Qualcomm

Demystifying Ray Tracing on Qualcomm® Adreno™ GPUs

Aleksandra Krstic

Director of Engineering, GPU Research Team, QTI

akrstic@qti.qualcomm.com

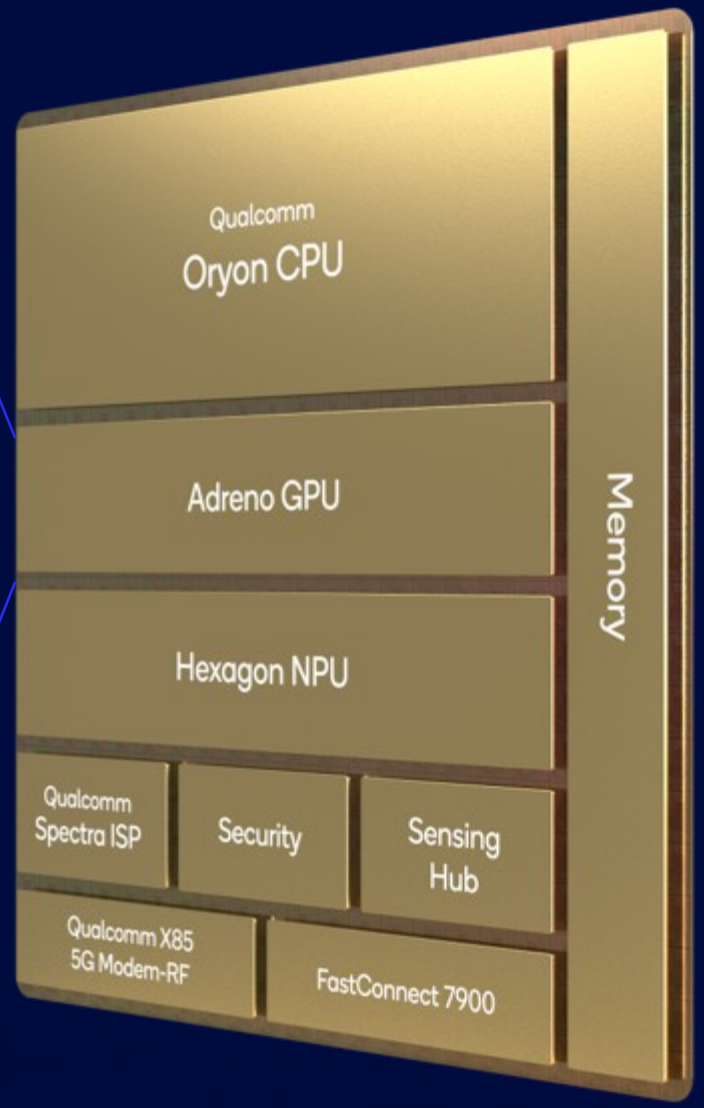
Acknowledgements

Thank you to my management and all hardworking RT engineers at Qualcomm!

- Qualcomm® Adreno™ GPUs
- Acceleration Structure
- Traversal
- Tools

Qualcomm® Adreno™ GPUs: Game smarter, not harder

- Qualcomm Adreno High Performance Memory (HPM)
- Tile Memory Heap
- Unreal Engine 5 Advanced Optimizations
  - ✓ Real-time Hardware-Accelerated Ray Tracing with Lumen
  - ✓ Global Illumination
  - ✓ Nanite Virtualized Geometry
  - ✓ Screen Space Ambient Occlusion
  - ✓ Temporal Anti-Aliasing
  - ✓ Temporal Super Resolution
- Snapdragon Adaptive Game Configuration
  - ✓ Gaming Frame Rate Conversion
  - ✓ Auto Variable Rate Shading
- Snapdragon Game Super Resolution 2.0
- Adreno Frame Motion Engine 3.0
- Snapdragon Game Post-Processing Accelerator
- HDR gaming (10-bit color depth, Rec. 2020 color gamut)
- Snapdragon Shadow Denoiser
- API support: OpenGL® ES 3.2, OpenCL™ 3.0 FP, Vulkan® 1.4, DirectX12.2 Ultimate



