The Vulkan Sessions

The Vulkan Working Group (and friends)
SIGGRAPH - July 31, 2019
# The Vulkan Sessions (part I)

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:00</td>
<td><strong>Vulkan Update</strong>&lt;br&gt;• Tom Olson (Arm), Piers Daniell (NVIDIA), Yuriy O’Donnell (Epic Games)**</td>
</tr>
<tr>
<td></td>
<td><strong>Vulkan Timeline Semaphores</strong>&lt;br&gt;• James Jones (NVIDIA)**</td>
</tr>
<tr>
<td></td>
<td><strong>OctaneRender on Vulkan</strong>&lt;br&gt;• Jules Urbach (OTOY)**</td>
</tr>
<tr>
<td></td>
<td><strong>DXC Update: HLSL to SPIR-V for Vulkan</strong>&lt;br&gt;• Ehsan Nasiri (Google)**</td>
</tr>
<tr>
<td>3:40</td>
<td><strong>Break</strong></td>
</tr>
</tbody>
</table>
## The Vulkan Sessions (part II)

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>4:05</td>
<td><strong>Ecosystem Advancements to Aid Vulkan Developers</strong></td>
</tr>
<tr>
<td></td>
<td>• Karen Ghavam &amp; Mark Lobodzinski (LunarG)</td>
</tr>
<tr>
<td>4:05</td>
<td><strong>OpenGL / OpenGl ES Update</strong></td>
</tr>
<tr>
<td></td>
<td>• Piers Daniell (NVIDIA)</td>
</tr>
<tr>
<td>4:05</td>
<td><strong>Zink – OpenGL on Vulkan</strong></td>
</tr>
<tr>
<td></td>
<td>• Eric Faye-Lund (Collabora)</td>
</tr>
<tr>
<td>4:05</td>
<td><strong>ANGLE on Vulkan: Portable OpenGL</strong></td>
</tr>
<tr>
<td></td>
<td>• Jamie Madill (Google)</td>
</tr>
<tr>
<td>4:05</td>
<td><strong>SwiftShader Vulkan</strong></td>
</tr>
<tr>
<td></td>
<td>• Alexis Hetu (Google)</td>
</tr>
<tr>
<td>5:20</td>
<td><strong>Break</strong></td>
</tr>
<tr>
<td>5:30</td>
<td><strong>Par-tay!</strong></td>
</tr>
</tbody>
</table>
Vulkan Update

Tom Olson, Arm
Chair, Vulkan Working Group

Piers Daniell, NVIDIA
Member, Machine Learning TSG
Chair, OpenGL / OpenGL ES WG

Yuriy O’Donnell, Epic Games
Member, Ray Tracing TSG
Vulkan Update - Outline

Platform News

New Applications

Working Group Activities

Task Sub Group (TSG) Updates
Platform news: Stadia

Google’s game streaming platform
- Announced at GDC 2019
- Will go live in November 2019
- Console-in-the-cloud
- Vulkan 1.1 is standard

GDC 2019: The line to get into the keynote...

Stadia support in Vulkan
- VK_EXT_pipeline_creation_feedback
Platform news: Android Q

Introducing Android Q Beta
13 March 2019

Posted by Dave Burke, VP of Engineering


Vulkan everywhere

We're continuing to expand the impact of Vulkan on Android, our implementation of the low-overhead, cross-platform API for high-performance 3D graphics. Our goal is to make Vulkan on Android a broadly supported and consistent developer API for graphics. We're working together with our device manufacturer partners to make Vulkan 1.1 a requirement on all 64-bit devices running Android Q and higher.
Platform news: Vulkan on MacOS / iOS

Vulkan Portability Initiative
- Expose a Vulkan subset on closed platforms
- Shim libraries
- Shader translation tools
- SDK and validation

Current focus
- Vulkan conformance test
Lots of new games!
Looking back to SIGGRAPH 2018...
Looking back to SIGGRAPH 2018…

Project Rush – Unveiled at VidCon 2018

Eric Berdahl
Adobe
SIGGRAPH 2019: It’s a product!

