

Vulkan Multipass Mobile Deferred Done Right

ARM

Hans-Kristian Arntzen
Senior Engineer

GDC 2017
02 / 28 / 2017

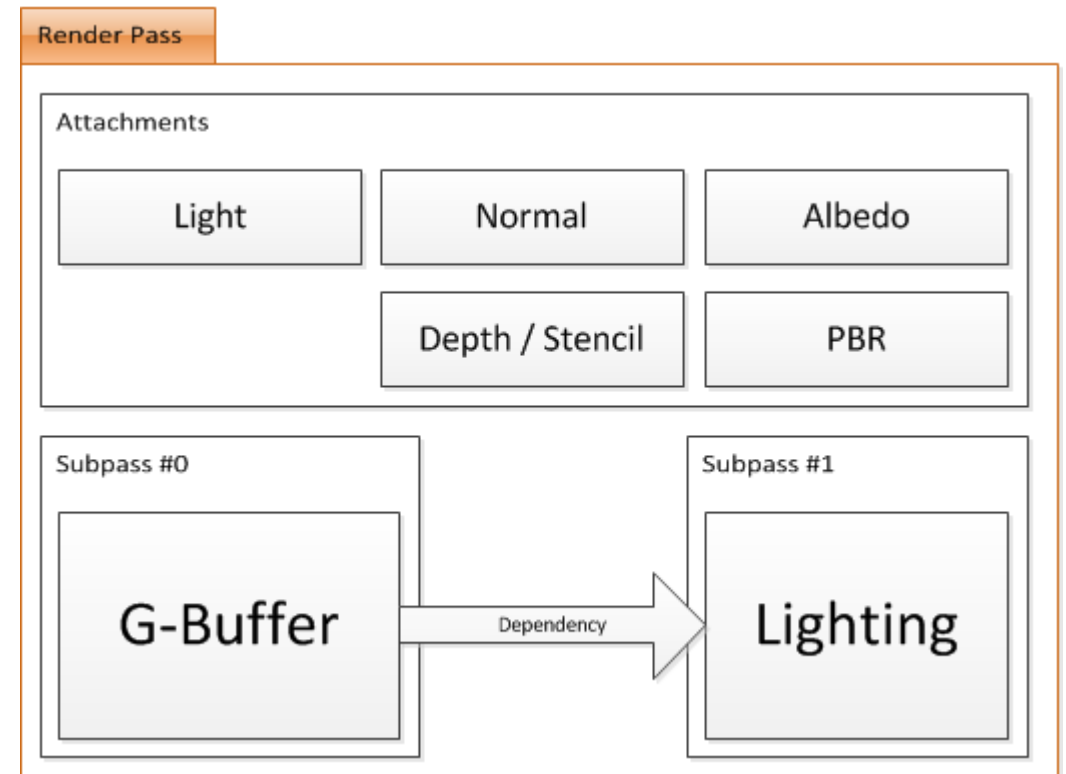
©ARM 2017

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

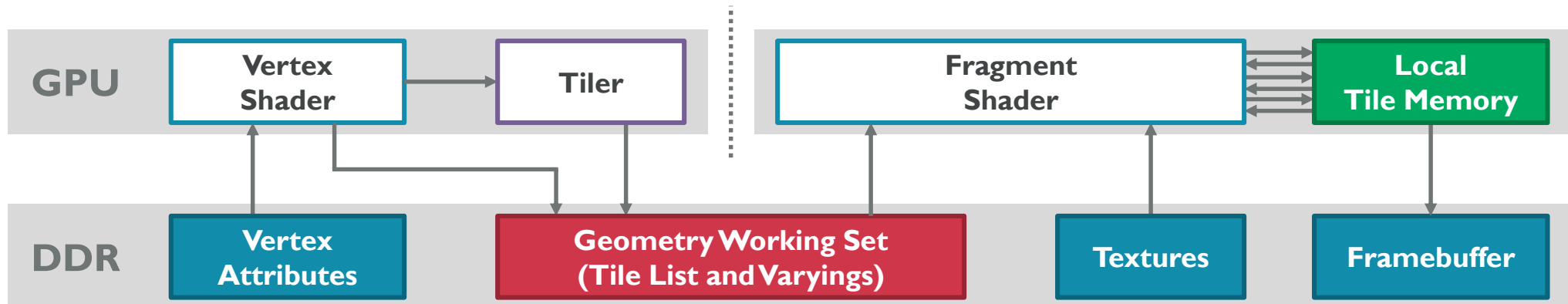
What is Multipass?

