



SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

K H R O N O S
GROUP

ANARI

OpenXR

WebGL

3DCommerce

glTF

Vulkan

Vulkan: Forging Ahead

August 2023



SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

K H R O N O S
GROUP

Vulkan

Vulkan Update

Tom Olson, Arm
Vulkan Working Group Chair

Outline

What is Vulkan? (the short version)

What's new?

- Adoption
- Functionality
- Documentation and support
- Profiles

What's next?

- Roadmap and future specifications



What is Vulkan

Vulkan

A modern API for graphics and compute on GPUs

- Descended from OpenGL / OpenGL ES
- Radically cross-platform
- One API across desktop and mobile

No-compromise focus on performance

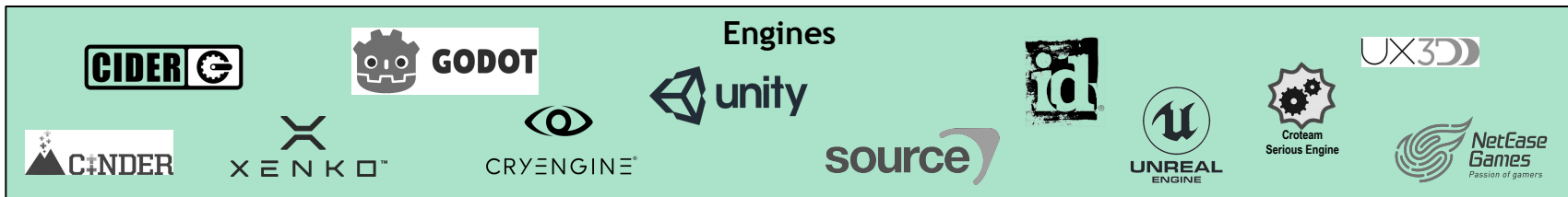
- Driving use case is AAA games

Developer has control / responsibility for

- Memory and object management
- Scheduling and synchronization
- Multithreading
- Error checking



Vulkan Adoption



<http://vulkan.gpuinfo.org/>

Note: The version of Vulkan available will depend on platform and vendor

Vulkan Applications

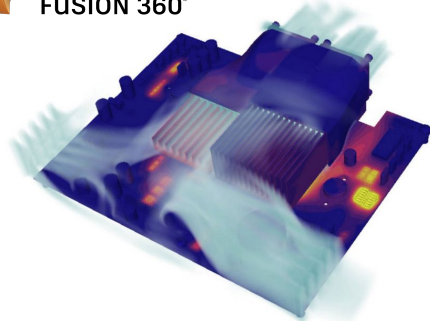


Desktop Games

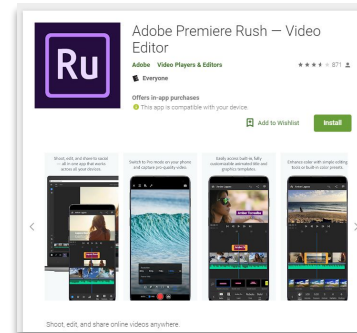


Mobile Games

F AUTODESK®
FUSION 360™



Not Games at All



Vulkan Evolution

Vulkan[®] Roadmap 2022



1.0

2016



1.1

2018



1.2

2020



1.3

2022



What's New: Adoption

Vulkan on MacOS / iOS

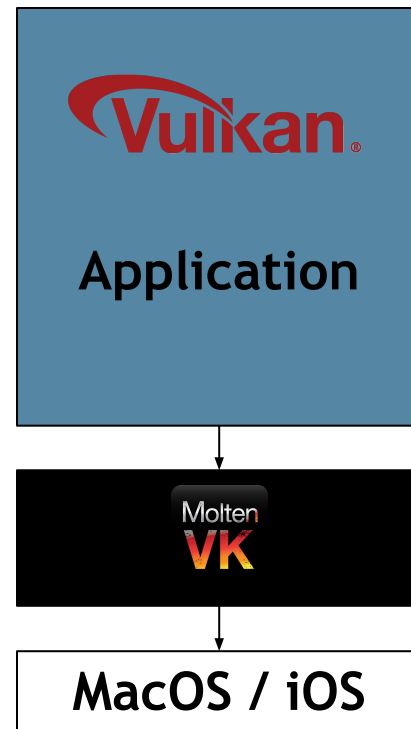


MoltenVK

- Shim library for MacOS/iOS
- Maps Vulkan calls to native Metal API
- Shaders translated using spirv-cross

Current status

- Fully supported in LunarG SDK
- Provides Vulkan 1.2 (portability subset)



Applications and Engines Using MoltenVK

• Games shipping with MoltenVK

- | | | | |
|--------------------------|------------------------------|---------------------|--------------------|
| • DOTA 2 | • DOTA Underlords | • Celeste | • Wreckfest |
| • Metro Exodus | • Aerofly Flight Simulator 2 | • Transport Fever 2 | • Victoria 3 |
| • Final Fantasy XIV | • Path of Exile | • Shadow Warrior 2 | • Artifact |
| • Dark Souls: Remastered | • Raft | • Streets of Rage 4 | • GZDOOM |
| • Dark Souls III | • The Elder Scrolls Online | • Jupiter Hell | • vkQuake/vkQuake2 |

• Games runnable by users via Crossover and MoltenVK

- | | | | |
|------------------------|------------------------------|------------------|---|
| • Halo: Combat Evolved | • World Of Tanks | • Guild Wars 2 | • Age of Empires II: Definitive Edition |
| • God of War (2018) | • Forsaken Remastered | • Battlefield 1 | • Witcher 3 |
| • Grand Theft Auto V | • Elder Scrolls V Skyrim: SE | • Battlefront II | |

• Applications shipping with MoltenVK

- Autodesk Fusion 360
- NAP

• Engines using MoltenVK

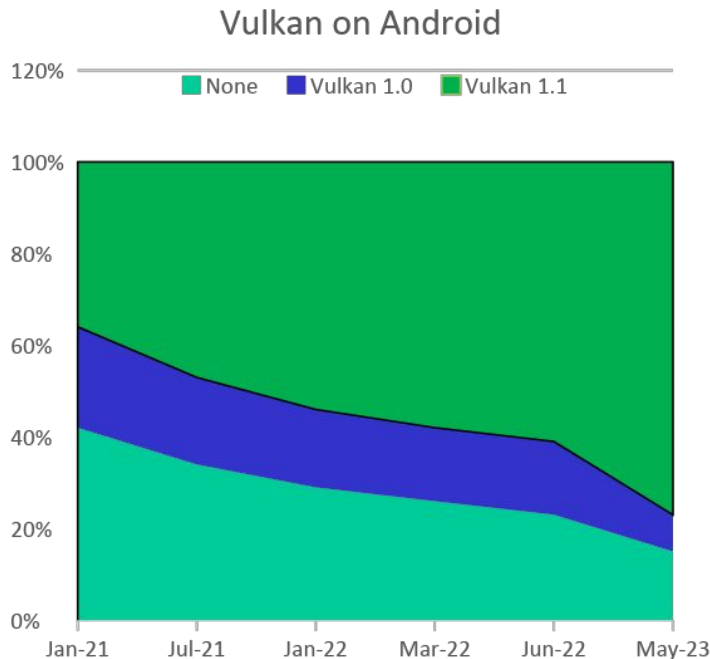
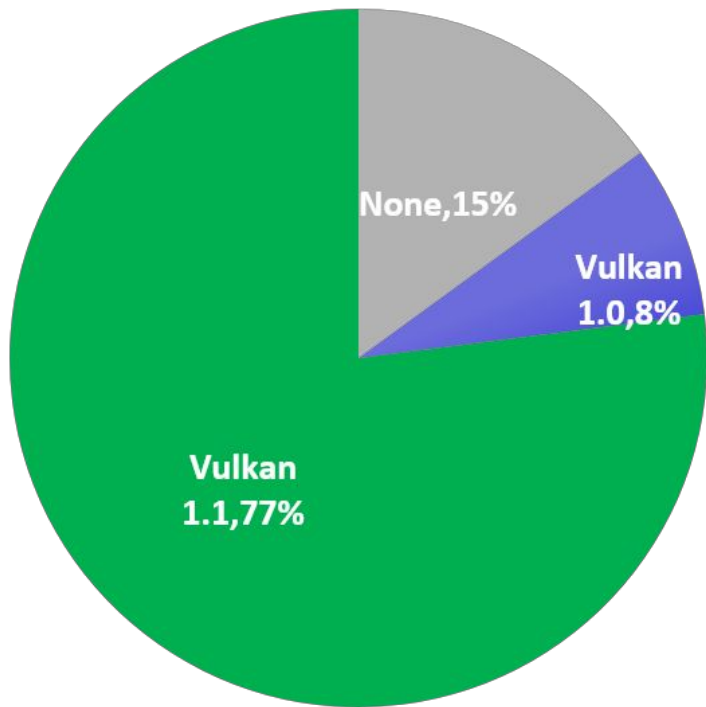
- | | | | |
|-------------------|-------------------------------|-------------------|--------|
| • Google Filament | • Blender Vulkan (PoC) | • Ultra Engine | • Ncnn |
| • Defold | • Clausewitz Engine (Paradox) | • Diligent Engine | • Qt |
| • Acid | • Flax | • Godot | |

• Platform emulators using MoltenVK

- | | | | |
|---------------------------|----------------------------|--------------------|---------------|
| • VKD3D (Direct3D 12) | • Google Android Emulator | • Ryujinx (Switch) | • RPCS3 (PS3) |
| • DXVK (Direct3D 9/10/11) | • Dolphin (Wii & GameCube) | • Cemu (Wii U) | • PCSX2 (PS2) |



Vulkan Adoption on Android



<https://developer.android.com/about/dashboards>

Vulkan is available on 85% of active Android devices



What's New: Functionality

New Extensions

Vulkan Video

VK_KHR_video_queue
VK_KHR_video_decode_queue
VK_KHR_video_decode_h264
VK_KHR_video_decode_h265

Programming model improvements

VK_EXT_attachment_feedback_loop_dynamic_state
VK_EXT_extended_dynamic_state3
VK_EXT_descriptor_buffer
VK_EXT_mutable_descriptor_type

Maintenance

VK_KHR_maintenance5
VK_EXT_depth_bias_control
VK_KHR_map_memory2
VK_EXT_legacy_dithering
VK_EXT_depth_clamp_01
VK_EXT_image_sliced_view_of_3D

Tile-based optimizations

VK_EXT_rasterization_order_attachment_access
VK_EXT_shader_tile_image

Debugging DEVICE_LOST

VK_EXT_device_fault
VK_EXT_device_address_binding_report

Window System Integration

VK_EXT_surface_maintenance1
VK_EXT_swapchain_maintenance1
VK_EXT_pipeline_protected_access

Exploratory / Experimental

VK_EXT_mesh_shader
VK_EXT_shader_object
VK_AMD_shader_enqueue

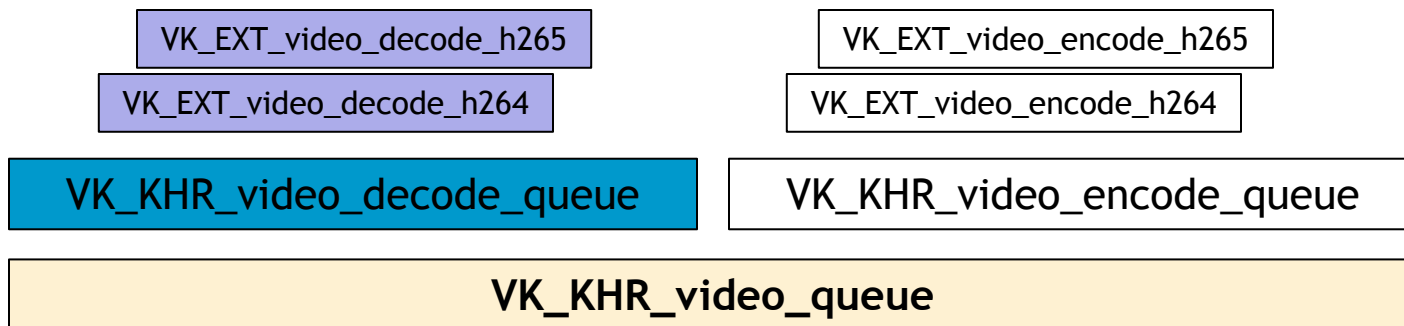
Other

VK_EXT_host_image_copy
VK_KHR_ray_tracing_position_fetch
VK_EXT_pipeline_library_group_handles

Vulkan Video Extensions

Decode stack is now final

- AMD, Intel, NVIDIA shipping drivers
- Adopted in FFMPEG, used in the MPV player
- Work ongoing in GStreamer
- Support coming in MESA ANV and RADV
- Final encode extensions are progressing well



Tile-based optimizations

VK_EXT_rasterization_order_attachment_access

- Equivalent to GLES Framebuffer Fetch
- Fetches become input attachment reads - requires subpass dependencies

VK_EXT_shader_tile_image

- Functionality of GLES Framebuffer Fetch and Pixel Local Storage
- Works with dynamic rendering!

Debugging GPU crashes



Extensions to the rescue:

- VK_EXT_device_fault
- VK_EXT_device_address_binding_report



What's New: Documentation and Outreach

Vulkanised 2023

Full-scale Vulkan conference in February 2023

- Hosted by Google in Munich, Germany
- Three days of talks, panels, demos, and a Vulkan course
 - All on line at <https://vulkan.org/learn#videos>
- Now planning Vulkanised 2024!



Vulkan Documentation Project

Bring Vulkan documentation together in one place

- Specification, Vulkan Guide, Proposal documents, ...
- Easy navigation and cross-linking
- <https://registry.khronos.org/vulkan/site/spec/latest/index.html>
- Experimental - feedback please! (github.com/KhronosGroup/Vulkan-Docs)


Vulkan Documentation Project Demo

Search the docs

Specs ▾

Guides ▾

Help ▾

 Vulkan Specification and Proposals / Drawing Commands

Edit this Page

Drawing Commands

Fixed-Function Vertex Processing

Tessellation

Geometry Shading

Mesh Shading

Cluster Culling Shading

Fixed-Function Vertex Post-Processing

Rasterization

Fragment Operations

Drawing Commands

Drawing commands (commands with `Draw` in the name) provoke work in a graphics pipeline. Drawing commands are recorded into a command buffer and when executed by a queue, will produce work which executes according to the bound graphics pipeline, or if the `shaderObject` feature is enabled, any [shader objects](#)

Contents

Primitive Topologies

Topology Class

Point Lists

Line Lists

Line Strips

Triangle Lists

Triangle Strips

Triangle Fans

An idea: Codified VUs?

(AKA Procedural Valid Usage statements)

Valid Usage

- VUID-vkCmdClearColorImage-image-01993
The **format features** of image **must** contain VK_FORMAT_FEATURE_TRANSFER_DST_BIT
- VUID-vkCmdClearColorImage-image-00002
image **must** have been created with VK_IMAGE_USAGE_TRANSFER_DST_BIT usage flag
- VUID-vkCmdClearColorImage-image-01545
image **must** not use any of the **formats that require a sampler Y'C_BC_R conversion**
- VUID-vkCmdClearColorImage-image-00003
If image is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdClearColorImage-imageLayout-00004
imageLayout **must** specify the layout of the image subresource ranges of image specified in pRanges at the time this command is executed on a VkDevice

Valid Usage Today

Procedural Valid Usage

What if we express VU statements as actual code?

i.e. instead of this:

If `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `imageOffset.y` **must** be 0 and `imageExtent.height` **must** be 1

write this:

```
if dstImage.create_info().imageType == VK_IMAGE_TYPE_1D:
    for region in pRegions:
        require(region.imageOffset.y == 0)
        require(region.imageExtent.height == 1)
```

Procedural Valid Usage

Advantages

- Easier to read?
- Less error-prone to write
- Machine readable / parseable /

Status

- Not committed, but under serious consideration
- Would like community feedback

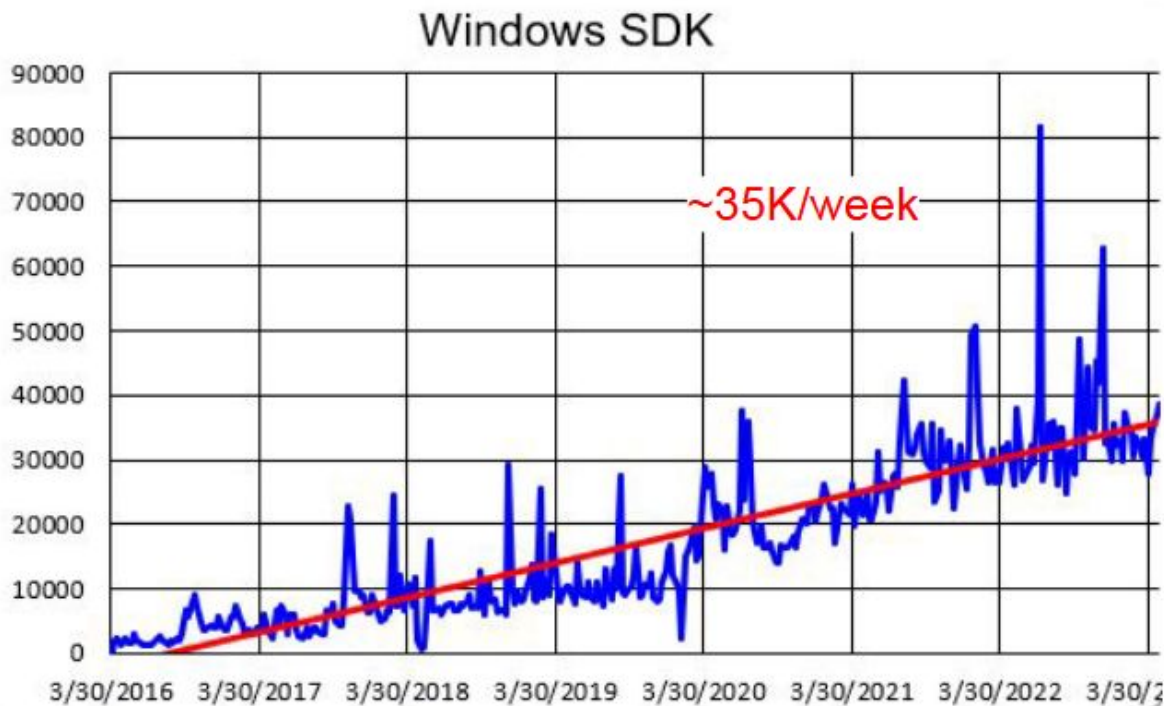
Would like community feedback:

- Proposal: <https://github.com/KhronosGroup/Vulkan-Docs/pull/2043>
- Language: <https://github.com/KhronosGroup/Vulkan-Docs/pull/2064>



What's New: SDK and Tools

SDK and Tools News





What's (kinda) New: Vulkan Profiles

Vulkan is caps-intensive

Implementations can differ in many, many ways

- Core version: 1.0, 1.1., 1.2, 1.3
- Extensions:
- Capabilities:
- Properties: how many render targets? How big can they be? ...
- Formats: What pixel formats can I use? Which can I sample? Which can I render to?
- And so on...

Result

Very hard to write portable code.

- Unfortunately, it has to be this way...

The good news

- Feature support isn't completely random!
- De facto standards exist for specific markets

Leads to the idea of profiles

Vulkan Profiles

Minimum capabilities across a set of Vulkan implementations

- Core version
- List of additional requirements for feature, property, and format support
- List of required extensions

External to the Vulkan specification

- Spec doesn't know about them
- Vulkan drivers do not know what profiles they support
- You can write new profiles to describe old hardware.

Vulkan Profile Specification

JSON schema

- Machine-readable

Enables code generation

- Support queries
- Device creation
- Set operations
- ...

```
"capabilities": {
  "baseline": {
    "extensions": {
      "VK_KHR_surface": 1,
      "VK_KHR_android_surface": 1,
      "VK_KHR_swapchain": 1,
      "VK_KHR_get_physical_device_properties2": 1,
      "VK_KHR_maintenance1": 1,
      ....
    },
    "features": {
      "VkPhysicalDeviceFeatures": {
        "depthBiasClamp": true,
        "fragmentStoresAndAtomics": true,
        "fullDrawIndexUint32": true,
        "imageCubeArray": true,
        "independentBlend": true,
        "robustBufferAccess": true,
        ....
      },
      "VkPhysicalDeviceMultiviewFeatures": {
        "multiview": true
      },
      ...
    },
    "properties": {
      "VkPhysicalDeviceProperties": {
        "limits": {
          "maxImageDimension1D": 4096,
```

Why Profiles are Awesome

It's like having your own personal Vulkan spec

- All your favorite extensions and features are supported

No driver update required!

- You can start using it today

No code changes required!

- (almost)

For example

Say you're writing an Android game...

