()
if (!BB)
  BB=new Dummy
InsertRHICmd()
return BB
```

RHI Thread:

```
AdvanceBackBuffer()
BB=nullptr
```

ExecCmd: `vkAcquireNextImage()`
Use Acquired Image Index
Vulkan RHI: Rendering (State)

- High-level Renderer:
  - SetBoundShaderState(VS, PS)
  - SetDepthStencilState(...)
  - Draw(A)
  - Draw(B)
  - SetRasterizerState(...)
  - Draw(C)
Vulkan RHI: Rendering (State)

- High-level Renderer:
  - SetBoundShaderState(VS, PS)
    - Reset BSS state for this thread, mark all state flags dirty
  - SetDepthStencilState(…)
    - Set DepthStencil state flags dirty
  - Draw[A]
    - PrepareDraw
      - Find PSO with all state flags in cache, or create if needed
      - State flags marked as no longer dirty
    - vkCmdDraw[]
Vulkan RHI: Rendering (State)

• [...]  
  – Draw(B)  
    • PrepareDraw  
      – NoOp (no dirty flags), use current PSO  
    • vkCmdDraw()  
  – SetRasterizerState(...)  
    • Mark Rasterizer state flags as dirty  
  – Draw(C)  
    • PrepareDraw  
      – Find PSO with all state flags in cache, or create if needed  
      – State flags marked as no longer dirty  
    • vkCmdDraw()
Vulkan RHI: Rendering (Resources)

- High-level Renderer:
  - `SetBoundShaderState(VS, PS)`
  - `Draw[A]`
  - `Draw[B]`
  - `SetTexture[]`
  - `Draw[C]`
Vulkan RHI: Rendering (Resources)

- High-level Renderer:
  - SetBoundShaderState(VS, PS)
    - Mark dirty DescriptorSet Write list
  - Draw[A]
    - PrepareDraw()
      - If dirty Write list
        » Get new DescriptorSets from Pool, update and bind
        » Set Write list to not dirty
    - vkCmdDraw[...]
  - Draw[B]
    - PrepareDraw()
      - NoOp as no dirty write list
    - vkCmdDraw[...]
Vulkan RHI: Rendering (Resources)

- [...] 
  - SetTexture()
    - Update Write list and set to dirty
  - Draw(C)
    - PrepareDraw()
      - If dirty Write list
        » Get new DescriptorSets from Pool, update and bind
        » Set Write list to not dirty
    - vkCmdDraw(...) and set not dirty Write list
Vulkan RHI: Render Passes

- UE4 has no concept of Render Passes
  - SetRenderTarget(...) 
  - Draw(...) 
  - CopyToResolveTarget(...) 
  - SetRenderTarget(...) 
  - Draw(...) 
  - Dispatch() [Compute] 
  - Draw(...) 
  - SetRenderTarget(...) 
  - Draw(...)
Vulkan RHI: Render Passes

- No good way (yet) for tracking transitions
  - The Renderer can also be multithreaded!
  - Renderer can switch to compute workloads w/o knowledge of previous state
- Tied also to resource/layout transitions/barriers
  - Started exposing resource transitions in the RHI but not enough info
- Still active area of research
  - Might need to expose it at the higher level
Vulkan RHI: Shaders

• Shaders are written in hlsl (usf files)
• Use hlslcc to convert from hlsl->glsl
  – Then converted to SPIR-V using glslang lib from the VulkanSDK linked into the Engine
• Might have a direct SPIR-V backend for hlslcc
  – Will depend on extensions/features
Vulkan RHI: PSOs

- UE4 compiles shaders conservatively
  - Runtime matching of vertex/pixel shaders
    - Any combination can be done at runtime
      - eg Blueprint dynamically adds a point light
    - Might have N vertex shaders, M pixel shaders
      - Unfeasible to pre-compile all combinations!
  - Have to create at runtime, causing hitches
Vulkan RHI: Shader Pipelines

- We already had added support for ShaderPipelines
  - Declare Vertex+Pixel stages at compile time
    - But not all passes support it yet (only Depth and Velocity currently)
  - Used to remove unused interpolators between Pixel & Vertex shaders as some architectures benefit from it
  - Original plan was to migrate this into PSOs
    - But still need all the rest of the state specified to be useful!
Vulkan RHI: Protostar

• We needed something so the demo wouldn’t hitch
  – First run-through experience not awesome due to so many PSOs being created
  – Couldn’t use ShaderPipelines as many passes not yet converted
  – Solution: Pipeline Cache!
Vulkan RHI: PSO Cache

• Cache:
  – Add every new unique PSO to a runtime cache off a hash from the render states and shader microcode’s CRC
  – Trigger a save command from console and serialize to disk
  – At load time if the file is there, pre-create the PSOs
  – Two levels: Local cache inside BoundShaderState, and global one
    • Is PSO key inside local BSS? Yes -> return local BSS copy
    • Is PSO key inside global BSS? Yes->copy to local BSS and return
    • Otherwise, create new PSO and add to both global and local caches
  – Virtually hitch-free in the final demo!
Vulkan RHI: PSO Cache

• Issues:
  – Shader code changes all the time
  – Out of sync whenever materials get tweaked
  – Doesn’t catch all cases... gotta catch ‘em all!
  – Some studios don’t have the resources to have QA running through the full game
  – Cache can be YUGE

• Really need a better solution...
Vulkan RHI: PSO Plans

- Plan A: Started prototyping real PSO support
  - Still researching API and impact to codebase
- Plan B: Doing research for specifying a ‘general’ PSO with some common/default state
  - Use derived pipelines [VK_PIPELINE_CREATE_DERIVATIVE_BIT] to get faster compiles
  - We do know *some* PSOs that might be needed at load time
    - Just not all of them
Vulkan RHI: PSO Plans

- Plan C: On the RenderThread, when creating a PSO we can start compiling an unoptimized version `VK_PIPELINE_CREATE_DISABLE_OPTIMIZATION_BIT` in another thread
  - Hopefully it compiles faster!
  - With enough latency between RenderThread and RHI Thread, might be enough time to hide the hitch!
