Vulkanised 2025
The 7th Vulkan Developer Conference
Cambridge, UK | February 11-13, 2025

Arm ASR: Platform Agnostic Efficient Temporal Upscaling on Mobile

Sergio Alapont Granero, Arm

Agenda

1. Some context
2. Performance & Image Quality
3. Integrating Arm ASR
4. Arm ASR meets Vulkan
5. Conclusions

Some context

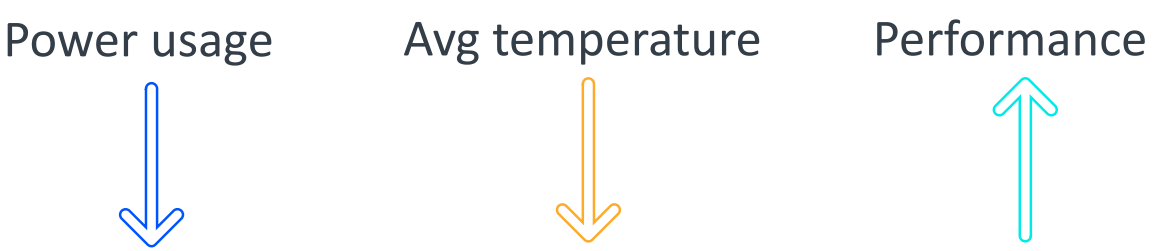
arm

Need to do things in a smarter way

- Techniques like lighting or Screen-Space effects scale proportionally to resolution
  - The higher the resolution, the more expensive these become and the higher the used memory and bandwidth cost

- Super-Resolution seemed the way to go. Render a reduced % of the pixels and reconstruct the rest
  - I.e., 2x upscaling (per-dimension) allows to render 25% of the pixels and reconstruct the rest.

- With Arm ASR we wanted to fill existing gaps on mobile around upscaling



Requirements for our upscaler

- **Upscaler type**. There are multiple good spatial upscalers in the market. There is a gap on mobile when it comes to more advanced solutions (… spatio-temporal) that allow for more aggressive upscaling workflows

- **Simplify developer adoption**. Reduce to the bare minimum the learning curve and time taken to adopt our solution

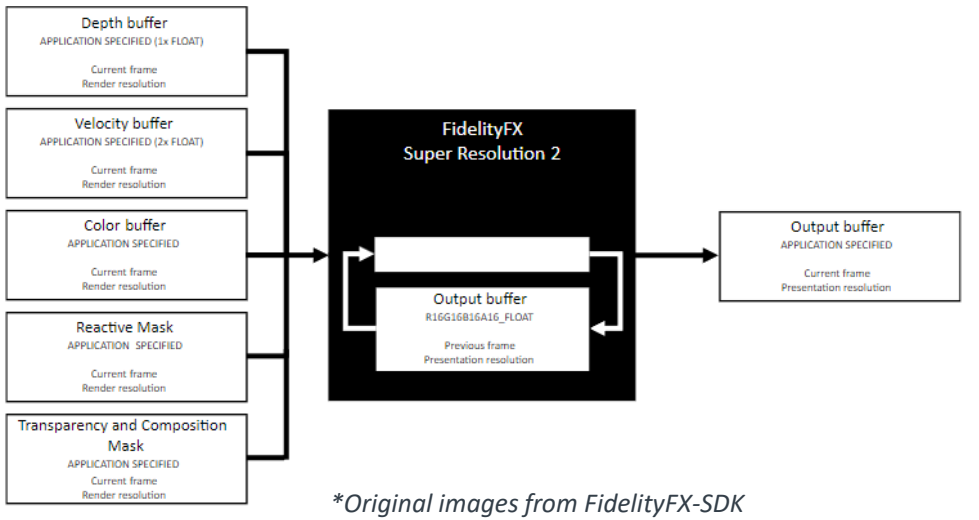
- **Compatible API to other solutions**. Avoid integration API fragmentation as much as possible

Using AMD's FSR2 as our starting point

- Engine needs:
  - Jitter the frames (Optimal Halton sequence)
  - Produce motion vectors
  - Global mip bias for texturing to compensate for the rendering resolution

```
mipBias = log2(renderResolution/displayResolution) - 1.0;
```

- Inputs
  - Src-res input color: Jittered, linearRGB or tonemapped
  - Src-res depth: Supports reversed-Z
  - Src/Dst-res motion vectors: RG16 in UV space
  - Optional reactive mask
- Outputs
  - Dst-res color in either linearRGB or tonemapped