- Targeting 3-4 years of devices, 4 GPU vendors, 12 handset OEMS
- Capability management is going to be fun!

Suppose Google says

- “if you target *this* profile, you'll reach 90% of devices”
- Seems like a win!

For example

Say you're writing an Android game...

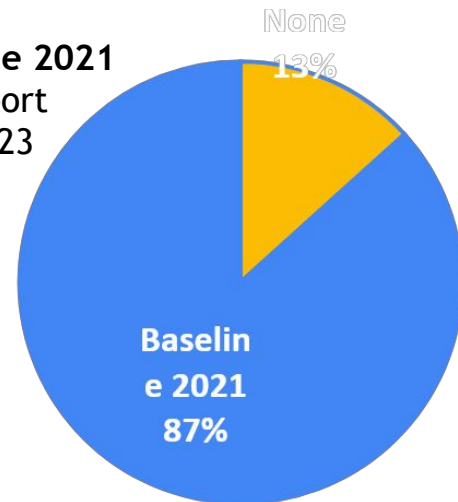
- Targeting 3-4 years of devices, 4 GPU vendors, 12 handset OEMS
- Capability management is going to be fun!

Suppose Google says

- “if you target *this* profile, you'll reach 90% of devices”
- Seems like a win!

They already have...

Android Baseline 2021
Profile Support
January 2023





What's Next

Ground rules for future releases (2022)

Core versions

- Must run on all devices
- Will not require new hardware
- Will provide quality-of-life / programming model features
 - E.g. synchronization2, timeline semaphore, dynamic rendering, GPL,
- All features will be required
- Only extensions remain optional

Profile specifications

- Will describe what is (or will be) available in specific markets
- May require new hardware functionality

Vulkan Roadmap Goals

Address fragmentation within market segments

- Short term: express what is common across upcoming devices
- Long term: encourage more commonality in future products

Allow a longer term, more coherent approach

- Groups of extensions that work together to solve a larger problem
- Avoid offering multiple solutions to the same problem.

The Vulkan Roadmap

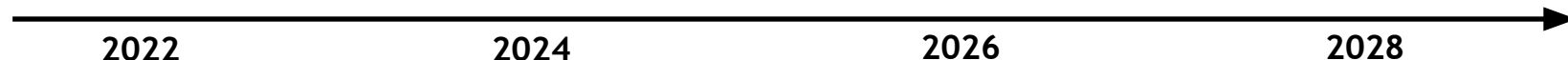
What market segment?

- “immersive graphics”
- Aka “mid- to high-end devices across smartphone, tablet, laptop, console, and desktop”
- Or, consumer devices that support an active 3rd party software market for games and media applications

Other roadmap market segments are possible

- High-end gaming on desktop (“Vulkan Ultimate”)
- Core

Roadmap Deliverables



Vulkan 1.3

- Functionality expected in all new Vulkan devices

Roadmap 2022 Profile

- Functionality expected in “immersive graphics” devices starting in 2022
- Every vendor in this market will have at least one product by e/o year

Roadmap Deliverables - 2024



Roadmap 2022



Roadmap 2024



1.3

2022

2024

2026

2028

Roadmap 2024 Profile

- Functionality expected in “immersive graphics” devices starting in 2024
- Every vendor in this market will have at least one product by e/o year

Roadmap Deliverables - 2024



Roadmap 2022



Roadmap 2024



1.3

2022

2024

2026

2028

No Vulkan 1.4 in 2024

- Eligible feature set is not compelling
- Value would not justify the overhead of a new major release

Roadmap Deliverables - Future


Roadmap 2022


Roadmap 2024


1.3

 - - - - - ► ?
1.4

2022

2024

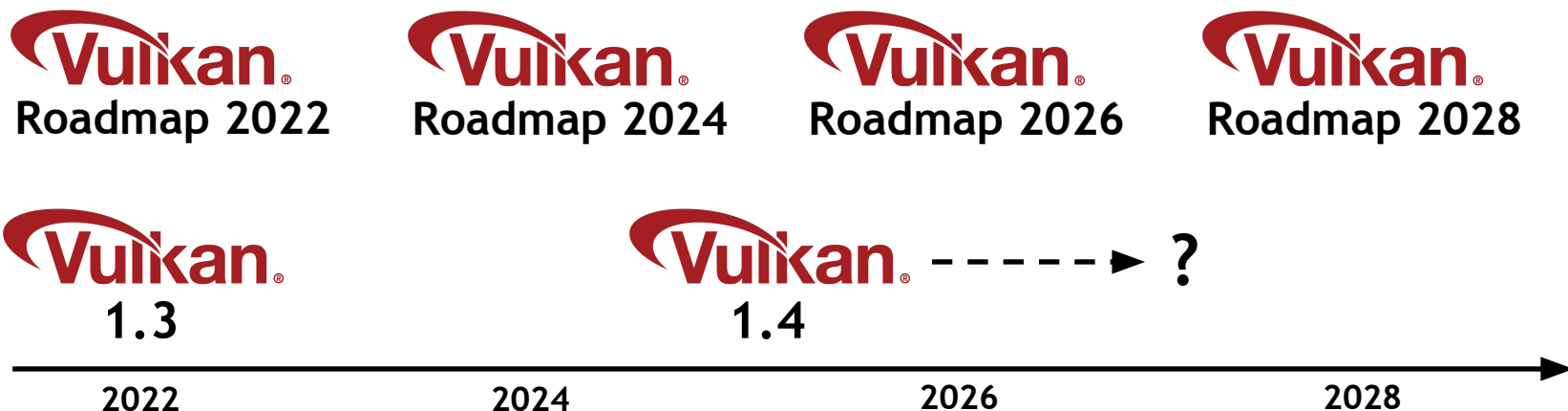
2026

2028

No Vulkan 1.4 in 2024

- Eligible feature set is not compelling
- Value would not justify the overhead of a new major release
- Will issue Vulkan 1.4 when needed - date not set

Roadmap Deliverables - Future

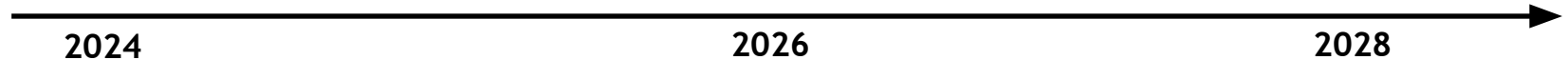


Will issue Vulkan 1.4 when it offers compelling value

- Date not set

Roadmap profiles will continue the current two-year cadence

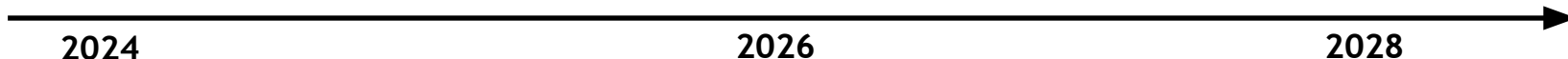
What we're working on now



Roadmap 2024 is ~done

- Content frozen, working on packaging / testing / tooling

What we're working on now



Roadmap 2024 is ~done

- Content frozen, working on packaging / testing / tooling

Roadmap 2026 is under active discussion

- Thinking in terms of topic areas / problems to solve

What we're working on now



2024



2026



2028

Roadmap 2024 is ~done

- Content frozen, working on packaging / testing / tooling

Roadmap 2026 is under active discussion

- Thinking in terms of topic areas / problems to solve

We're also thinking about Roadmap 2028...

We would value your input

Got a view on what problems we should be solving?

Raise an issue at the spec repository:

<https://github.com/KhronosGroup/Vulkan-Docs/>

Ping us on Discord:

<https://discord.com/invite/vulkan>

Talk to us here / at GDC / at Vulkanised 2024...



SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

K H R O N O S
GROUP

Vulkan®

Vulkan SDK and Ecosystem Tools

Karen Ghavam
CEO and Engineering Director
LunarG, Inc



Vulkan SDK and Ecosystem Tools

Today's
Presentation:



Karen Ghavam
CEO and Engineering Director
LunarG, Inc



Who is LunarG?

- **An Independent, private company with Khronos membership**
 - Specializing in 3D graphics software solutions for our clients
- **Developing Vulkan Ecosystem components since 2015**
 - Generous sponsorship from Valve and Google
- **Vulkan Ecosystem Projects**
 - Vulkan SDK
 - Vulkan Loader
 - Vulkan Validation Layers
 - Vulkan Profiles Toolset
 - Vulkan Extension Layer
 - GFXReconstruct
 - glslang
 - ...

Today's
Presentation:



2023 Ecosystem Survey Highlights

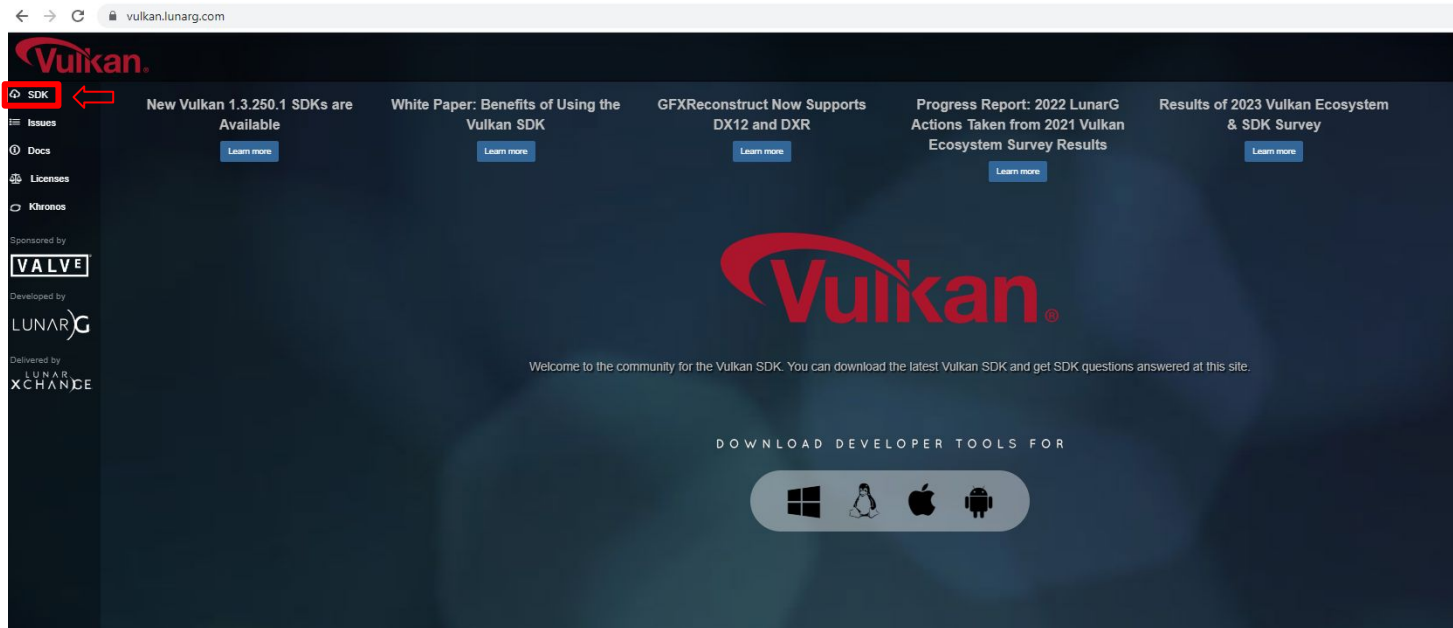
- 275 respondents. 48.3/50.7% split between self-study/commercial developers
- Amount of released content increased from 28% to 36%
- Themes:
 - Shader tool chain needs more development and maintenance
 - DXC usage was 20% of population
 - glslangValidator/shaderc (glsl->SPIR-V) was 60+% of the population
 - Validation Layers
 - Invaluable!
 - Continue to increase coverage
 - Error messages are very verbose and could be formatted better for easier reading
 - Interpreting errors (finding my root cause) is difficult (Synchronization & GPU-AV in particular)
 - Improve the performance
 - Would like to have MoltenVK to move forward more quickly

Full report [here](#)

Today's
Presentation:



The Vulkan SDK (Vulkan.lunarg.com)



Delivered by LunarG in close coordination with the Khronos Vulkan working group

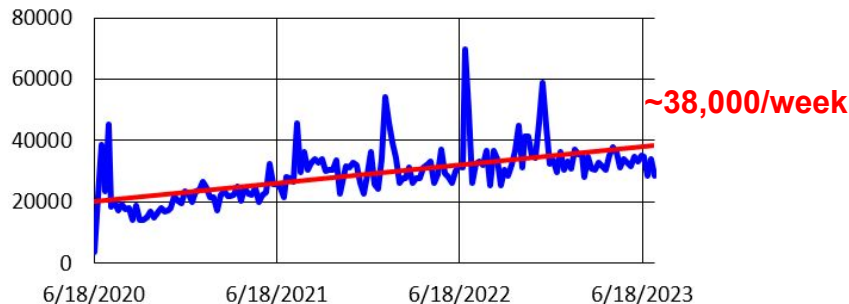


KHRONOS
GROUP

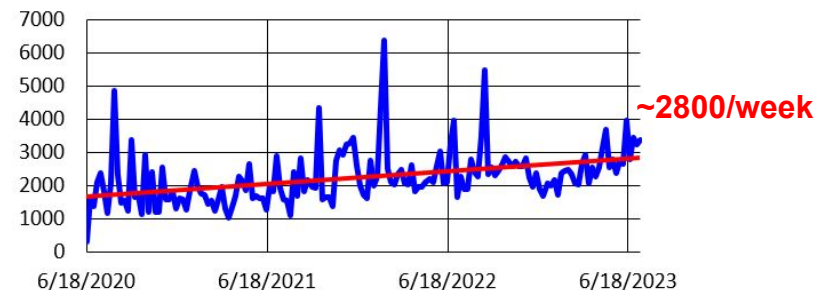


Vulkan SDK Downloads are Healthy

Windows SDK

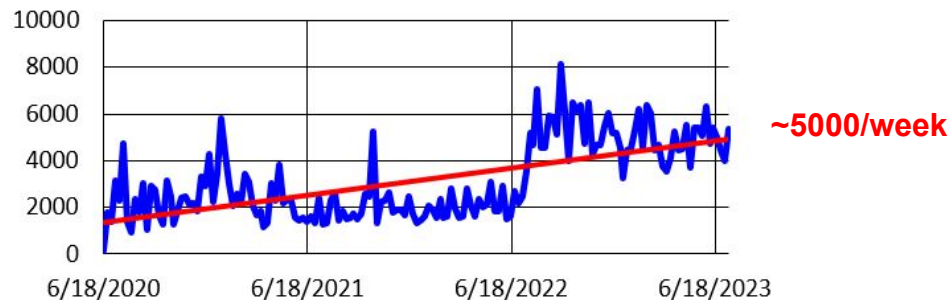


Linux SDK



Note: Numbers are for Linux "Tarball" only and don't include Ubuntu packages also available from LunarG

Mac SDK



Why Use the SDK?

- An installation process that is easy and fast
 - Windows, Linux, and macOS versions
- Pre-built tools installed into the correct system locations, ready for use.
- Vetted and curated content to ensure compatibility and seamless integration
- Ready-to-use versions of the Vulkan Configurator
- SDK release notes and user documentation
- License Registry
 - Details ALL of the open-source licenses present in the SDK

Vulkan Configurator

- **Vulkan Configurator**
 - Greatly simplifies experience with layer enablement and configuration!
- **Multiple preset default configurations**
 - Ability to create your own layer configurations as well
- **Recent addition: physical device selection**
- **Next major release preview**
 - Better control of the layers
 - Multiple versions of a layer
 - Full ordering of the layers
 - Including implicit layers
 - Improved UI: Tab based redesign
 - Diagnostics tab driven by Vulkan loader diagnostic information
- **Resources:**
 - Munich Vulkanise 2023:
 - [Using the Vulkan Configurator for Daily Vulkan Development](#)
 - [Khronos Youtube Video](#)

Developer tools in the Vulkan SDK

- **VK_LAYER_KHRONOS_validation** - validate correct API usage
 - GPU Assisted Validation
 - Best Practices for
 - Nvidia (new as of August 2022), ARM, Imagination, and AMD
 - Synchronization Validation
 - Debug Printf - “printf inside a shader”
- **VK_LAYER_LUNARG_api_dump**
 - Ascii output of Vulkan API calls
- **Vulkaninfo**
 - Show GPU device properties and extensions, installed layers, supported image formats, properties...
- **Emulation Layers**
 - VK_LAYER_KHRONOS_Synchronization2
 - VK_LAYER_KHRONOS_shader_object



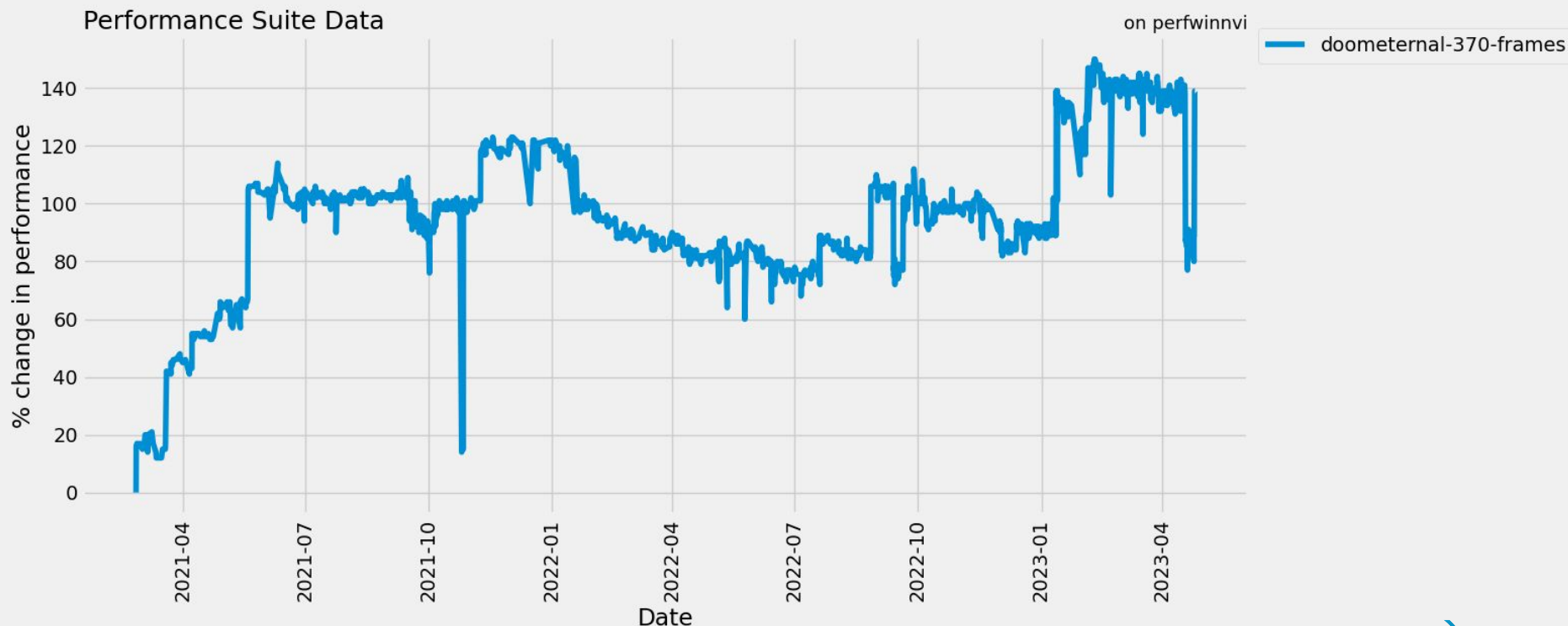
New as of
May 2023

Validation Layer Performance Improvement

- **Current Focus: Improve descriptor indexing validation performance**
 - Existing CPU side implementation has performance and correctness problems
 - Cannot determine which descriptors are 'dynamically used' and therefore need validation
 - Moving descriptor validation to GPU-AV
 - Refactor to better scale for 1M+ descriptor arrays
 - Estimate this will be ready Q3 2023
- **Some past performance improvements**
 - Linear Memory Mapping in GPU-AV (18% to 314% improvement)
 - Fine grained locking (60% to 250% improvement)



Validation Layer Performance Improvements



Synchronization Validation

- **Synchronization Validation**

- Identifies resource access conflicts due to incorrect synchronization operations between (draw, copy, dispatch, blit) reading or writing the same regions of memory.
 - Within a single buffer
 - Within and between queue submissions, and across multiple queues

- **Implementations complete with some limitations**

- **Current focus**




- Fixing community found issues
- Performance tuning

- **Resources:**

- Siggraph Birds of a Feather Presentations:
 - [Ensure Correct Vulkan Synchronization by Using Synchronization Validation](#)
 - [Khronos Youtube video](#)
 - [Correct Vulkan Synchronization with Extended Synchronization Validation](#)
 - [Khronos Youtube video](#)



Developer tools in the Vulkan SDK

- VOLK 
 - A Vulkan entry point meta-loader
- HW Capabilities Viewer from gpuinfo.org 
- Vulkan Portability Solution 
 - Vulkan® Portability™ enables the consistent use of layered implementations of Vulkan functionality over Metal and other APIs
 - Vulkan Loader: VK_KHR_portability_enumeration
 - API: VK_KHR_portability_subset
 - Portability Profiles (VP_LUNARG_desktop_portability_2022)

The macOS SDK

- iOS Vulkan Loader support will be coming soon! 
 - Enables running the validation layers on your iOS device with your app
- All mac SDK components now support both Apple Silicon and Intel Architectures
 - No longer need Rosetta emulation environment
- Resources:
 - LunarG White Paper - [The state of Vulkan on Apple Devices](#)
 - Munich 2023 Vulkanise Presentation
 - [Vulkan Development for Apple Devices](#)
 - [Khronos Youtube video](#)
 - 2023 Siggraph BoF
 - [Vulkan Development in Apple Environments](#)



New in the next
LunarG SDK!