    - Meanwhile on another thread compile the optimized version and swap once its done

- Plans orthogonal and final solution probably a mix of all
Vulkan RHI: Tools

- You’re only as good as your tools ;)
- Use Vulkan’s Validation Layers!
  - BOLO for yesterday’s BoF on Vulkan Tools Loader and Validation session from Khronos
Vulkan RHI: Tools

• Use RenderDoc!
  – https://renderdoc.org/builds
  – Vital on UE4 for tracking/diagnosing issues
    • Not just for Vulkan! (D3D11, OpenGL)
  – Use Debug Markers and Object Names
    • http://www.saschawillems.de/?page_id=2017
Vulkan RHI: Closing...

• But wait, there’s more!
  – Plans on investigating:
    • Render Subpasses
    • Push Constants
    • Reworking Descriptor Set Layouts

• Drivers are greatly improved, but you’ll still run into BSODs
  – Report bugs to IHVs with repro steps
  – At least get one card from each major vendor
    • Helps you determine if it’s a driver issue or a bug in your code
Thanks!

Rendering, Core Rendering, Mobile Rendering & Platform Teams

Samsung, Qualcomm & Confetti
Porting DOOM to Vulkan

SIGGRAPH 2016
Axel Gneiting
id Software
Agenda

• Demo & short idTech 6 overview
• Porting to Vulkan
  • Shaders, pipelines & states
  • Descriptor Sets
  • Multithreading
  • Image layouts & barriers
  • Memory & synchronization
  • Asynchronous compute
• Results & Future Work
idTech 6

- PC OpenGL & Vulkan, PS4, Xbox One
- DOOM and future id Software titles
- 60+ Hz on all Platforms
- Shader syntax similar to HLSL
  - Translated to PSSL/HLSL/GLSL at build time
CPU

- Parallel command buffer generation
  - Split up into several “contexts” per frame
  - Each contexts owns command buffer
  - For each context we run multiple jobs to fill CB
  - Last job in frame submits command buffers to GPU

- OpenGL runs sequential on one thread
  - Some scene preparation work is still in jobs
GPU

- Clustered forward shading with some deferred
- Same shader for most of the geometry
  - Same set of textures too (virtual texturing)
  - Very few state changes
- Extensive post process
  - DoF, Temporal AA, SSDO, motion blur, etc.
- Lots of asynchronous compute
  - DXT encode, particles & post processing
Porting to Vulkan

- Started 2015 with an early version
  - Wrote most of the Vulkan backend code
  - Got first triangle rendering
- Picked it up in late March 2016 again
- Was mostly running at game launch
  - RenderDoc helps, even better now!
- Small issues delaying release 😞
  - Driver issues
  - Swap chain surprisingly hard to get right
Porting to Vulkan

- Validation layers were unreliable back then
- Lots of false errors
- Had to write some validation code ourselves
- Validation layers much better now
- Still good to have own validation for debugging
Shaders

- Already had GLSL translator
  - But OpenGL was binding by name
  - Vulkan uses binding IDs at pipeline creation
- Using AMD extensions if available
  - Variant for all shaders
  - AMD_shader_ballot & AMD_gcn_shader
Shaders

• Normalized clip space is upside down
  • Shader generator adds gl_Position.y = -gl_Position.y at end of every vertex program
  • Can we please have an extension that fixes this?
  • Platform differences are a waste of time

• Z range is good: [0,1] 😊
Pipelines & States

- Abstraction layer still old style API like
- Need to emulate stateful API & track states
- Hash table for pipelines, render passes & frame buffer states
  - Way smaller perf overhead than thought
- Dynamic state for scissor/viewport/stencil and depth bias
- Only ~350 total graphics pipelines for entire game
Pipelines & States

• Pipeline creation expensive
  • Lookup misses unacceptable at runtime
  • Some pipelines take 100+ ms to compile

• Solution
  • Play game and serialize states to disk
  • On startup launch jobs to compile pipelines
  • Fairly robust, missed pipelines would just cause stalls for player
Descriptor Sets

• No deletion of Vulkan objects while playing
  • Geometry statically loaded
  • Textures virtualized
• Got away with a descriptor hash table
• One big descriptor set for each combination
• Complete table flush if a Vulkan handle gets deleted
  • Level load & unload, etc.
• About 3-4k descriptor sets usually
Descriptor Sets

- Dynamic uniforms written to ring buffer
- Thread safe allocation from ring with atomics
  - 256 byte align allocations for simplicity
- Bound with UNIFORM_BUFFER_DYNAMIC
  - Offset set as vkCmdBindDescriptorSets parameter
- Also used UNIFORM_BUFFER_DYNAMIC for skinning data
  - Baked range problematic
  - Got away with 64kB range for everything
  - Alternative would have been way more descriptor sets
Multithreading

• Mostly straight forward port from consoles
• Image layouts problematic (more soon)
• Double buffered CBs per context
• Read/write locks for state hash tables
  • Never blocks if no state misses
Image layouts & barriers

• Image layouts were a big headache
  • 25+ barriers per frame
  • Hundreds of layout changes

• Combining as many barriers as possible

• Knowing last image state difficult
  • We only specify the new state in code

• But parallelism makes complete automatic tracking impossible
Image layouts & barriers

- Automatic tracking inside each context / CB
- Not many images used across CBs
- Start of frame: Set state for start of CB to fix up missing tracking
- End of frame:
  - Go over transitions & determine initial next frame state
  - Validate image transitions
- No vkCmdSetEvent/vkCmdWaitEvents right now
