Vulkanised 2023

The 5th Vulkan Developer Conference
Munich, Germany / February 7–9

# Ray Tracing: delivering immersive gaming experiences on mobile


## Jose-Emilio Muñoz-Lopez

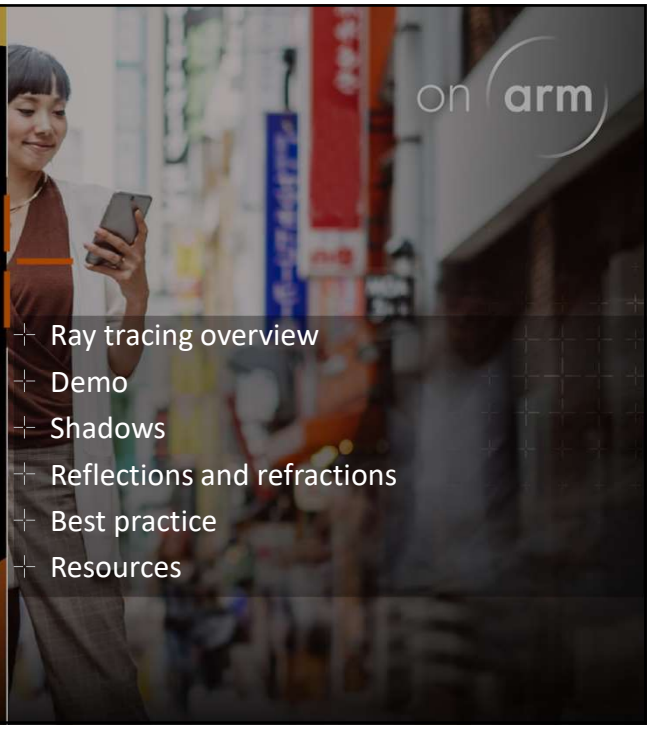
**Staff Software Engineer, Arm**


Platinum Sponsors: AMD arm Google LUNARG KHRONOS GROUP SAMSUNG

1

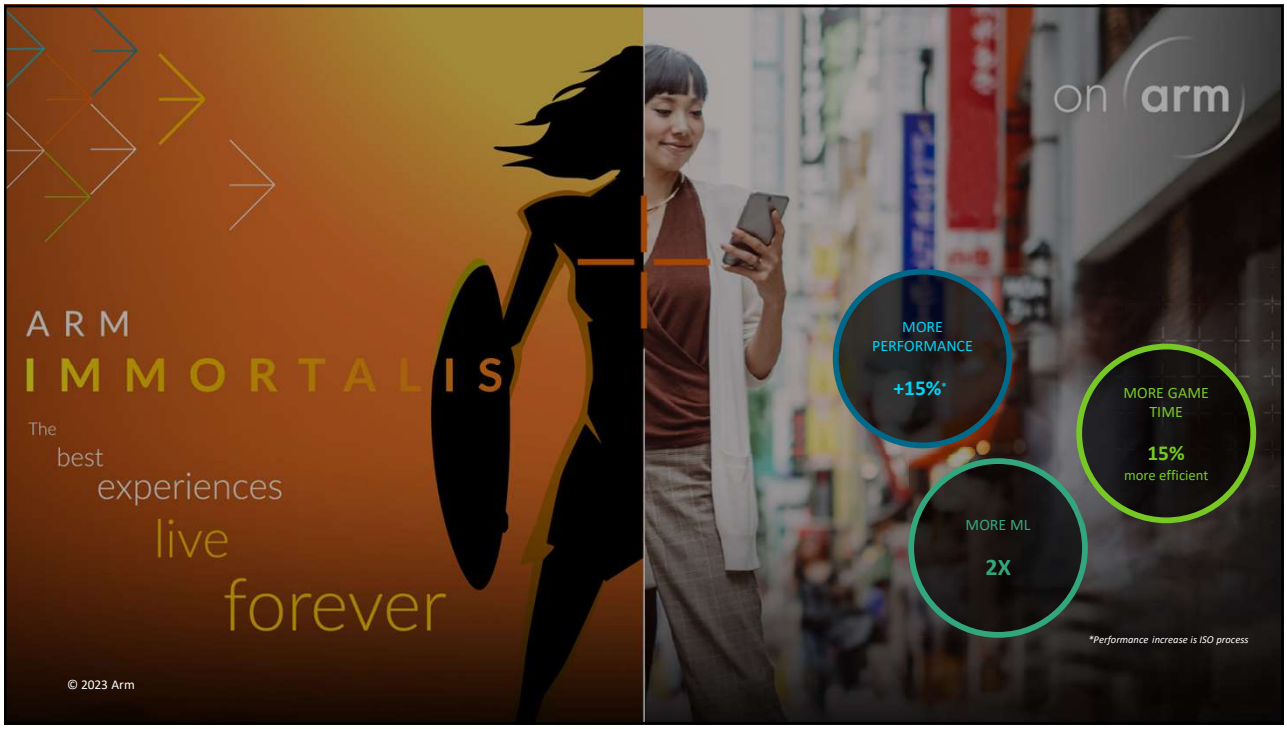
## Agenda




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

2




3

arm

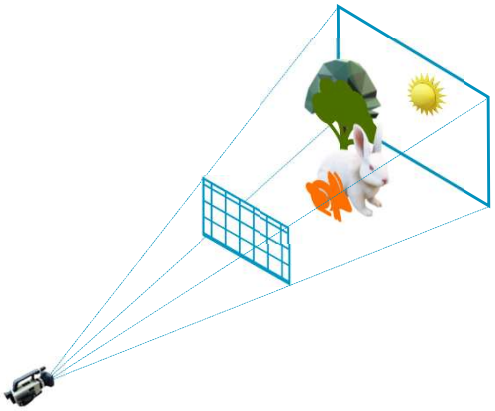
# Ray tracing overview



4

## Ray tracing vs Rasterization

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