Qualcomm® Adreno™ GPUs: Ray Tracing support

- Ray Tracing is…
  - Given a group of triangles and a ray: Origin + Direction + [tmin ,tmax]…
    - Does the ray hit anything?
    - What is the first thing it hits?
  - Used in rendering algorithms to model light transport
  - Very computationally intensive
    - 100K → 100M triangles in scene, 1M → 1B rays per frame
  - Amenable to parallelization
  - Accelerated on GPUs

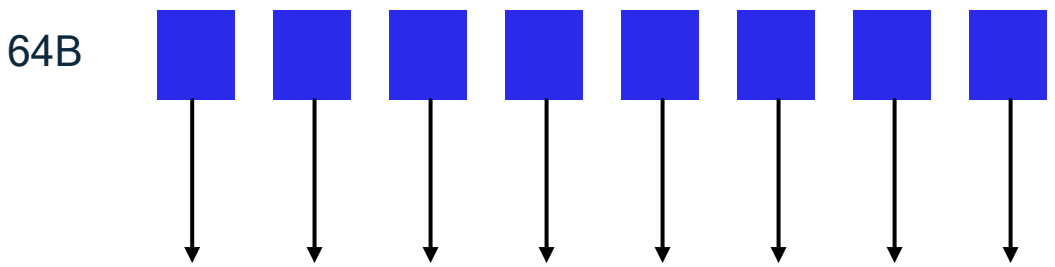
- Adreno GPUs support:
  - Vulkan
    - VK_KHR_acceleration_structure
    - VK_KHR_ray_query
    - VK_KHR_deferred_host_operations
    - VK_KHR_ray_tracing_pipeline
    - VK_KHR_pipeline_library
    - VK_KHR_ray_tracing_position_fetch
    - Corresponding SPIR-V and GLSL extensions
  - **DXR1.1 → First Windows on ARM**

Acceleration Structures

Why do we need acceleration structures?

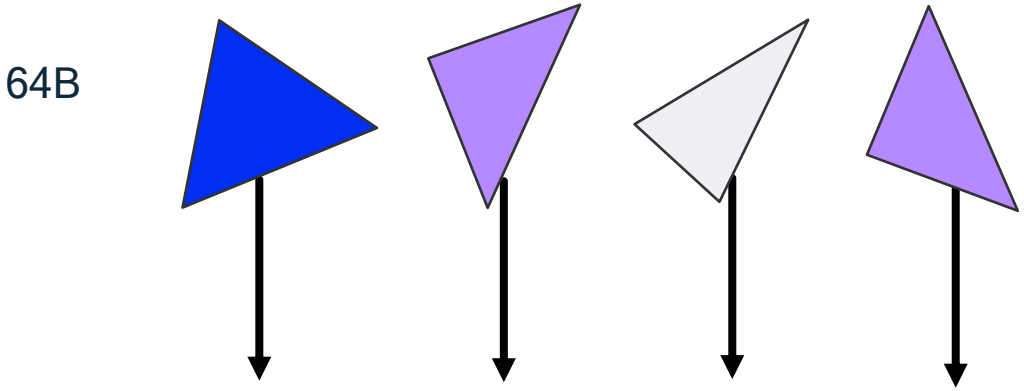
- Any piece of geometry within the scene can potentially be intersected by a ray
  - Naïve: 1M rays traced against 1M polygons → 1 trillion ray-triangle tests!
- API defines Top Level (TLAS) and Bottom Level (BLAS) Acceleration Structures
  - BLAS is built over an individual geometry
  - TLAS is built over instances of BLASes
- Bounding Volume Hierarchies (BVH) used to represent acceleration structures
  - Wrap geometry into recursive spatial structures, such as axis-aligned bounding boxes (AABBs)
  - Boxes can be of arbitrary size and can overlap
  - Boxes are organized into a tree structure
  - Brings down complexity of the query operation to log(n)
- There are many variants of BVHs (binary, 4-way, compressed, etc.)
- BVH type has a massive impact on tree traversal performance!