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



Tiled GPUs I/O

- Tiled 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!
- Main memory is written to when tile is complete
- Having framebuffer in on-chip SRAM has practical benefits
 - **Read/Write to it is cheap, no external bandwidth cost**

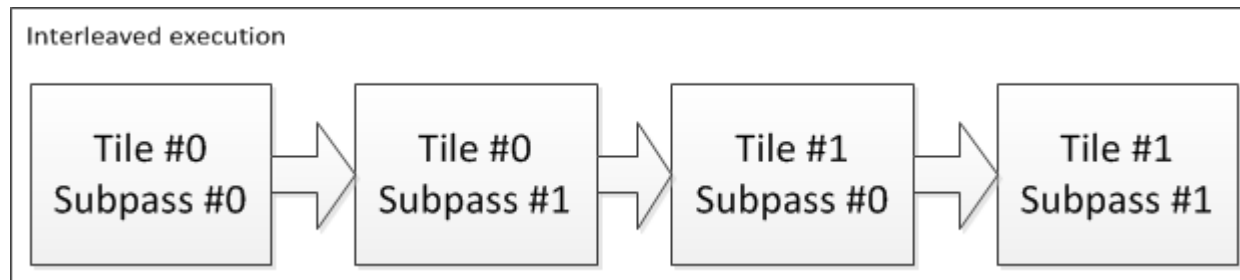


Improved MRT Deferred Shading in Vulkan

- 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_WRITE | DEPTH_WRITE -> INPUT_ATTACHMENT | DEPTH_READ
 - **VK_DEPENDENCY_BY_REGION_BIT**