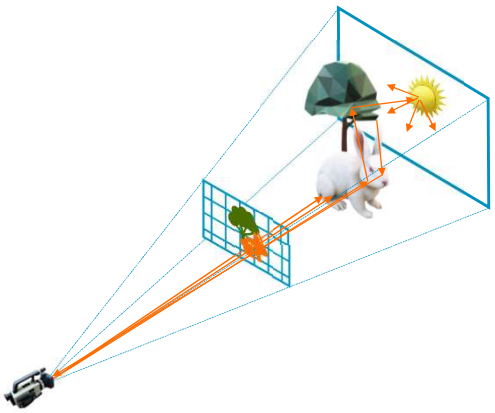
 

arm

5

## 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



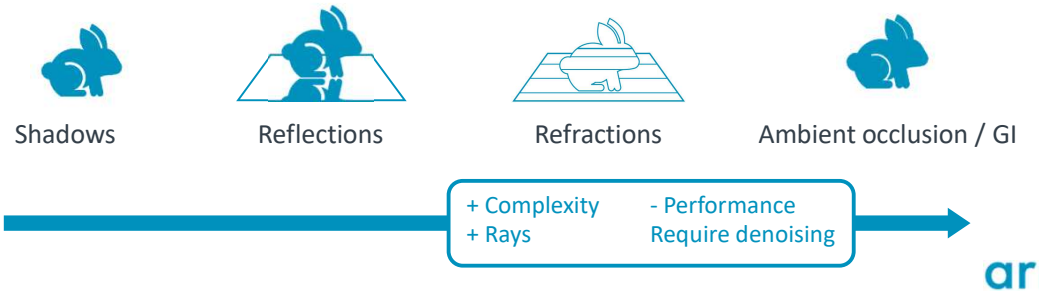
 

arm

6

## 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)






arm

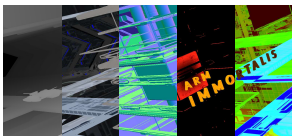
7


8

# Ray tracing demo

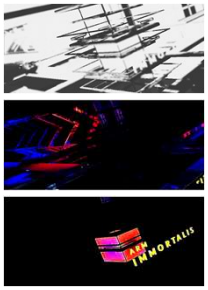
- Scene and similar samples available in Khronos' Vulkan Samples
  - https://github.com/KhronosGroup/Vulkan-Samples
- Deferred PBR rendering with ray tracing as post-processing
- On Arm Immortalis, 40-60 FPS, 1600x720 with shadows, reflections and refractions