Acceleration Structures



Internal Node has child AABBs and index of first child

or



Leaf Node has triangle coordinates
and indices of each triangle

BVH stats	Value
Internal node width	8
Leaf node width	1-4 triangles
Node size	64B
Bytes/tri	35 (average)
Bytes/instance	78 (average)

- All nodes are packed tightly with no empty space
- Children are adjacent, but can be anywhere
- Random access to contiguous blocks

Best Practices for Acceleration Structures on Adreno GPUs

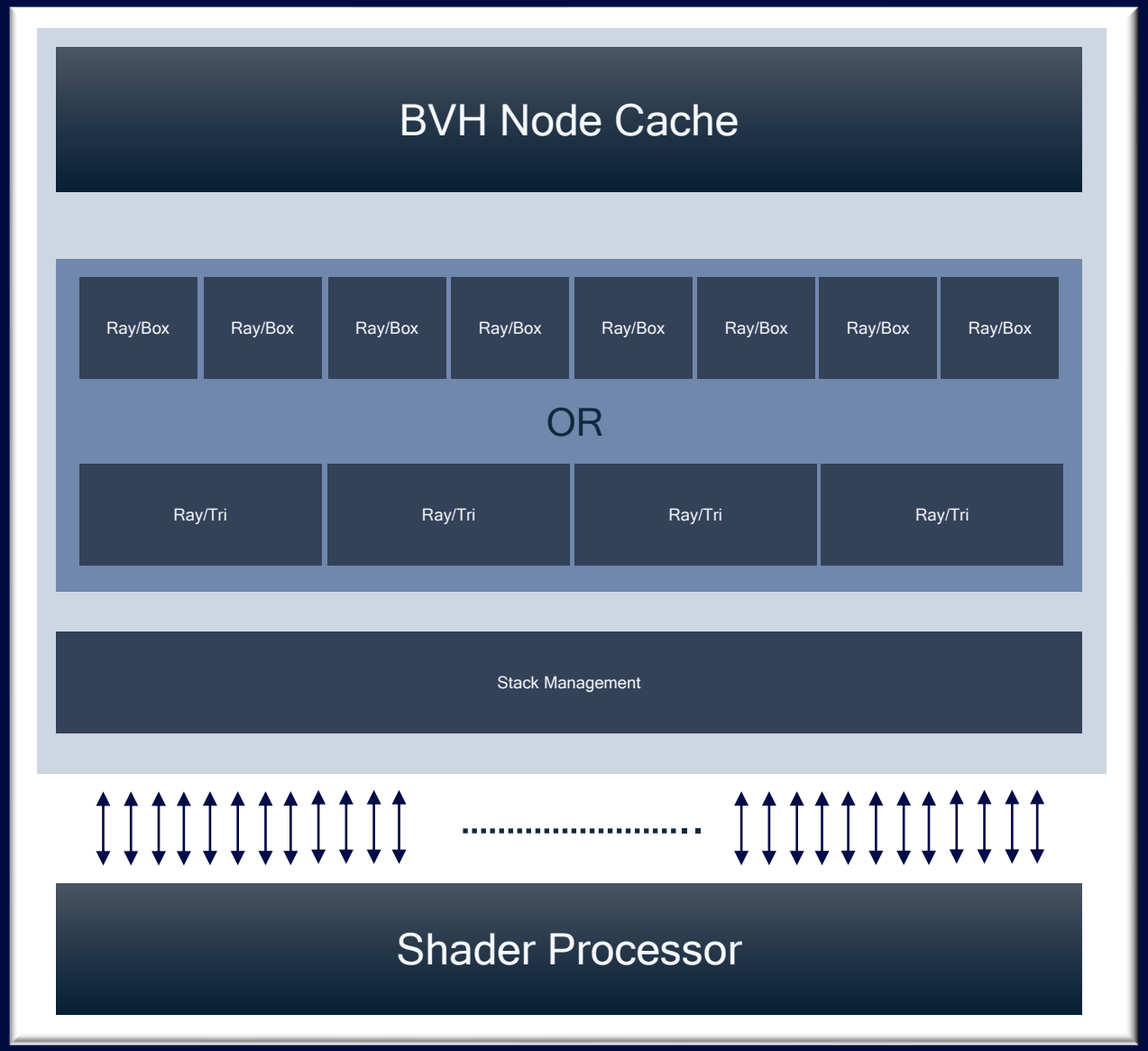
- Build on GPU
- For full throughput target about 500K prims per build
  - Combine smaller BLASes into larger ones
  - Build BLASes in parallel
- Refit rather than rebuild
  - FAST_BUILD is about 2.5x faster than FAST_TRACE
  - Refit is about 9x faster than FAST_TRACE
  - Amortize the cost of refit/rebuild over multiple frames
- Use FAST_TRACE flag whenever possible
  - Especially for static geometry!
- Simplify your acceleration structures
  - Aggressively cull your geometry as much as your content allows: frustum culling, portal culling, level-of-detail, etc.