2023 Siggraph
BoF:



LUNAR)G

GFXReconstruct

- Capture Vulkan API calls in a file with VK_LAYER_LUNARG_gfxreconstruct
 - Replay with gfxrecon-replay
- Linux, Windows, Android
- API-agnostic; Vulkan and Direct3D 12 so far!
- Use cases
 - Silicon development
 - Driver quality testing
 - Bug reporting
 - App debugging
- Resources:
 - Siggraph 2023:
 - [Capture & Replay with Vulkan & DX12: GFXReconstruct](#)
 - Munich Vulkanise 2023:
 - [GFXReconstruct- Tools to Capture and Replay Graphics API Calls](#)
 - [Youtube Video](#)

2023 Siggraph
BoF:



Vulkan Profiles



Essential for Roadmap
2024

Vulkan SDK

- A mechanism that enables the precise specification of capabilities
 - Communication of capabilities to participants in the Vulkan ecosystem
 - Easier Vulkan development for a selected range of actual ecosystem devices

Example Profiles Usage

- **Roadmap profiles**: To express guidance on the future direction of Vulkan devices
 - In the SDK: VK_KHR_roadmap_2022
 - Upcoming: Roadmap 2024
- **Platform profiles**: To express Vulkan support available on different platforms
 - In the SDK:
 - LunarG Desktop Baseline 2022 (Vulkan 1.1)
 - LunarG Desktop Baseline 2023 (Vulkan 1.2)
 - Android Baseline 2021 v2 (Vulkan 1.0)
 - Android Baseline 2022 (Vulkan 1.1)
- **Device Profiles**: To express Vulkan support of a single Vulkan device
 - Gpuinfo.org provides device profiles
- **Engine Profiles**: To express requirements of the rendering code path
 - Prevent application from requiring unavailable features on devices



Vulkan Profiles Toolset

- **Profiles Schema** - A JSON data format to communicate about Vulkan capabilities
 - Extensions, features, properties, formats, and queue properties
 - Schema for each Vulkan API revision (KhronosGroup/Khronos-Schemas)
- **VK_LAYER_KHRONOS_profiles**
 - Enables downgrading the Vulkan developer's system capabilities using a Vulkan Profile
- **Vulkan Profiles Library**
 - A header-only C++ library to use Vulkan Profiles in Vulkan applications
 - Checking Profile support on a device.
 - Create a vkDevice instance w/ features/extensions enabled
- **The Vulkan Profiles JSON file generation**
 - Generate profiles file by combining multiple existing profiles
 - Union and intersection of Vulkan capabilities



Profiles Tutorials

- [LunarG White Paper: Vulkan Profiles](#)
- 2023 Munich Vulkanise
 - [Creating Vulkan Profiles](#)
 - [Khronos Youtube Video](#)
- 2022 Khronos Vulkanise
 - [Vulkan SDK Tools to Use and Create Vulkan Profiles](#)
 - [Khronos Youtube Video](#)

Shader Tool Chain

- Offline executables and API libraries for:
 - SPIRV-Tools
 - Validator, optimizer, assembler, disassembler, diff, Remapper
 - GLSL->SPIR-V
 - glslang SPIR-V generator
 - HLSL->SPIR-V
 - Glslang SPIR-V generator (up to shader model 5)
 - DXC (Microsoft DirectX Shader Compiler)
 - Shaderc
 - Glslang and SPIRV-Tools wrapper for better integration with build tools
 - SPIRV-CROSS
 - SPIR-V shaders -> HLSL/Metal/GLSL shaders
 - SPIRV-Reflect
 - Provides a C/C++ reflection API for SPIR-V shader bytecode
- Did you know? A really handy tool to visualize your SPIR-V
 - <https://www.khronos.org/spir/visualizer/>



Today's
Presentation:



Stop by and see our demos!

Vulkan demos at the LunarG table during the Networking Event

Demo 1 - Using GFXReconstruct to Capture, Replay, and Inspect an Application's Graphics Commands

Demo 2 - Vulkan Validation on Apple Devices, a Vulkan Configurator Demo



SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

K H R O N O S
GROUP

Vulkan®

Vulkan and Open Source Graphics at Autodesk

Henrik Edström
Distinguished Architect, Graphics - Autodesk

Safe Harbor Statement

The presentations during this event may contain forward-looking statements about our outlook, future results and related assumptions, total addressable markets, acquisitions, products and product capabilities, and strategies. These statements reflect our best judgment based on currently known factors. Actual events or results could differ materially. Please refer to our SEC filings, including our most recent Form 10-K and Form 10-Q filings available at www.sec.gov, for important risks and other factors that may cause our actual results to differ from those in our forward-looking statements.

The forward-looking statements made in these presentations are being made as of the time and date of their live presentation. If these presentations are reviewed after the time and date of their live presentation, even if subsequently made available by us, on our website or otherwise, these presentations may not contain current or accurate information. We disclaim any obligation to update or revise any forward-looking statements.

Statements regarding planned or future development efforts for our products and services are not intended to be a promise or guarantee of future availability of products, services, or features but merely reflect our current plans and based on factors currently known to us. Purchasing decisions should not be made based upon reliance on these statements.

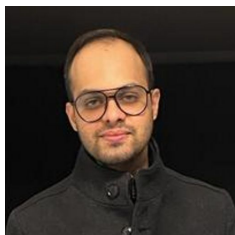
PLEASE NOTE: All Autodesk content is proprietary. Please Do Not Copy, Post or Distribute without authorization.

Three Vulkan presentations from Autodesk



Vulkan Ray Tracing in Aurora: An Open Source Real-Time Path Tracer

Mauricio Vives
Sr. Principal Engineer, Graphics



Vulkan for Cross-Platform Viewing of Large AEC Models

Vipul Kapoor
Senior Software Engineer



Porting Autodesk Flame from OpenGL to Vulkan

Jasmin Roy
Software Development Manager



“Autodesk makes software for people who design and make things”



ARCHITECTURE, ENGINEERING &
CONSTRUCTION



PRODUCT DESIGN &
MANUFACTURING

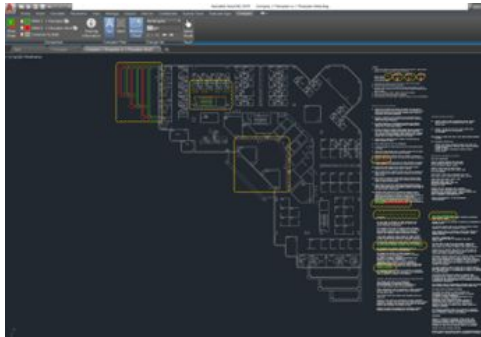
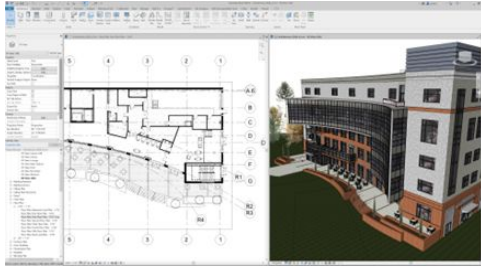


MEDIA & ENTERTAINMENT

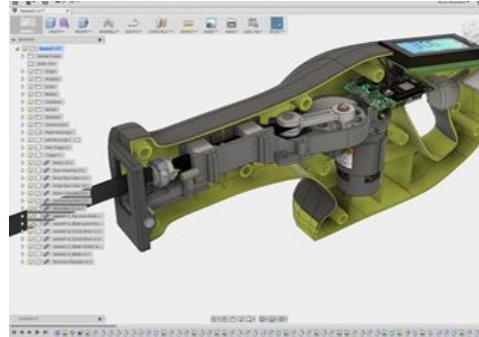


Need a wide range of graphics capabilities

2D & Simple 3D



3D Modeling



Realistic Rendering



Autodesk Graphics Platform Objectives

Modern APIs



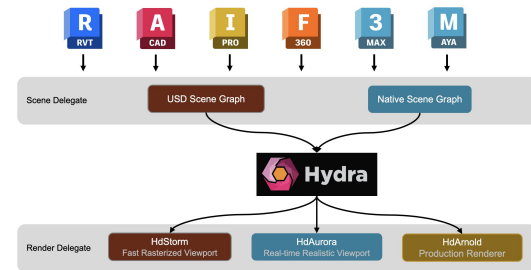
Open Standards



OpenPBR

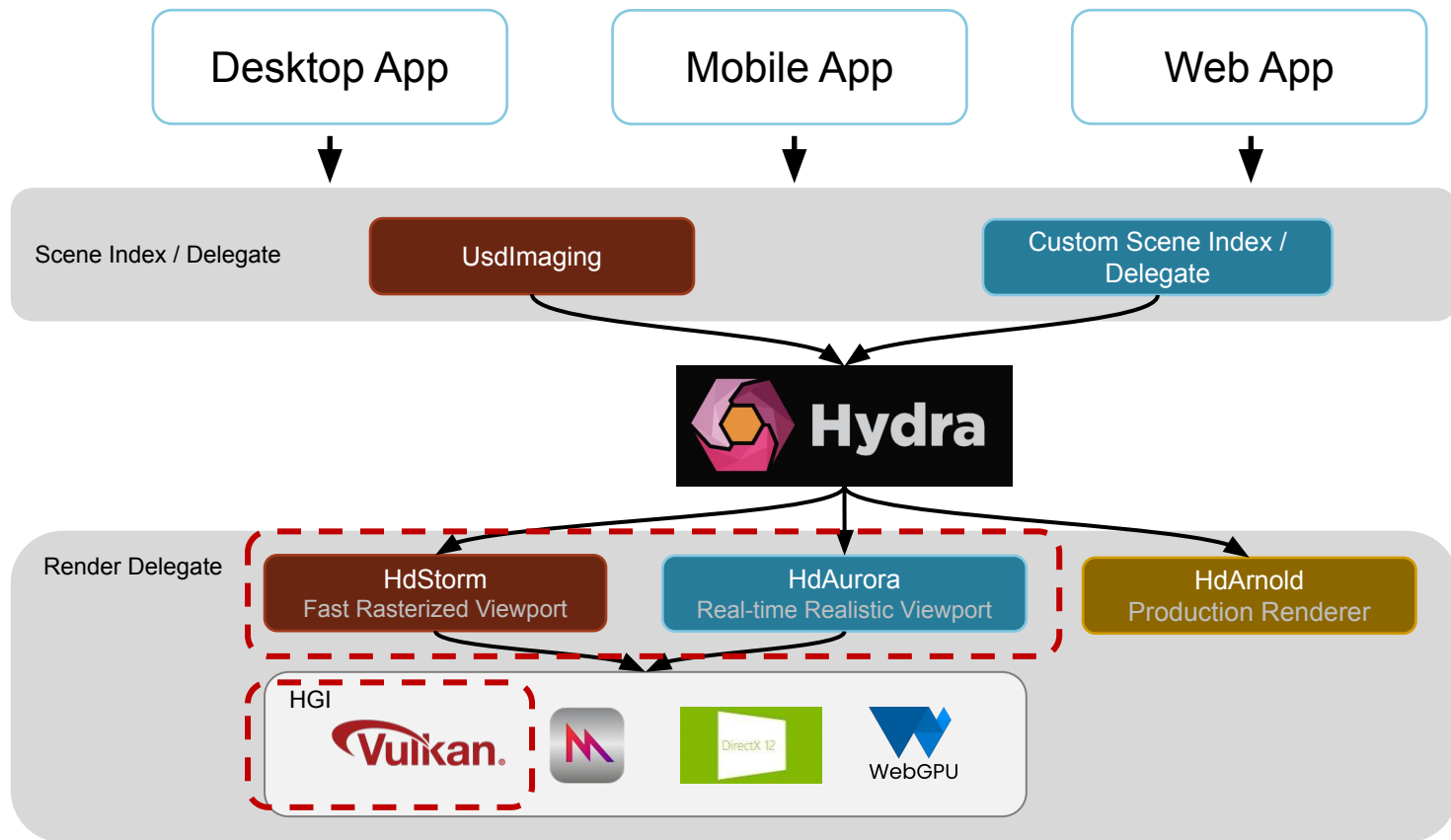


Decoupled Architecture



Available on Desktop, Mobile, and Web

Hydra for Desktop, Mobile, and Web





SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

K H R  N O S
GROUP

 **Vulkan**®

Vulkan Ray Tracing in Aurora: An Open Source Real-Time Path Tracer

Mauricio Vives & Gareth Morgan
Sr. Principal Engineers, Autodesk

Safe Harbor Statement

The presentations during this event may contain forward-looking statements about our outlook, future results and related assumptions, total addressable markets, acquisitions, products and product capabilities, and strategies. These statements reflect our best judgment based on currently known factors. Actual events or results could differ materially. Please refer to our SEC filings, including our most recent Form 10-K and Form 10-Q filings available at www.sec.gov, for important risks and other factors that may cause our actual results to differ from those in our forward-looking statements.

The forward-looking statements made in these presentations are being made as of the time and date of their live presentation. If these presentations are reviewed after the time and date of their live presentation, even if subsequently made available by us, on our website or otherwise, these presentations may not contain current or accurate information. We disclaim any obligation to update or revise any forward-looking statements.

Statements regarding planned or future development efforts for our products and services are not intended to be a promise or guarantee of future availability of products, services, or features but merely reflect our current plans and based on factors currently known to us. Purchasing decisions should not be made based upon reliance on these statements.

PLEASE NOTE: All Autodesk content is proprietary. Please Do Not Copy, Post or Distribute without authorization.

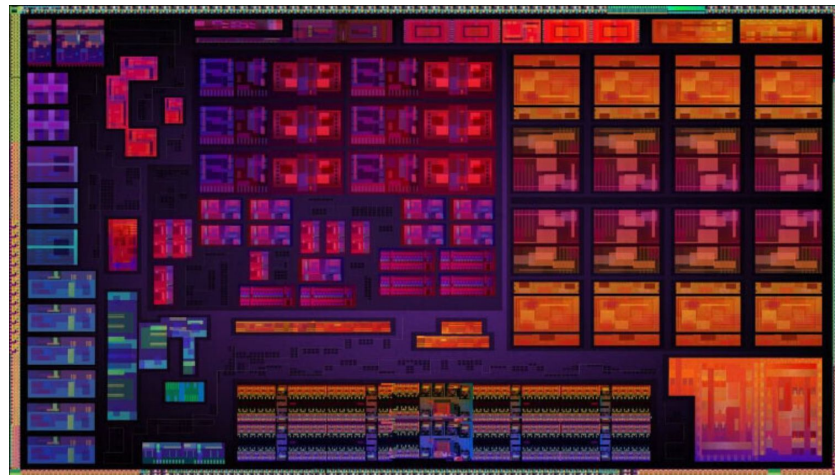
What is Aurora?

- Open source and \$Free (more later)
- Path tracing renderer
- Real-time
- Physically-based
- Noise-free (optional)
- Using hardware ray tracing
- Autodesk Standard Surface w/ MaterialX
- OpenUSD Hydra render delegate
- Cross-platform + cross-vendor
- *Rapid design iteration, not final frames*



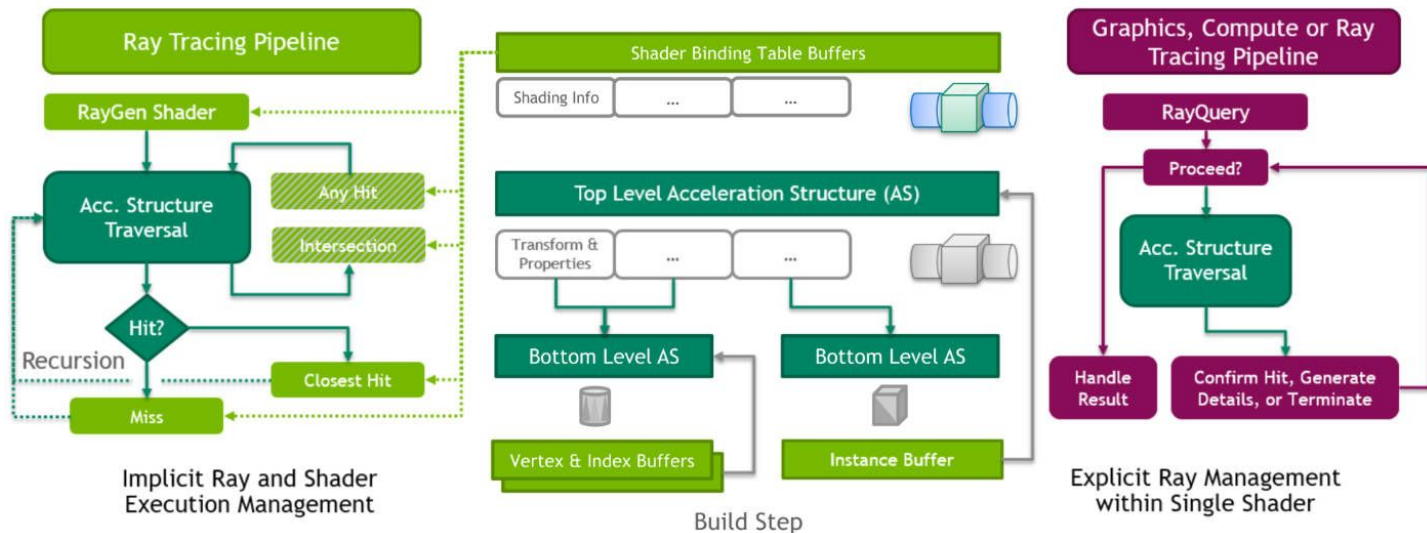
What is hardware ray tracing?

- Graphics API support for ray tracing operations.
 - May or may not be backed by actual dedicated hardware.
 - Introduced with DirectX Raytracing (DXR) at GDC 2018.
 - Hardware from NVIDIA followed shortly after (Turing / “RTX”).
 - Now widely supported by NVIDIA, AMD, and Intel.
 - ... even in integrated GPUs starting with the AMD Radeon 680M.
 - Here to stay! 🙌
-
- Dedicated hardware normally handles BVH traversal + triangle intersection.
 - API introduces:
 - Acceleration structures.
 - Ray tracing shaders.
 - Shader binding tables.
 - Special commands.



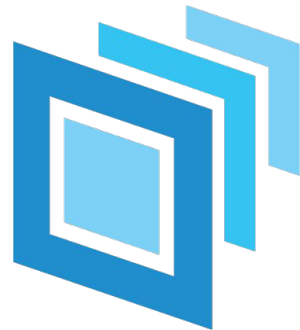
What about Vulkan?

- Aurora development started with DXR support, for Inventor on Windows.
- Want Linux support for Autodesk products, so... Vulkan it is!
- Ray tracing support finalized in December 2020. 🎉
- Conceptually similar to DirectX Raytracing... and just as verbose. 😊
- <https://www.khronos.org/blog/ray-tracing-in-vulkan>



How does Aurora support Vulkan ray tracing?

- Hgi instead of a direct integration (easier for our Hydra render delegate).
- OpenUSD > Hydra: Mix-and-match scene data and renderers.
- Hydra > Hgi: “Hydra Graphics Interface”
 - Abstraction layer over graphics backends: OpenGL, Metal, Vulkan.
 - Intended for Hydra render delegates, but can be used independently.
 - Hgi Vulkan support is incomplete... for now. (See the next talk!)
- Don't need all of Vulkan, just enough for ray tracing support.
- We extended the Hgi API and Vulkan backend to support ray tracing...
 - Acceleration structures.
 - Ray tracing shaders.
 - Shader binding tables.
 - Special commands.



What is in Hgi Ray Tracing?