GBuffer

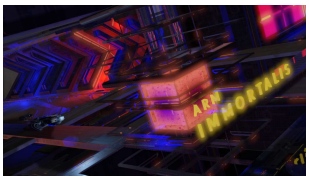


Lighting

RT Effects



Composition



arm

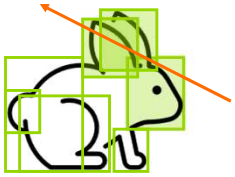
9

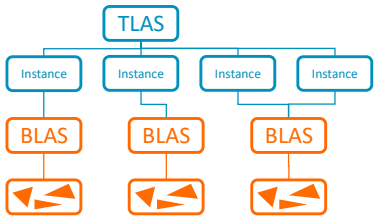
# Ray tracing in Vulkan

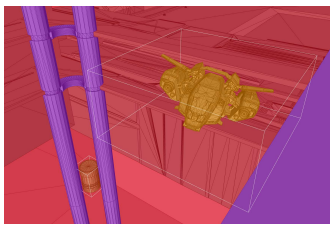
Acceleration structures (VK_KHR_acceleration_structure)

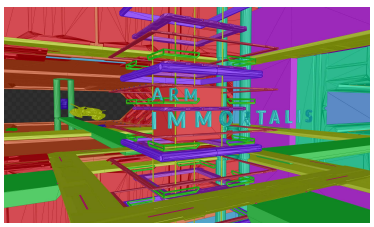
Vulkan

- 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)







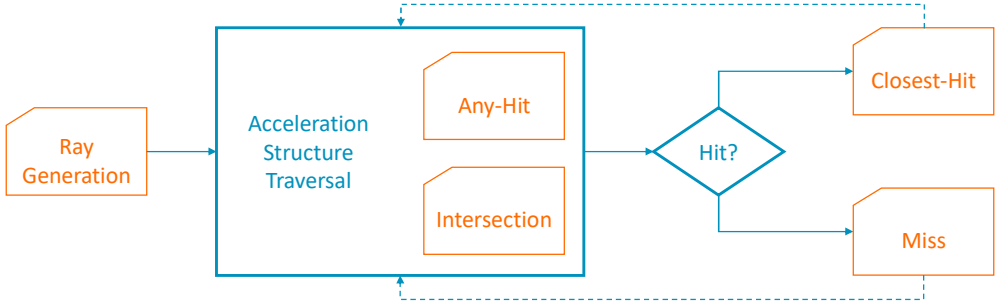
arm

10

# 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





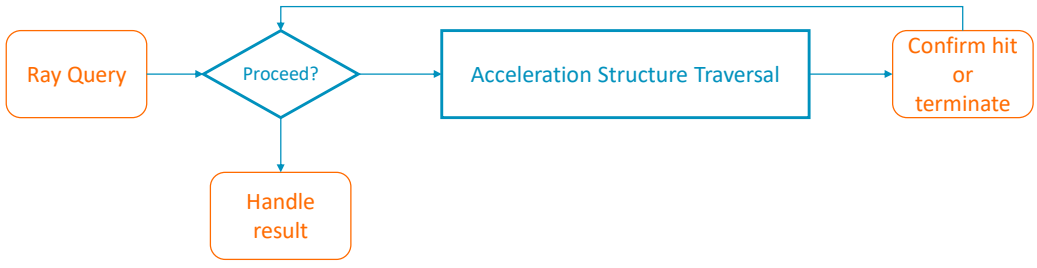


11

# Ray tracing in Vulkan

Ray query (VK_KHR_ray_query)