Tiled 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 😊

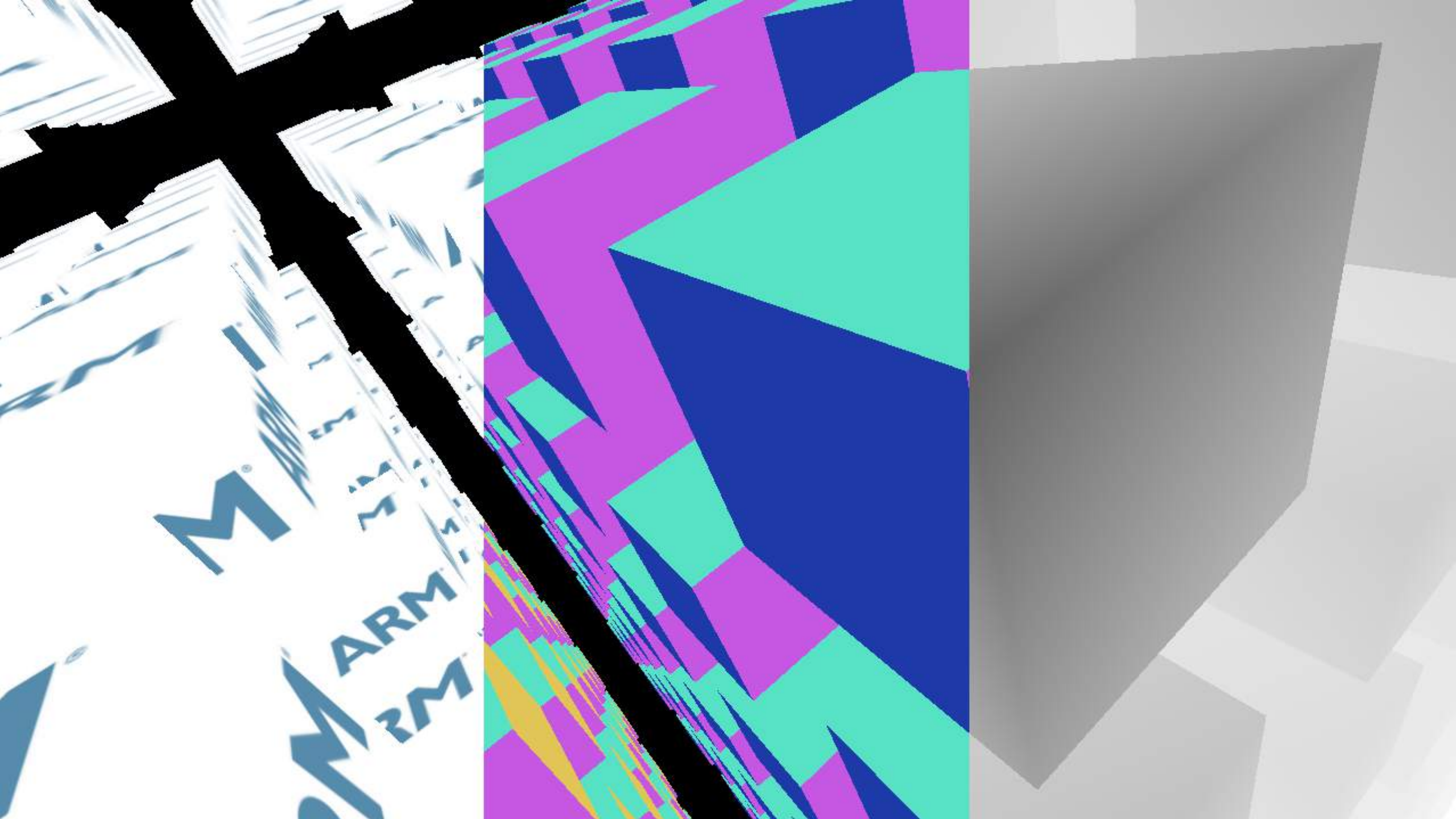
Multipass Benefits Everyone

- 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



Baseline Test

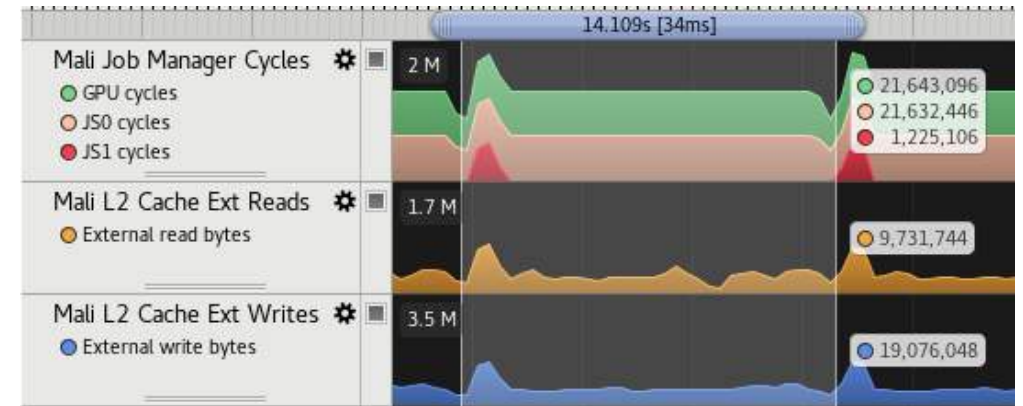
- Basic multipass sample
- One renderpass
- Light on geometry
- ~8 large lights
 - Simple shading
- Benchmark
 - Multipass (subpass fusing)
 - MRT
 - Overall performance
 - Bandwidth



