Vulkan Multipass mobile deferred done right

ARM

Hans-Kristian Arntzen
Marius Bjørge

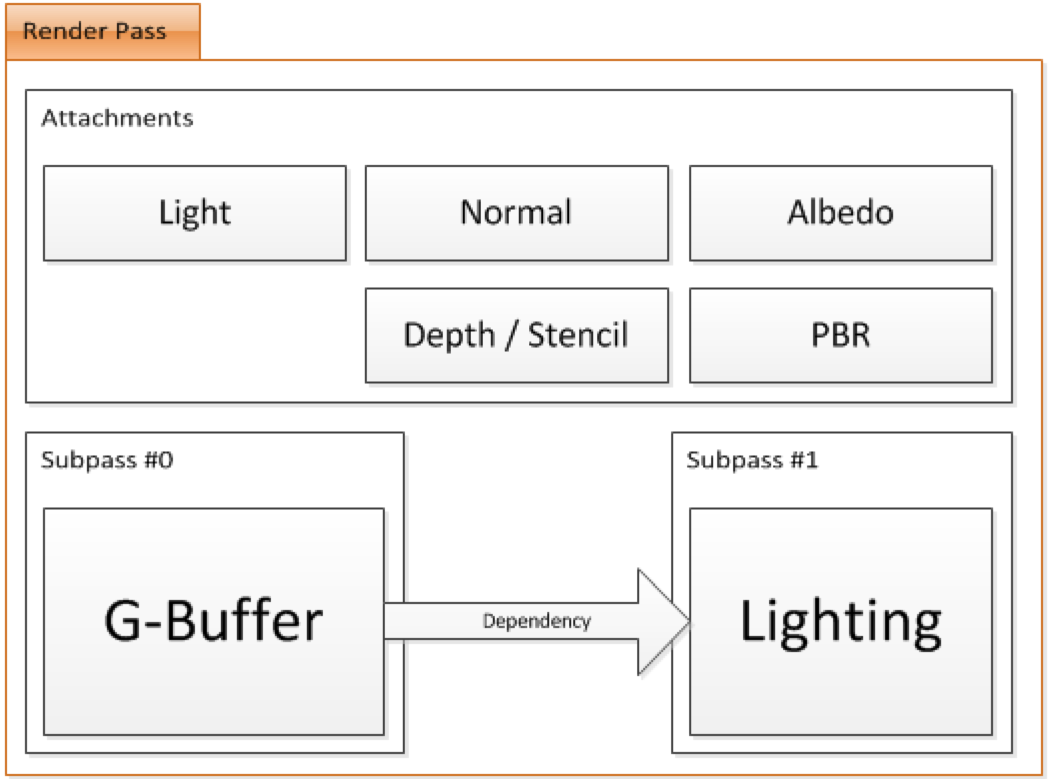
Khronos
5 / 25 / 2017

Content

- What is multipass?
- What multipass allows ...
  - A driver to do versus MRT
  - Developers to do
  - Transient images and lazy memory
- Case studies
  - Baseline app
  - Sponza
  - «Lofoten» demo

What is Multipass?

- Renderpasses can have multiple subpasses
- Subpasses can have dependencies between each other
  - Render pass graphs
- Subpasses refer to subset of attachments

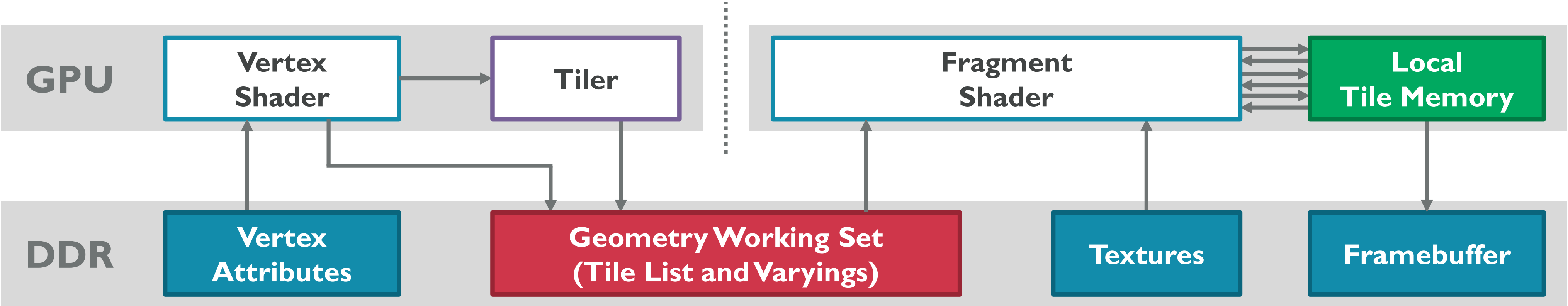


Improved MRT deferred shading in Vulkan

- Classic deferred has two render passes
  - G-Buffer pass, render to ~4 textures
  - Lighting pass, read from G-Buffer, accumulate light
- Lighting pass only reads G-Buffer at gl_FragCoord
- Rethinking this in terms of Vulkan multipass
  - Two subpasses
  - Dependencies
    - COLOR | DEPTH -> INPUT_ATTACHMENT | COLOR | DEPTH_READ
    - **VK_DEPENDENCY_BY_REGION_BIT**