HgiAccelerationStructure

HgiAccelerationStructureGeometry

HgiRayTracingPipeline

HgiAccelerationStructureCmds

HgiRayTracingCmds

HgiBufferUsageShaderBindingTable

HgiShaderStageRayGen

HgiShaderStageAnyHit

HgiShaderStageClosestHit

HgiShaderStageMiss

HgiShaderStageIntersection

HgiShaderStageCallable

... and more.

[https://github.com/autodesk-forks/USD](https://github.com/autodesk-forks/USD/tree/adsk/feature/hgiraytracing)
[/tree/adsk/feature/hgiraytracing](https://github.com/autodesk-forks/USD/tree/adsk/feature/hgiraytracing)

What about shader languages?

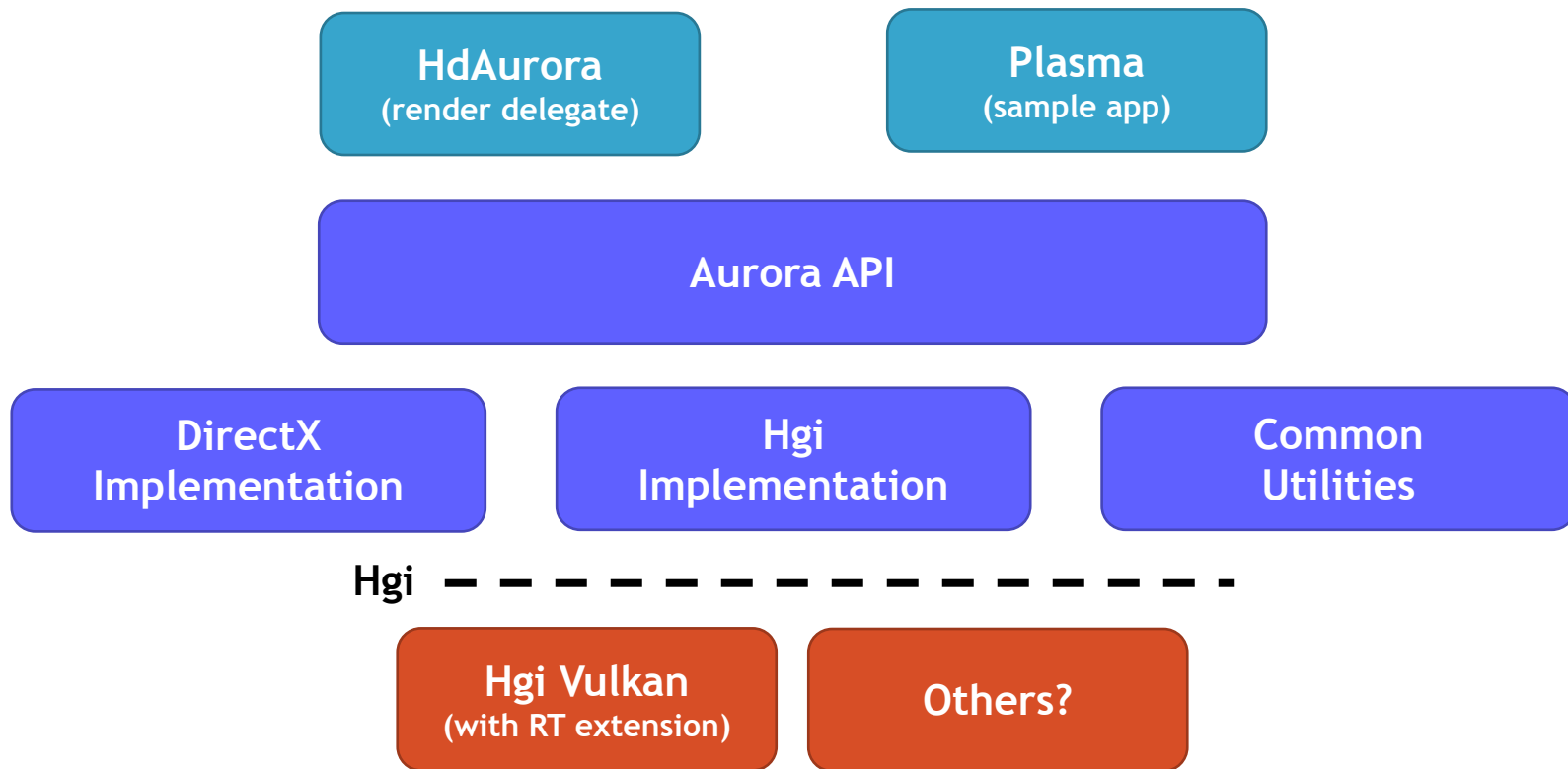
- Need both HLSL for DXR and GLSL for Vulkan.
- Don't want to maintain two copies...
- ... so we want shader language translation...
- ... but need support for ray tracing concepts like `TraceRay()`. 🤔
- Slang to the rescue! “Making it easier to work with shaders” 🌟
- Code written in Slang (HLSL with extensions).
- Runtime Slang transpiler converts to GLSL when Vulkan backend is used

Falcor: real-time path tracer from NVIDIA Research.

GDC Talk: https://research.nvidia.com/publication/2022-03_Research-Advances-Toward



What is the Aurora architecture?



What are the next steps?

- Address known gaps between DXR and Vulkan, Windows and Linux.
- Refactor integrator to be iterative instead of recursive.
- Denoising with NVIDIA NRD... Vulkan supported. ✓
- Upscaling with AMD FSR... Vulkan supported. ✓
- MaterialX... and OpenPBR?

Open source under Apache license... you can contribute!

<https://github.com/Autodesk/Aurora>

Reach out to aurora@autodesk.com



SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

K H R O N O S
GROUP

Vulkan®

Vulkan for Cross-Platform Viewing of Large AEC Models

Vipul Kapoor
Sr. Software Engineer, Autodesk

Safe Harbor Statement

The presentations during this event may contain forward-looking statements about our outlook, future results and related assumptions, total addressable markets, acquisitions, products and product capabilities, and strategies. These statements reflect our best judgment based on currently known factors. Actual events or results could differ materially. Please refer to our SEC filings, including our most recent Form 10-K and Form 10-Q filings available at www.sec.gov, for important risks and other factors that may cause our actual results to differ from those in our forward-looking statements.

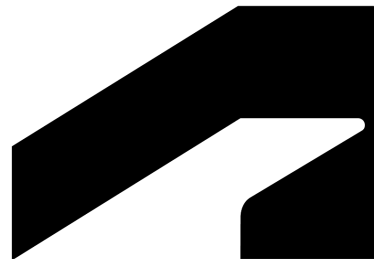
The forward-looking statements made in these presentations are being made as of the time and date of their live presentation. If these presentations are reviewed after the time and date of their live presentation, even if subsequently made available by us, on our website or otherwise, these presentations may not contain current or accurate information. We disclaim any obligation to update or revise any forward-looking statements.

Statements regarding planned or future development efforts for our products and services are not intended to be a promise or guarantee of future availability of products, services, or features but merely reflect our current plans and based on factors currently known to us. Purchasing decisions should not be made based upon reliance on these statements.

PLEASE NOTE: All Autodesk content is proprietary. Please Do Not Copy, Post or Distribute without authorization.

Presentation Agenda

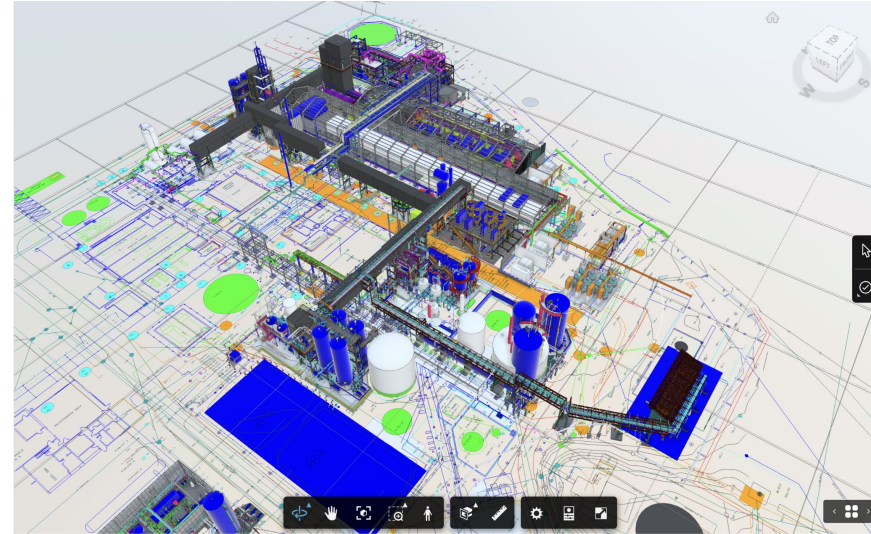
- AEC Models
- Cross Platform Strategy
- Current State and Progress
- Debugging and Tools
- Community and Collaboration



AUTODESK

AEC Models

- Design data for Building, Infrastructure, construction
- Optimised for Design, not Rendering
- 2D, 3D, Not texture Rich
- Asset Stats:
 - File Size - Up to >100GB
 - Polygon Count - Up to >1 Bil
 - Mesh Count - Up to >1 Mill
- Tech Problem Statement - Draw Call, IA + VS pressure, VMem consumption, etc



Cross Platform Strategy

- OpenUSD and Hydra Graphics Interface
 - HgiVulkan for Windows, Linux and Android
 - HgiMetal for macOS and iOS
 - HgiWebGPU for Web Platform
 - HgiDx12 for exclusive Windows Platform



Where we are on .. Windows

- Completed HgiVulkan path for USD's base-material pipeline.
*Note: barring some caveats
- Stabilized the HgiVulkan backend
- Vulkan Swap-chain and Presentation Layer (No cross-API use)
- OpenUSD Proposal PR - <https://github.com/PixarAnimationStudios/OpenUSD-proposals/pull/15>
- OpenUSD Implementation PR(Draft) - <https://github.com/PixarAnimationStudios/OpenUSD/pull/2553>

Windows Capture on

Device: AMD Desktop

CPU and Nvidia RTX

Discrete GPU

OS: Windows 10



Linux and Android

- Ported progress from Windows
- On Linux
 - Clang++ 14.0, CMake 3.27
- On Android
 - NDK 25.2 + Android Studio Flamingo
 - OpenUSD Proposal PR - <https://github.com/PixarAnimationStudios/OpenUSD-proposals/pull/17>
- OpenUSD Implementation PR - **Coming Soon**

Linux Capture on

Device: AMD Desktop
CPU and Nvidia RTX
Discrete GPU

OS: Ubuntu 22.04



Android Capture on

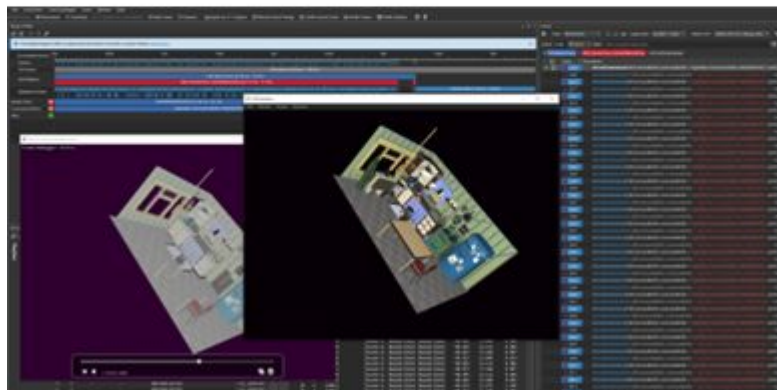
Device: Galaxy Tab S8+
(model: SM-X800)

OS: Android 13



Debugging and Tools

- Vulkan Debug and Validation Layer - “VK_LAYER_KHRONOS_validation”
- Debug Markers
 - Render Targets
 - Resource Buffers
 - Renderpasses, Pipelines etc
- Frame Debugging Tools
 - RenderDoc
 - Nvidia Nsight Graphics
- Vulkan Device Simulation Layer
 - Unit + Component Testing



Community and Collaboration

- **Contact Us**
 - Autodesk Graphics Platform Team - agp@autodesk.com
- **OpenUSD Forum**
 - AOUSD Forum - <https://forum.aousd.org/>
 - AOUSD Hydra Forum - <https://forum.aousd.org/c/community/hydra/8>





SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

K H R O N O S
GROUP

Vulkan®

Porting Autodesk Flame from OpenGL to Vulkan

Jasmin Roy
Software Development Manager, Autodesk

Safe Harbor Statement

The presentations during this event may contain forward-looking statements about our outlook, future results and related assumptions, total addressable markets, acquisitions, products and product capabilities, and strategies. These statements reflect our best judgment based on currently known factors. Actual events or results could differ materially. Please refer to our SEC filings, including our most recent Form 10-K and Form 10-Q filings available at www.sec.gov, for important risks and other factors that may cause our actual results to differ from those in our forward-looking statements.

The forward-looking statements made in these presentations are being made as of the time and date of their live presentation. If these presentations are reviewed after the time and date of their live presentation, even if subsequently made available by us, on our website or otherwise, these presentations may not contain current or accurate information. We disclaim any obligation to update or revise any forward-looking statements.

Statements regarding planned or future development efforts for our products and services are not intended to be a promise or guarantee of future availability of products, services, or features but merely reflect our current plans and based on factors currently known to us. Purchasing decisions should not be made based upon reliance on these statements.

PLEASE NOTE: All Autodesk content is proprietary. Please Do Not Copy, Post or Distribute without authorization.

Presentation Agenda

- Autodesk Flame Overview
- Project Conversion Strategy
- Technical Challenges & Solutions
- Project Outcome
- Next Steps!



AUTODESK
Flame **Vulkan.**

Autodesk Flame Overview

Timeline based visual effects and finishing solution

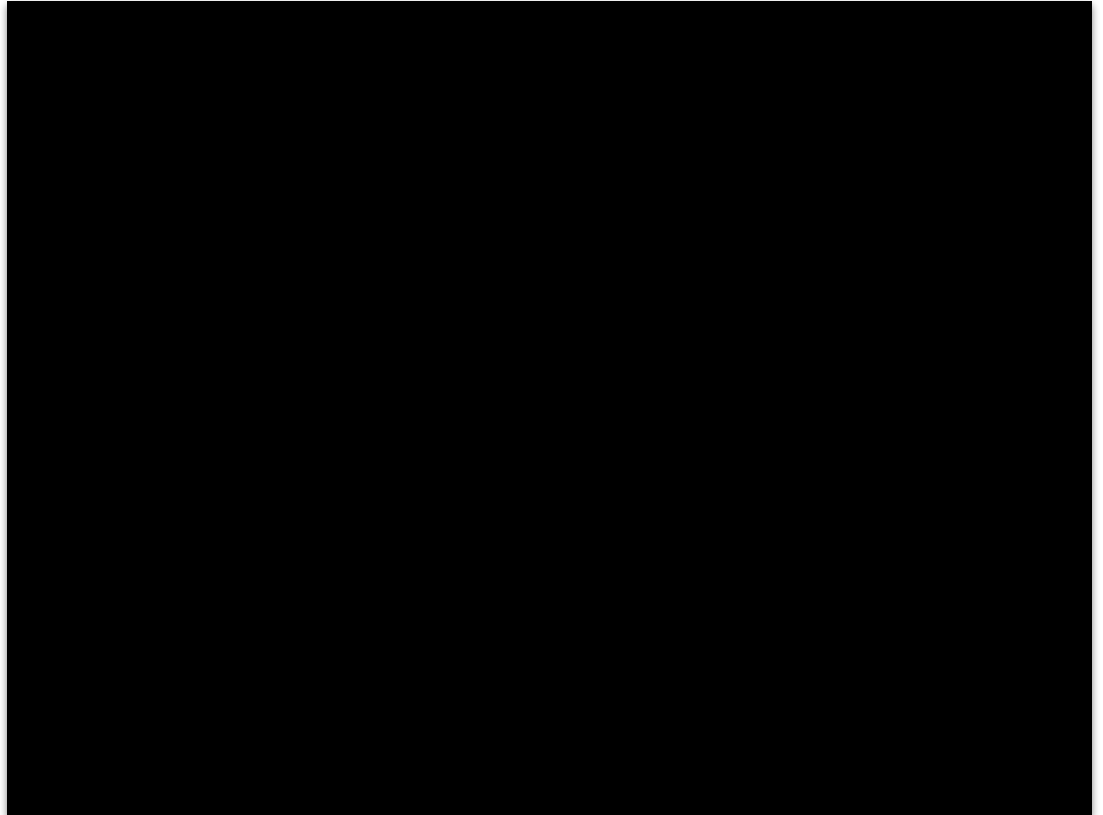
- Proprietary 3D engine which is optimized for real-time artistic visual effects.
- Used for commercials, TV episodics and features.
- Machine learning to accelerate and assist creative tasks.
- Used by many high-profile artists and studios that rely on its tools, performance and rendering quality.



Autodesk Flame Overview

Timeline based visual effects and finishing solution

- Proprietary 3D engine which is optimized for real-time artistic visual effects.
- Used for commercials, TV episodics and features.
- Machine learning to accelerate and assist creative tasks.
- Used by many high-profile artists and studios that rely on its tools, performance and rendering quality.





Project Conversion Strategy

Highlights

Challenges

- Huge Feature Set and Codebase
- Linux & Mac Platform Support
- Memory Management
- Multi-Year Project

Strategies

- MoltenVK on the Mac
- Create Abstraction Layers
- Master Branch Development
- Iterative / Automation Driven Approach



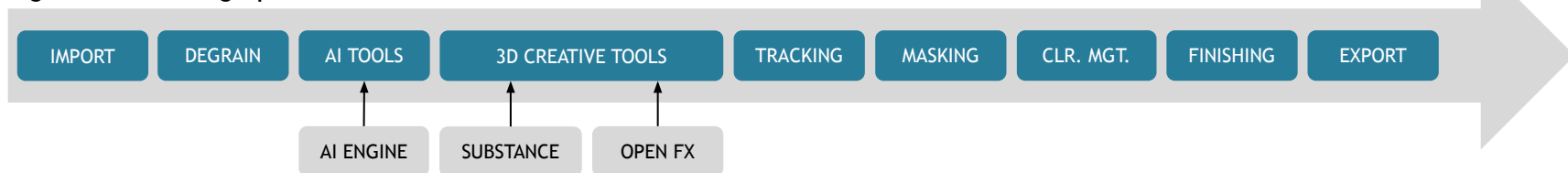
Conversion Strategy

Iterative Approach using OpenGL / Vulkan Interop

- OpenGL / Vulkan Interoperability Extensions
- Semaphores for Synchronization
- Support Mixed Graphics API Processing Pipeline
- Take Advantage of 20000+ Automation Tests Early

```
VkExternalMemoryImageCreateInfo extMemImgInfo = {};  
extMemImgInfo.sType = VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_IMAGE_CREATE_INFO;  
extMemImgInfo.handleTypes = VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT_KHR;  
  
VkImageCreateInfo imageInfo = {};  
imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;  
imageInfo.pNext = &extMemImgInfo;  
...  
  
vkCreateImage( *vkDevice(), &imageInfo, nullptr, &interopBuf.vkImage );  
  
VkExportMemoryAllocateInfoKHR exportMemInfo = {};  
exportMemInfo.sType = VK_STRUCTURE_TYPE_EXPORT_MEMORY_ALLOCATE_INFO_KHR;  
exportMemInfo.handleTypes = VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT_KHR;  
  
VkMemoryAllocateInfo imgAllocInfo = {};  
imgAllocInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;  
imgAllocInfo.pNext = &exportMemInfo;  
...  
  
vkAllocateMemory( *vkDevice(), &imgAllocInfo, nullptr, &interopBuf.vkMem );  
vkBindImageMemory( *vkDevice(), interopBuf.vkImage, interopBuf.vkMem, 0 );  
  
glCreateMemoryObjectsEXT( 1, &interopBuf.glMemObj );  
glImportMemoryFdEXT(  
    interopBuf.glMemObj,  
    interopBuf.vkRequirements.size,  
    HANDLE_TYPE_OPAQUE_FD_EXT,  
    interopBuf.memoryFd );
```

High Level Processing Pipeline



Technical Challenges & Solutions

Runtime Legacy GLSL Shader Conversion

- Matchbox Legacy GLSL Support
- Runtime Shader Converter

GLSL 1.2

```
#version 120

uniform sampler2D input;

uniform vec3 slope;
uniform vec3 offset;
uniform vec3 power;

uniform float width;
uniform float height;
uniform float saturation;

void main(void)
{
    vec2 coords = gl_FragCoord.xy / vec2( width, height );
    vec3 source = texture2D(input, coords).rgb;

    vec3 slopeRGB = slope;
    vec3 offsetRGB = offset;
    vec3 powerRGB = power;

    vec3 colour = source * slopeRGB + offsetRGB;
    float luma = 0.2126 * colour.r + 0.7152 * colour.g + 0.0722 * colour.b;
    colour = luma + (saturation*0.01) * (colour-luma);

    gl_FragColor = vec4(colour, 1.0);
}
```