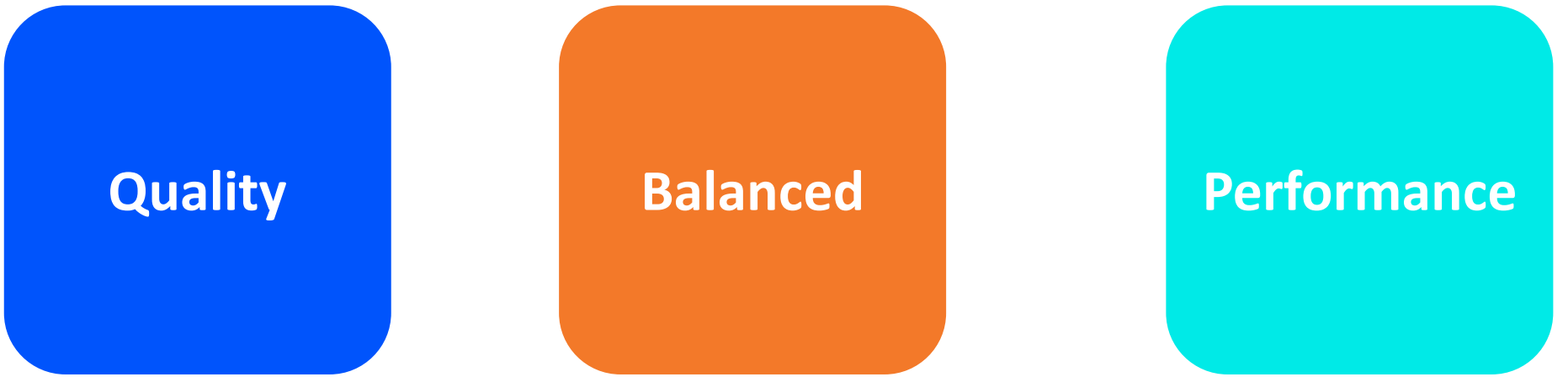
**Original images from FidelityFX-SDK*

Performance & Image Quality

arm

Shader Quality Presets

- Aim to provide further Performance/Quality scalability to developers.
  - Mobile has a wide range of devices
  - **Separate upscaling ratio from shader quality presets**
  - Do performance-quality tradeoffs



Starting Point - How we measure Image Quality?

- **PSNR**
  - Peak Signal to Noise Ratio. Spatial measure of image distortion between a reference and distorted image pair.
  - Evaluate the reconstruction.
  - >= 30 (dB) is great for upscalers

- **SSIM**
  - Structural Similarity Index Measure. Spatial measure that aims to better fit the human visual system.
  - Evaluate the reconstruction

- **tPSNR (Temporal PSNR)**
  - Residuals between frame t and t-1 for both the reference and the distorted sequence are computed. PSNR is then taken between the two residuals to get a rough estimate for temporal stability.
  - Good for tracking ghosting, temporal stability

- **STRRED**
  - Spatio-Temporal Reduced Reference Entropic Differencing
  - Good for tracking temporal instability, ghosting or shimmering.

Starting point – What's our performance baseline?

- Target res of **2400x1080**. Running on **Immortalis-G720** device
- High **duration (ms)** (measured with **timelines** in Streamline)
- High **bandwidth (MiB)** too (look at "Mali Memory Bandwidth Usage" **counters**)

FSR2 x2 (1200x540 -> 2400x1080)	Duration (ms)	Read BW (MiB)	Write BW (MiB)	Total BW (MiB)
LuminancePyramid	2.441	46.7	53.9	100.6
ReconstructPrevDepth	0.368	6.91	8.1	15.01
DepthClip	0.305	6.97	7.18	14.15
Lock	0.1	2.12	2.03	4.15
AccumulateSharpen	2.725	53.2	48.5	101.7
RCAS	0.611	18.9	10.6	29.5
TOTAL	**6.55**	**134.8**	**130.31**	**265.11**
TOTAL (No RCAS)	**5.939**	**115.9**	**119.71**	**235.61**