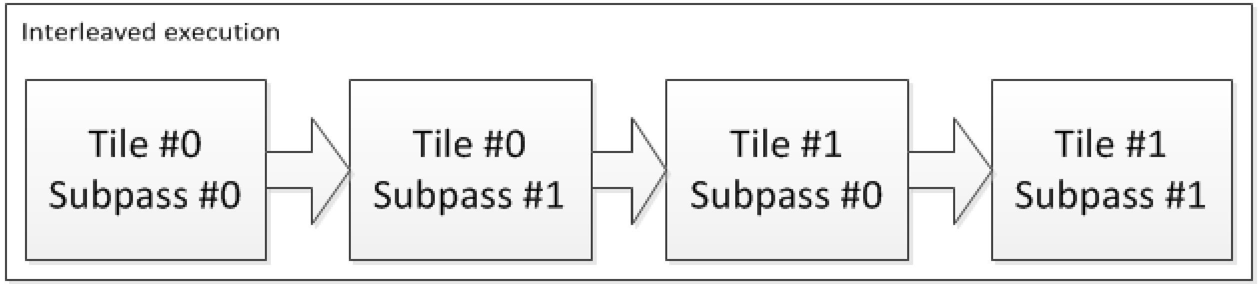
Tile-based GPUs 101

- Tile-based GPUs batch up and bin all primitives in a render pass to tiles
- In fragment processing later, render one tile at a time
  - Hardware knows all primitives which cover a tile
  - Advantage, framebuffer is now in fast and small SRAM!
- Having framebuffer in on-chip SRAM has practical benefits
  - **Read/write to it is cheap, no external bandwidth cost**
- Main memory is written to when tile is complete



Tile-based GPU subpass fusing

- Subpass information is known ahead of time
  - VkRenderPass
- Driver can find two or more sub-passes which have …
  - BY_REGION dependencies
  - no external side effects which might prevent fusing
- Fuse G-Buffer and Lighting passes
  - Combine draw calls from G-Buffer and Lighting into one "render pass"
  - G-Buffer content can remain in on-chip SRAM
  - Reading G-Buffer data in lighting pass just needs to read tile buffer
  - vkCmdNextSubpass essentially becomes a noop



Vulkan GLSL subpassLoad()

- Reading from input attachments in Vulkan is special
  - Special image type in SPIR-V
- On vkCreateGraphicsPipelines we know
  - renderPass
  - subpassIndex
- subpassLoad() either becomes
  - texelFetch()-like if subpasses were **not** fused
    - This is why we need VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT
  - magicReadFromTilebuffer() if subpasses were fused
- Compiler knows ahead of time
  - No last-minute shader patching required

Transient attachments

- After the lighting pass, G-Buffer data is not needed anymore
  - G-Buffer data only needs to live on the on-chip SRAM
  - Clear on render pass begin, no need to read from main memory
  - storeOp is DONT_CARE, so never actually written out to main memory
- Vulkan exposes lazily allocated memory
  - imageUsage = VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT
  - memoryProperty = VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT
  - On tilers, no need to back these images with physical memory ☺