Image layouts & barriers

Context 1

- **SHADER_READ**
- **ATTACHMENT_WRITE**

Context 2

- **ATTACHMENT_WRITE**
- **SHADER_READ**

CPU

---

**SIGGRAPH 2016**
Memory

• Simple block allocator
  • Split into max 128 MB pieces
  • Try smaller allocation until allocation succeeds
  • Or falls back to system memory if allocations fail in VRAM
  • Resizable images allocated individually

• NVIDIA problematic under pressure (2GB)
  • Lots of fixes in driver by now
  • Use NV_dedicated_allocation if possible
Memory

• All uploads through common manager
• Double buffered host staging memory
• Each staging buffer associated with
  • Command buffer
  • Fence
• If buffer is full, write fence at end of CB and submit
• Wait on fence before reuse
• Flush host visible ranges before graphics submits
Synchronization

• Double buffering everywhere
  • Wait for command buffer fence on CPU
  • Minimizes latency

• GPUView is your friend!
  • Much more useful than with OpenGL/DX11

• Swap chains are tricky
  • Make sure acquire & present always matching
  • Acquire as late as possible (avoids stalls)
Asynchronous Compute

- Useful for leveraging wasted GPU idle time
  - E.g. during shadow & depth pass
- GPU particles & post process
- Post process overlaps with beginning of next frame
  - Present from compute queue on AMD
  - NVIDIA still working on driver support
- Using SHARING_MODE_CONCURRENT for render targets
  - Careful, might be slower
Results

• Very pleased with performance gains
• 60%-70% in some scenes on AMD in GPU limit
  • Faster than OpenGL even without async/intrinsics
• NVIDIA GPU time about the same
• Render CPU limit is mostly gone
  • People reporting 60+ Hz in power saving mode
• Lots of potential
Future Work

• Prepare image barriers & layouts at beginning of frame
• Remove hashes and make high level code aware of states
• Know exactly what pipelines are used in game
• Better use of render passes (sub passes, layout transitions)
Future Work

- Split barriers (vkCmdSetEvent/vkCmdWaitEvents)
- Command buffer reuse (e.g. deferred passes & post process)
- More asynchronous compute
- Asynchronous transfers
Thanks

- Jean Geffroy, Tiago Sousa, Billy Khan & the whole team at id Software
- Baldur Karlsson for RenderDoc
- AMD and NVIDIA for help on Vulkan port
- Make sure to play the game!
We are Hiring

• Various openings across Zenimax Studios!
• Please visit https://jobs.zenimax.com
Panel: Best Practices for Programming to the Vulkan API

Chris Hebert  
Developer of Technology Engineer  
Optimizing Cuda, OpenGL, & Vulkan for ISVs targeting Nvidia HW

Axel Gneiting  
Senior Engine Programmer  
Ported Doom to Vulkan

Tobias Hector  
Software Design Engineer, PowerVR API and Extension Development

Rolando Caloca  
Sr. Rendering Engineer  
Vulkan port of Unreal Engine 4

Dan Archard  
Principal Engineer, ACG Team  
Getting the most out of Vulkan on Qualcomm HW
Memory Transfers and Pipeline Barriers

Chris Hebert
Developer of Technology Engineer
Moving Forward with Vulkan
Pipelining Memory Operations

Chris Hebert, Dev Tech Software Engineer, Professional Visualization
Agenda

- CPU -> GPU Transfers
- Pipeline Barriers
CPU->GPU Transfers
Low-level memory control

Console-like access to memory

Vulkan exposes several physical memory pools - device memory, host visible, etc.

Application binds buffer and image virtual memory to physical memory

Application is responsible for sub-allocation
## Resource management
### Allocation and Sub-allocation

<table>
<thead>
<tr>
<th>HEAP supporting A, B</th>
<th>HEAP supporting B</th>
</tr>
</thead>
</table>

Allocate memory type from heap

<table>
<thead>
<tr>
<th>Allocation Type A</th>
<th>Allocation Type B</th>
<th>...</th>
</tr>
</thead>
</table>

Query resource about size, alignment & type requirements

Assign memory subregion to a resource (allows aliasing)

<table>
<thead>
<tr>
<th>Image</th>
<th>...</th>
<th>...</th>
<th>Buffer</th>
</tr>
</thead>
</table>

Create resource views on subranges of a buffer or image (array slices...)

BufferView BufferView
Resources

Give Vulkan something to work with

Vulkan exposes several heaps of different types

Vulkan heaps support different properties

- **VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT**: Fastest to access from GPU
- **VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT**: Slower but visible from CPU
- **VK_MEMORY_PROPERTY_HOST_COHERENT_BIT**: No need to flush/invalidate
- **VK_MEMORY_PROPERTY_HOST_CACHED_BIT**: Faster, may need to flush/invalidate
- **VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT**: Device only, but allocated at a later time
Resources
PCIe vs SoC (UMA)

**HOST_VISIBLE OR DEVICE_LOCAL**

- Type 1: DEVICE_LOCAL
- Type 2: HOST_VISIBLE | HOST_COHERENT
- Type 3: HOST_COHERENT | LAZYILI_ALLOCATED

**HOST_VISIBLE AND DEVICE_LOCAL**

- Type 1: DEVICE_LOCAL
- Type 2: DEVICE_LOCAL | HOST_VISIBLE | HOST_COHERENT
- Type 3: DEVICE_LOCAL | HOST_VISIBLE | HOST_CACHED
Staging memory
Using staging buffers

Host Visible Memory (slower)

Map Memory & Copy

HOST

Copy using graphics or DMA queue

Copy

Device Local Memory (fast!)
Staging memory
Using staging buffers

HOST

Map Memory & Copy

Copy using graphics or DMA queue
Is my memory ready to copy to the device?
Not necessarily.....

Host Visible Memory (slower)

Copy

Device Local Memory (fast!)
If `VK_MEMORY_PROPERTY_HOST_COHERENT_BIT` is supported on the heap, then no need to flush.

Otherwise, blocking call to:

```c
VkResult vkFlushMappedMemoryRanges(
    VkDevice device,
    uint32_t memoryRangeCount,
    const VkMappedMemoryRange* pMemoryRanges);
```

Will flush any memory still to be written.
Staging memory
Using staging buffers

Now we know memory is written to host visible mem, Copy using graphics or DMA queue

HOST

Map Memory & Copy

Host Visible Memory (slower)

Copy

Device Local Memory (fast!)
Memory synchronisation
Using pipeline barriers

In any application, both reads from and writes to memory take place frequently. Potential for hazards even in single thread.

Examples (by no means exhaustive):

- Staging large uniform or vertex buffer updates
- Reading from texture rendered to in a previous pass
- Staging large buffer for compute work.
Staging memory
Using pipeline barriers

Copy using graphics or DMA queue

But is our memory actually here yet?

Command Buffer(s)

Read from device memory
In some pipeline stage
Staging memory
Using pipeline barriers

Insert a `vkCmdPipelineBarrier` into the command buffer

Memory → Copy → Device Local Memory (fast!) → Read from device memory in some pipeline stage
Staging memory
Using pipeline barriers

```c
void vkCmdPipelineBarrier(
  VkCommandBuffer commandBuffer,
  VkPipelineStageFlags srcStageMask,
  VkPipelineStageFlags dstStageMask,
  VkDependencyFlags dependencyFlags,
  uint32_t memoryBarrierCount, const VkMemoryBarrier* pMemoryBarriers,
  uint32_t bufferMemoryBarrierCount, const VkBufferMemoryBarrier* pBufferMemoryBarriers,
  uint32_t imageMemoryBarrierCount, const VkImageMemoryBarrier* pImageMemoryBarriers);
```

All of these must be complete.....
... before any of these execute.

(e.g.
VK_PIPELINE_STAGE_VERTEX_INPUT_BIT
VK_PIPELINE_STAGE_VERTEX_SHADER_BIT
VK_PIPELINE_STAGE_TRANSFER_BIT)
Staging memory
Using pipeline barriers

Can take arrays of:

- **VkMemoryBarrier**: Global barrier for all memory types
- **VkBufferMemoryBarrier**: Scoped to a range defined by the buffer
- **VkImageMemoryBarrier**: Can also perform layout transitions (where applicable)

```c
typedef struct VkMemoryBarrier {
    VkStructureType sType;
    const void* pNext;
    VkAccessFlags srcAccessMask;
    VkAccessFlags dstAccessMask;
} VkMemoryBarrier;
```

All of these must complete with the `srcStageMask` of the pipeline barrier
All of these must complete with the `dstStageMask` of the pipeline barrier

E.g.
- `VK_ACCESS_SHADER_READ_BIT`
- `VK_ACCESS_SHADER_WRITE_BIT`
- `VK_ACCESS_COLOR_ATTACHMENT_READ_BIT`
- `VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT`
Updating Buffers

vkCmdUpdateBuffer

Great for UBO’s or small VBO’s
No need to stage
Better for the performance path
Limited to 64k transfers
Still treated as transfer operation; use a memory barrier
Must take place outside of a render pass