- Use in any shader stage
- User code handles intersection logic







12

arm

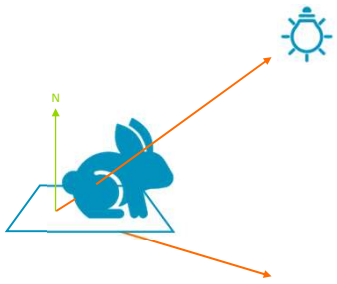
# Shadows

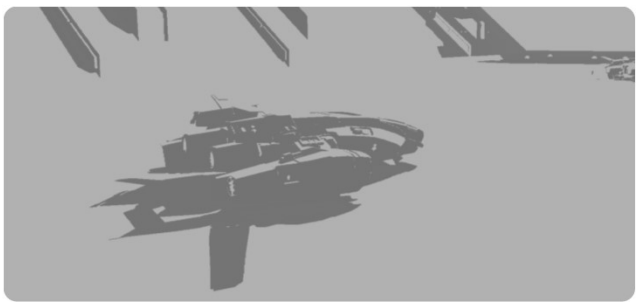


13

## 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



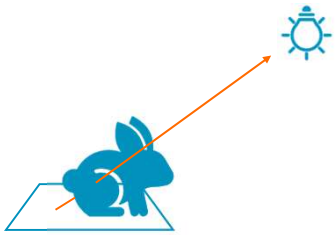



arm

14

## Ray traced shadows



```
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
}
```

arm

15

# Reflections and refractions

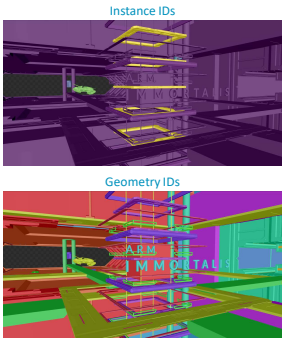


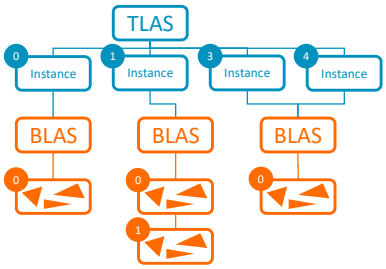
16

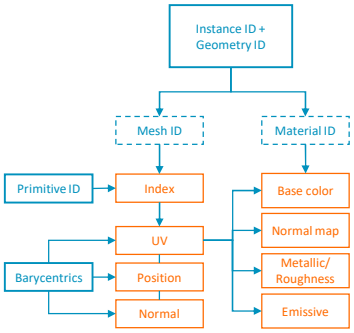
# Bindless resources

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)



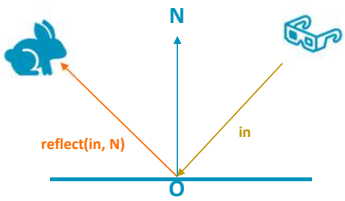

 

arm

17

# 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



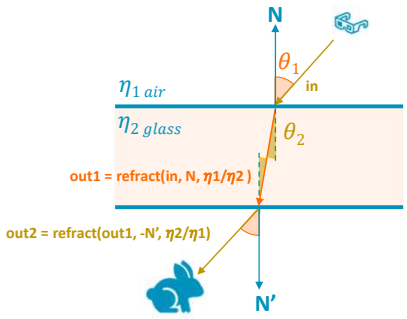

 