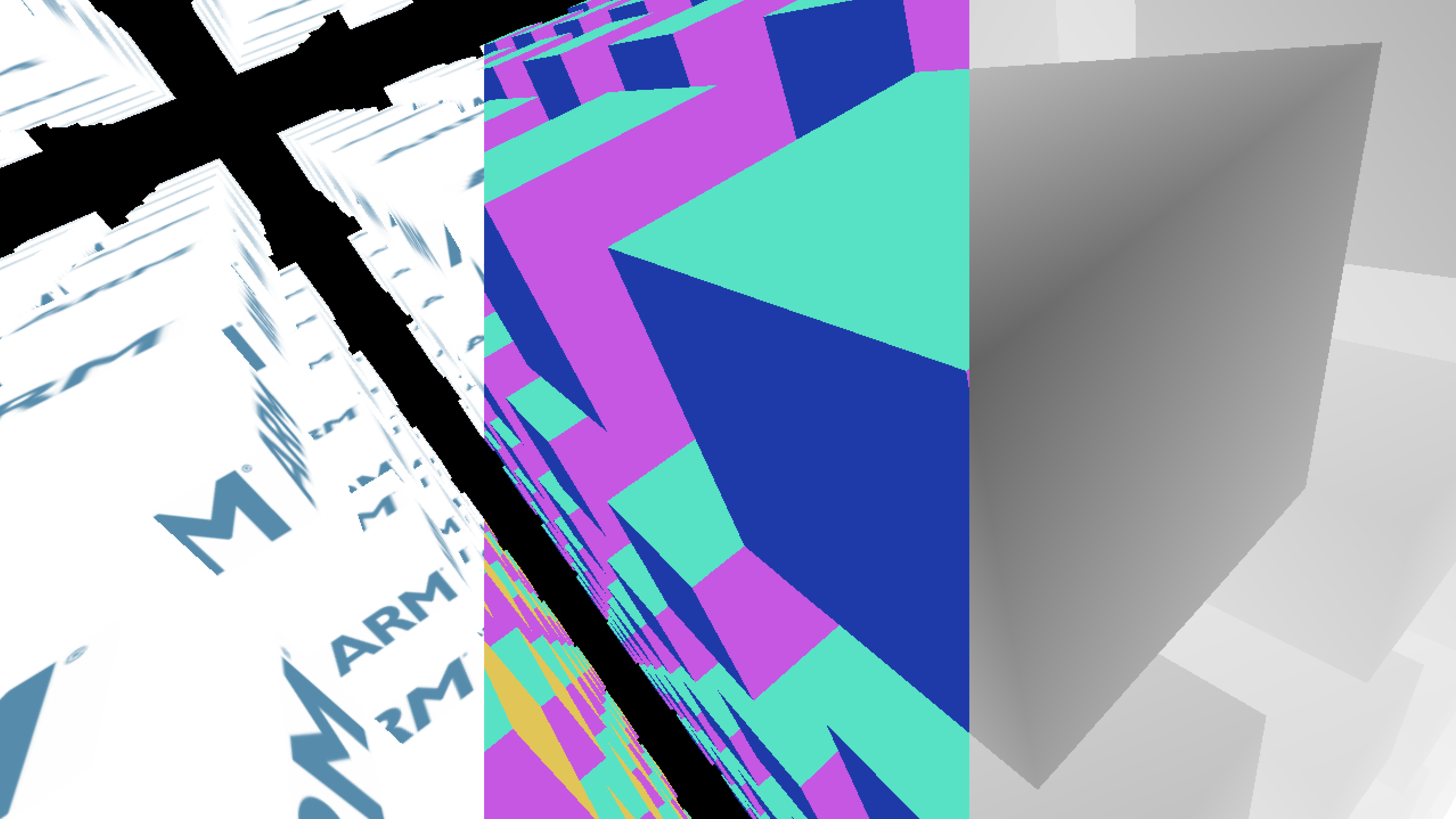
Multipass benefits everyone

- Deferred paths essentially same for mobile and desktop
  - Same Vulkan code (*)
  - Same shader code (*)
- VkRenderPass contains all information it needs
  - Desktop can enjoy more informed scheduling decisions
  - Latest desktop GPU iterations seem to be moving towards tile-based
  - At worst, it's just classic MRT
- (*) Minor tweaking to G-Buffer layout may apply


ARM

Baseline test

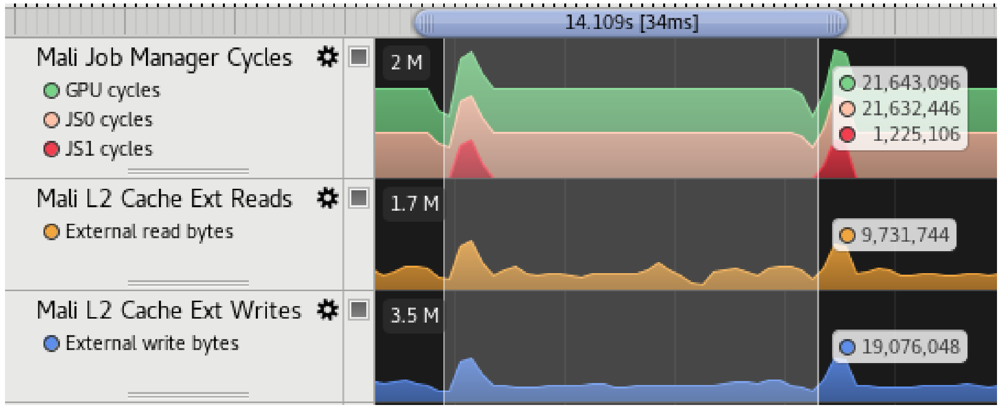
- Basic multipass sample
- One renderpass
- Light on geometry
- ~8 large lights
  - Simple shading
- Benchmark
  - Multipass (subpass fusing)
  - MRT
  - Overall performance
  - Bandwidth