```c
void vkCmdUpdateBuffer(
    VkCommandBuffer commandBuffer,
    VkBuffer dstBuffer,
    VkDeviceSize dstOffset,
    VkDeviceSize dataSize,
    const uint32_t* pData);
```
Optimal Transfers

A few tips.

- Keep transfers to a minimum
  - Batch if possible
- Keep data on the GPU if possible
  - Use compute for updates, pass parameters as push constants
- Try to keep transfers off the performance path
  - Transfer when you have time.
- Use barriers as late as possible
  - Don’t hold up the queue unnecessarily
- Ping Pong/Double Buffer
  - Use one buffer while the other transfers
Conclusion

Takeaways

Vulkan memory is programmable
Sub allocate whenever feasible
Use the right heap for the right job
Stage memory to fastest heap where appropriate
Make sure caches are flushed when you need the memory
Make sure transfers are complete when you need the memory
Keep transfers to a minimum and off the performance path
Thank You

Enjoy Vulkan!!
Questions?

Chris Hebert, Dev Tech Software Engineer, Professional Visualization
RenderPass Usage

Tobias Hector
Software Design Engineer
Best Practices: Render Passes & Scheduling

Tobias Hector, Leading Software Design Engineer
27th July, 2016
What is a Render Pass?

- **Unique feature of Vulkan**
  - Allows multiple passes to be scheduled efficiently
  - Explicitly calls out how tile-based GPUs should operate

- **Benefits across all GPUs**
  - Scheduling benefits on all GPUs
  - Bandwidth and memory savings on tile-based GPUs

- **Huge enabler for portability**
  - Best way to do e.g. Deferred Shading, for all vendors
  - No need for vendor-specific extensions (e.g. Pixel Local Storage)
Efficient scheduling

- Scheduling work is involved
  - Need to consider exactly when things need to happen

- Scheduling effectively means having knowledge of the future
  - Synchronization primitives describe the present and past
  - Requires very careful app management
Render pass dependencies

- Render passes describe future work
  - Dependencies between sub passes
  - No implicit order between sub passes

- Drivers can compile these structures
  - Can construct an optimised dependency graph
  - Future work can be scheduled extremely efficiently

- Render pass instances use this graph
  - Acts as a framework in which to execute draw commands
Additional benefits

As if that wasn’t enough…

- **Tile-based GPUs get an extra boost**
  - Sub passes can be merged – keeping G-Buffer-like data completely on-chip
  - No bandwidth required!
  - Some direct renderers may avoid cache flushes
  - Savings on the order of GB/s

- **If you don’t need to read/write from RAM…**
  - Then don’t even allocate attachments in the first place
  - Can represent significant memory savings for high resolutions
  - E.g. One 1080p RGBA8 attachment is ~8MB
Best Practices

- **Put as much possible in as few render passes as possible**
  - Even passes that don’t depend on each other!
  - E.g. Multiple shadow map generation passes
  - Most apps should need just 1 or 2!

- **Use subpass dependencies**
  - Instead of barriers or events

- **Use initialLayout/finalLayout**
  - Instead of explicit image transitions
Best Practices

- Use Load and Store Ops!
  - Use DONT_CARE liberally
  - Use CLEAR instead of vkCmdClearAttachment/vkCmdClearImage

- Use MSAA resolve attachments
  - Instead of vkCmdResolveImage

- Use TRANSIENT_ATTACHMENT_BIT and LAZILY_ALLOCATED_MEMORY
  - No need to allocate memory on some architectures!
Conclusion

- Render passes are awesome
  - We’re going to continue to make them even more awesome

- You should definitely use them
  - They are not scary or difficult, I promise
    - (well, no more than Vulkan already is…)

- If you have any questions, please ask me!
  - Either during the panel or afterwards
  - I’m very friendly
  - Also on twitter: @TobskiHectov
Pipeline State Object Caching

Dan Archard
Principal Engineer, ACG
QCT
July 11, 2016
Why do we care?

• ... because it’s one of the easiest optimizations you’ll ever make!
• Perfect PSO creation isn’t always viable
  • DX9/DX11 rendering interface, script driven rendering state etc.
  • PSOs created on the fly are the reality
• Creating pipelines can be SLOOOOOOOOOOOWWWWWWW!
  • ... so it hitches like crazy
• There’s a bunch of redundant work happening during PSO creation
  • GLES took care of this for you
• Use case from Epic Games Protostar
PSO create time break-down

Epic Games Protostar*

- Linking: 56%
- Compilation: 42%
- All other PSO processing: 2%
Redundancy
Epic Game Protostar*

Compile
- Unique Compile 38%
- Redundant Compile 62%

Link
- Unique Link 54%
- Redundant Link 46%
Possible solutions to speed up PSO creation

Dynamic Pipeline State
- Limited what state can change

Derived Pipelines
- Vendor specific
- Difficult to plug in to most engines

Pipeline State Cache
Creating a pipeline

 VkGraphicsPipelineCreateInfo pipelineCreateInfo = {};
createInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
// ...

VkPipeline pipeline;

VkResult vkCreateGraphicsPipelines(
    device,
    VK_NULL_HANDLE,
    1,
    &pipelineCreateInfo,
    nullptr,
    pipeline);
Pipeline Cache
Creating a pipeline using a cache

```c
static VkPipelineCache pipelineCache;

VkPipelineCacheCreateInfo pipelineCacheCreateInfo = {};    
pipelineCacheCreateInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_CACHE_CREATE_INFO;

VkResult result = vkCreatePipelineCache(          
    device, // VkDevice          
    &pipelineCacheCreateInfo, // const VkPipelineCacheCreateInfo*    
    nullptr, // const VkAllocationCallbacks*  
    &pipelineCache); // VkPipelineCache*

// ....