REGEX Converter

GLSL 4.30

```
#version 430

layout ( location = 0 ) out vec4 adsk_FragColor;
layout( binding = 1 ) uniform UniformBlock
{
    vec3 slope;
    vec3 offset;
    vec3 power;
    float width;
    float height;
    float saturation;
};

layout( binding = 2 ) uniform sampler2D input;

void main(void)
{
    vec2 coords = gl_FragCoord . xy / vec2(width, height);
    vec3 source = texture(input, coords). rgb;

    vec3 slopeRGB = slope;
    vec3 offsetRGB = offset;
    vec3 powerRGB = power;

    vec3 colour = source * slopeRGB + offsetRGB;
    float luma = 0.2126 * colour . r + 0.7152 * colour . g + 0.0722 * colour . b;
    colour = luma +(saturation * 0.01)*(colour - luma);

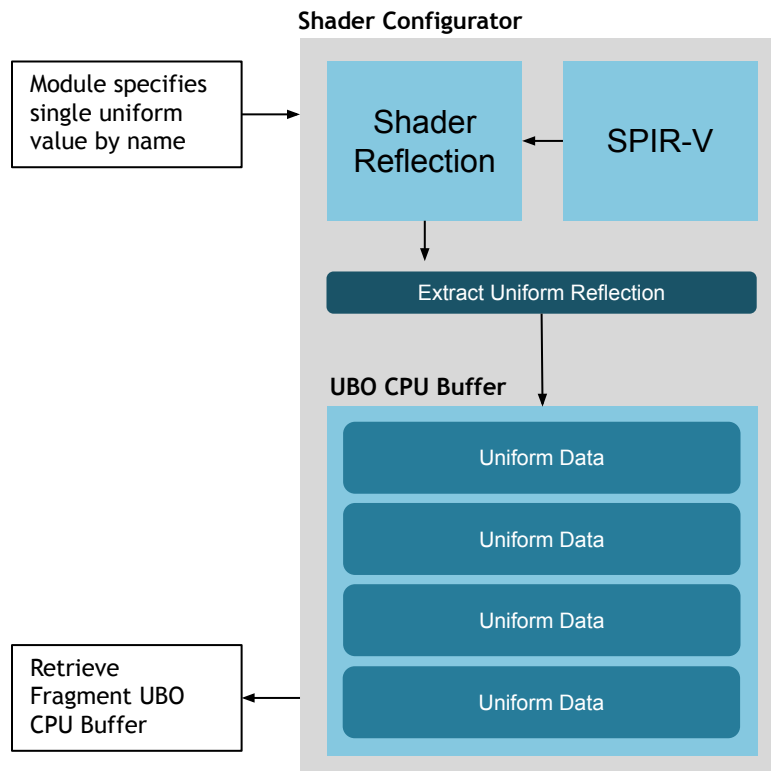
    adsk_FragColor = vec4(colour, 1.0);
}
```

GLSLang SPIR-V Compiler

Technical Challenges & Solutions

Abstraction Layer / Shader Configurator

- Streamline Component Conversion
- Help Distribute Work
- Reduce Lines of Code
- Using Shader Reflection
- Manage UBO Buffers

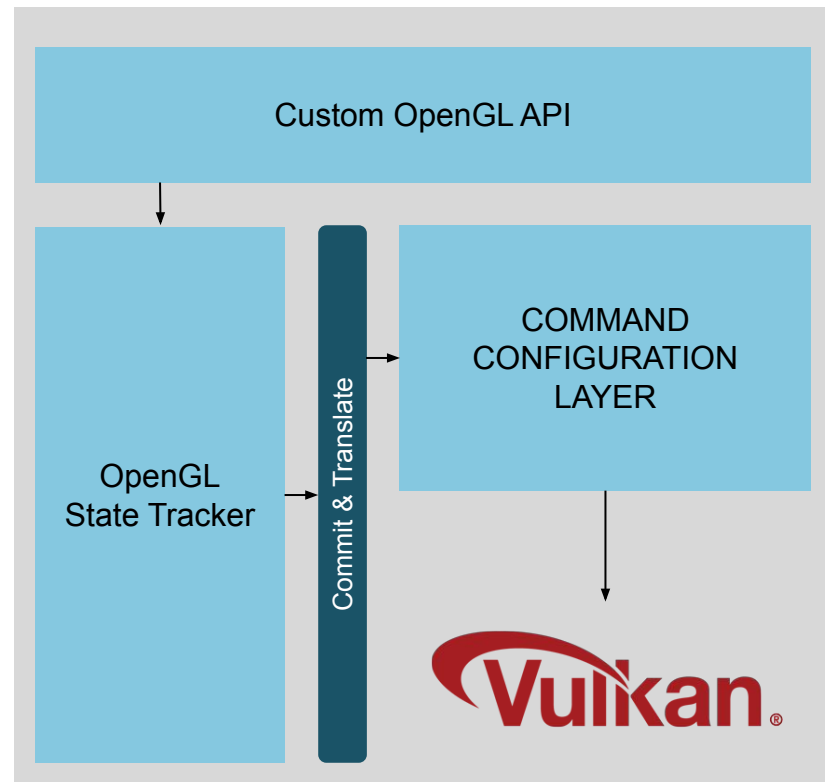


Technical Challenges & Solutions

OpenGL Translation Layer

- For Non-Critical Components
 - User Interface
 - Overlays
 - Diagnostic Tools
- Help Concentrate on Critical Components
- Custom OpenGL API
- OpenGL State Tracker
- Commit & Translate to Vulkan Pipeline

OpenGL Translation Layer (State Based)



Artist Testimonials

“It was immediately clear that the move to Vulkan brings a significant speed boost. Flame already is the fastest and most versatile creative system out there, but this brings things to a whole new level. This is very exciting because every creative process benefits from iterations, and more speed allows for more iterations, thus adding to the creative result. Vulkan should also pave the way for the addition of new and exciting creative tools, and I can’t wait to see what’s in store next.”

Ton Habraken, VFX supervisor at SquarePXL.

“We found that most of the work we do renders faster. Some render times are faster by 50%. The interface is snappy and overall, it feels like a huge improvement to the working experience. I know getting Flame up to Metal was a long undertaking and I hope the user base understands this is a huge step for the future of the product”.

Bryan Bayley, senior Flame artist and colorist at Republic Editorial.

Project Outcome & Next Steps

Outcome

- Performance Improvements
- Better **Memory Management and Stability**
- Successful **Flame 2024.1** Update

Next Steps

- Vulkan Sub-Passes
- Multi-Threaded Processing Pipeline
- Improved Multi GPU Support
- OpenCL / CUDA to Vulkan Compute



AUTODESK
Flame  **Vulkan**



SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

K H R O N O S
GROUP

Vulkan®

Basic Ray Trace Debugging in Vulkan

Hai Nguyen
(just some guy who really likes graphics)



Hai Nguyen

- Graphics Lead for XR at Google
- [DirectX Shader Compiler for SPIR-V aka DXC](#)
- [SPIRV-Reflect](#)
- Computer Graphics Enthusiast



Went from this...



UV Coordinates For Ray Generation

...to this in a couple of months. There was debugging.



Real Time Progressive PBR Path Tracer (It's interactive!)

What We're Discussing Today

- **Who is the target audience of this talk?**
 - Anyone new to Vulkan ray tracing
- **Primary focus of this talk is on basic debugging of Vulkan's ray tracing API**
 - Assumes you're somewhat familiar with Vulkan's ray tracing
 - Acceleration structures (BLAS and TLAS)
 - GLSL or HLSL ray tracing data structures and functions
 - Ray tracing pipeline creation
 - Assumes you're familiar with Vulkan's descriptor management for resources
 - Image and image view
 - Uniform buffer / constant buffer
 - Storage buffer / structured buffer
 - **vkCmdTraceRaysKHR** is main API function we'll cover
 - Unfortunately we don't have enough time to cover BLAS and TLAS debugging
- **Will talk a bit about debugging rendered pixels**

Basic Ray Tracing Debugging in Vulkan

- **Two types of debugging when it comes to ray tracing**
 - Debugging Vulkan's ray tracing API
 - Debugging rendered pixels
- **Each type has different goals but the basic approach is kind of the same**
 - What tools can I use for debugging?
 - Show me what's happening in my app!
 - What pipeline / shader stage information can I see?
 - Can I debug my shaders?
 - What's some useful tidbits to keep in mind for debugging?
 - Can I examine the pixel values?
 - Can I see how Vulkan interpreted my scene?
 - Can I graphics printf()?
 - How can I debug crashes?

Lets Answer These Questions

- **Debugging Vulkan's ray tracing API**
 - What tools can I use for debugging?
 - Show me what's happening in my app!
 - What pipeline / shader stage information can I see?
 - Can I debug my shaders?
- **Debugging rendered pixels**
 - What's some useful tidbits to keep in mind for debugging?
 - Can I examine the pixel values?
 - Can I see how Vulkan interpreted my scene?
 - Can I graphics printf()?
- **How can I debug crashes?**

Debugging Vulkan's ray tracing API...using a graphics debugger

What tools can I use for debugging?

- **RenderDoc does not currently support ray tracing**
 - Applications using ray tracing API will fail to launch
 - I don't know if or when ray tracing support will happen
- **NVIDIA Nsight Graphics is the only tool that I've found that works (most of the time)**
 - Some of the next few topics will use Nsight Graphics as an example
 - No, this is not a tutorial on Nsight Graphics
 - Okay, maybe a little bit
 - No, this is not a sponsored talk by NVIDIA
 - It's the only tool I've found that works - hopefully it works for you as well!

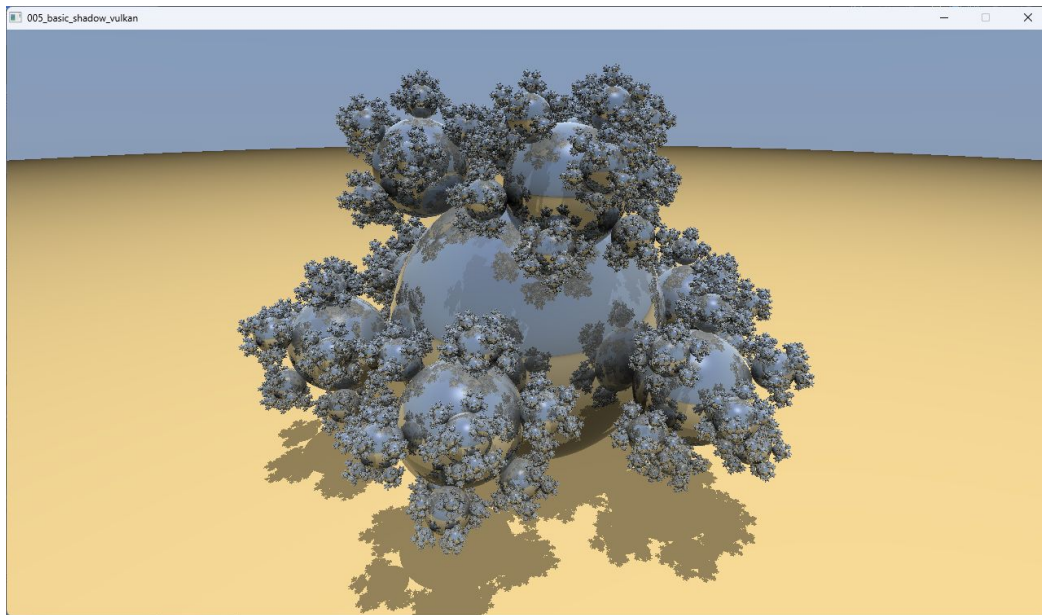
Our example ray tracing scene ([Sphreflake by Eric Haines](#))

We'll look at a capture of this scene over the next few examples.

Some stuff about [our Sphreflake](#):

- **1 RAYGEN shader**
- **2 MISS shaders**
 - Shadowed
 - Not shadowed
- **2 shaders in the hit group**
 - CLOSEST_HIT
 - INTERSECTION
- **3 resources**
 - Acceleration structure
 - Output image
 - Buffer for sphreflake nodes
- **Max recursion depth is set to 5**
- **66431 spheres**
 - Ground is also a sphere

By the Unwritten Laws of Computer Graphics, all ray tracing examples must include either Sphreflake or a reflective checker floor. We'll go with Sphreflake.

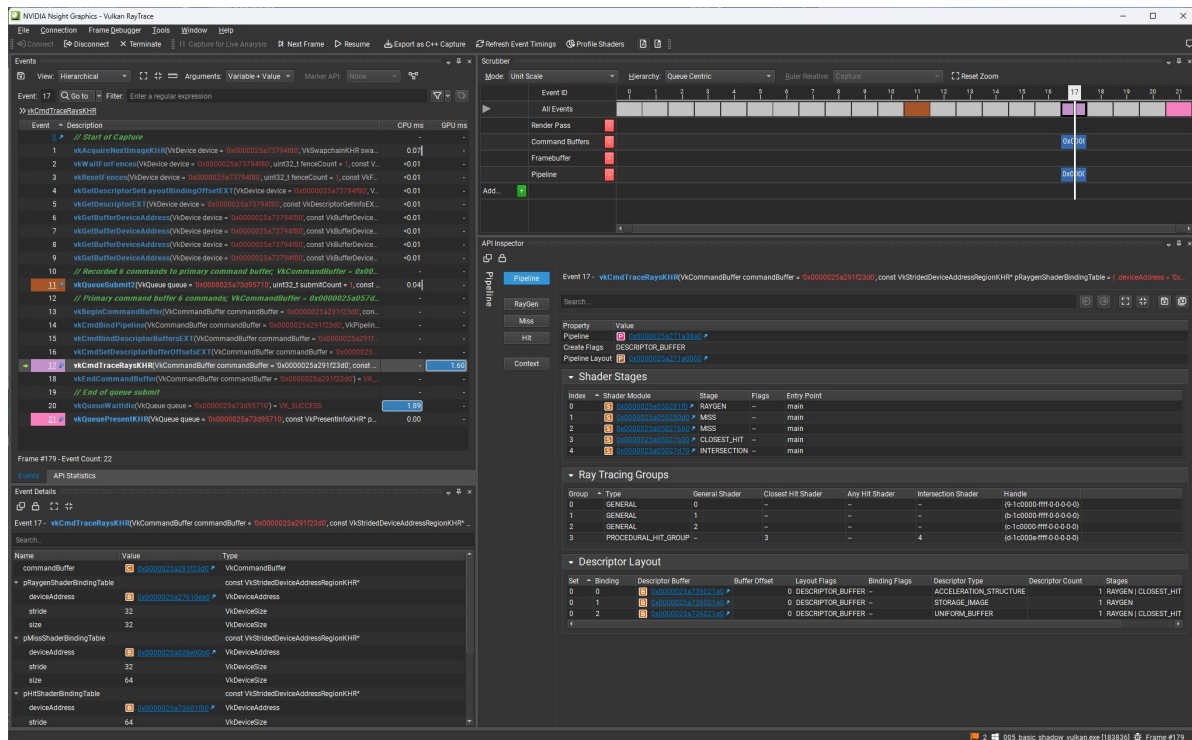


NVIDIA Nsight Graphics

Basic Nsight Graphics capture.

Like other graphics debuggers, Nsight has a events, event details, timeline, API inspection, etc

We'll take a closer look at some of these and see how they're useful for debugging ray tracing.



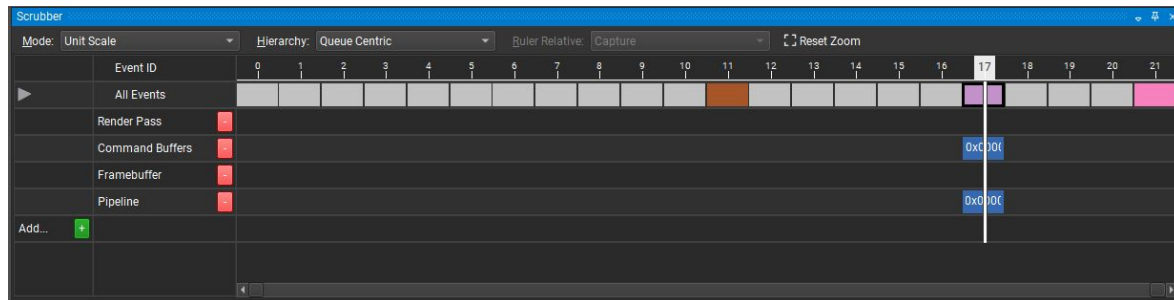
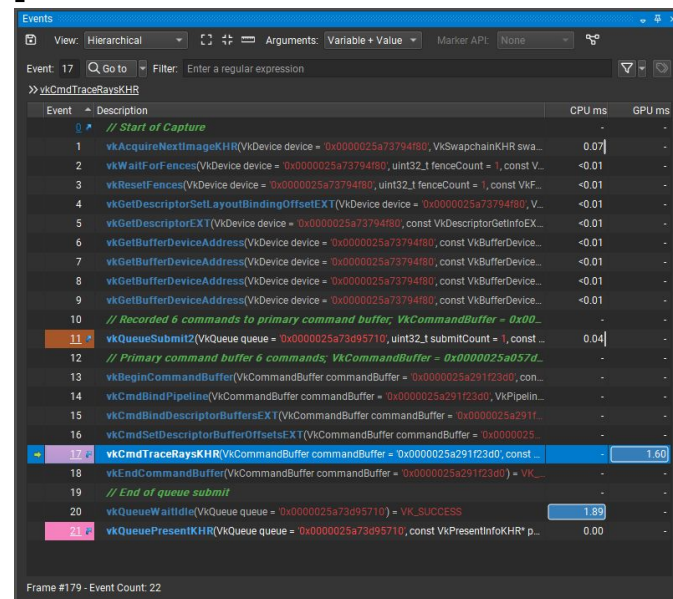
Show me what's happening in my app!

Nsight's Event viewer lets you see the Vulkan calls and commands in the captured frame.

Since Nsight is also a profiler, you'll also find rough CPU and GPU time for draw, dispatch, and ray trace commands in the Event viewer.

These same events also appear in the Scrubber, which shows a timeline version of the events.

In this example, the event we're interested in is Event 17, since it's the call to `vkCmdTraceRaysKHR()`.



vkCmdTraceRaysKHR Example



Will Usher has a pretty cool SBT visualizer here



<https://www.willusher.io/graphics/2019/11/20/the-sbt-three-ways>

```
// As a reminder...
//   shaderGroupHandleSize is the size in bytes of the shader header
//   shaderGroupHandleAlignment is the required alignment in bytes for each shader binding table entry

// The device for the example below has the following properties:
//   VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupHandleSize = 32
//   VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupHandleAlignment = 32

// 1 shader and no shader record data.
VkStridedDeviceAddressRegionKHR rgenSBT = {};
rgenSBT.deviceAddress = GetDeviceAddress(rgenSBTBuffer);
rgenSBT.stride = rayTracingProperties.shaderGroupHandleAlignment; // 32
rgenSBT.size = rayTracingProperties.shaderGroupHandleSize; // 32

// 2 shaders and no shader record data.
VkStridedDeviceAddressRegionKHR missSBT = {};
missSBT.deviceAddress = GetDeviceAddress(missSBTBuffer);
missSBT.stride = rayTracingProperties.shaderGroupHandleAlignment; // 32
missSBT.size = 2 * rayTracingProperties.shaderGroupHandleSize; // 64

// 2 shader and 1 shader record parameter.
// Add 8 bytes for buffer in shader record.
// Align both stride and size to shaderGroupHandleAlignment.
//
// NOTE: If shader record parameter was absent, this would look like the other two SBTs.
//
VkStridedDeviceAddressRegionKHR hitgSBT = {};
hitgSBT.deviceAddress = GetDeviceAddress(&hitgSBTBuffer);
hitgSBT.stride = Align(alignedHandleSize + 8, rayTracingProperties.shaderGroupHandleAlignment); // 64
hitgSBT.size = Align(alignedHandleSize + 8, rayTracingProperties.shaderGroupHandleAlignment); // 64

// Nothing for callable SBT
VkStridedDeviceAddressRegionKHR callableSBT = {};

vkCmdTraceRaysKHR(cmdBuf, &rgenSBT, &missSBT, &hitgSBT, &callableSBT, imageWidth, imageHeight, 1);
```

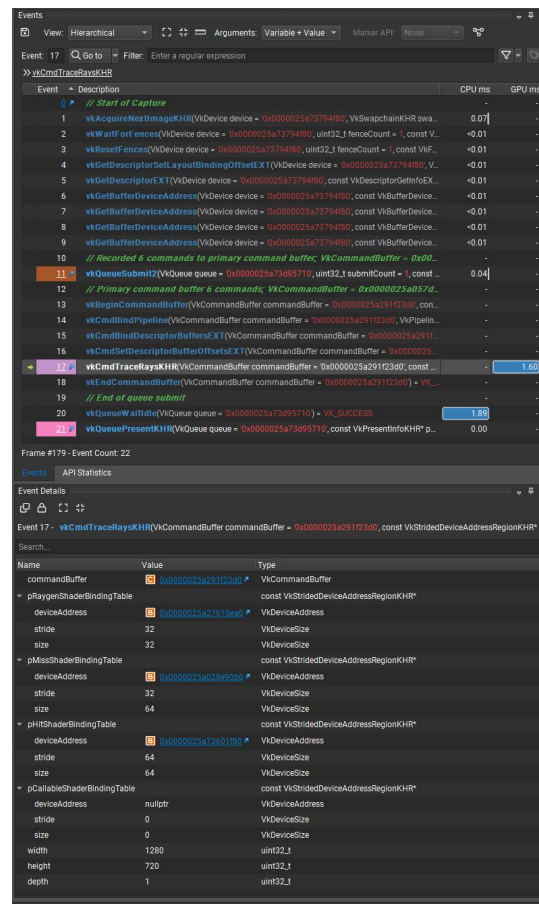
What does vkCmdTraceRaysKHR look like in the debugger?

