Open Standards for Real Time Ray Tracing
VP NVIDIA, President Khronos
June 2020
Khronos Connects Software to Silicon

Open interoperability standards to enable software to effectively harness the power of multiprocessors and accelerator silicon

3D graphics, XR, parallel programming, vision acceleration and machine learning

Non-profit, member-driven standards-defining industry consortium

Open to any interested company

All Khronos standards are royalty-free

Well-defined IP Framework protects participant’s intellectual property

Founded in 2000

>150 Members - 40% US, 30% Europe, 30% Asia
Khronos Active Initiatives

- **3D Graphics**
  - Desktop, Mobile, Web
  - Embedded and Safety Critical

- **3D Assets**
  - Authoring and Delivery

- **Portable XR**
  - Augmented and Virtual Reality

- **Parallel Computation**
  - Vision, Inferencing, Machine Learning

Guidelines for creating APIs to streamline system safety certification
Pervasive Vulkan

Desktop and Mobile GPUs

Platforms

- Desktop
- Android (Android 7.0+)
- Apple (via porting layers)
- Media Players
- Consoles
- Virtual Reality
- Cloud Services
- Game Streaming
- Embedded

Engines

Cider | Godot | Unity | Unreal | Cryengine

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

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

Vulkan 1.1 Extensions
- Maintenance updates plus additional functionality
- Timeline semaphores
- DX/HLSL compatibility
- Bindless resources
- Reduced precision arithmetic
- Formal memory model
- Buffer references
- SPIR-V 1.5

Roadmap Discussions
- Ray Tracing
- Variable Rate Shading
- Accelerated Video Encode/Decode
- Machine Learning Primitives
- Mesh Shaders

January 2020

This work is licensed under a Creative Commons Attribution 4.0 International License
Vulkan Ray Tracing

Coherent ray tracing framework with flexible merging of rasterization and ray tracing

Set of Extensions to Vulkan, GLSL and SPIR-V seamlessly integrates ray tracing into Vulkan 1.X

Provisional Extensions - released March 2020 - enabling developer feedback before finalization

Familiar to users of existing proprietary ray tracing APIs but also introduces new implementation flexibility

Hardware agnostic - can be accelerated on existing GPU compute and dedicated ray tracing cores

Primary focus on meeting desktop market demand for both real-time and offline rendering

The industry’s first open, cross-vendor, cross-platform standard for ray tracing acceleration
Step 1: Create Efficient Scene Geometry

Ray tracing may use a huge numbers of rays
Specialized data structures for interrogating scene geometry are necessary for efficient acceleration

Acceleration Structures
Contains low-level 3D geometry to be ray traced and high-level references into the geometry
Opaque internal organization details
Each vendor can optimize for processing for their hardware
E.g. Bounding Volume Hierarchy (BVH) for rapidly determining if there is any geometry in the path of a ray

Build Acceleration Structure
Vulkan driver integrates supplied geometry into its two-level Acceleration Structure

Using a BVH data structure to enable efficient ray tracing through a 3D scene
Step 2: Traverse Scene with Rays

Two ways to launch rays into the scene

Ray Tracing Pipelines
A new type of graphics pipeline
Implicit management of ray intersections
Application compiles a set of shaders into the pipeline to provide desired ray and material processing

Ray Queries
Any type of shader can launch a ray at any time
Shader can process intersection data however it wishes
Shader controls how traversal proceeds

Vulkan Ray Tracing Pipeline

1. Launch 2D/3D grid of rays into scene contained in an Acceleration Structure

2. ‘Intersection’ Shader computes ray intersections Ray-triangle intersections are built-in

3. Invoke ‘Any Hit’ Shader if intersection is found. Multiple intersections possible - arbitrary order

4. Invoke ‘Closest Hit’ shader on the closest intersection of the ray OR Invoke ‘Miss’ Shader if no hit is found. Can trace more rays
Vulkan Ray Tracing includes GLSL and SPIR-V Extensions
Enabling compiled GLSL/SPIR-V shaders to operate in a Ray Tracing Pipeline - similar to HLSL features used in Direct3D’s DXR

HLSL and Vulkan with DXC
Microsoft’s DXC HLSL compiler was open sourced in Jan 2017
Google and others have added SPIR-V code generation to DXC with Microsoft’s knowledge and approval
Vulkan developers can now choose between GLSL and HLSL!

HLSL for Vulkan Ray Tracing
NVIDIA added code generation to DXC to generate SPIR-V for the Vulkan Ray Tracing extension from HLSL

Developers can port HLSL shaders with minimal changes between Vulkan Ray Tracing and DXR
Pipeline Libraries

Ray Tracing Pipelines can use many shaders
Potentially orders of magnitude more shaders
(1000s) than traditional applications to handle
diverse tracing techniques and material types

Compilation Bottleneck
Compiling many shaders into a Ray Tracing
Pipeline can be computationally intensive and
cause application bottlenecks and stuttering

Vulkan Pipeline Library Extension
Enables a library of SPIR-V shaders to be
incrementally compiled into an existing Ray
Tracing Pipeline saving significant processing load

Primary Ray + Ambient Light
Reflection + Direct Local light / Shadow
Indirect Bounce
Indirect Sun Light / Shadow

Multiple shaders used to build complex lighting in a Quake 2 scene
Host Offload of Setup Operations