VkGraphicsPipelineCreateInfo createInfo = {};    
createInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;  
// ...

VkPipeline pipeline;

VkResult result = vkCreateGraphicsPipelines(          
    device, // VkDevice          
    &pipelineCache, // VkPipelineCache    
    1, // uint32_t    
    &createInfo, // const VkGraphicsPipelineCreateInfo*  
    nullptr, // const VkAllocationCallbacks*  
    pipeline); // VkPipeline*  
```

40
Pipeline Cache
Creating a pipeline using a cache

Total PSO Create Time - Epic Games Protostar*

No Cache Using Cache

Compile Link Driver Overhead Cache Overhead
Pipeline Cache

Loading from disk

- Pipeline cache can take initial data on create
- Save & Restore cache across runs:

```cpp
VkPipelineCache pipelineCache;

VkPipelineCacheCreateInfo createInfo = {};  // VkPipelineCacheCreateInfo
createInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_CACHE_CREATE_INFO;
createInfo.pInitialData = LoadPipelineCacheFromDisk(createInfo.initialDataSize);

VkResult result = vkCreatePipelineCache(device,  // VkDevice
&createInfo,  // const VkPipelineCacheCreateInfo* pCreateInfo,
nullptr,  // const VkAllocationCallbacks* pAllocator,
&pipelineCache);  // VkPipelineCache* pPipelineCache);
```
Pipeline Cache

Loading from disk

Total PSO Create Time - Epic Games Protostar*

No Cache

Using Cache

Cache With Initial Data

Compile  Link  Driver Overhead  Cache Overhead
Thank you

Follow us on: f t i t
For more information, visit us at:
www.qualcomm.com & www.qualcomm.com/blog
Panel: Tools for the Vulkan Ecosystem

Bill Hollings
Architect
MoltenVK: Vulkan on iOS/macOS

Karl Schultz
Principal Engineer
LunarG SDK and Tools

Kyle Spagnoli
Engineer
Bringing Vulkan support to NVIDIA® Nsight™

Andrew Woloszyn
Software Engineer
SPIR-V Tools
Vulkan on iOS/macOS

Bill Hollings
Architect
Vulkan on iOS & macOS

Bill Hollings, The Brenwill Workshop Ltd.
July 2016
MoltenVK

- MoltenVK is an implementation of Vulkan on iOS & macOS
  - Built on Metal

- Vulkan & Metal are static-state, command-buffer APIs
  - Very little friction
  - MoltenVK minimal overhead

- MoltenVK feature set dependent on Metal
  - Metal’s focus is on providing a convenient API
  - MoltenVK helps define x-platform compatibility
Xcode Profiling Tools - GPU Frame Capture

- Apple’s strong focus on ecosystem developer tools
  - Apple committed to Metal
  - MoltenVK leverages this

- GPU Frame Capture
  - Vulkan command sequence
  - Capture rendering stages
  - Cmd buffs & renderpasses
  - Pipeline state & shaders
  - Resources & render state
  - Identifies inefficiencies

- Manual or programmatic
  - Trace setup activity
Xcode Profiling Tools - Metal System Trace

- **Metal System Trace**
  - Detailed tracing of CPU & GPU activity per frame
  - Separates per-frame loads
  - Identifies utilization shortfalls:
    - blocking,
    - device starvation
    - sync issues
Xcode Profiling Tools - Other

- **GPU Driver**
  - CPU & GPU performance monitoring

- **Allocations and Leaks**
  - CPU memory allocation details
  - Identify memory leak details

- **These tools available to Vulkan developers**
  - Apple provides a sophisticated suite of tools for graphics developers using Apple’s ecosystem.
  - MoltenVK makes all of these tools available to Vulkan developers.
Bringing Vulkan Support to NVIDIA® Nsight™

Kyle Spagnoli
Engineer
NSIGHT VISUAL STUDIO EDITION 5.2
Vulkan, VR, and Advanced Graphics Profiling

- Vulkan API support
- New Range Profiler, including DX12
- New Geometry View
- Oculus VR SDK support
- CUDA 8.0 support
MULTI-THREAD / MULTI-QUEUE

Recording Command Buffers

Scrubby shows all threads for command buffer construction

Events view shows entry for in-frame command buffer construction
## MULTI-THREAD/MULTI-QUEUE

**Executing Command Buffers**

Scrubber highlights multiple queues. This application uses one for compute and one for graphics.

Scrubber shows queue as it migrates from thread to thread.
CURRENT RENDER TARGET DISPLAY

Dig Into Per Pass Rendering Results

View each render target for any draw call in flight

Wireframe highlights rendered geometry
BARRIER INFORMATION
Managing Rendering Passes & Resource Transitions

Details for each pipeline barrier and what resources/stages are impacted