Adobe Premiere Rush
- A pro-quality, cross-platform video capture and editing system

Vulkan powered on Android
- Supported on select Samsung, Google, and OnePlus phones
- Several hundred thousand lines of OpenCL C compute shaders
- Translated to SPIR-V by clspv
- Run in Vulkan command buffers

A testament to the power of SPIR-V!
CAD on Vulkan?

People are starting to ask the question...

CAD on Vulkan makes sense
- CAD apps make *lots* of draw calls
- CAD apps are often CPU bound

But CAD needs things Vulkan doesn’t have
- E.g., stippled lines
- VK_EXT_line_rasterization shipped this week
- Other CAD support features under consideration
Emulating other APIs on Vulkan

Vulkan as Hardware Abstraction Layer (HAL)
- A very thin layer over the hardware
- A good target for middleware
- E.g., game engines!

Using Vulkan as a HAL to support other APIs
- ANGLE: OpenGL ES
- Zink: OpenGL
- DXVK: D3D 10/11
- D9vk: D3D 9
- Vkd3d: D3D 12

I’m afraid I can’t do that, Dave
DXVK

DX emulator running on Vulkan
- Supports D3D 10 and D3D 11
- Developed by Philip Rebohle with support from Valve Software
- In open source on GitHub

Extensions created in response to DXVK issues
- VK_EXT_transform_feedback
- VK_EXT_depth_clip_enable
- VK_EXT_host_query_reset
- VK_EXT_texel_buffer_alignment
- VK_EXT_shader_demote_to_helper_invocation
DXVK in action

Valve Software’s Proton for Steam Play
- Supports native PC games on Linux
- See https://www.protondb.com
- Powered by DXVK and WINE

With Proton and Steam Play, many Windows games now work on Linux!

Since the release of Proton only 11 months ago...

51,447 8,833 5,739
reports written games reported games work

https://www.protondb.com
Ecosystem, Tools, Developer Support

Lots of news on Tools, Validation, SDK
- Validation layer consolidation / configuration
- GPU-assisted validation
- SDK and tools progress

Developer Support
- Welcome Kris Rose!
- Dedicated DevRel resource for Vulkan / OpenXR
New functionality highlights

VK_KHR_shader_float16_int8, VK_KHR_shader_float_controls
- Shader arithmetic on reduced precision types
- Query and override rounding mode, denorm handling, etc

VK_EXT_descriptor_indexing
- Bindless access to resources

VK_EXT_buffer_device_address (note: KHR version coming soon)
- Reference objects in buffers via 64-bit handles
- E.g., create linked data structures in GPU memory

VK_KHR_timeline_semaphore (note: Not quite public; coming soon)
New functionality since SIGGRAPH 2018

VK_KHR_depth_stencil_resolve       VK_EXT_fragment_shader_interlock
VK_KHR_driver_properties           VK_EXT_host_query_reset
VK_KHR_imageless_framebuffer       VK_EXT_index_type_uint8
VK_KHR_memory_model                VK_EXT_inline_uniform_block
VK_KHR_shader_atomic_int64         VK_EXT_line_rasterization
VK_KHR_shader_float16_int8         VK_EXT_memory_budget
VK_KHR_shader_float_controls       VK_EXT_memory_priority
VK_KHR_surface_protected_capabilities VK_EXT_pci_bus_info
VK_KHR_uniform_buffer_standard_layout VK_EXT_pipeline_creation_feedback
VK_EXT_astc_decode_mode           VK_EXT_scalar_block_layout
VK_EXT_astc_texture_compression_astc_hdr VK_EXT_separate_stencil_usage
VK_EXT_buffer_device_address      VK_EXT_shader_demote_to_helper_invocation
VK_EXT_calibrated_timestamps     VK_EXT_subgroup_size_control
VK_EXT_depth_clip_enable          VK_EXT_texel_buffer_alignment
VK_EXT_filter_cubic              VK_EXT_transform_feedback
VK_EXT_fragment_density_map      VK_EXT_validation_features
VK_EXT_fragment_density_map      VK_EXT_yccbcrc_image_arrays
Vulkan Video TSG Update

Gabor Sines - AMD
SIGGRAPH - July 31, 2019
Vulkan Video TSG Update

Goal
- Create a series of extensions for video encode/decode in Vulkan

Target release date
- First half 2020

Codec roadmap
- H.264, HEVC in first release
- VP9, AV1 will follow in later releases
Vulkan ML TSG Update

Piers Daniell - NVIDIA
SIGGRAPH - July 31, 2019
Why machine learning in Vulkan?