Ray tracing setup compute workloads can be intensive
Building Acceleration Structures and compiling Ray Tracing Pipelines
Two Vulkan mechanisms to offload and control setup workloads
on the host CPU(s) for smoother, faster rendering

Build Acceleration Structure on Host
Use the host to build Acceleration Structure in host memory and then
copy to the GPU - rather than build directly on the GPU
Final size of Application Structure is known before copying to the GPU
- enabling optimized GPU memory allocation

Deferred Host Operations
Driver returns deferred work handle to application for later execution
Application controls work execution and can chose to distribute onto
multiple cores and background threads

Deferred Host Operations can be used to
asynchronously use multiple CPU cores to build
Acceleration Structures on the host

Using Deferred Host Operations to build a
complex Acceleration Structure using
multiple CPU cores to offload the work from
the GPU for faster, smoother framerates
# Vulkan Ray Tracing and DXR

<table>
<thead>
<tr>
<th>Feature</th>
<th>Vulkan Ray Tracing</th>
<th>DX12 / DXR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ray Tracing Pipelines</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Ray Queries</td>
<td>Optional</td>
<td>DXR Tier 1.1 Inline ray tracing</td>
</tr>
<tr>
<td>Language for Ray Tracing Shaders</td>
<td>GLSL or HLSL</td>
<td>HLSL</td>
</tr>
<tr>
<td>Pipeline Libraries</td>
<td>Yes</td>
<td>DXR Tier 1.1 AddToStateObject()</td>
</tr>
<tr>
<td>Build Acceleration Structure on Host</td>
<td>Optional</td>
<td>No</td>
</tr>
<tr>
<td>Deferred Host Operations</td>
<td>Optional</td>
<td>No</td>
</tr>
<tr>
<td>Capture/Replay Support for Tools (e.g. RenderDoc)</td>
<td>Optional</td>
<td>No</td>
</tr>
</tbody>
</table>

**Straightforward to port code between Vulkan Ray Tracing and DXR**

Including re-use of ray tracing shaders written in HLSL
API Layering

<table>
<thead>
<tr>
<th>Layers Over</th>
<th>Vulkan</th>
<th>OpenGL</th>
<th>OpenCL</th>
<th>OpenGL ES</th>
<th>DX12</th>
<th>DX9-11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vulkan</td>
<td></td>
<td></td>
<td>Zink</td>
<td>clspv</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>gfx-rs</td>
<td></td>
<td></td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ashes</td>
<td></td>
<td></td>
<td>GLOVE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>clspv</td>
<td>Angle</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>vkd3d</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DXVK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>WineD3D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenGL</td>
<td>gfx-rs</td>
<td></td>
<td></td>
<td>Angle</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ashes</td>
<td></td>
<td></td>
<td>Microsoft</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>‘GLOn12’</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Microsoft</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>‘CLOn12’</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Microsoft</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D3D11On12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DX12</td>
<td>gfx-rs</td>
<td></td>
<td></td>
<td>Angle</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ashes</td>
<td></td>
<td></td>
<td>Microsoft</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>‘GLOn12’</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Microsoft</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>‘CLOn12’</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Microsoft</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D3D11On12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DX9-11</td>
<td>gfx-rs</td>
<td></td>
<td></td>
<td>Angle</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ashes</td>
<td></td>
<td></td>
<td>Microsoft</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>‘GLOn12’</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Microsoft</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>‘CLOn12’</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Microsoft</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D3D11On12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metal</td>
<td>MoltenVK</td>
<td></td>
<td></td>
<td>clspv</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>gfx-rs</td>
<td></td>
<td></td>
<td>+ SPIRV-Cross?</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MoltenGL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Angle</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**COLUMNS** Benefit ISVs by making an API available everywhere

**ROWS** Benefit Platforms by adding APIs

- **vkd3d used to port DX12 titles to Linux**
- **Enabled by growing robustness of open source compiler ecosystem**
Vulkan Ray Tracing Extension Schedule

January 2018
Vulkan Ray Tracing Subgroup Created

March 2020
Vulkan Ray Tracing Provisional Spec
- Extension Specifications
- Press Release
- Blog
- Beta Drivers

Review and Integration of IHV and Developer Feedback
https://khr.io/vkrayprovfeedback
- Streamline layering of DXR over Vulkan Ray Tracing
- Multiple Usability Tweaks

Conformance Test
- Development
- Multiple Implementations
- Added support for Provisional Spec to DXC HLSL compiler
- Sample code using Provisional Spec

2H20
Updated Specification

Expecting only minor application updates to migrate from March 2020 Provisional Spec
ANARI - Analytic Rendering API

Scientific Visualization Portability
Common API to describe objects in a scene
The renderer takes care of generating imagery
Ray tracing was catalyst to create a standard
But ANARI design will enable any style of renderer
Not limited to scientific visualization
E.g. Data Analytics and other domains with lots of data

Acceleration APIs:
Embree, OptiX, Radeon Rays, CUDA, OpenCL, Vulkan etc.

Renderers:
Intel OSPray, Radeon ProRender, NVIDIA VisRTX etc.

SciViz Apps and Engines

Scene Graphs

Hardware:
CPUs, GPUs etc.

Industry Support

Working Group Announced 3rd March 2020
Thank you!
Any questions?

www.khronos.org
ntrevett@nvidia.com
@neilt3d