Ray Tracing: delivering immersive gaming experiences on mobile

Jose-Emilio Muñoz-Lopez
Staff Software Engineer, Arm

Agenda
- Ray tracing overview
- Demo
- Shadows
- Reflections and refractions
- Best practice
- Resources
Ray tracing overview
Ray tracing vs Rasterization

- Rasterization
  - Object by object
  - Triangles projected onto screen
  - Check pixel coverage
  - Use Z-Buffer for visibility

- Ray Tracing
  - Pixel by pixel
  - Cast a ray from camera to pixel
  - Check triangle intersection
  - Use closest-hit for visibility
  - More rays for more complex rendering

- Hybrid
  - Rasterize the scene, enhance with ray tracing
Hybrid ray tracing effects

- Hard shadows (≤1 ray per pixel), soft shadows (>1rpp)
- Mirror reflections (1rpp), glossy reflections (>1rpp)
- Smooth refractions (≥2rpp), glossy refractions (>2rpp)
- Ambient occlusion and global illumination (≥2rpp)

- Complexity
- Rays
- Performance
- Require denoising
Ray tracing demo

- Scene and similar samples available in Khronos’ Vulkan Samples
  - https://github.com/KhronosGroup/Vulkan-Samples
- Deferred PBR rendering with ray tracing as post-processing
- On Arm Immortalis, 40-60 FPS, 1600x720 with shadows, reflections and refractions

Ray tracing in Vulkan

Acceleration structures (VK_KHR_acceleration_structure)

- Optimised data structure
  - Minimises intersection tests
  - Quickly find what a ray has hit
  - User can control the topology
- Bottom Level (BLAS)
  - Contain index and vertex data
  - Hierarchical bounding volumes
- Top Level (TLAS)
  - BLAS grouped in instances with
    - Transform data (animations)
    - Custom ID (materials)
Ray tracing in Vulkan
Ray tracing pipeline (VK_KHR_ray_tracing_pipeline)
- Implementation managed traversal and shader dispatch
- New shader stages: ray generation, intersection/any-hit, closest-hit/miss

Ray tracing in Vulkan
Ray query (VK_KHR_ray_query)
- Use in any shader stage
- User code handles intersection logic
Ray traced shadows

- Trace ray towards the light, if it intersects with anything, surface is in shadow
- Simpler and more accurate than traditional shadow mapping techniques
- To improve performance, skip ray tracing if the surface faces away from light
Ray traced shadows

```c
rayQueryEXT rq;
rayQueryInitializeEXT(rq, accStruct,
  gl_RayFlagsTerminateOnFirstHitEXT |
  gl_RayFlagsOpaqueEXT,
cullMask,
origin, tMin, direction, tMax);

// Traverse the acceleration structure
rayQueryProceedEXT(rq);

// Check intersections (if any)
if (rayQueryGetIntersectionTypeEXT(rq, true)
  != gl_RayQueryCommittedIntersectionNoneEXT) {
  // In shadow
}
```

Reflections and refractions
Bindless resources
Descriptor indexing (VK_EXT_descriptor_indexing, core since 1.2)

- Need to know not only that the ray intersected an object, but which object
- All vertices, normals, UV-coordinates and textures need to be accessible
- Descriptor arrays for buffers and textures, indexed with look-up tables
- API provides intersection IDs for instance, geometry, primitive (and barycentric coordinates)

Ray traced reflections

- Simpler and less limited than traditional techniques like Screen-Space Reflections
- Can handle animations, occluded faces and objects outside frustrum
- To improve performance, simplify rendering for reflected surfaces
Ray traced refractions

- Calculate new direction for light transmitted through a material
- Accurately simulate real light refraction unlike rasterization techniques
- Requires careful handling of back-faces and blending of multiple layers

\[
\begin{align*}
\eta_1 \text{ air} & \quad \theta_1 \quad \text{in} \\
\eta_2 \text{ glass} & \quad \theta_2 \\
N & \quad N' \\
\text{out1} = \text{refract}(\text{in}, N, \eta_1/\eta_2) & \\
\text{out2} = \text{refract}(\text{out1}, -N', \eta_2/\eta_1) \\
\end{align*}
\]
Ray query best practice

- Profile early and often
- Use the Mali Offline Compiler
- Some usages of the ray query API force the compiler to fallback to a slow multi-context traversal
- Use a single call to rayQueryProceed for each rayQueryInitialize, and call them unconditionally

**Example 1 (slow)**
```c
rayQueryEXT ray_query;
rayQueryInitializeEXT(ray_query, ...);
if (cond)
    rayQueryProceed(ray_query);
rayQueryProceed(ray_query);
```

- Main shader
  - Work registers: 64 (100% used at 50% occupancy)
  - Uniform registers: 16 (25% used)
  - Ray traversal contexts: 16 objects
  - Stack spilling: 36 bytes
  - 16-bit arithmetic: 0%
  - Total instruction cycles: 9.39
  - Shortest path cycles: 0.29
  - Longest path cycles: N/A

**Shader properties**
- Has uniform computation: true
- Has side-effects: false
- Modifies coverage: false
- Uses late ZS test: false
- Uses late ZS update: false
- Reads color buffer: false
- Has slow ray traversal: true

**Example 2 (fast)**
```c
rayQueryEXT ray_query;
rayQueryInitializeEXT(ray_query, ...);
rayQueryProceed(ray_query);
```

- Main shader
  - Work registers: 64 (100% used at 50% occupancy)
  - Uniform registers: 16 (25% used)
  - Ray traversal contexts: 1 objects
  - Stack spilling: 36 bytes
  - 16-bit arithmetic: 0%
  - Total instruction cycles: 3.20
  - Shortest path cycles: 3.20
  - Longest path cycles: 3.20

**Shader properties**
- Has uniform computation: true
- Has side-effects: false
- Modifies coverage: false
- Uses late ZS test: false
- Uses late ZS update: false
- Reads color buffer: false
- Has slow ray traversal: false
Ray tracing best practice

For both Ray Query and Ray Tracing Pipeline:

- Minimize rays per pixel
- Maximize ray coherency
- Optimize traversal with gl_RayFlags

For Ray Tracing Pipeline:

- Only trace rays in the Ray Generation shaders (avoid recursion)
- Avoid using Ray Query in ray tracing pipeline shaders
- Keep a small payload size
- Limit use of Any Hit shaders
- Consider moving to Ray Query
Acceleration structures best practice

- Minimize the count of dynamic meshes
  - Try in-place updates as opposed to re-building
- Build options
  - PREFER_FAST_TRACE for static structures
  - PREFER_FAST_BUILD for animated structures
  - Consider CPU (host) build if application is GPU bound
- Minimize geometry overlap, decompose scene if needed
- Arm GPU best practices developer guide

Build the future on Arm

- Join our developer community
- Connect with like-minded developers
- Get fresh insights from Arm experts
- Sign up: arm.com/developerprogram

Find out more

- Forums and blogs: community.arm.com
- Documentation: developer.arm.com/graphics
Thank You
Danke
Gracias
Grazie
谢谢
ありがとうございます
Asante
Merci
감사합니다
धन्यवाद
Kìitos
شكرًا
شكر
תודה

The Arm trademarks featured in this presentation are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

www.arm.com/company/policies/trademarks