The Events Details, as well as the tool tip over function calls, shows what was passed into `vkCmdTraceRaysKHR()`.

For debugging purposes, the Events Details lets you see the deviceAddress, stride, and size of what was passed in.

Keep in mind that if Nsight is able to capture a frame, it doesn't mean that the values are 100% correct. It just means that they work well enough not to cause a crash. If the ray tracing output isn't what you expect, this is another place you can look at to see if there's an error.

The validation layers should catch any egregious errors in stride and size. If validation is off or fails to catch the error, the application might crash.



What pipeline / shader stage information can I see?

API Inspector allows examination of the ray tracing pipeline, shader stages, shader groups, descriptor information, etc.

GLSL based shaders will have main as the entry point for all stages. Maybe one day we can have more descriptive entry point names 😊

Shader groups shows the indices for the shader stages they reference. Hit group shows CLOSEST_HIT and INTERSECTION shaders since this example is ray tracing procedurals.

If you need to check the indices for the pipeline's shader groups, here's where to do it.

The screenshot displays the API Inspector window for a Vulkan application. The left sidebar shows the 'Pipeline' tab selected. The main area shows details for 'Event 17 - vkCmdTraceRaysKHR'. The 'Shader Stages' table is highlighted, showing five stages: RAYGEN, MISS, MISS, CLOSEST_HIT, and INTERSECTION, all with 'main' as the entry point. The 'Ray Tracing Groups' table shows three groups: GENERAL (index 0), GENERAL (index 2), and PROCEDURAL_HIT_GROUP (index 3). The 'Descriptor Layout' table shows three sets: 0 (ACCELERATION_STRUCTURE), 1 (STORAGE_IMAGE), and 2 (UNIFORM_BUFFER). The bottom status bar indicates '005_basic_shadow_vulkan.exe [183836] Frame #179'.

Index	Shader Module	Stage	Flags	Entry Point
0	0x0000025a05028100	RAYGEN	-	main
1	0x0000025a05028040	MISS	-	main
2	0x0000025a05027640	MISS	-	main
3	0x0000025a05027630	CLOSEST_HIT	-	main
4	0x0000025a05027d10	INTERSECTION	-	main

Group	Type	General Shader	Closest Hit Shader	Any Hit Shader	Intersection Shader	Handle
0	GENERAL	0	-	-	-	(p-1c0000-ffff-0-0-0-0-0)
1	GENERAL	1	-	-	-	(p-1c0000-ffff-0-0-0-0-0)
2	GENERAL	2	-	-	-	(c-1c0000-ffff-0-0-0-0-0)
3	PROCEDURAL_HIT_GROUP	-	3	-	4	(p-1c000e-ffff-0-0-0-0-0)

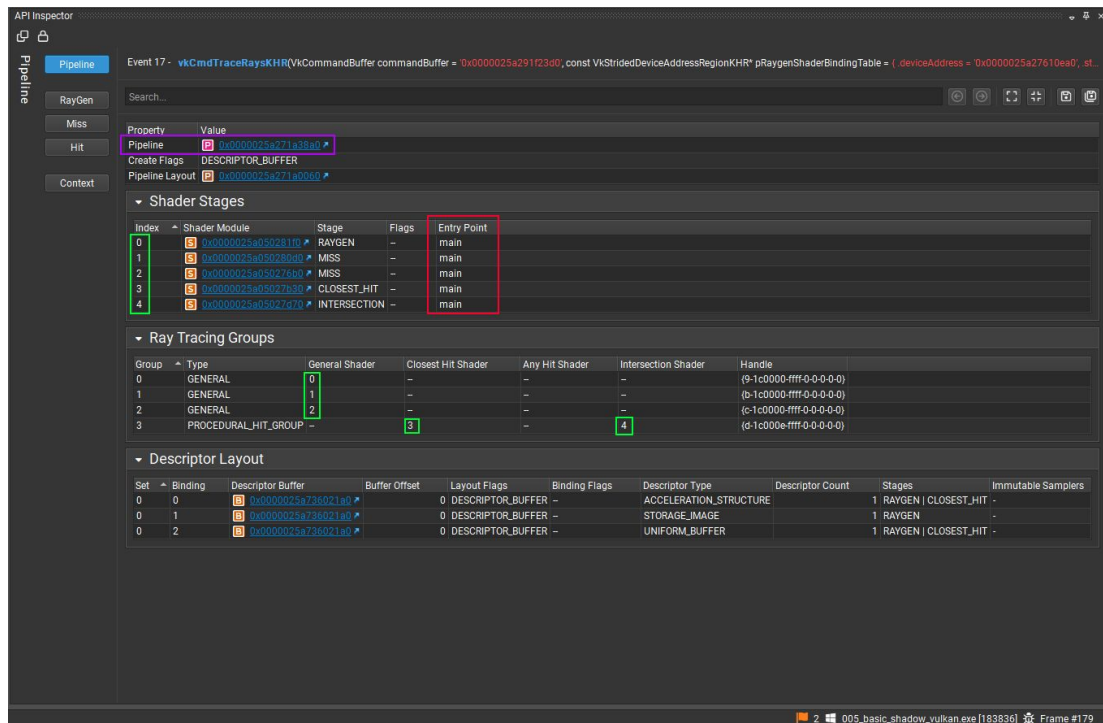
Set	Binding	Descriptor Buffer	Buffer Offset	Layout Flags	Binding Flags	Descriptor Type	Descriptor Count	Stages	Immutable Samplers
0	0	0x0000025a736021a0	0	DESCRIPTOR_BUFFER	-	ACCELERATION_STRUCTURE	1	RAYGEN CLOSEST_HIT	-
0	1	0x0000025a736021a0	0	DESCRIPTOR_BUFFER	-	STORAGE_IMAGE	1	RAYGEN	-
0	2	0x0000025a736021a0	0	DESCRIPTOR_BUFFER	-	UNIFORM_BUFFER	1	RAYGEN CLOSEST_HIT	-

What pipeline / shader stage information can I see?

Descriptor information shows there are 3 resources bound to this pipeline and to which stages these resources are visible.

Max recursion depth isn't visible from here but you can see it in Object Browser by clicking on the pipeline.

It would be mighty useful if Nsight could grab the variable names for the resources - using some type of [SPIRV-Reflect](#).



What pipeline / shader stage information can I see? (HLSL)

Lets take a look at an HLSL based pipeline too.

HLSL based shaders will have the target entry points listed for each stage.

Hit group here, shows only closest hit since this example is ray tracing triangles.

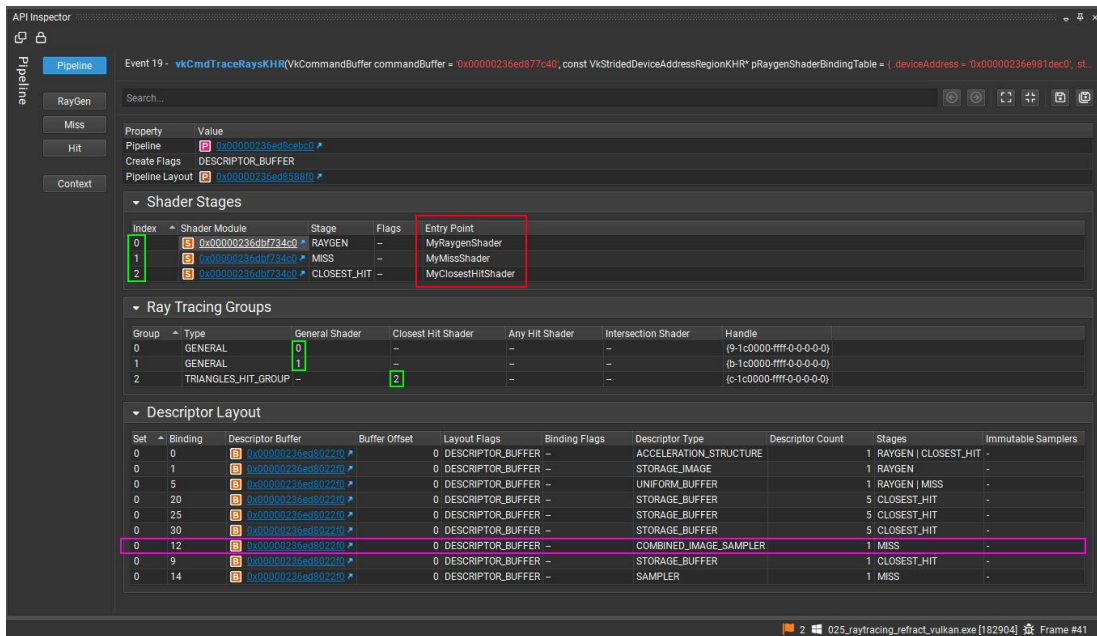
Descriptor type for binding 12 shows COMBINED_IMAGE_SAMPLER. While this information is correct, it's not what we want since HLSL doesn't easily support COMBINED_IMAGE_SAMPLER for Vulkan.

In this case, it turned out to be a bug in the sample with the descriptor type and also the descriptor buffer location. By some weird coincidence it just happened to work.

Again, it would be mighty useful if Nsight could grab the variable names for the resources - using some type of [SPIRV-Reflect](#). When there's more than a few resources, it can get a bit difficult to remember exactly which is at which set/binding.

PR of bug fix

<https://github.com/chaoticbob/GraphicsExperiments/pull/43>



What pipeline / shader stage information can I see?

Ray Generation Shaders

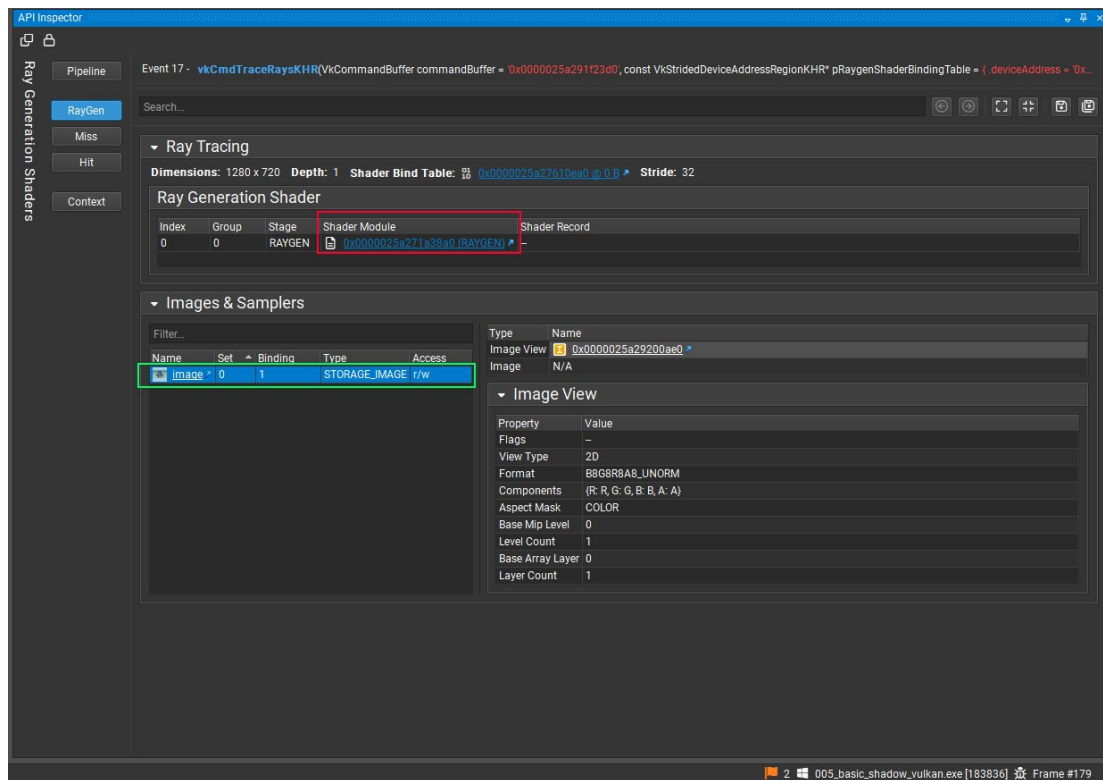
Shader Module object is visible in all shader stages.

Clicking on the shader module will show the source for the shader. More on this in a moment.

Output image(s) are visible in this view. Clicking on an image shows the results for the current vkCmdTraceRaysKHR call.

No data in the shader record for this stage.

It would be mighty useful to see the entry point(s) here as well.



What pipeline / shader stage information can I see?

Miss Shaders

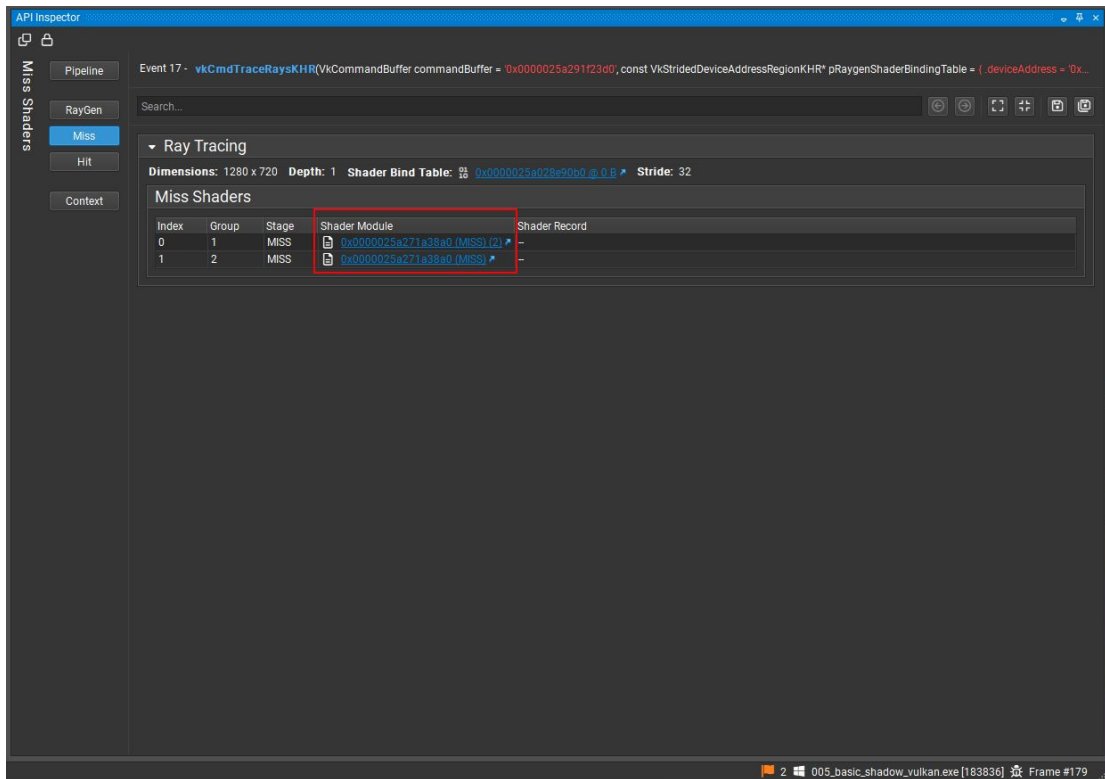
Two MISS shaders in this pipeline.
One for primary rays and the other for shadow.

Like other stages, SBT stride is visible.

Clicking on the shader module will show the source for the shader.

No data in the shader record data for shaders in MISS stage.

It would be mighty useful to see the entry point(s) here as well.



What pipeline / shader stage information can I see?

Hit Shaders

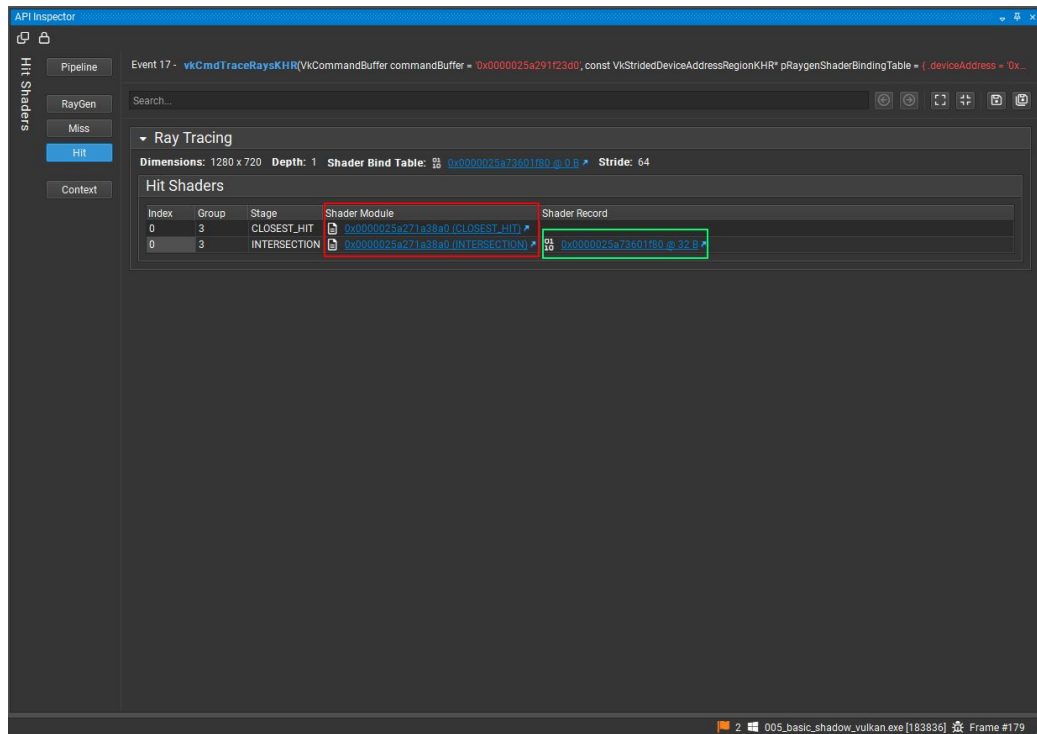
This hit group has CLOSESET_HIT shader and INTERSECTION shader since we're tracing procedurals.

Finally there's something in the shader record data! The INTERSECTION shader access a storage buffer that stores the bounding boxes for each node of the sphere flake. It uses **gl_PrimitiveID** to look up which node.

Key takeaway is that you can use these views to check the shaders and shader record data, and shader source, coming up next.

Don't get too excited about being able to click the resource in the shader record though. Clicking on the buffer in the shader record opens up a data view that's a bit hard to navigate. *For the buffer in the shader record, it seems to interpret the size of the shader record data as the buffer size.*

It would be mighty useful to see the entry point(s) here as well.



Can I debug my shaders?

Not in a way that you can step through it as you do a C/C++ program.

You can view the shader source as SPIR-V or GLSL cross compiled using SPIRV-Cross. *HLSL cross compile didn't work for me.*

This allows you to verify the shaders in each group are the ones that are suppose to be there.

Even if the original shader is in HLSL, it can still be helpful to see the code as GLSL. At very least you can verify it the logic matches the original shader so it's the correct shader.