Traversal

Ray Traversal Steps

- Rays traverse through BVH nodes applying ray-box or ray-triangle intersection at each step
- Tons of floating-point math
- Divergence
  - Rays like to fetch different sections of the BVH → memory divergence
  - Rays have variable lifetimes → traversal divergence
- Requires traversal stack on GPU for every ray in flight

Ray Tracing Unit



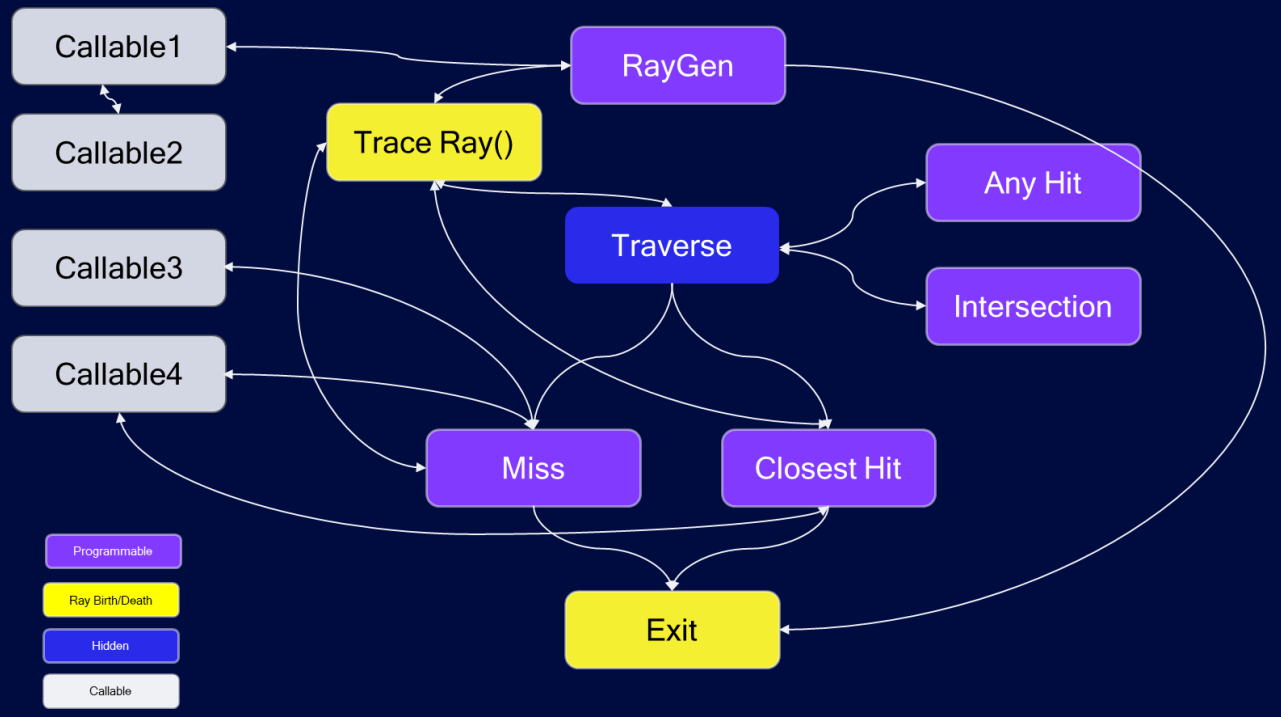
- Wave size: 64
- Every ray traverses independently
- Cache size is 16KB
- @1GHz we can do:
  - 12B Rays/sec
  - 96B Ray/Box Intersections/sec
  - 48B Ray/Tri Intersections/sec

Best Practices for Inline Ray Tracing (Ray Query)