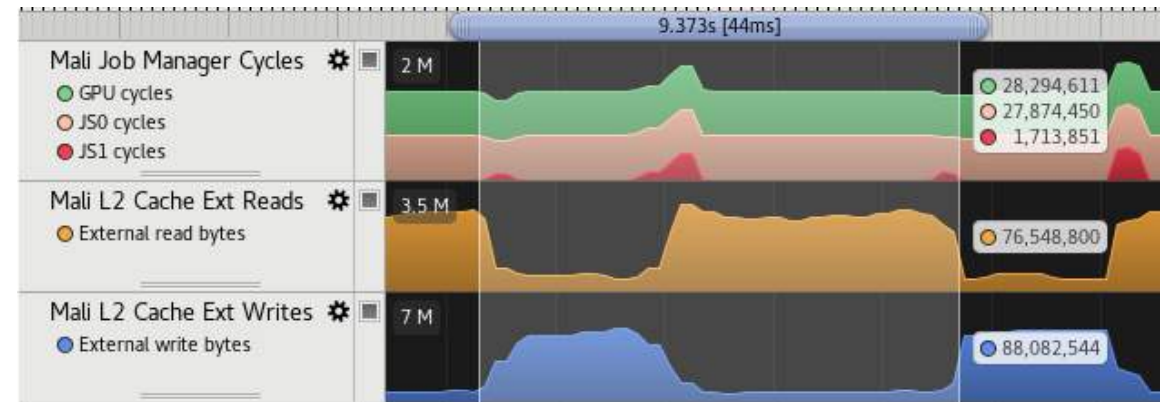
Baseline Test Data

- 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



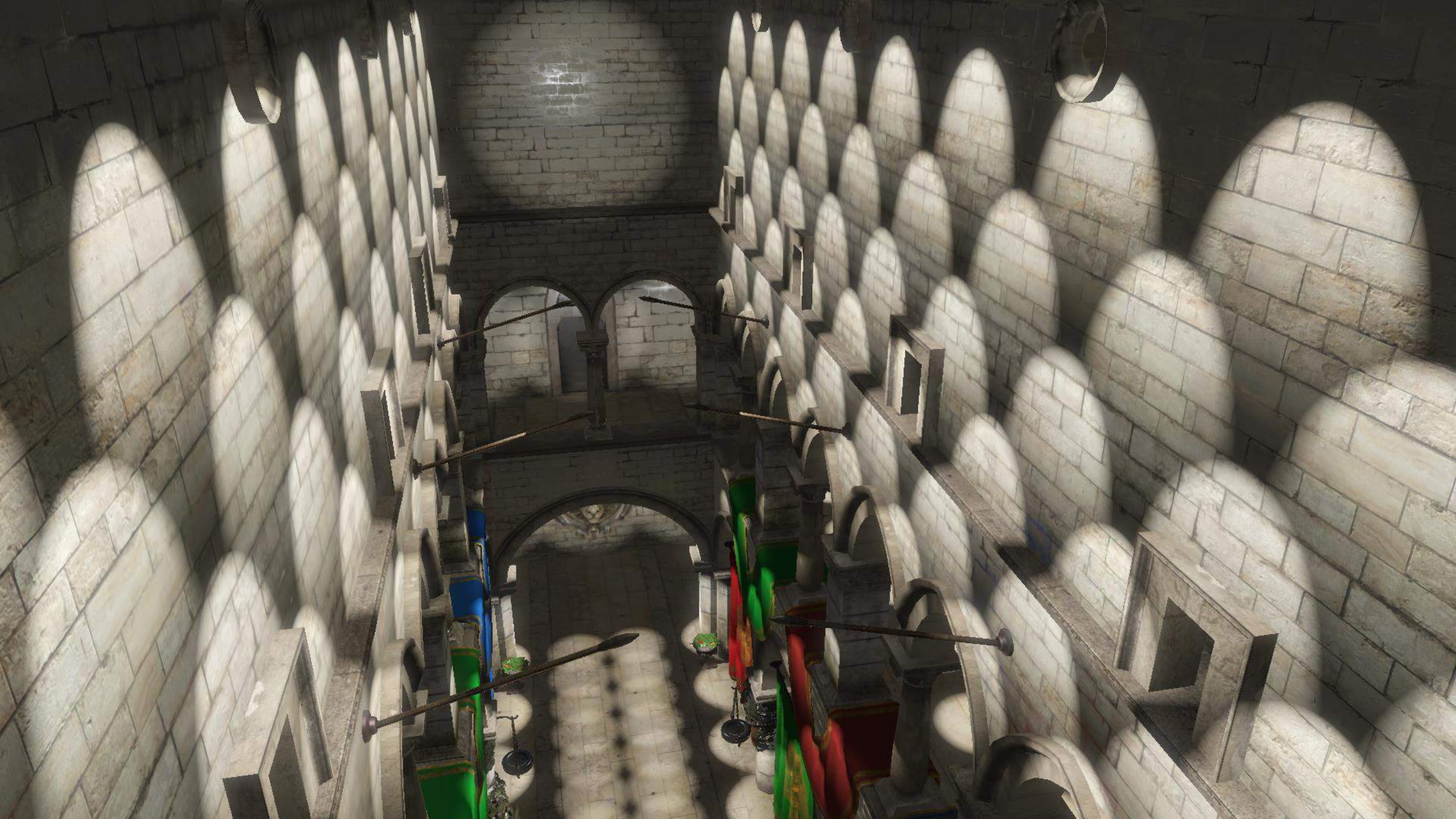
MRT



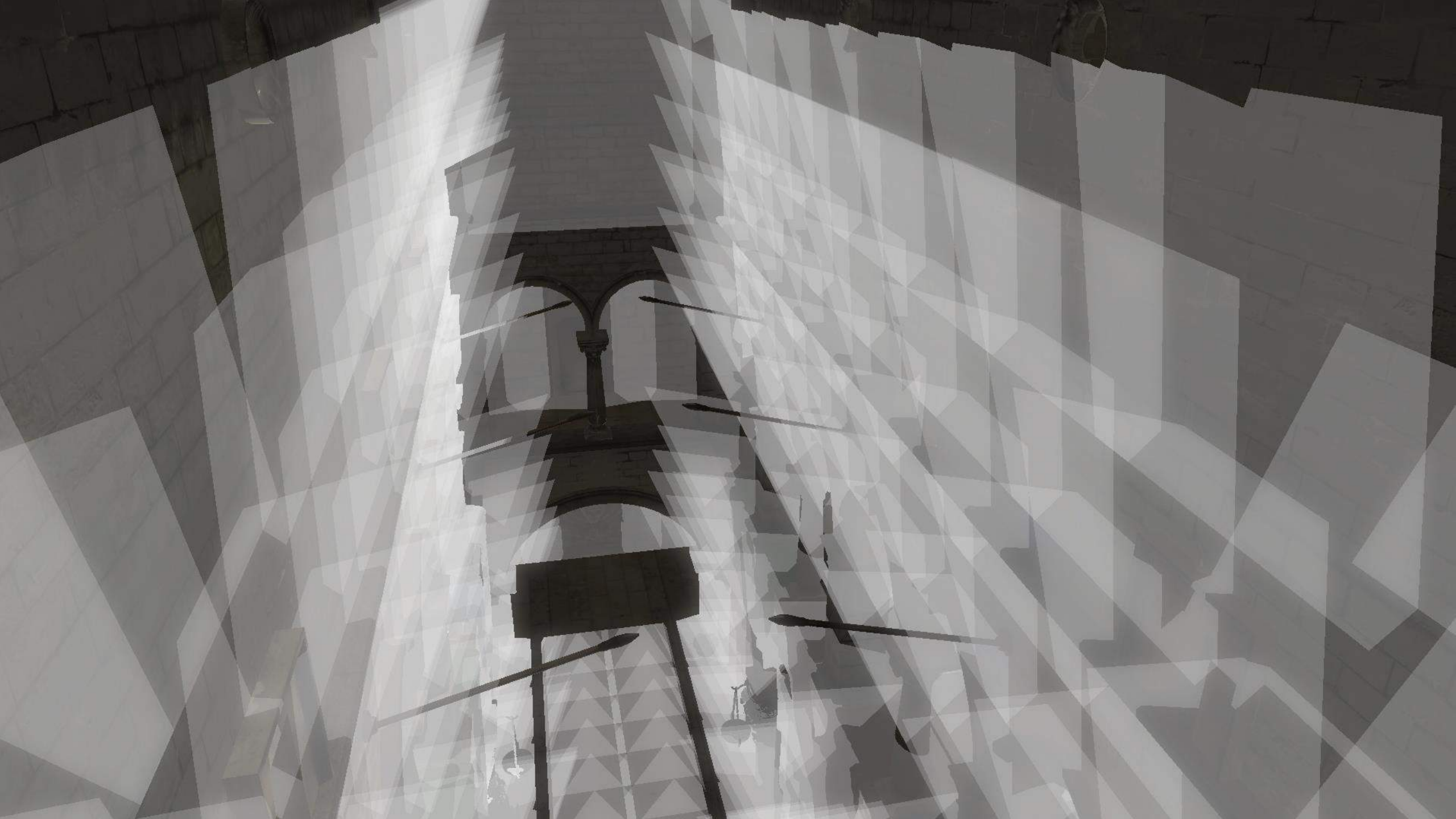
Sponza Test

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