```c
void vkCmdPipelineBarrier(
    VkCommandBuffer commandBuffer = 0x0d376c20,
    VkPipelineStageFlags srcStageMask = VkPipelineStageFlags(
        VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT | VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT),
    VkPipelineStageFlags dstStageMask = VkPipelineStageFlags(
        VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT),
    VkDependencyFlags dependencyFlags = VkDependencyFlags(0),
    uint32_t memoryBarrierCount = 1u,
    VkMemoryBarrier pMemoryBarriers = {{VkMemoryBarrier::sType = VK_STRUCTURE_TYPE_MEMORY_BARRIER,
    VkMemoryBarrier::pNext = nullptr,
    VkMemoryBarrier::srcAccessMask = VkAccessFlags(
        VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT | VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT),
    VkMemoryBarrier::dstAccessMask = VkAccessFlags(
        VK_ACCESS_INPUT_ATTACHMENT_READ_BIT | VK_ACCESS_SHADER_READ_BIT)),
    uint32_t bufferMemoryBarrierCount = 0u,
    VkBufferMemoryBarrier pBufferMemoryBarriers = nullptr,
    uint32_t imageMemoryBarrierCount = 0u,
    VkImageMemoryBarrier pImageMemoryBarriers = nullptr))
```
FENCES, SIGNALS & SEMAPHORES

Synchronization Primitives

Highlight synchronization points involving fences, events, and semaphores
API INSPECTOR

View API State
## DEVICE MEMORY

### Visualize Memory Usage & Layout

All memory at a glance

Listing of contained resources

Visual resource layout
SERIALIZATION

Generate Source Code For A Single Frame

C++ code compiles into...
ROADMAP & AVAILABILITY

Upcoming release

NSIGHT Visual Studio Edition 5.2 with Vulkan Support

• Available when you return from SIGGRAPH
• C++ Serialization is a beta feature

Additions to come:

• Performance Info & Range Profiler
• Android Support
• Linux Support
• Shader Editing
• Analysis & Hints

• Shader Reflection Information
• Sparse Texture
• Improved Barrier GUI
• Support Future Extensions
Thank you!

Check out our demo during the Khronos After Party for a hands on Vulkan demo of Nsight + DOOM

Test Drive Vulkan Support @ Booth #509
LunarG Vulkan SDK and Tools

Karl Schultz
Principal Engineer
LunarG SDK and Tools

Karl Schultz, LunarG, Inc.
SIGGRAPH – Vulkan Tools Roundtable
July 2016
Vulkan SDK

• Current release based on Vulkan spec/header 1.0.21
  – Released on July 21
• Cadence is approximately monthly right now
• Derived from public GitHub repos
• Value-add:
  – Components tested and verified
  – “One-stop shop”
  – Easy install
Vulkan SDK Tools

• We'll be talking about:
  – API Dump, Screenshot, vktrace/vkreplay, vktraceviewer, RenderDoc

• Other parts of the SDK, not discussed here:
  – Loader and Validation Layers
    • Covered in Tuesday BOF
    • Check out recordings if you missed it
    • “Vulkan Validation Layers Deep Dive” Webinar coming, probably September 27
  – Vulkan header files
  – Vulkan Spec docs
  – Samples / demos
$ VK_INSTANCE_LAYERS=VK_LAYER_LUNARG_api_dump ./tri

vkCreateInstance(pCreateInfo = 0x7ffedd58e9c0, pAllocator = 0x0, *pInstance = 0x2014710) = VK_SUCCESS

pCreateInfo:
  sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO
  pNext = 0x7ffedd58e9a0
  flags = 0x0
  pApplicationInfo = 0x7ffedd58eba0
  enabledLayerCount = 0x0
  ppEnabledLayerNames = 0x0
  enabledExtensionCount = 0x2
  ppEnabledExtensionNames = 0x7ffedd58f140

pApplicationInfo:
  sType = VK_STRUCTURE_TYPE_APPLICATION_INFO
  pNext = 0x0
  pApplicationName = tri
  applicationVersion = 0
  pEngineName = tri
  engineVersion = 0
  apiVersion = 4194304
  pNext:

t{0} vkEnumeratePhysicalDevices(instance = 0x2014710, *pPhysicalDeviceCount = 0x1, pPhysicalDevices = 0x0) = VK_SUCCESS

t{0} vkEnumeratePhysicalDevices(instance = 0x2014710, *pPhysicalDeviceCount = 0x1, *pPhysicalDevices = 0x2216600) = VK_SUCCESS

- Implemented as a Vulkan layer
- Writes API calls out as text output
- Good for seeing what led up to a problem
• Implemented as a Vulkan layer
• These commands capture the 5\textsuperscript{th} frame and store it in 5.ppm
• Vktrace (next slide) can also take screenshots using this layer
 Vktrace sets environment to load vktrace layer and then launches app as a child process
 Vktrace layer serializes Vulkan API calls and records them into a file
 Vkreplay plays back the vktrace file
 Work in Progress:
  - WSI mapping – allows recording on one window system and playback on another
  - OS mapping – handle OS-specific issues like structure packing
  - GPU mapping – handle differences in GPU capabilities and physical limits
  - Other issues and features – See VulkanTools GitHub
VkTrace Viewer – Interactive vktrace File Explorer

• Developer: Peter Lohrmann
• Pretty cool tool to look at vktrace files
• Coming in future LunarG SDK
• But code is in the LunarG VulkanTools repo
  – Windows version currently in better shape than the Linux version
  – Needs Qt to build
• Features
  – Load existing vktrace files
  – Start an app to generate a vktrace file
  – Replay a vktrace file