ARM

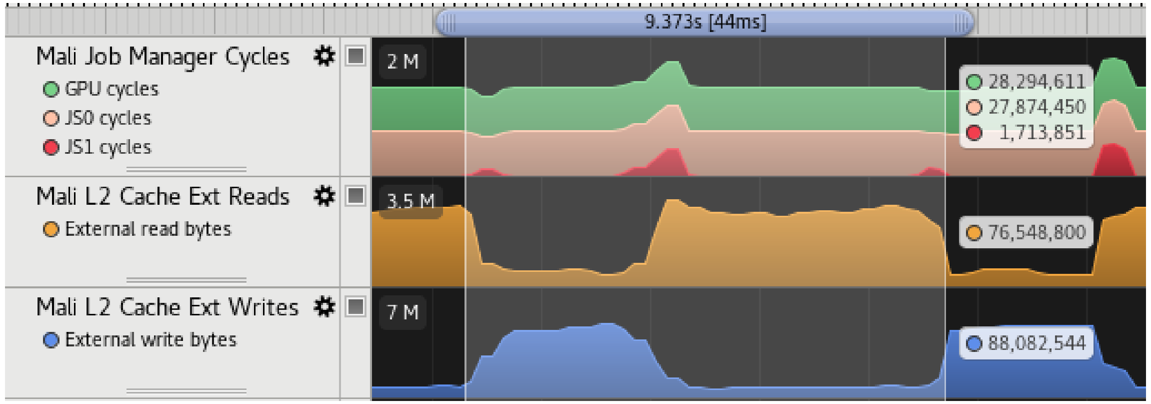
Baseline test data

- Measured on Galaxy S7 (Exynos)
- 4096x2048 resolution
  - Hit V-Sync at native 1440p
- ~30% FPS improvement
- ~80% bandwidth reduction
  - Only using albedo and normals
  - Saving bandwidth is vital for mobile GPUs