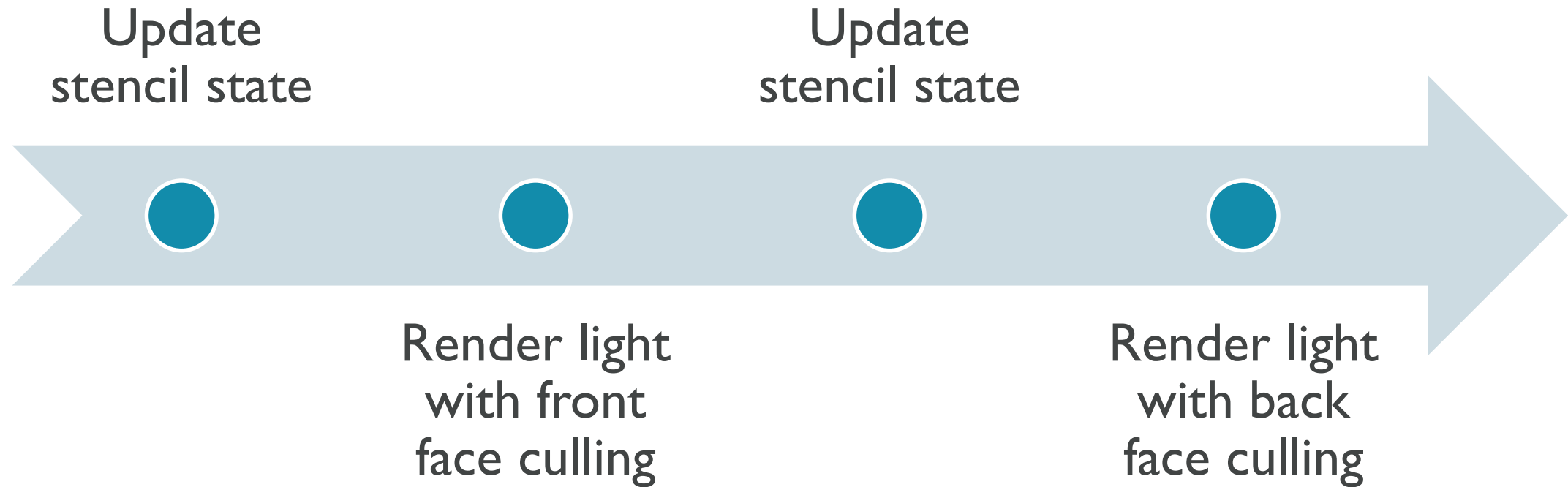








Stencil Culling



Clustered Stencil Culling

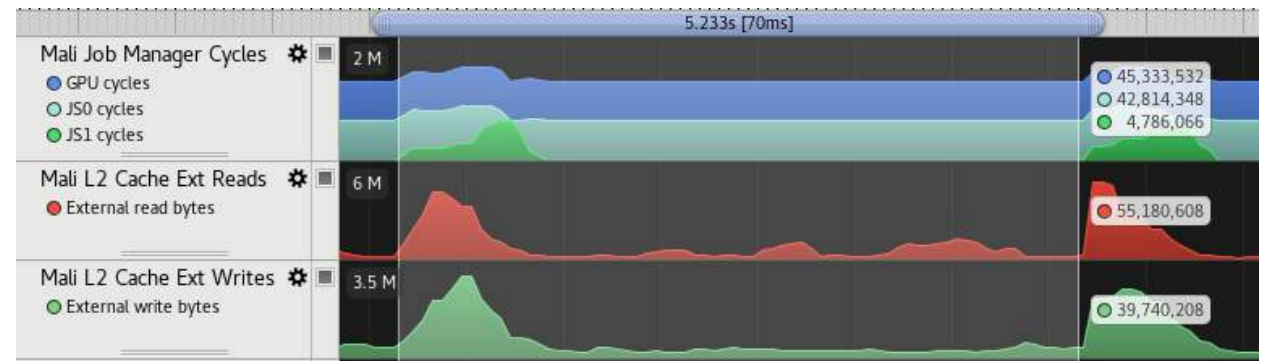
- Classic method of per light stencil culling involves a lot of state toggling
 - Can be expensive even on Vulkan
 - Usually performs poorly on GPU
- Instead, we cluster local lights along camera Z axis
 - Each light is assigned to a cluster using conservative depth
 - Stencil is written for all local 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