arm

18

## 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





arm

19

arm

# Ray tracing best practice



20

# 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)

```
rayQueryEXT ray_query;
rayQueryInitializeEXT(ray_query, ...);
if (cond)
    rayQueryProceed(ray_query);
rayQueryProceed(ray_query);
```

```
> .\malioc.exe --vulkan ray_query.frag


Main shader
===========

Work registers: 64 (100% used at 50% occupancy)
Uniform registers: 12 (18% used)
Ray traversal contexts: 16 objects
Stack spilling: 36 bytes
16-bit arithmetic: 0%

                                   A      LS      V      T    Bound
Total instruction cycles:       9.39  127.00    0.00   0.00      LS
Shortest path cycles:           0.29   13.00    0.00   0.00      LS
Longest path cycles:             N/A     N/A     N/A    N/A     N/A

A = Arithmetic, LS = Load/Store, V = Varying, T = Texture

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
```



21

# 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 2 (fast)

```
rayQueryEXT ray_query;
rayQueryInitializeEXT(ray_query, ...);
rayQueryProceed(ray_query);
```

```
> .\malioc.exe --vulkan ray_query.frag


Main shader
===========

Work registers: 64 (100% used at 50% occupancy)
Uniform registers: 16 (25% used)
Ray traversal contexts: 1 objects
Stack spilling: 80 bytes
16-bit arithmetic: 0%

                                   A      LS      V      T    Bound
Total instruction cycles:       3.20   26.80    0.00   0.00      LS
Shortest path cycles:           3.20   38.00    0.00   0.00      LS
Longest path cycles:            3.20   38.00    0.00   0.00      LS

A = Arithmetic, LS = Load/Store, V = Varying, T = Texture

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
```



22

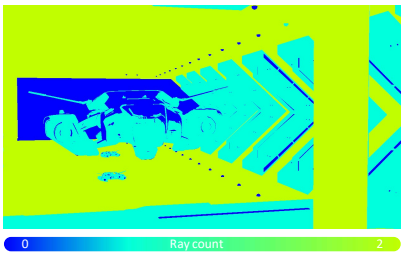


arm



# 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

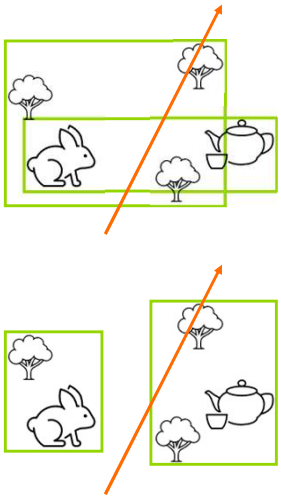





arm

## 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
  - https://developer.arm.com/documentation/101897/latest/






arm

25

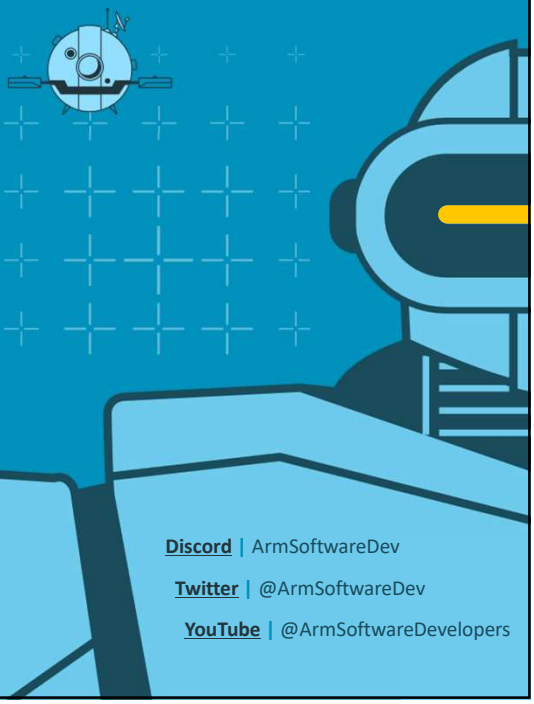
## arm Developer Program

## 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



**Discord** | ArmSoftwareDev

**Twitter** | @ArmSoftwareDev

**YouTube** | @ArmSoftwareDevelopers



26

arm

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה



27

arm



28