Realistic graphics with Ray Tracing on Mobile

Iago Calvo Lista, Arm
Agenda

- Ray tracing overview
- Advanced ray tracing
  - Transparencies
  - Skinned animation
- Hybrid ray tracing (SS+RQ)
  - Hybrid reflection
  - Hybrid shadows
- VRS
- Conclusion
Ray tracing on mobile is here!

- Last year's talk
  - Devices with HW ray tracing just appeared
  - 40-60 FPS
  - Introduction to shadows, reflections and refractions

- Today
  - More devices with HW ray tracing
  - 60-90 FPS
  - More data and advanced techniques to share

<table>
<thead>
<tr>
<th>Model</th>
<th>SoC</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vivo X90</td>
<td>D9200</td>
<td>Immortalis-G715</td>
</tr>
<tr>
<td>Vivo X90s</td>
<td>D9200+</td>
<td>Immortalis-G715</td>
</tr>
<tr>
<td>Oppo Find X6</td>
<td>D9200</td>
<td>Immortalis-G715</td>
</tr>
<tr>
<td>Vivo X100</td>
<td>D9300</td>
<td>Immortalis-G720</td>
</tr>
</tbody>
</table>
Ray tracing overview
What is Ray tracing

- API for ray intersection tests
  - User launches a ray
  - Test the ray against the scene geometry
  - Returns the ray closest hit

- Main use case: rendering
  - Other: physics simulations
Ray tracing in Vulkan

+ **VK_KHR_acceleration_structure**
  - Optimised data structure
  - Bottom Level AS (BLAS)
  - Top Level AS (TLAS)

+ **Two options**
  - **VK_KHR_ray_query**
    + Existing shaders
  - **VK_KHR_ray_tracing_pipeline**
    + Driver managed

+ **VK_EXT_descriptor_indexing**
  - Bindless resources
  - Access the material of intercepted hit
  - Core Vulkan, not part of ray tracing
Ray tracing demo

- Use of PBR rendering
  - Easy integration with ray tracing
- Hybrid rendering:
  - Rasterized Deferred GBuffer
  - Ray tracing effects as post-processing (1 render pass per effect)
- Shadows (hard vs soft)
- Reflections (glossy and mirror)
- Refractions
<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of objects</td>
<td>1102</td>
</tr>
<tr>
<td>Number of meshes</td>
<td>517</td>
</tr>
<tr>
<td>Number of triangles</td>
<td>1.5M</td>
</tr>
<tr>
<td>Number of triangles without instancing</td>
<td>1.65M</td>
</tr>
<tr>
<td>Number of vertices</td>
<td>762K</td>
</tr>
<tr>
<td>Number of vertices without instancing</td>
<td>1.07M</td>
</tr>
<tr>
<td>Number of materials</td>
<td>190</td>
</tr>
<tr>
<td>Number of Lights</td>
<td>16</td>
</tr>
<tr>
<td>Number of casting shadow lights</td>
<td>1 punctual</td>
</tr>
<tr>
<td>Resolution</td>
<td>1600x900</td>
</tr>
<tr>
<td>Reflections resolution</td>
<td>1600x900</td>
</tr>
<tr>
<td>Shadows resolution</td>
<td>800x600</td>
</tr>
</tbody>
</table>
Advanced Ray Tracing
Ray Query Shadows - Alpha test OFF
Ray Query Shadows - Alpha test ON
Alpha test

- Access to bindless materials
  - Extra memory bandwidth
  - Only use transparencies necessary
- RayQueryProceedEXT will return false once it has a confirmed hit
  - Non opaque materials need confirmation to be considered
  - The first hit candidate might not be the closest
- RT pipeline invokes Any-hit shader
- For opaque materials set:
  - GEOMETRY_OPAQUE_BIT_KHR (low priority)
  - GEOMETRY_INSTANCE_FORCE_OPAQUE_BIT_KHR
  - gl_RayFlagsOpaqueEXT (priority, all materials)

```c
rayQueryInitializeEXT(…);
#if defined(USE_RQ_ALPHA_TEST)
void main() {
  // Retrieve material ID, barycentrics and sample bindless alpha texture
  const bool opaqueHit = bindless_alpha_test(rayQuery);
  if (!opaqueHit) {
    ignoreIntersectionEXT;
  }
}
#else
rayQueryProceedEXT(rayQuery);
#endif
```
Skinned animations

- Update the BLAS
  - Expensive
  - Minimize number of skinned meshes

- Distribute update across multiple frames

- `VkBuildAccelerationStructureFlagBitsKHR`
  - `FAST_TRACE` Only static geometry
  - `FAST_BUILD` Dynamic geometry

- `vkCmdBuildAccelerationStructuresKHR`
  - Build a new AS is slower
  - Try update/refit an existing one
Hybrid Ray tracing
Reflections

Try to minimize number of rays

- Retrieve GBuffer info
- Roughness for number of rays
- Generate ray info
- PBR data
- Trace ray
- RQ/RT Pipeline API
- Hit
- Fail
- Bindless material
- Environment map/ skybox
- Multiple Bounces
Screen Space Reflections

Implementation
- We use a stochastic/physically based approach
- Transform ray origin and direction to screen space
- Use the depth buffer to check each pixel
- Advance ray and check each pixel
- Compare ray position and depth buffer
  - Values are close – Hit
  - Depth is lower – Continue
  - Depth is higher – Not enough information (fail)

![Diagram of Screen Space Reflections](image)
Screen Space Reflections

- Can be integrated easily in PBR
  - Fallback for RT reflections

- Compared to RT reflections
  - Better performance
  - Lower quality - Only objects on screen