Multipass

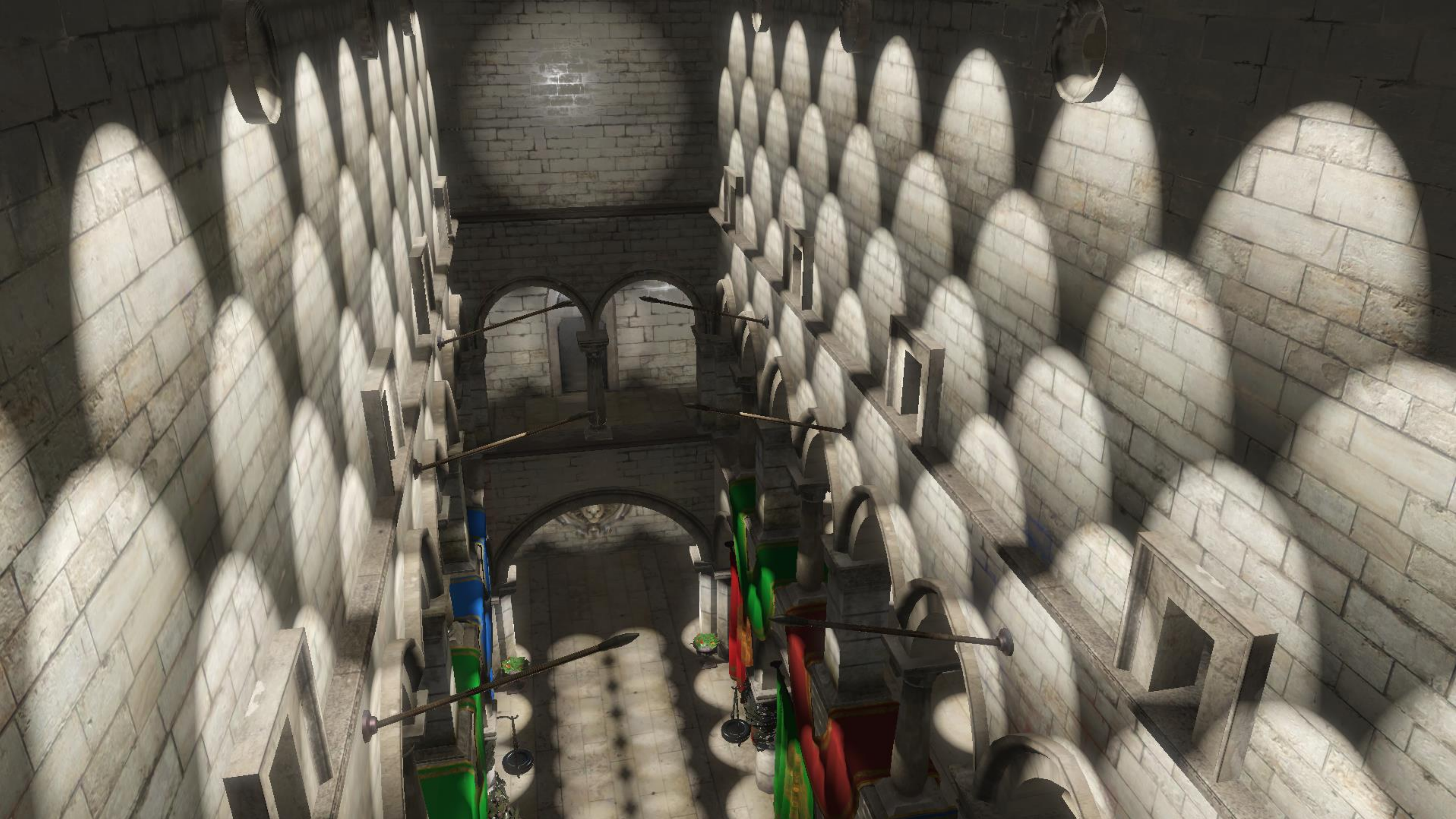


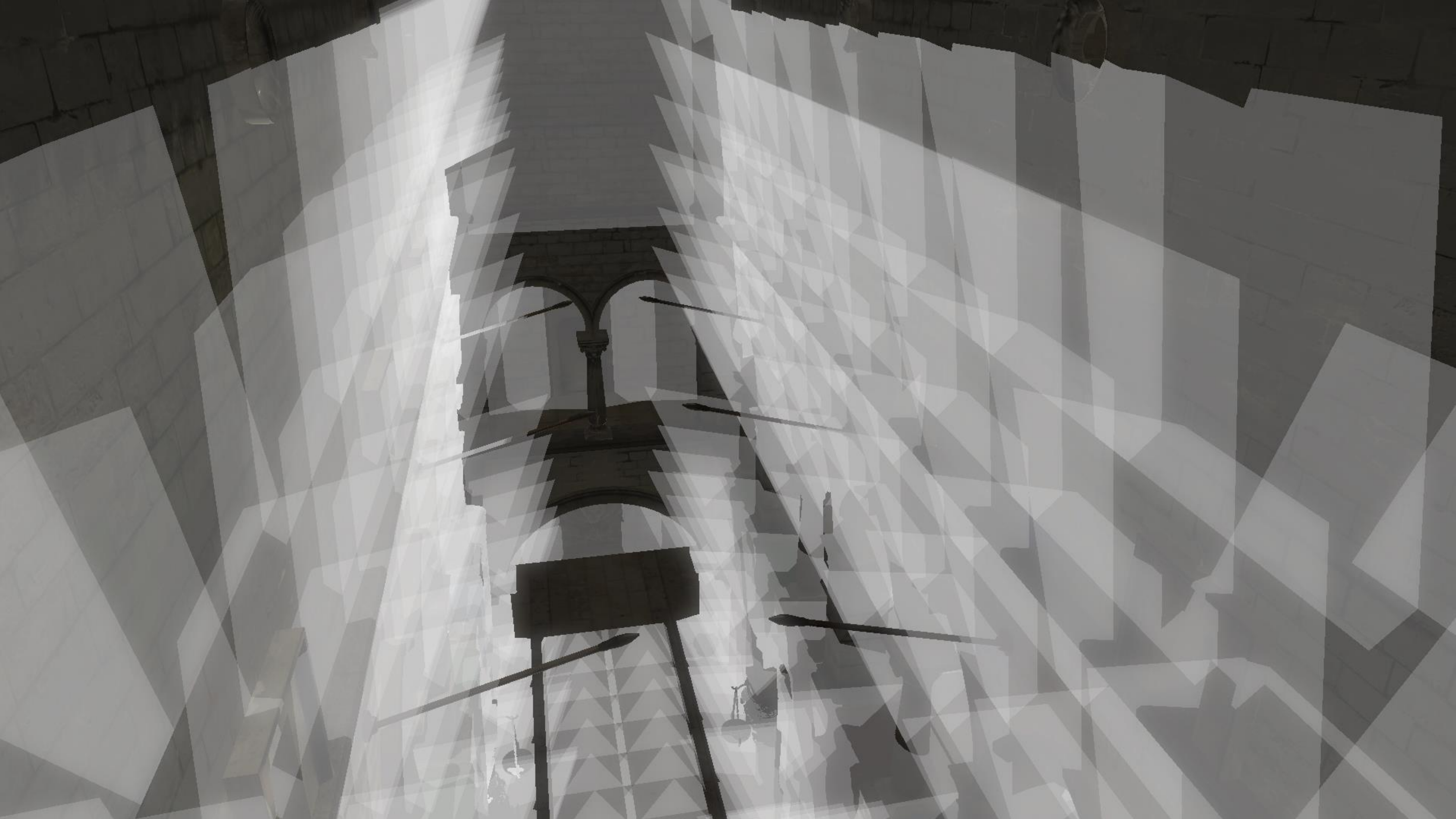
MRT



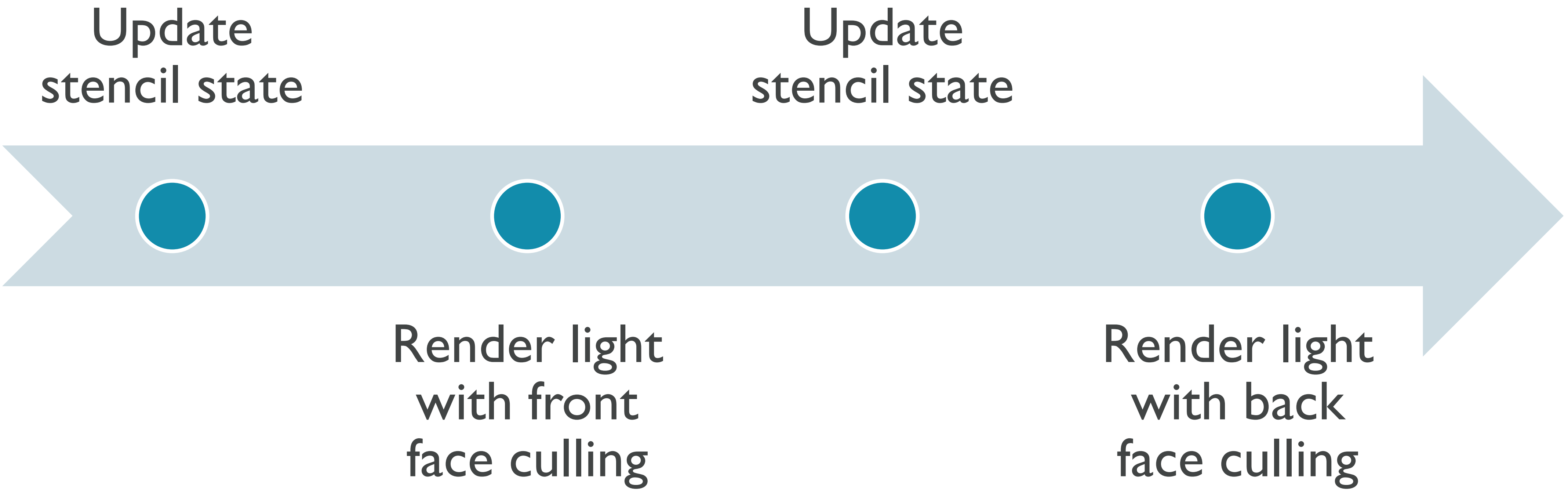
Sponza test

- Mid-high complexity geometry for mobile
  - ~200K tris drawn / frame
- ~610 spot lights
  - Full PBR
  - Shadowmaps on some of the lights
- 8 reflection probes
- Directional light w/ shadows
- Full HDR pipeline
  - Adaptive tonemapping
  - Bloom





Stencil culling



ARM

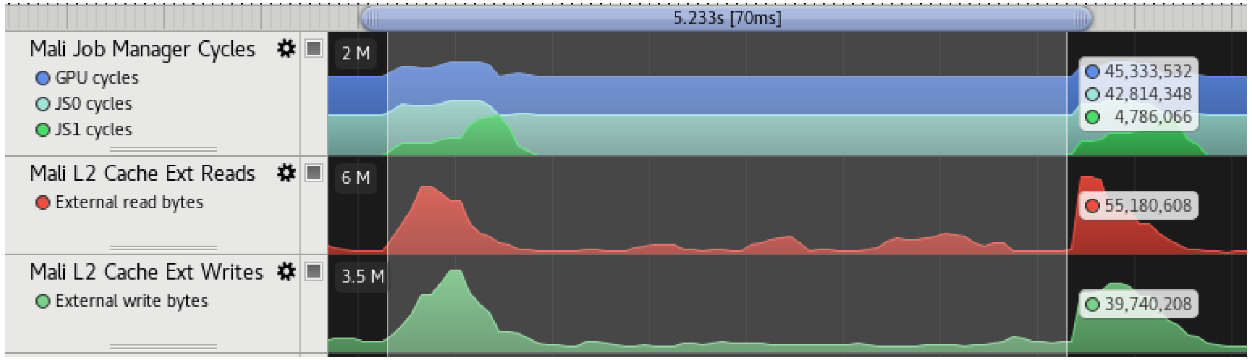
Clustered stencil culling

- Classic method of per light stencil culling involves a lot of state toggling
  - Can be expensive even on Vulkan
  - Usually performs poorly on GPU
- Instead, cluster local lights along camera Z axis
  - Each light is assigned to a cluster using conservative depth
  - Stencil is written for all local lights in a single pass
  - Peel depth for an extra 2-bit "depth buffer" in stencil
  - Lights are finally rendered using stencil with double sided depth test

Sponza test data

- Far, far heavier than sensible mobile content
- 1440p (native)
  - Overkill
- ~50-60% bandwidth reduction
- ~18% FPS increase