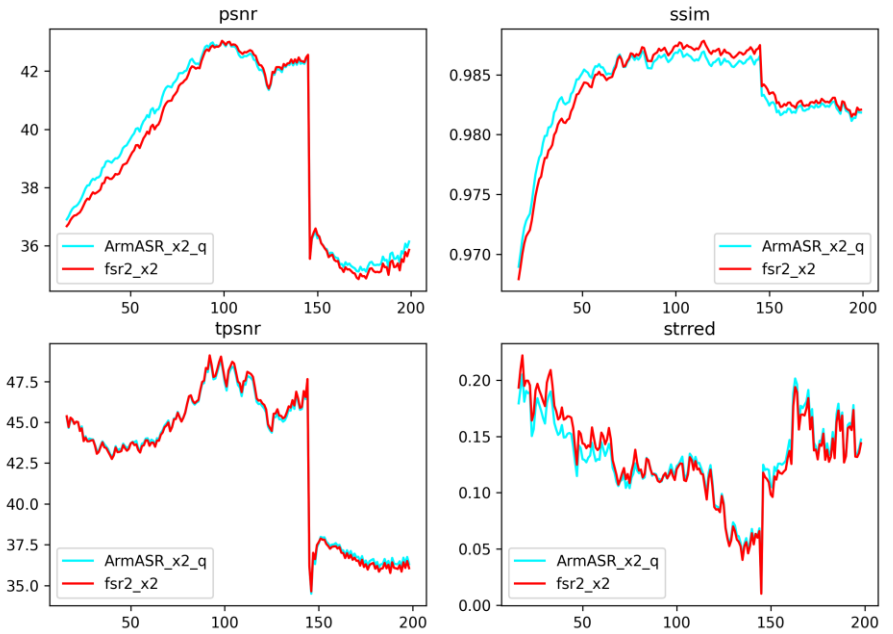
FSR2 x1.7 (1412x635 -> 2400x1080)	Duration (ms)	Read BW (MiB)	Write BW (MiB)	Total BW (MiB)
LuminancePyramid	3.161	58.7	49.3	108
ReconstructPrevDepth	0.527	7.7	10.8	18.5
DepthClip	0.444	11.7	9.3	21
Lock	0.137	1.84	2.35	4.19
AccumulateSharpen	2.804	55.9	45.5	101.4
RCAS	0.664	19.8	10.4	30.2
TOTAL	**7.737**	**155.64**	**127.65**	**283.29**
TOTAL (No RCAS)	**7.073**	**135.84**	**117.25**	**253.09**

FSR2 x1.5 (1600x720 -> 2400x1080)	Duration (ms)	Read BW (MiB)	Write BW (MiB)	Total BW (MiB)
LuminancePyramid	3.676	88.4	99.7	188.1
ReconstructPrevDepth	0.724	9.2	13.5	22.7
DepthClip	0.644	14.4	11.6	26
Lock	0.171	2.98	3.03	6.01
AccumulateSharpen	2.822	61.1	49.3	110.4
RCAS	0.681	21.7	10.8	32.5
TOTAL	**8.718**	**197.78**	**187.93**	**385.71**
TOTAL (No RCAS)	**8.037**	**176.08**	**177.13**	**353.21**

Arm ASR – Final “Quality” preset

Quality

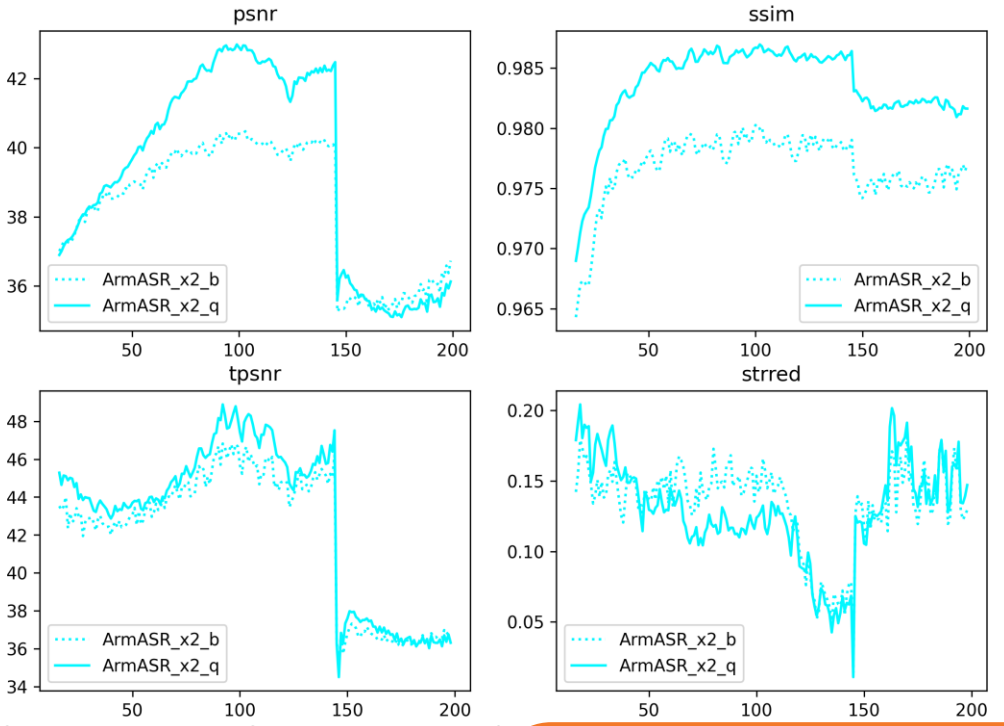
- The “Quality” shader preset was an ultra-optimized FSR2. Roughly 100% IQ match
- This can be used on production by high-end content
  - Duration within a reasonable budget, we can still do better about bandwidth though
- Time to scale things down and make controlled sacrifices