- Ray Query can be used in any shader stage, but compute is your friend
- Keep instruction cache in mind
  - Minimize call sites to `proceed()`
- Be mindful of your GPR usage
  - Create one ray query object and reuse as needed
  - Use ray query data directly through intrinsics
- Use Ray flags
  - If tracing shadow rays: gl_RayFlagsTerminateOnFirstHitEXT
  - If all geometry is opaque: gl_RayFlagsOpaqueEXT or gl_RayFlagsCullNoOpaqueEXT
  - If any of the above, no need to use `<while(rayQueryProceedEXT(rayQuery)){}>`; simplify code by directly calling `<rayQueryProceedEXT(rayQuery);>`
  - Remove dead code, factor away non-uniform branches and looping, and shrink each shader's total instruction as much as possible
- Use 16-bit precision in your shader
  - Vulkan API supports this with VK_KHR_shader_float16_int8

Ray Pipelines on Adreno GPUs

- Complex Graph which consists of multiple shader stages and supports recursion
- Depending on the app (and compiler heuristics)
  - Ubershader
  - Shader graph
- Ubershader primarily used for “simpler” pipelines
  - Not many hit groups
  - No recursion
  - Smaller shaders
- Shader graph approach
  - For complex pipelines
  - Capable of repacking



Tools

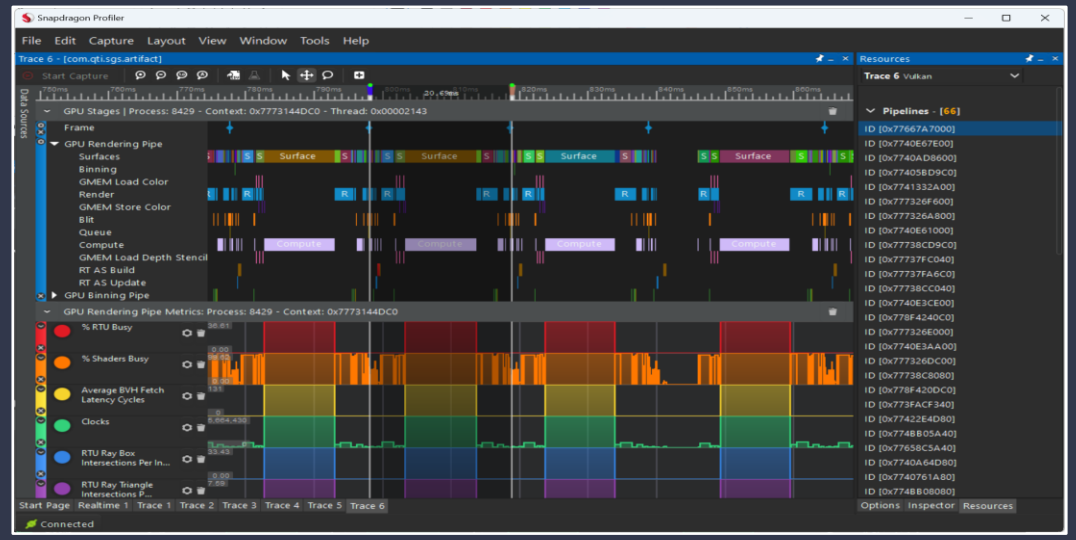
Snapdragon Profiler: Ray Tracing

Realtime

- Current metrics
  - % RTU Busy
  - Average BVH Fetch Latency Cycles
  - RTU Ray Box Intersections
  - RTU Ray Triangle Intersections
  - % BVH Fetch Stall

Trace

- Ray Tracing Related Stages
  - Acceleration Structure Operations:
    - Build / Copy / Update / Refit / etc.
  - Ray Gen / Dispatch stages
- Trace Stage Metrics
  - All available metrics (including Ray Tracing specific metrics shown on the left)