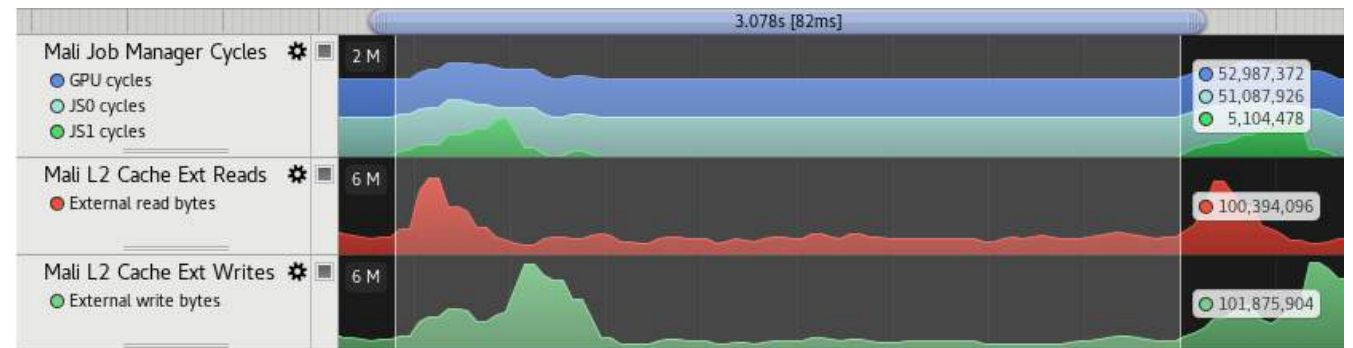
Sponza Test Data

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

Multipass



MRT

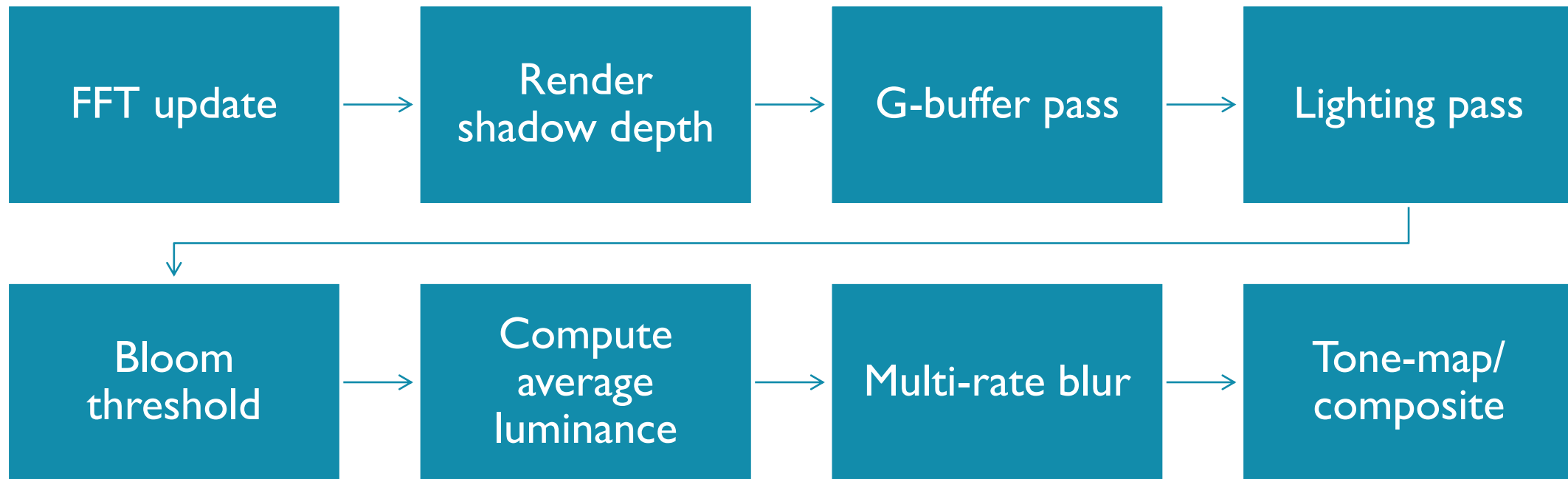




«Lofoten»

- 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



Implementation Details

- G-buffer pass
 - Only light buffer attachment backed by physical memory
 - Emissive materials written directly to light buffer (no extra bandwidth cost)
 - Stencil set to mark reflection influence
- Lighting pass
 - Lighting accumulated using additive blending to lighting attachment
 - Clustered stencil culling used for local lights
- Environment pass
 - Fog applied after shading is complete
- Only committed to memory once the entire render pass is complete

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 };
multiPass.numSubpasses = 2;
multiPass.subpass[0].colorTargets = { RT0, VIRTUAL0, VIRTUAL1, VIRTUAL2 };
multiPass.subpass[0].depthStencil = DEPTH_STENCIL_READ_WRITE;
multiPass.subpass[1].colorTargets = { RT0 };
multiPass.subpass[1].inputs = { VIRTUAL0, VIRTUAL1, VIRTUAL2, DEPTH };
multiPass.subpass[1].depthStencil = DEPTH_STENCIL_READ_ONLY;
renderPass.multiPass = &multiPass;