- More difficult (more magic numbers)

- Consider Hierarchical SSR
  - Mipmap with maximum value of depth-buffer
  - Allows to skip a lot of empty space
Hybrid reflections (SSR+RQ)

+ We can retrieve SSR hit material from G-buffer
  • Avoid bindless
+ Different view direction
  • Don’t reuse illuminated result for SSR
+ Terminate (as fail) if occluded
+ Mali offline compiler
  • Ensure data is reused and not copied
  • Easy to create a lot of spilling

Retrace:
- Retrieve GBuffer info
- Roughness for number of rays
- Generate ray info
- Trace SSR ray
  - Fail
  - Hit
- Trace RQ
- Retrieve material bindless
- Multiple Bounces
- Retrieve material GBuffer
Hybrid reflections (SSR+RQ) improvements

- **Self intersection**
  - Common problem when using G-buffer to launch rays

- **Solutions**
  - `gl_RayFlagsCullFrontFacingTrianglesEXT`
    - Extra GPU work
  - Modify ray - Increase $t_{\text{min}}$ or add bias to ray origin
  - Use a Screen space ray

- **Hybrid reflections always launch a SSR ray**
  - We know that we cannot intersect anything until the ray fails
  - Relaunch a ray query ray from the last texel visited by SSR
  - Transform from screen space to world space

- **Useful in model with transparencies or high geometry**
  - Vegetation, hair, etc
  - Allows to skip the close objects
Hybrid reflections (SSR+RQ)

- Glossy reflections
  - Roughness to control direction
  - Warp divergence in ray directions
- One single ray doing can slow down the entire warp
  - Less relevant in mirror reflections
- We can use subgroup operation to control occupancy
  - WaveActiveCount(true)
  - subgroupBallotBitCount(subgroupBallot(true))
- Check during SSR iterations to see occupancy
- Check occupancy before launching a RQ ray

### Flowchart

<table>
<thead>
<tr>
<th>Retrieve GBuffer info</th>
<th>Generate ray info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roughness for number of rays</td>
<td>Trace SSR ray – with early termination</td>
</tr>
<tr>
<td>Fail - enough occupancy</td>
<td>Hit</td>
</tr>
</tbody>
</table>

### Adjust ray info origin

- Retrieve material GBuffer
  - Multiple Bounces

- Trace RQ
- Retrieve material bindless
No reflections
Screen Space Reflections (SSR)
RQ reflections
Hybrid (RQ+SSR) reflections
Hybrid (RQ+SSR) reflections

SSR hit
RQ hit
Miss (skybox)
Hybrid Reflections

- SSR: 10.1ms (1.17 GPU Active)
- RQ: 10.1ms (1.36 GPU Active)
- SSR+RQ: 10.1ms (1.37 GPU Active)
- SSR+RQ (Adjusting Origin): 10.1ms (1.27 GPU Active)
- SSR+RQ (Subgroup): 10.1ms (1.23 GPU Active)
Avoid unnecessary rays if the light is behind

We do not need the closest hit - `gl_RayFlagsTerminateOnFirstHitEXT`

If $(\dot{N, \text{light\_direction}}) \leq 0$ {
  visibility *= MAX_SHADOW_STRENGTH;
  continue;
}

Trace ray
RQ/RTPipeline API

Hit
Fail
In shadow
Illuminated

Retrieves GBuffer info
Generate ray info
PBR data

Behind the light
In shadow – no ray
Hybrid shadows

+ Hybrid reflections are faster to traverse
+ Shadows follow a similar algorithm
+ Can the same idea be applied to shadows
Facing away from the light
Screen Space Shadows (SSS)
Hybrid (SSS+RQ) Shadows
SSR: 10.1ms (1.24 GPU Active)

RQ: 10.1ms (1.49 GPU Active)

SSR+RQ: 11.6ms (1.69 GPU Active)

SSR+RQ (Adjusting Origin): 11.5ms (1.69 GPU Active)

SSR+RQ (Subgroup): 10.1ms (1.48 GPU Active)
Variable Rate Shading (VRS)

- **VK_KHR_fragment_shading_rate** extension

- Render some elements at lower resolution
  - Concentrate Fragment density were is noticeable

- Multiple modes
  - Per draw call
  - Per primitive
  - Shading rate attachment (most flexible)

- Option to combine multiple methods
VRS + RT

- We can combine VRS and RT
- Screen-space shading attachment
  - Generated by a compute pass
- Choose a heuristic to lower rate in less noticeable areas
- IE: Luminance to have lower rate in darker areas
  - Detail is less noticeable in darker areas
  - Useful for reflections
- IE: Concentrate in edge of shadows
  - Use previous frame
  - Light shadowed areas should remain relatively constant
  - Concentrate shading rate in borders
  - Big enough margins avoids flickering
Ray Query shadows

- Half Res: 10.1ms (1.47 GPU Active)
- VRS: 12.2ms (1.78 GPU Active)
- Full Res: 17.3ms (2.47 GPU Active)
Conclusion

+ Ray tracing can provide a huge increase of visual quality to games
+ Ray tracing makes it easier to implement complex realistic effects
+ Devices in the market are capable to run ray tracing content
+ Vulkan uses a flexible API for RT
  - Lots of space for optimization for your application
  - Same API for mobile and desktop – easy to port your content
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

- **Talk to us via Discord** ArmSoftwareDev
- **Docs & Resources:** arm.com/vulkan
- **Forums and blogs:** community.arm.com
Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
ありがとう
Asante
Merci
감사합니다
धन्यवाद
شكرًا
ধন্যবাদ
תודה