Wrong shaders do wrong things.

```
0x000001b42b4b330 (INTERSECTION) - main - RAYGEN
GLSL (SPIRV-Cross)  Edit Shader  Compile Shader  Promote Shader  Compile Tool: Glslang (GLSL)  + - -
Ray Tracing Pipeline: 0x000001b42b4b330 Stage: Intersection Shader Module: 0x000001b42b4b330 (INTERSECTION) - main
  GLSL generated using SPIRV-Cross, might not be representative of original shader source
1 #version 460
2 #extension GL_EXT_ray_tracing : require
3 #extension GL_EXT_buffer_reference : require
4 struct Sphere
5 {
6     float minx;
7     float miny;
8     float minz;
9     float maxx;
10    float maxy;
11    float maxz;
12 };
13
14 layout(buffer_reference) buffer SphereBuffer;
15 layout(buffer_reference) buffer _l01;
16 layout(Cbuffer_reference, buffer_reference_align = 8, std430) buffer SphereBuffer
17 {
18     _l01 spheres[];
19 };
20
21 layout(buffer_reference, std430) buffer _l01
22 {
23     float minx;
24     float miny;
25     float minz;
26     float maxx;
27     float maxy;
28     float maxz;
29 };
30
31
32 layout(shader_recordEXT, std430) buffer ShaderRecord
33 {
34     SphereBuffer sphereBuffer;
35 } _l05;
36 hitAttributesEXT vec3 hitNormal;
37
38 vec3 guess_intersections(vec3 orig, vec3 dir, vec3 center, float radius)
39 {
40     vec3 f = orig - center;
41     float a = dot(f, dir);
42     float b = dot(f, f) - (radius * radius);
43     float c = dot(f, dir) * (b / a);
44     float disc = (radius * radius) - dot(s, s);
45     vec2 t = vec2(-c, 0);
46     if (disc <= 0)
47     {
48         float q = b1 + (sqrt(b1) * sqrt(a + disc));
49         float t1 = c / q;
50         float t2 = q / a;
51         t = vec2(t1, t2);
52     }
53     return t;
54 }
55
56 void main()
57 {
58     vec3 orig = gl_WorldRayOriginEXT;
59     vec3 dir = gl_WorldRayDirectionEXT;
60     _l01 _l17;
61     _l17.minx = _l05.sphereBuffer.spheres[gl_PrimitiveID].minx;
62     _l17.miny = _l05.sphereBuffer.spheres[gl_PrimitiveID].miny;
63     _l17.minz = _l05.sphereBuffer.spheres[gl_PrimitiveID].minz;
64     _l17.maxx = _l05.sphereBuffer.spheres[gl_PrimitiveID].maxx;
65     _l17.maxy = _l05.sphereBuffer.spheres[gl_PrimitiveID].maxy;
66     _l17.maxz = _l05.sphereBuffer.spheres[gl_PrimitiveID].maxz;
67     _l01 sphere = _l17;
68     vec3 aabb_min = vec3(sphere.minx, sphere.miny, sphere.minz);
69     vec3 aabb_max = vec3(sphere.maxx, sphere.maxy, sphere.maxz);
70     vec3 center = (aabb_max + aabb_min) / vec3(2);
71     float radius = (aabb_max.x - aabb_min.x) / 2.0;
72     vec3 param = orig;
73     vec3 param_1 = dir;
74     vec3 param_2 = center;
75     float param_3 = radius;
76     vec3 t = guess_intersections(param, param_1, param_2, param_3);
77     if (t.x > 0)
78     {
79         hitNormal = normalize((orig + (dir * t.x) - center);
80         bool _l03 = reportIntersectionEXT(t.x, 0);
81     }
82     if (t.y > 0)
83     {
84         hitNormal = normalize((orig + (dir * t.y) - center);
85         bool _l01 = reportIntersectionEXT(t.y, 0);
86     }
87 }
88
89 }
90
```

```
0x000001b42b4b330 (RAYGEN) - main - RAYGEN
GLSL (SPIRV-Cross)  Edit Shader  Compile Shader  Promote Shader  Compile Tool: Glslang (GLSL)  + - -
Ray Tracing Pipeline: 0x000001b42b4b330 Stage: Ray Generation Shader Module: 0x000001b42b4b330 (RAYGEN) - main
  GLSL generated using SPIRV-Cross, might not be representative of original shader source
1 #version 460
2 #extension GL_EXT_ray_tracing : require
3
4 struct RayPayload
5 {
6     vec3 hitValue;
7     uint recursionDepth;
8 };
9
10 layout(set = 0, binding = 2, std140) uniform CameraProperties
11 {
12     mat4 viewInverse;
13     mat3 projInverse;
14     vec3 eyePosition;
15 };
16
17 layout(location = 0) rayPayloadEXT RayPayload payload;
18 layout(set = 0, binding = 0) uniform accelerationStructureEXT topLevelAS;
19 layout(set = 0, binding = 1, rgba8) uniform writable Image2D image;
20
21 void main()
22 {
23     vec3 pixelCenter = vec3(gl_LaunchIDEXT.xy) + vec3(0.5);
24     vec2 inUV = pixelCenter / vec2(gl_LaunchSizeEXT.xy);
25     vec2 d = (inUV * 2.0) - vec2(1.0);
26     d.y = -d.y;
27     vec3 origin = can.viewInverse * vec3(0.0, 0.0, 0.0, 1.0);
28     vec3 target = can.projInverse * vec3(d.x, d.y, 1.0, 1.0);
29     vec3 direction = normalize((can.viewInverse * vec3(normalize(target.xyz, 0.0);
30     float tmin = 0.00000000479795130538949429687;
31     float tmax = 1000.0;
32     payload.hitValue = vec3(0.0);
33     payload.recursionDepth = 0;
34     traceRayEXT(topLevelAS, 0, 255, 0u, 0u, 0u, origin.xyz, tmin, direction.xyz, tmax, 0);
35     imageStore(image, ivec2(gl_LaunchIDEXT.xy), vec4(payload.hitValue, 1.0));
36 }
37
```

```
0x000001b42b4b330 (MISS) - main - MISS
GLSL (SPIRV-Cross)  Edit Shader  Compile Shader  Promote Shader  Compile Tool: Glslang (GLSL)  + - -
Ray Tracing Pipeline: 0x000001b42b4b330 Stage: Miss Shader Module: 0x000001b42b4b330 (MISS) - main
  GLSL generated using SPIRV-Cross, might not be representative of original shader source
1 #version 460
2 #extension GL_EXT_ray_tracing : require
3
4 struct RayPayload
5 {
6     vec3 hitValue;
7     uint recursionDepth;
8 };
9
10 layout(location = 0) rayPayloadInEXT RayPayload payload;
11
12 vec3 CubicBezier(float t, vec3 P0, vec3 P1, vec3 P2, vec3 P3)
13 {
14     float s = 1.0 - t;
15     float a = (s * s) * s;
16     float b = ((3.0 * s) * s) * t;
17     float c = ((3.0 * s) * t) * t;
18     float d = (t * t) * t;
19     return (((P0 * a) + (P1 * b)) + (P2 * c) + (P3 * d));
20 }
21
22 void main()
23 {
24     float t = (0.0 + 1.0) / 2.0;
25     vec3 C0 = vec3(0.0099999997764825828922815625, 0.00999999997764825828922815625, 0.01999999995526651641840793123);
26     vec3 C1 = vec3(0.926888616249108137109375, 0.926888616249108137109375, 0.9999999953741160662);
27     vec3 C2 = vec3(-0.7708886437701632921875, 0.6888886277971801721875, 0.76999997981316581063);
28     vec3 C3 = vec3(0.189999997415814268906375, 0.112088886437701632921875, 0.57899999185302734375);
29     float param = t;
30     vec3 param_1 = C0;
31     vec3 param_2 = C1;
32     vec3 param_3 = C2;
33     vec3 param_4 = C3;
34     vec3 C = CubicBezier(param, param_1, param_2, param_3, param_4);
35     payload.hitValue = C;
36 }
37
```

In place edit of a MISS shader

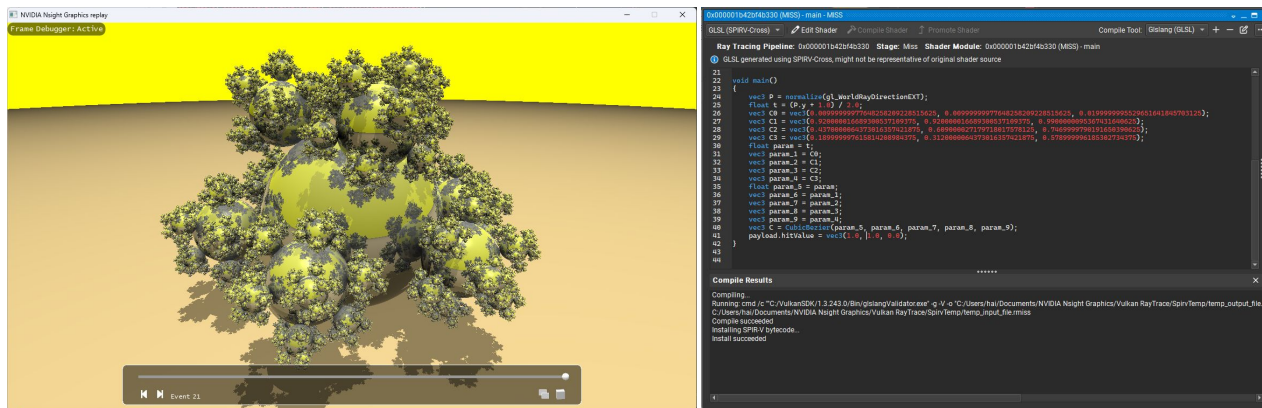
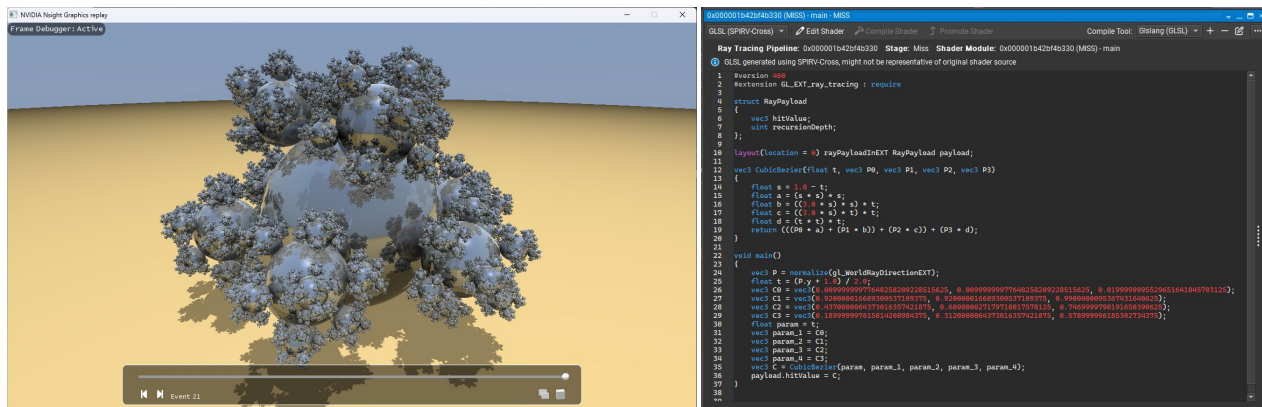
Like other graphics debuggers, Nsight lets you in place edit a shader in ray tracing pipelines.

Lets change the payload color of a MISS shader from its original value to yellow.

Keep in mind that this is a very simple and small program, results from larger and more complex applications may vary.

Even with the above caveat, pretty neat feature to have for debugging.

This feature was not available in the D3D12 mode for me.



Debugging rendered pixels

What's some useful tidbits to keep in mind for debugging?

- **Make it easy to enable Vulkan Validation Layers**
- **Coordinate system of output image is (0, 0) Upper Left**
 - True for both Vulkan and D3D12
 - As a result, RAYGEN shaders often have some version of $d.y = -d.y$
- **Most Vulkan swapchain implementations support STORAGE_IMAGE**
 - This means you can write ray traced output to swapchain images
 - Remember to note the swapchain image format if you're planning to copy it back to CPU
- **Using HLSL makes it easier to go to D3D12 for second opinion**
- **Note the conventions that your program uses**
 - Left hand or right hand 3D coordinate system
 - Shading done in world space, view space, or object space
 - Triangle winding order
 - Pre or post matrix multiplication order
- **Have graphics printf() shaders ready to go**
- **Max Recursion Depth**
 - NVIDIA = 31
 - AMD = 1
 - Intel = ??? (sorry, I don't have access to an Intel GPU)
- **Remember that this real time ray tracing**
 - You can add real time debugging utilities the same way you would a raster graphics program

Can I examine the pixel values?

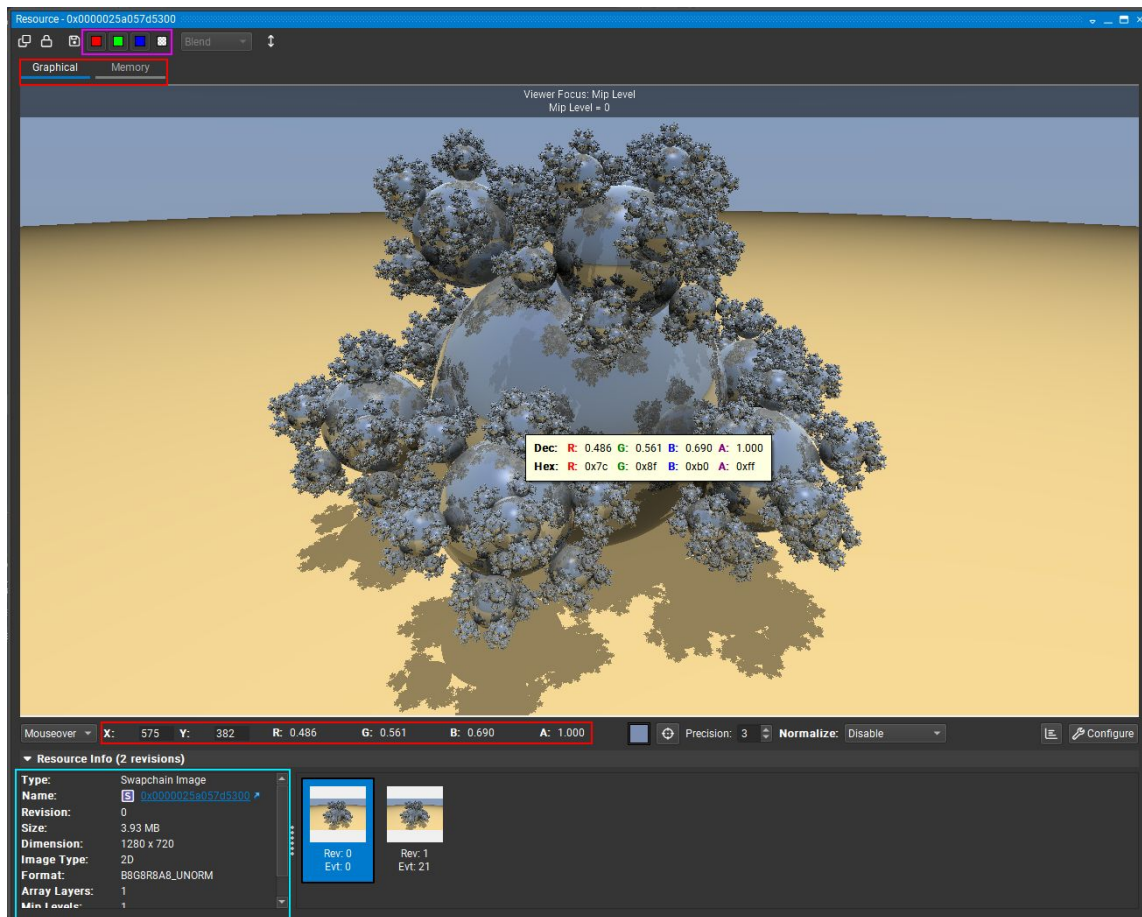
Nsight's resource viewer lets you click to view pixel values in decimal floating point or hex integer. There doesn't seem to be a setting for decimal integer.

Like other graphics debugging tools, there's channel selector in case you want view particular channels.

A histogram (not pictured) is available by clicking on the graph button next to the Configure button on the right.

Lower panel left shows the properties of the image you're viewing.

Nsight calls this the Graphical view of the image.



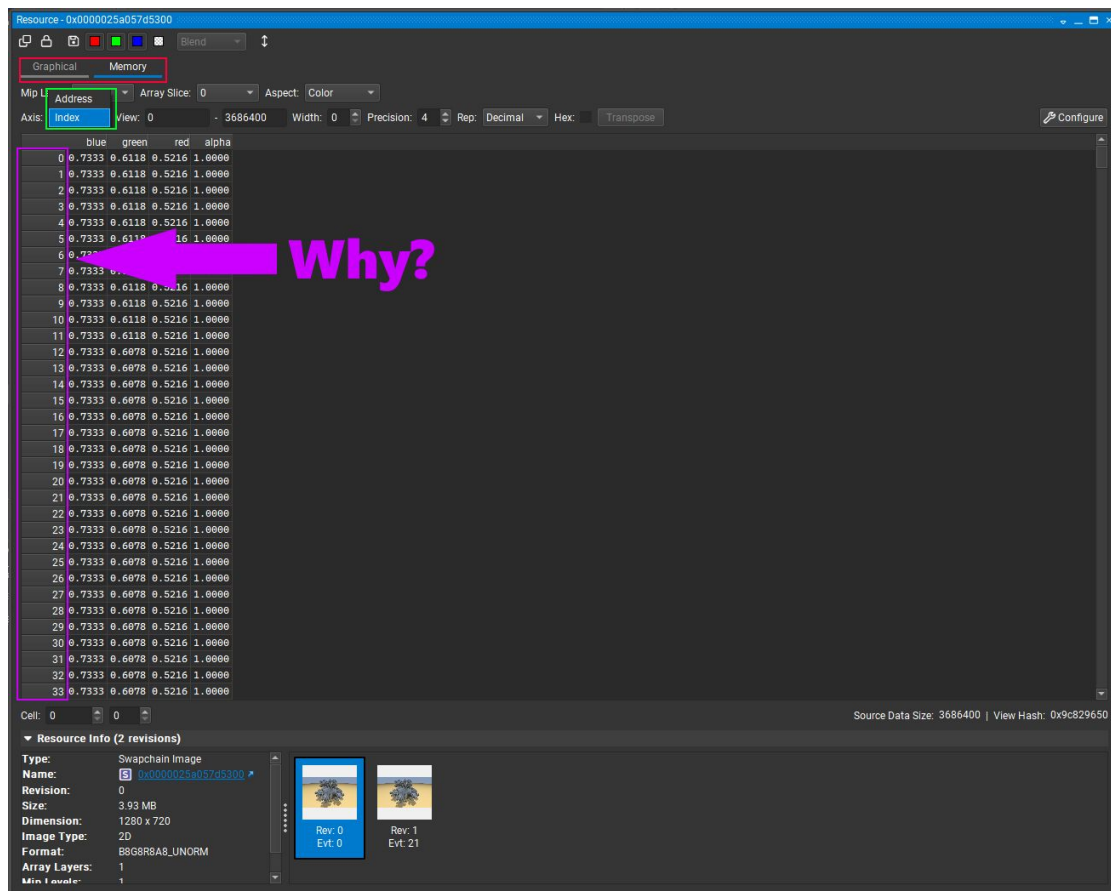
Can I examine the pixel values?

Additionally, there is also the Memory view of the image.

In this mode, you can select between Address or Index for the Axis. This flips between memory address and indices for the first column.

There's no pixel coordinate mode for Axis, so you'll need to do a little math to figure out which address or index refers to a pixel at particular (x,y).

Keep your calculator handy.

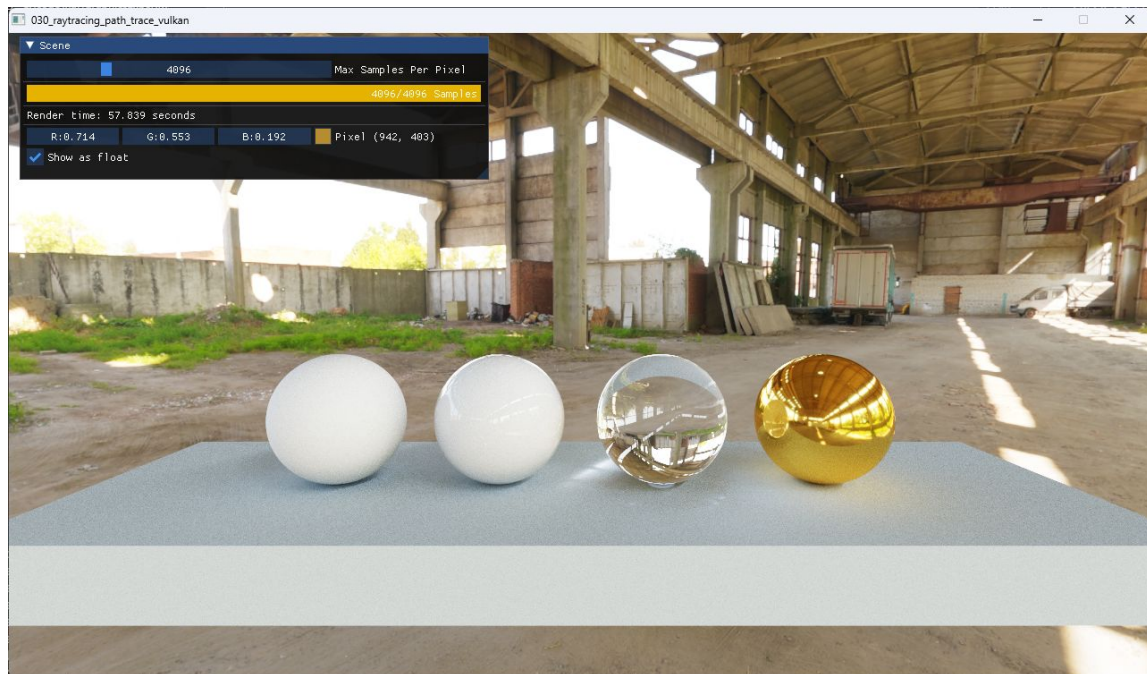


Can I examine the pixel values?

