Vulkan Ray Tracing Overview
Neil Trevett
VP, NVIDIA | President, Khronos
April 2021
Vulkan Ray Tracing

The industry’s first open, cross-vendor, cross-platform standard for ray tracing acceleration

Released November 2020

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

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 today - but designed to encourage mobile ray tracing too
Vulkan Ray Tracing Development Timeline

January 2018
Vulkan Ray Tracing Subgroup Created

March 2020
Provisional Extension Specifications
- Extension Specifications
- Press Release
- Blog
- Beta Drivers

March 2020
Review and Integration of IHV and Developer Feedback
- https://khr.io/vkrayprovfeedback
- Streamline layering of DXR over Vulkan Ray Tracing
- Multiple Usability Tweaks

November 2020
Final Extension Specifications
- Conformance Test Development
- Multiple Implementations
- Added support for Provisional Spec to DXC HLSL compiler
  - Sample code using Provisional Spec

December 2020
Vulkan SDK with Ray Tracing Updated Samples
Ray Tracing Refresher

Rasterization and ray tracing can both use triangles to describe scene geometry - but only ray tracing calculates physical phenomena such as shadows, reflections and refractions.

Ray tracing calculates how rays intersect and interact with scene geometry, materials and light sources.

Ray Tracing is a Flexible Technique

Programmers need programmable flexibility to trace rays through scenes for a wide variety of visual effects - some examples…
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 traverse Acceleration Structure
Launching rays into scene to generate results

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

Model courtesy of PTC
Traversal with Ray Tracing Pipelines

Implicit Ray and Shader Execution Management

Application compiles collection of shaders to be invoked on ray/geometry intersection into the Ray Tracing Pipeline.

Controlling which shaders are invoked during traversal enables a wide variety of ray tracing techniques.

Hit Shaders can query the materials they intersect e.g., transparent materials can be handled differently than opaque.

Intersection and Hit shaders can control how traversal proceeds.

Shader stages can communicate parameters and results through ray payload structures.

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.
Traversals with Ray Queries

**Explicit Ray Management within a Single Shader**

Any standard shader (e.g., compute, vertex and fragment shaders) can invoke a single ray traversal at any time.

Uses an Acceleration Structure and a geometric description of the ray being traced.

Shader reads intersection properties during traversal and controls how materials are processed and how the traversal proceeds.

1. Shader launches a single ray into scene contained in an Acceleration Structure

2. Shader takes action depending on intersection properties
   - Can trace more rays

Graphics, Compute or Ray Tracing Pipeline

1. Shader launches a single ray into scene contained in an Acceleration Structure
2. Shader takes action depending on intersection properties
   - Can trace more rays

Ray Query

Handle Result
No
Proceed?

Acceleration Structure Traversal

Confirm Hit, Generate Details or Terminate
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

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
### 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></td>
<td>vkd3d-Proton</td>
<td>DXVK</td>
<td>WineD3D</td>
</tr>
<tr>
<td>OpenGL</td>
<td>gfx-rs</td>
<td>Zink</td>
<td>clspv</td>
<td>GLOVE Angle</td>
<td>Angle</td>
<td>WineD3D</td>
</tr>
<tr>
<td>DX12</td>
<td>gfx-rs</td>
<td></td>
<td></td>
<td></td>
<td>Angle</td>
<td>Microsoft D3D11On12</td>
</tr>
<tr>
<td>DX9-11</td>
<td>gfx-rs</td>
<td></td>
<td></td>
<td></td>
<td>Angle</td>
<td></td>
</tr>
<tr>
<td>Metal</td>
<td>MoltenVK</td>
<td>gfx-rs</td>
<td></td>
<td>clspv + SPIRV-Cross?</td>
<td>MoltenGL Angle</td>
<td></td>
</tr>
</tbody>
</table>

Vulkan Ray Tracing designed to efficiently support layered DirectX 12 DXR. **Wine 6.0** will support Vulkan specification version 1.2.162 which includes Vulkan Ray Tracing.

E.g. Vkd3d-Proton used to port DX12 titles to Linux with Valve Proton.

Vulkan is an effective porting target for multiple APIs e.g., for bringing DX12 games to Linux.
Vulkan Ray Tracing and Shading Languages

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
Vulkan Ray Tracing and DirectX 12 DXR

<table>
<thead>
<tr>
<th>Feature</th>
<th>Vulkan Ray Tracing</th>
<th>DirectX12 DXR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ray Tracing Pipelines</td>
<td>At least one must be available</td>
<td>Yes</td>
</tr>
<tr>
<td>Ray Queries</td>
<td></td>
<td>DXR Tier 1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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>

Porting code between Vulkan Ray Tracing and DXR is straightforward
Including re-use of ray tracing shaders written in HLSL

Final extensions designed to enable layering DXR over Vulkan Ray Tracing
To enable ray tracing in DX12 over Vulkan Layers such as vkd3d
Diverse Applications Using Vulkan Ray Tracing

Quake II RTX
Source code available on GitHub. Get it on Steam

aiSim 3.0
All sensor modalities accurately simulated in driving sim

Horizontal light field on Looking Glass
Complete Light field Rendered as fast as single view

Helios Renderer
Realtime display of VFX effects displayed on movie studio LED walls
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

Scene Graphs
SciViz Apps and Engines

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

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

Hardware:
CPUs, GPUs etc.

Industry Support

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

Hardware:
CPUs, GPUs etc.

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

Scene Graphs
SciViz Apps and Engines

Industry Support

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

© The Khronos® Group Inc. 2021 - Page 16
Developer Vulkan Ray Tracing Resources

Production Vulkan Drivers with Vulkan Ray Tracing are shipping

AMD Radeon Adrenalin 20.11.3 drivers for Radeon RX 6000 Series

NVIDIA R460 drivers for All RTX GPUs
GeForce GTX 1660 with 6GB+ of memory
GeForce GTX 1060+ with 6GB+ of memory

Intel Xe-HPG GPUs, available in 2021

Khronos welcomes developer feedback on Vulkan GitHub issues tracker

Vulkan Ray Tracing Samples
Vulkan Ray Tracing Guide
How to use the Vulkan Ray Tracing extensions
Exploring deeper technical details of the Vulkan Ray Tracing specifications
Best practices for blending Vulkan rasterization and ray tracing techniques

Khronos Member Materials
Deep dive Vulkan Ray Tracing Tutorial
How to use Vulkan Ray Tracing to Create a complete mini-path tracer
A Vulkan-based glTF ray tracing viewer with open source on GitHub

www.khronos.org