  – Single-step through a vktrace file
  – Examine vktrace packet detail
  – Run to a specific packet
Essentially the same as running `vktrace` from the command line

Or open an existing `vktrace` file from the File menu
VkTrace Viewer
Examine Trace Initial Screen

• Comes up right after you create the trace
• Packets are shown in the bar graph
• A red packet is taking a long time
• This one is the first Present
• Note API call list panel
• “Prev DC” and “Next DC” are for Draw Calls
VkTrace Viewer
Examine Trace
One Frame

- Zoomed in graph to show about 3 frames
- API call window shows calls for 1 frame
- 12 API calls
- Present through QueueSubmit shown here
- Note Trace Stats panel
VkTrace Viewer
Examine Trace with Hover

- Hover over a call in the API Call frame
- Packet header info displays
- Also some parameter and structure data
VkTrace Viewer
Replay / Step
RenderDoc

- Developer: Baldur Karlsson
- Shipped in LunarG Windows SDK
- https://github.com/baldurk/renderdoc
- Popular for D3D11 and OpenGL
- Vulkan Support has been added
- No Linux GUI yet
- Cannot possibly do justice to it here – check out video tutorials on YouTube, etc
SPIR-V Tools

Andrew Woloszyn
Software Engineer
SPIR-V Tooling

- SPIR-V is the binary intermediate language used for Compute Kernels in OpenCL and Shaders in Vulkan.
  - Easy to parse SSA form.
  - Retains high-level information.
  - Contains enough information to allow useful reflection of the binary.
Compilation

- **Glslang** [https://github.com/khronosgroup/glslang](https://github.com/khronosgroup/glslang)
  - Reference Glsl -> SPIR-V compiler.
  - Compile a fragment shader: `glslangValidator -V foo.frag -o output.spv`
  - Output generated assembly: `glslangValidator -H foo.frag`
  - Can be used as a library for online compilation.

- **Shaderc** [https://github.com/google/shaderc]
  - Wrapper around the reference compiler (glslang)
  - Provides a gcc/clang-like command-line interface.
  - Adds support for both `<>` and `""` includes.
  - Adds command-line preprocessor defines.
  - Adds `-M` dependency generation.
  - Adds a C and C++ library interface that has all of the functionality of the command-line tool.
  - Compile a fragment shader: `glslc -fshader-stage=fragment foo.glsl -o a.spv`
void main() {
    out_color = in_color;
}

ccat /Sansbox/shaderc/build/install/bin

ccat in_out.glsl

ifdef COLOR_LOCATION
layout(location = COLOR_LOCATION) in vec4 in_color;
#endif
layout(location = 0) out vec4 out_color;

ccat /Sansbox/shaderc/build/install/bin

glslc -o foo.spy -fshader-stage=fragment foo.glsl -DCOLOR_LOCATION=2

ccat /Sansbox/shaderc/build/install/bin
SPIRV-Tools

• A collection of command-line tools and libraries for handling SPIR-V.

  • spirv-dis
    - Takes a SPIR-V module and produces a human-readable format similar to llvm.

  • spirv-as
    - Takes the human-readable format and turns it back into a SPIR-V module.

  • spirv-val (Not Yet Complete)
    - Validates that a given SPIR-V module follows all of the rules set out in the spec.

  • spirv-opt
    - Optimization tool and framework for transforming SPIR-V.
      - Currently has a debug info stripping pass.

• Library interfaces to all of these.
OpCapability Shader

OpExtInstImport "GLSL.std.450"
OpMemoryModel Logical GLSL450
OpEntryPoint Fragment %main "main" %out_color %in_color
OpExecutionMode %main OriginUpperLeft
OpSource GLSL 450
OpSourceExtension "GLOGLE_cpp_style_line_directive"
OpSourceExtension "GLOGLE_include_directive"
OpName %main "main"
OpName %out_color "out_color"
OpName %in_color "in_color"
OpDecorate %out_color Location 0
OpDecorate %in_color Location 2

%void = OpTypeVoid
%3 = OpTypeFunction %void
%float = OpTypeFloat 32
%4float = OpTypeVector %float 4
%ptr_Output_v4float = OpTypePointer Output %v4float
%out_color = OpVariable %ptr_Output_v4float Output
%ptr_Input_v4float = OpTypePointer Input %v4float
%in_color = OpVariable %ptr_Input_v4float Input
%main = OpFunction %void None %3
%5 = OpLabel
%12 = OpLoad %v4float %in_color
OpStore %out_color %12
OpReturn
OpFunctionEnd
SPIRV-Cross

- SPIR-V to higher level language conversion tool
  - SPIR-V to GLSL
  - SPIR-V to MSL
  - SPIR-V to C++

- Library interface to do the same
- Reflection api for determining shader resources
layout(location = 0) out vec4 out_color;
layout(location = 2) in vec4 in_color;

void main()
{
    out_color = in_color;
}

awoloszyn ~/Sandbox/spirv-cross/build
What’s needed for the future?

- **Linker**
  - Turn multiple SPIR-V modules into one larger module
  - Size improvements due to merged constants/globals/functions

- **Debug Info**
  - More complete debug information in generated SPIR-V

- **Simulation/Debugging tools**
  - Single-stepping SPIR-V, value examination, ...

- **Optimization Passes**
  - Architecture agnostic optimizations
  - Constant folding, Variable elimination, etc
  - Constant Specialization pass

- **More high-level language support**
  - Work is being done in glslang to support HLSL