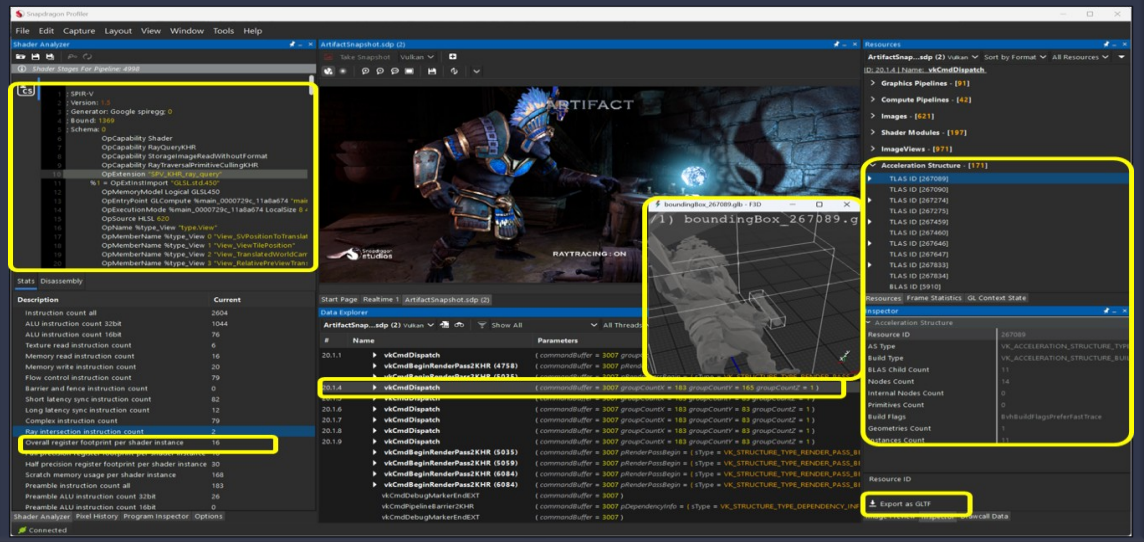


Access at www.qualcomm.com/developer/software/snapdragon-profiler

Snapdragon Profiler: Ray Tracing (cont'd.)

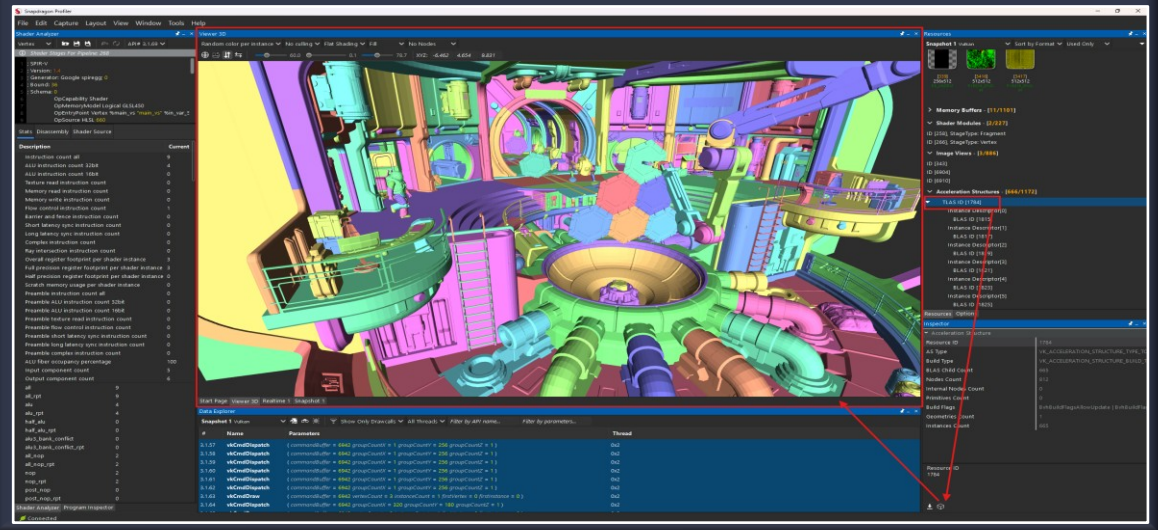
Snapshot

- Capture / Replay
- Data Explorer
  - vkCmdTraceRays / DispatchRays
- Resource View
  - TLAS / BLAS Node Details
  - AS Export to glTF
  - Launch external viewer
- Shader Analyzer
  - Shader Source (SpirV / HLSL / GLSL)
  - Shader Disassembly
  - Shader Stats
    - Ray Query Instruction count



Built-In Acceleration Structure Viewer

- Visualize Acceleration Structure
  - Colorize individual nodes
  - AABB hierarchy
  - Various shading and camera options
- Help identify areas / clusters / overlapping regions within an acceleration structure that may benefit from additional optimization efforts



Thank you

Follow us on:

For more information, visit us at qualcomm.com & qualcomm.com/blog