Multipass



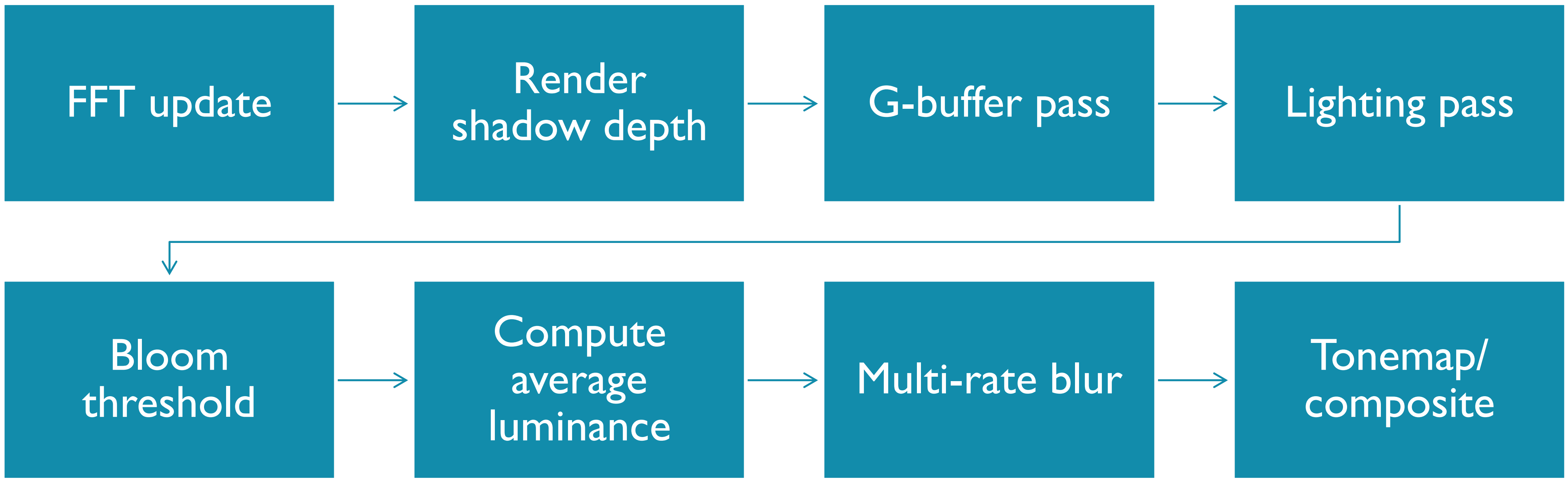
MRT


THEFT OR
VANDALISM
CRIME
STOPPERS
WARNING
t Side
STOP
STOP
PAY PHONE

«Lofoten»

- City scene with ~2.5 million primitives
  - Relies heavily on CPU based culling techniques to reduce geometry load
- 100 spot lights and point lights w/shadow maps
- Reflection probes
- Sun light with cascaded shadow maps
- Atmospheric scattering
- Ocean simulation
  - FFT running GPU compute
  - Separate refraction pass
  - Screen space reflections
- Bloom post-processing w/adaptive luminance tone mapping

Not your typical mobile graphics pipeline



ARM

Implementation details

- G-buffer pass
  - On tile-based GPUs, **fill rate != bandwidth**
  - Emissive/forward materials written directly to light buffer
  - Stencil set to mark reflection influence
  - Depth peeling for clustered stencil culling
- Lighting pass
  - Lighting accumulated using additive blending to lighting attachment
  - Clustered stencil culling used for local lights
  - Transparent objects
  - Fog applied after shading is complete
- Only commits light buffer to memory

ARM

Render passes

- Early decision to also make the high level interface explicit in terms of defining render passes
  - Possible to back-port to OpenGL etc
- BeginRenderPass()
  NextSubPass()
  EndRenderPass()

Integration for transient and lazy images

- Add support for “virtual” attachments
  - Keep a pool of images allocated with TRANSIENT_ATTACHMENT_BIT
  - Actual image handles not visible to the API user

```
multipass.virtualAttachments = { RGBA8, RGBA8, RGB10A2 };
mutipass.numSubpasses = 2;
multipass.subpass[0].colorTargets = { RT0, VIRTUAL0, VIRTUAL1, VIRTUAL2 };
multipass.subpass[1].colorTargets = { RT0 };
multipass.subpass[1].inputs = { VIRTUAL0, VIRTUAL1, VIRTUAL2, DEPTH };
renderpass.multipass = &multipass;

BeginRenderPass(&renderpass);
```

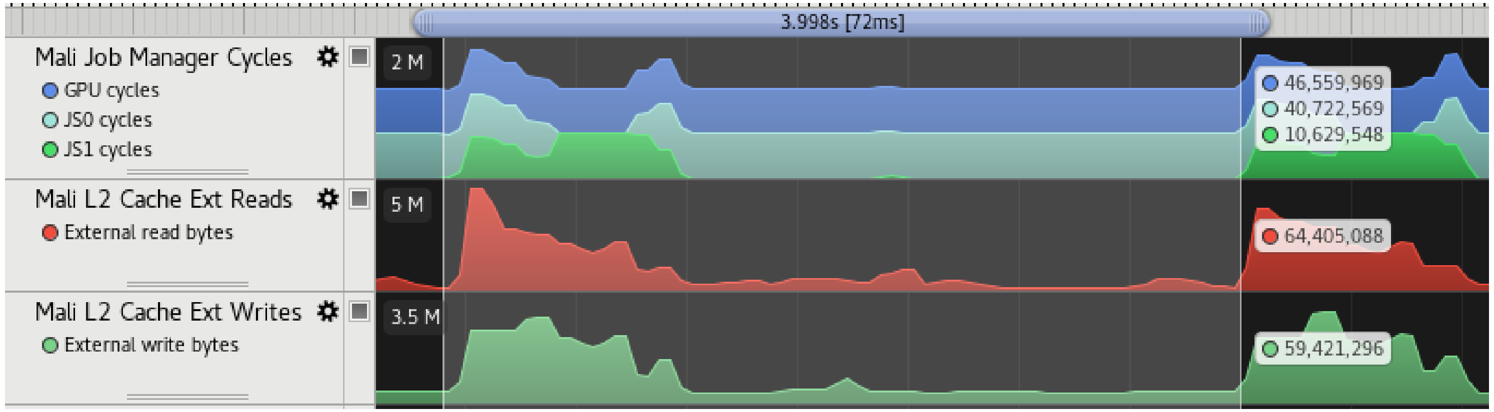
Multipass – virtual attachments

```
// Render G-buffer
pCB->BeginRenderPass(pCommandBuffer,
    KINSO_RENDERPASS_LOAD_CLEAR_ALL | KINSO_RENDERPASS_STORE_COLOR,
    Vec4(0.0f), 1.0f, 0, &m_Multipass);

DrawGeometry();
// Increment to the next subpass.
pCB->NextSubpass();

// Render lights additively.
DrawLightGeometry();
// Finally apply environment effects
ApplyEnv();

pCB->EndRenderPass();
```