BeginRenderPass(&renderPass);
```

Multipass – Virtual Attachments

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

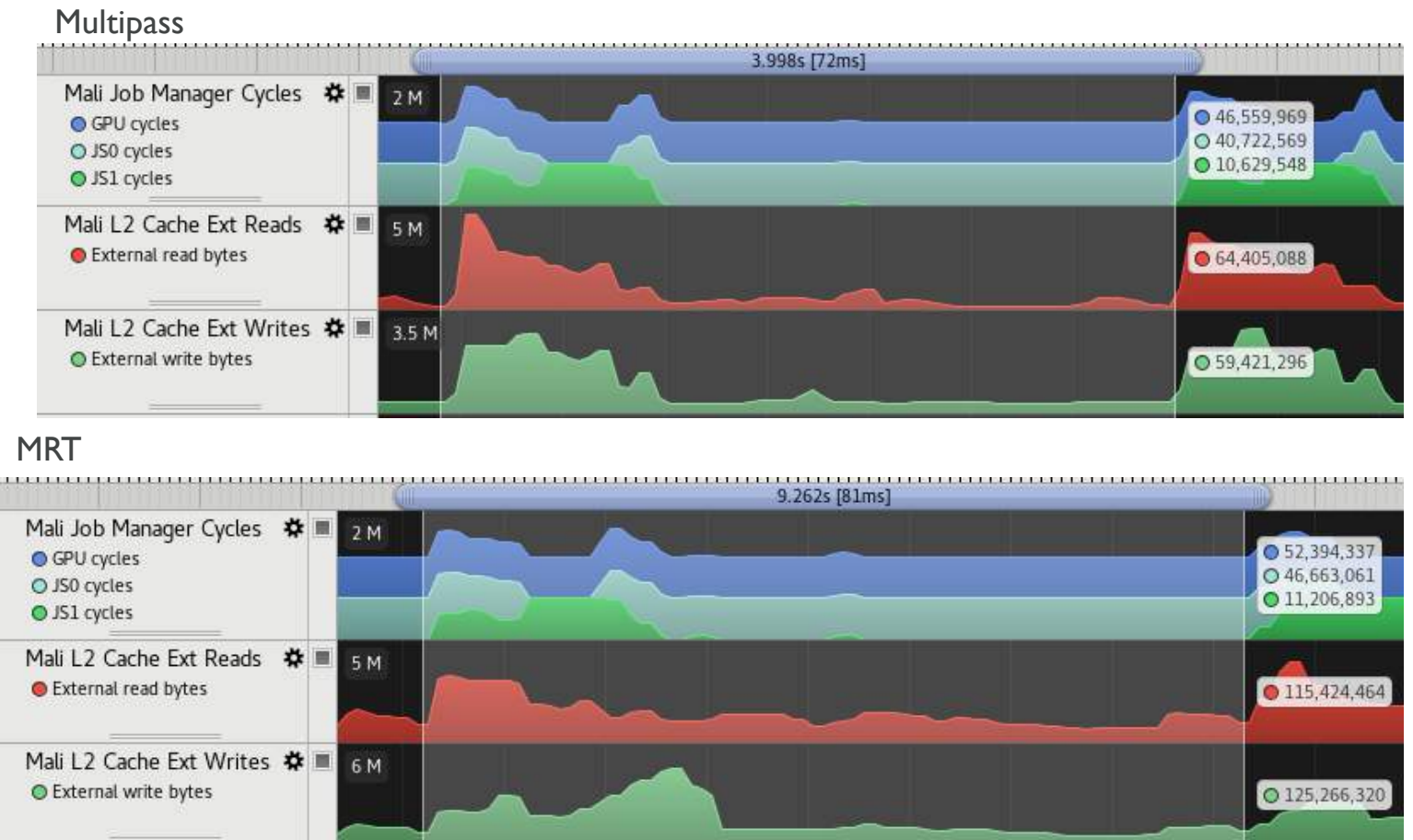
DrawGeometry(pCB);
// Increment to the next subpass.
pCB->NextSubpass();

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

pCB->EndRenderPass();
```

«Lofoten» Test Data

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





Questions?



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

Copyright © 2017 ARM Limited

©ARM 2017

BONUS SLIDES

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

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

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;
```