Alternatively, one can rig up a custom solution with ImGui:

- Copy the image from the GPU to CPU
- Determine pixel coordinates from mouse event(s)
- Read values from CPU image
- Update ImGui color values
 - Visualize color with color picker

Brad Loos was nice enough to [do just this](#) to one of the existing samples.



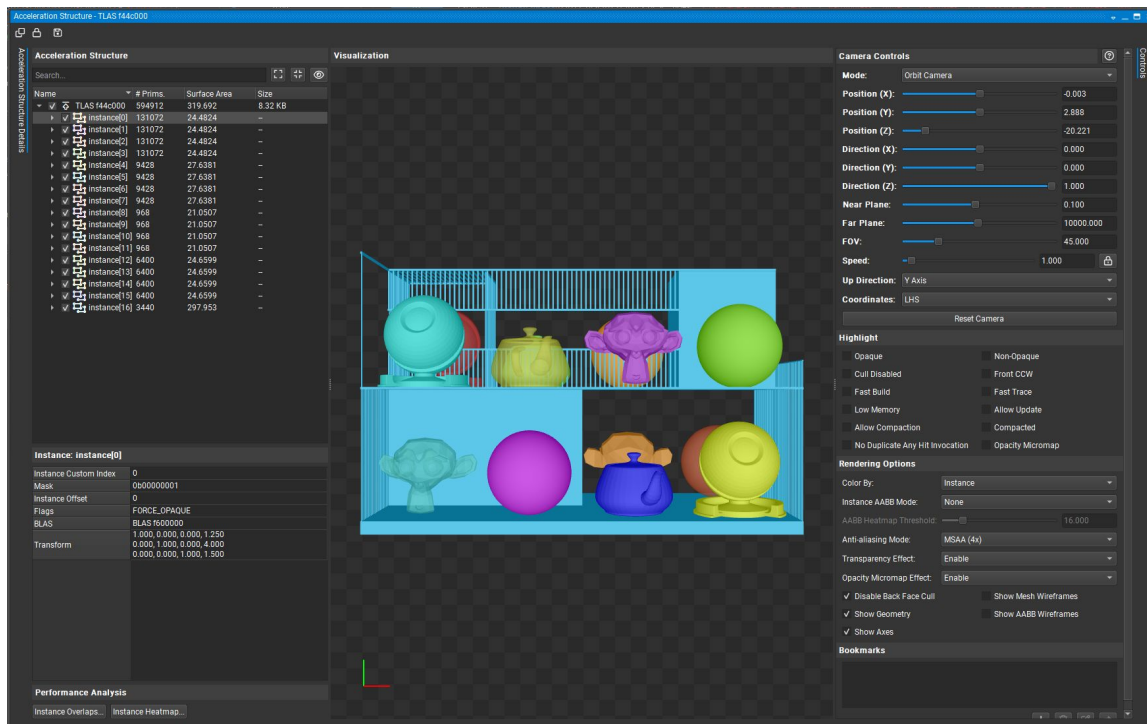
Can I see how Vulkan interpreted my scene?

Sort of using graphics printf() techniques - more on this next but before that...

Nsight's D3D12 mode has an Acceleration Structure Details view that visualizes the scene for a DispatchRays call. PIX also has a similar feature.

This is an incredibly useful debugging feature to have help developers see how the API interpreted the scene based on the data provided by the application.

This visualization lets the developers know if their geometry, instances, and acceleration structure setups are correct. Or at the very least, what the developer was expecting.



Can I graphics printf()?

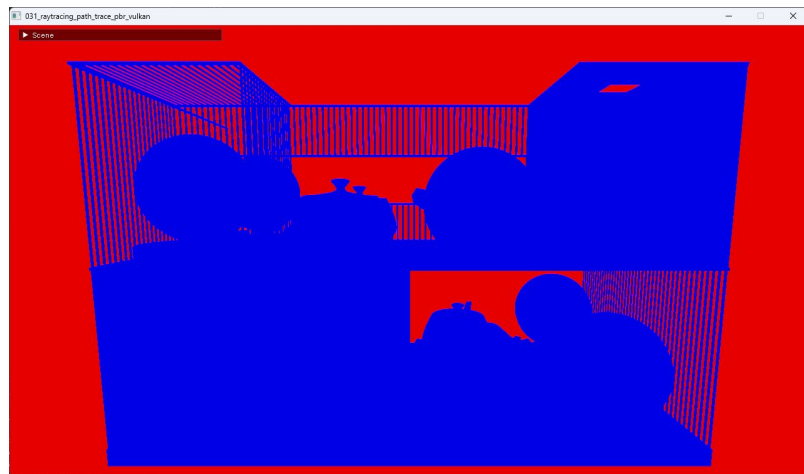
Short of having a debugger with scene visualization, we have to resort to more tried and true techniques of computer science: **printf()**.

In the case of graphics, we print/write colors instead of variable values. Although, it's possible to **printf()** too - but that's beyond the scope of this presentation.

This image shows the most basic version of a ray tracing printf() using **red** for misses and **blue** for hits:

```
[shader("miss")]
void MyMissShader(inout RayPayload payload)
{
    payload.color = float4(1, 0, 0, 1);
}

[shader("closesthit")]
void MyClosestHitShader(inout RayPayload payload, in MyAttributes attr)
{
    payload.color = float4(0, 0, 0, 1);
}
```



Can I graphics printf()?

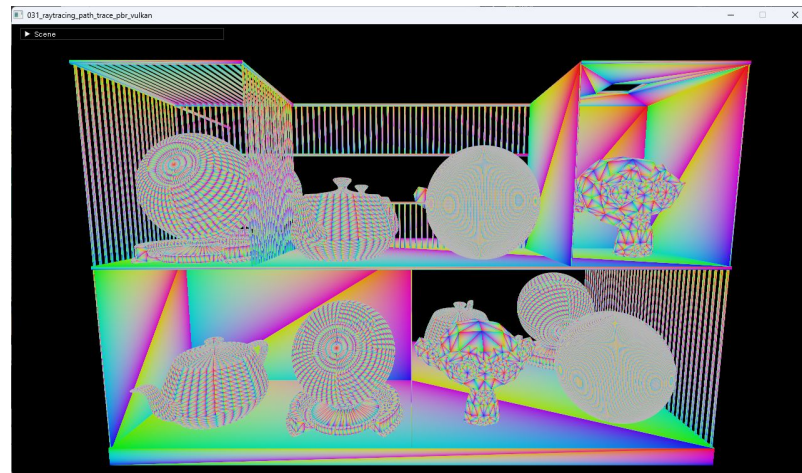
If it's necessary to see some separation in the objects, barycentrics or hit position is helpful.

This shader writes out the barycentric values for each intersection with TRIANGLE geometry:

```
[shader("closesthit")]
void MyClosestHitShader(inout RayPayload payload, in MyAttributes attr)
{
    float3 bc = float3(1 - attr.barycentrics.x - attr.barycentrics.y,
                      attr.barycentrics.x,
                      attr.barycentrics.y);
    payload.color = float4(bc, 1);
}
```

Barycentrics and hit position (in world space) can be derived without needing access to vertex attribute data. This makes it quick and easy to do without additional setup.

You can also use instance or primitive indices with a little more math.



Can I graphics printf()??

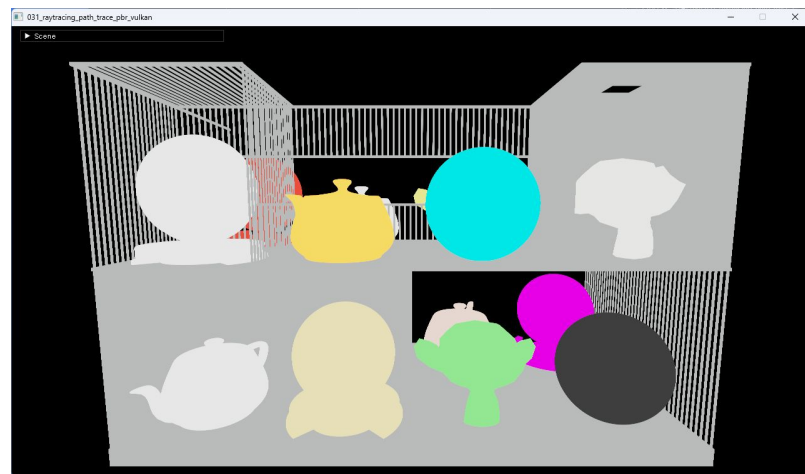
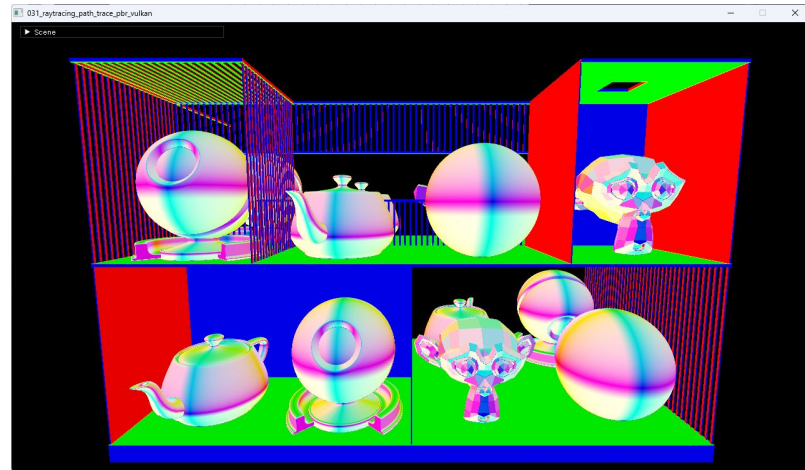
Normals or material properties can also be useful depending on what you're looking to debug. This approach may seem a bit adhoc, but having suite of copy/paste shaders ready-to-go for debugging can make life in Vulkan ray tracing a bit easier.

Sometimes, a copy/paste might be quicker than fighting with a graphics debugger 🤔

However, if you're already setup in the debugger (such as Nsight) - you can just copy/paste the shaders and see results immediately using the shader edit feature. Just make sure your ready-to-go shaders are in GLSL.

Some ideas for ready-to-go printf shaders:

- **Color values for hit and miss shaders**
- **Barycentrics**
 - Not available with PROCEDURAL geometry
- **Hit position**
- **Hit distance**
 - May need to normalize using near/far and scale afterwards for better visualization
- **Instance / primitive IDs**
- **Vertex attributes (assuming these are accessible)**
 - Positions
 - Normals
 - Texture coordinates
- **Material properties**
 - Base color
 - Roughness
 - Metallic



Something looks off...sometimes a second opinion is helpful

Top image is Vulkan

Bottom image is D3D12

This is a super simple Gouraud shader:

```
// Lambert shading
float3 lightPos = float3(2, 5, 5);
float3 L = normalize(lightPos - hitPosition);
float d = 0.8 * saturate(dot(L, N));
float a = 0.2;
// Multiply diffuse + ambient by material color
float3 color = (float3)(d + a) * Materials[geoIdx];
```

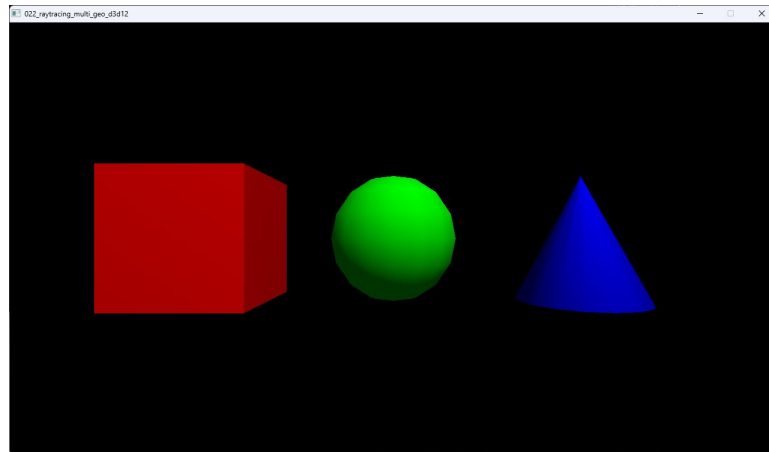
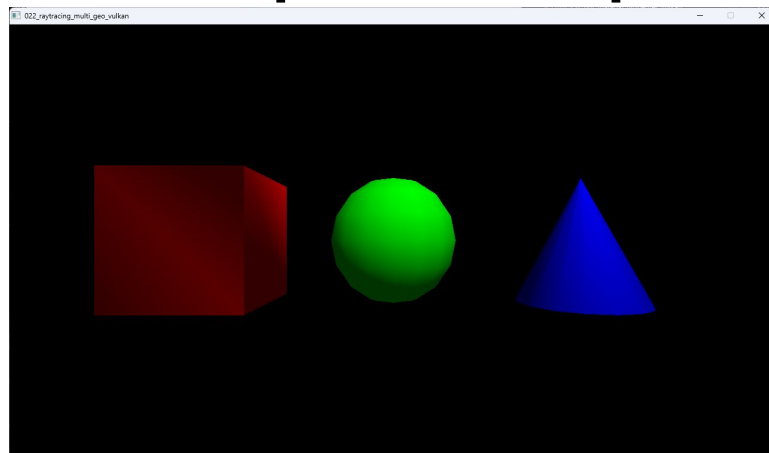
Red cube's color is a bit off in Vulkan version due to a bug in the HLSL and C++ code:

```
// HLSL
StructuredBuffer<Triangle> Triangles[2] : register(t4); // Index buffers
StructuredBuffer<float3> Positions[2] : register(t7); // Position buffers
StructuredBuffer<float3> Normals[2] : register(t10); // Normal buffers
// C++
binding.descriptorCount = 2;
```

The **2** is supposed to be a **3** since there are **3** objects in the scene. HLSL code is shared by both APIs, but D3D12 was a lot more forgiving about the bug than Vulkan.

🐛 PR for bug fix 🐛:

<https://github.com/chaoticbob/GraphicsExperiments/pull/40>



How can I debug crashes?

Debugging Vulkan's ray tracing API...by thinking very hard.

Debugging crashes

Lets break crashes down into two categories

1. Crashes before ray tracing starts
2. Crashes while ray tracing

Short of a driver bug, both categories of crashes are most likely invalid access or invalid API usage.

Where you start the investigation is what differentiates them.

Lets take a very quick look at some starting points for each case.

Crashes before ray tracing starts

- Vulkan's Validation Layers can help!
- If a crash occurs while building acceleration structures (1/2)
 - Check the geometry setup
 - Device addresses, strides, vertex format, etc.
 - Did **vkGetAccelerationStructureBuildSizesKHR** return an error?
 - Does the argument for **pBuildInfo** have the correct information?
 - Is **geometryCount** correct?
 - Is **pGeometries** not NULL?
 - Are values in the **pMaxPrimitiveCounts** array argument correct?
 - **pMaxPrimitiveCounts** can be a bit confusing at first, but it's an array of the number of triangles or AABBs you have in each **pBuildInfo->pGeometries** entry.
 - For example, if you have 3 geometry entries that are all triangles and they have 3, 4, 5 triangles respectively then:
 - **pBuildInfo->geometryCount** = 3
 - **pBuildInfo->pGeometry** would point to an array containing 3 **VkAccelerationStructureGeometryKHR** filled out with buffer device address, vertex, index information corresponding the number of triangles
 - **pGeometries** would point to an array containing [3, 4, 5]

Crashes before ray tracing starts

- If a crash occurs while building acceleration structures (2/2)
 - Does the build size information returned by **vkGetAccelerationStructureBuildSizesKHR** make sense?
 - Does the scratch buffer size make sense?
 - Does the acceleration structure size make sense?
 - Has memory been allocated and bound to the buffer objects for the scratch buffer and acceleration structure?
 - Did **vkCreateAccelerationStructureKHR** return an error?
 - All the information in the create info valid?
 - Does the build geometry info for **vkCmdBuildAccelerationStructuresKHR** matches what was passed to **vkGetAccelerationStructureBuildSizesKHR**?
 - Is the data in arguments for **pInfos** and **ppBuildRangeInfos** correct?
 - Did you wait for the GPU to finish processing the command buffer that's building the acceleration structure?
 - In the case of TLAS, is the device addresses for the BLASes correct?

Crashes while ray tracing

- **Vulkan's Validation Layers can also help here!**
- **Check pipeline shader group shader stage indices**
- **Revisit our old friend vkCmdTraceRaysKHR**
 - Are the device addresses, strides, and sizes for the SBTs correct?
 - Are argument values for width, height, and depth correct?
- **Check the same things that you would check in a raster program**
 - Pipeline creation parameters
 - Descriptor set layout configurations
- **Do all the resources used by the shader have actually resources bound?**
 - Vulkan Validation Layer should catch this case, but it never hurt to double check
- **Try moving buffer and parameters in the shader records to global descriptors and/or push constants**
 - Especially if you're using descriptor buffers and the buffer in the shader record refers to a bindless resource
 - As of this presentation, descriptor buffers is still somewhat new and not all drivers have been thoroughly battle tested with program that use them

Crashes while ray tracing

- **Try simplifying the ray tracing pipeline to track down which stage could be causing the crash**
 - Simplify RAYGEN shader so it just writes out the UV coordinate for the pixel to the output image
 - Simplify the MISS and HIT shaders to write out colors
 - Simplify INTERSECTION shaders to always return true or false
 - Bring back resource access one by one for each shader stage to see if there's an invalid resource access
 - In most cases you can just read from a resource and write to the payload so that value gets written out to the output.
 - This doesn't produce anything meaningful visually but it prevents the compiler from Dead Code Eliminating (DCE) the resource access.
- **It's unlikely that a shader arithmetic operations can cause a crash**
 - But it's possible that the HLL shader compiler or the driver shader compiler might have a bug.
 - If you have a large shader and suspect a crash might be somewhere in its code
 - Try the tried-and-true method of commenting out code and slowly reintroducing it.
- **For large command buffers**
 - Might be worth investigating a buffer marking extension
 - Or a tool like NVIDIA's AfterMath to track down exactly which vkCmdTraceRaysKHR call could be causing the crash

Unsolicited Feedback

Nsight is definitely useful for Vulkan ray trace debugging...but some small quality of life changes would be great...or maybe just some explanation of why somethings work the way they do.

- **API Inspector in D3D12 offers a bit richer information about the pipeline than than the Vulkan version**
 - Such as Shader names and offsets in the table
- **Descriptor Layout view should show the variable names for the resource from the SPIR-V**
 - [SPIRV-Reflect](#) and libraries like it offers extensive information about the descriptors that would be useful to see in the debugger
 - Buffer offsets, member data types, etc
 - Am sure you're aware, just a friendly reminder 😊
- **Acceleration Structure Details view would be awesome to have for Vulkan**
 - This would be immensely helpful with understanding how Vulkan interpreted the BLAS/TLAS data
- **When some of the windows are undocked, they seem to be mildly obsessive about their sizing**
 - Object Browser and Resource viewer do not remember their size and opens up to some ridiculously obtrusive size
- **Max recursion depth should be visible without needing to inspect the properties in the Object Browser**
- **Resource viewer should be able to show buffer data passed into via shader record correctly**
 - This works as expected in D3D12 mode
- **Auto data formatting of data in buffers would be nice**
 - Data layouts of the buffer can also be determined via SPIRV-Reflect or other reflection libraries

Thank You! + Resources

- **Thanks to the following folks for their help on this ray tracing journey**
 - Brad Loos
 - Matt Pettineo
 - Tobias Hector
- **Resources**
 - Eric Haines' Sphereflake:
 - <https://erich.realtimerendering.com/rtrt/index.html>
 - Will Usher's awesome blog post on SBTs with visualizer:
 - <https://www.willusher.io/graphics/2019/11/20/the-sbt-three-ways>
 - Images used in this presentation are from samples that can be found here:
 - <https://github.com/chaoticbob/GraphicsExperiments>
 - Collection of graphics samples done in Vulkan, D3D12, and Metal
 - Computer graphics is fun!