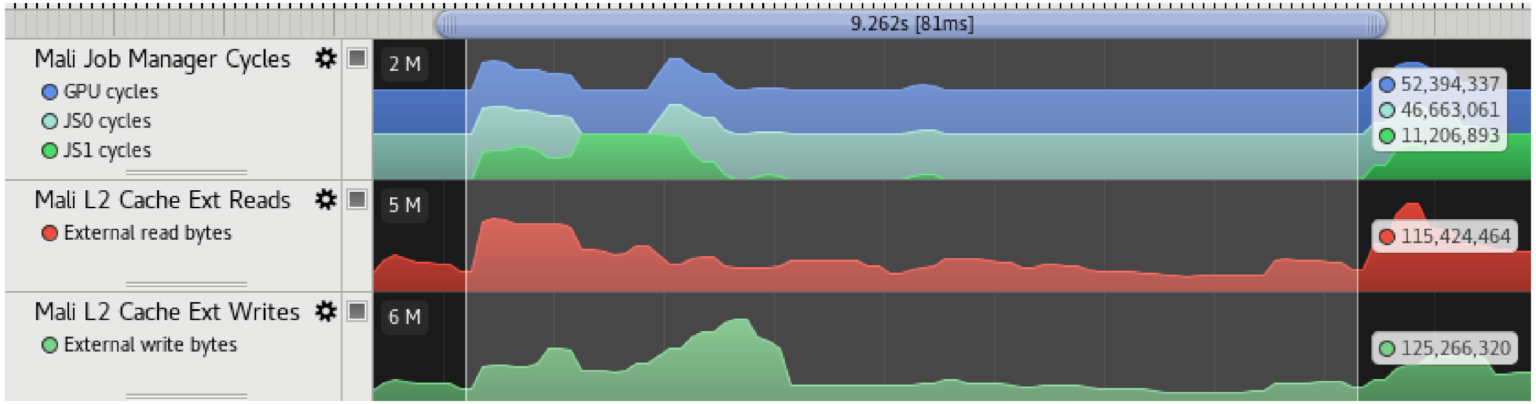
«Lofoten» test data

- Even heavier than Sponza
- 1440p (native)
  - Overkill
- ~50% bandwidth save
- ~12% FPS increase
- ~25% GPU energy save!

Multipass



MRT




ARM

Performance considerations for tilers

- With on-chip SRAM, per-pixel buffer size is limited
  - G-Buffer size is therefore limited
- On current Mali hardware, 128 bits color targets per pixel
  - May vary between GPUs and vendors
- Smaller tiles may allow for larger G-Buffer
  - At a quite large performance penalty
  - Fewer threads active, worse occupancy on shader core
  - Need to scan through tile list more

ARM

Engine integration for multiple APIs

- Render pass concept in engine is a must
  - Cannot express multiple subpasses otherwise
- subpassLoad() is unique to Vulkan GLSL
  - Solution #1, Vulkan GLSL is main shading language
    - SPIRV-Cross can remap subpassLoad() to MRT-style texelFetch
  - Solution #2, HLSL or similar
    - Make your own "intrinsic" which emits subpassLoad in Vulkan
- Unroll multipass to multiple passes in other APIs
  - Change render targets on NextSubpass()
  - Bind input attachments for subpass to texture units
  - Statically remap input_attachment_index -> texture unit in shader

ARM

Handling image layouts

- G-Buffer images are by design only used temporarily
- No need to track layouts
  - Application should not have direct access to these images!
- Can use external subpass dependencies for transition

```
VkSubpassDependency dep = {};
dep.srcSubpass = VK_SUBPASS_EXTERNAL;
dep.dstSubpass = 0;

attachment.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
reference.layout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;

dep.srcAccessMask = COLOR_ATTACHMENT_WRITE;
dep.dstAccessMask = COLOR_ATTACHMENT_READ | WRITE;
dep.srcStageMask = COLOR_ATTACHMENT_OUTPUT;
dep.dstStageMask = COLOR_ATTACHMENT_OUTPUT;
```


ARM

ARM

Questions?