- Research showcases potential use of machine learning in interactive and high frame rate applications
  - Character animation (phase function neural network, etc.)
  - Full screen image processing (antialiasing, upscaling, inpainting, DLSS, etc.)
  - Non-Player Character bots (AlphaStar, OpenAI Five, etc.)
  - Image generations (GAN, fire & smoke & clouds, etc.)

- Current machine learning solutions have relatively high interop overhead
  - Interop with third party framework (Python TensorFlow, PyTorch, OpenCL, etc.) introduces bubbles where the CPU/GPU are not doing useful work
  - Sharing data with external APIs can be challenging due to difference in memory models and may require additional copies

![Cat! (grumpy?)](image-url)
How to do machine learning in Vulkan?

• This is possible already today
  - Just use compute shaders to implement the various algebra operations.
  - Examples: Tencent/ncnn, Alibaba/MNN, Unity ML, etc.

• Or use compilers which will generate SPIR-V code for you
  - Examples: TVM.AI

• But
  - Writing high efficiency layered matrix multiplications, with various activation functions requires some advanced GPU programming skills, with different solutions for different hardware.
The Vulkan ML TSG (Task SubGroup)

- A new task subgroup at Khronos has been formed to improve the solution space for machine learning in Vulkan
- Includes representatives from many companies
- Goals
  - Investigate proprietary extensions for inclusion into core Vulkan (VK_NV_cooperative_matrix, etc.)
  - Improvements to compute shaders specific to ML needs
  - New cross vendor extensions (meta-commands, etc.)
- If you are interested, please reach out to us: pboudier@nvidia.com

This work is licensed under a Creative Commons Attribution 4.0 International License
VK_NV_cooperative_matrix

- Exposes NVIDIA’s Turing Tensor Cores to Vulkan/SPIR-V
- Accelerates large, low-precision matrix multiplies
- Core compute function for deep learning
- FP16 supported today, UINT8/SINT8 coming soon
- Sample code: https://github.com/jeffbolznv/vk_cooperative_matrix_perf
- Performance on NVIDIA TITAN RTX
  - fp16 matrix math with fp16 accumulation: 100 TFLOPS
Vulkan Ray Tracing TSG Update

Yuriy O’Donnell - Epic Games
SIGGRAPH - July 31, 2019
About Ray Tracing Task SubGroup (TSG)

- Part of the Vulkan Working Group
- Collaboration between many hardware and software vendors
- Immediate and future-looking specification design
- Weekly meetings to discuss proposals and track progress
- Face-to-face meetings several times per year
Ray Tracing TSG Goals

• Standard Vulkan API extension for ray tracing on desktop
  - Path for ray tracing techniques in existing Vulkan renderers
  - Consistent with Vulkan principles
  - High-level shading language and SPIR-V extensions
  - Validation, debugging and profiling support

• Primary target use cases
  - Real-time rendering
  - Hybrid ray tracing / rasterization pipelines

• Other possible applications
  - Interactive visualizations and content workflows
  - Offline rendering and baking pipelines
Ray Tracing TSG Status

• Current specification draft based on VK_NV_ray_tracing extension
  - Working towards few additions
    - Capture/replay support
    - Serialization/deserialization of acceleration structures
    - CPU acceleration structure build
    - Indirect build and trace
    - etc.
  - Recursive ray tracing made optional

• Aiming for substantial compatibility with DirectX Ray Tracing (DXR)
  - While keeping to established Vulkan principles

• Release when 2 conformant implementations available
Give us feedback!

General issues
• https://github.com/KhronosGroup/Vulkan-Ecosystem

Spec and functionality issues
• https://github.com/KhronosGroup/Vulkan-Docs

Validation issues
• https://github.com/KhronosGroup/Vulkan-ValidationLayers

SDK issues
• http://vulkan.lunarg.com

We look forward to hearing from you!