TOTAL	FSR2		Arm ASR (Quality)			
	Duration (ms)	Total BW (MiB)	Duration (ms)	Relative (%)	Total BW (MiB)	Relative (%)
x2 (1200x540 -> 2400x1080)	6.55	265.11	2.453	-63%	76.222	-71%
No RCAS, x2 (1200x540 -> 2400x1080)	5.939	235.61	2.046	-66%	63.382	-73%
x1.7 (1412x635 -> 2400x1080)	7.737	283.29	2.586	-67%	90.14	-68%
No RCAS, x1.7 (1412x635 -> 2400x1080)	7.073	253.09	2.242	-68%	75.85	-70%
x1.5 (1600x720 -> 2400x1080)	8.718	385.71	2.786	-68%	96.43	-75%
No RCAS, x1.5 (1600x720 -> 2400x1080)	8.037	353.21	2.467	-69%	82.71	-77%

Balanced Preset - Results

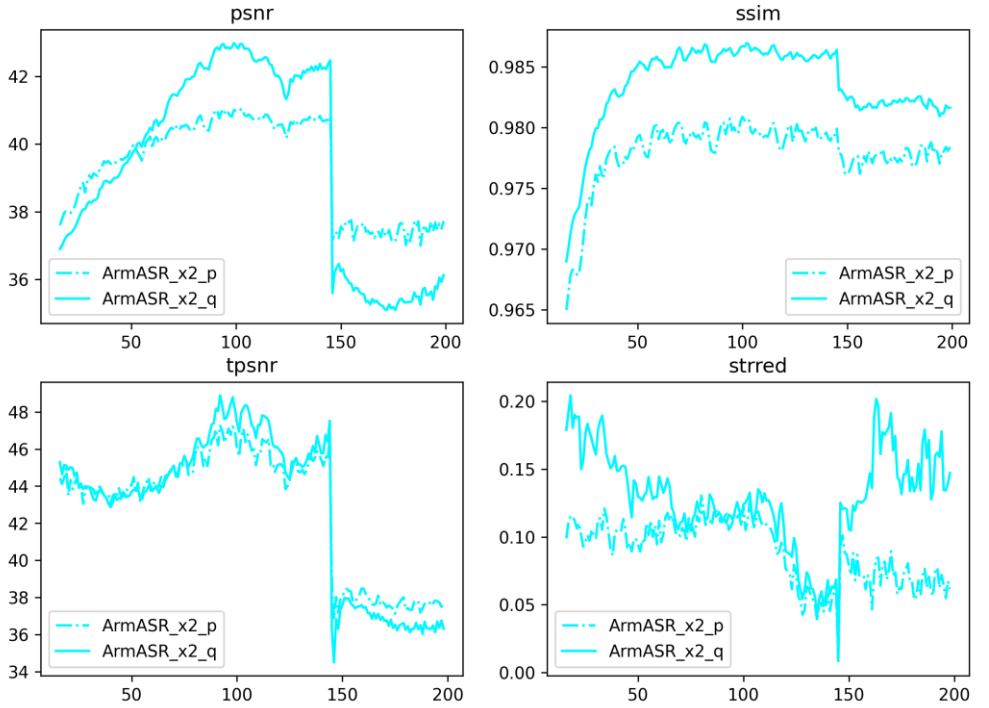
Balanced



TOTAL	Arm ASR (Quality)		Arm ASR (Balanced)			
	Duration (ms)	Total BW (MiB)	Duration (ms)	Relative (%)	Total BW (MiB)	Relative (%)
x2 (1200x540 -> 2400x1080)	2.453	76.222	1.929	-21%	58.73	-23%
No RCAS, x2 (1200x540 -> 2400x1080)	2.046	63.382	1.632	-20%	48.29	-24%
x1.7 (1412x635 -> 2400x1080)	2.586	90.14	2.2	-15%	72.26	-20%
No RCAS, x1.7 (1412x635 -> 2400x1080)	2.242	75.85	1.9	-15%	61.71	-19%
x1.5 (1600x720 -> 2400x1080)	2.786	96.43	2.459	-12%	80.26	-17%
No RCAS, x1.5 (1600x720 -> 2400x1080)	2.467	82.71	2.162	-12%	69.15	-16%

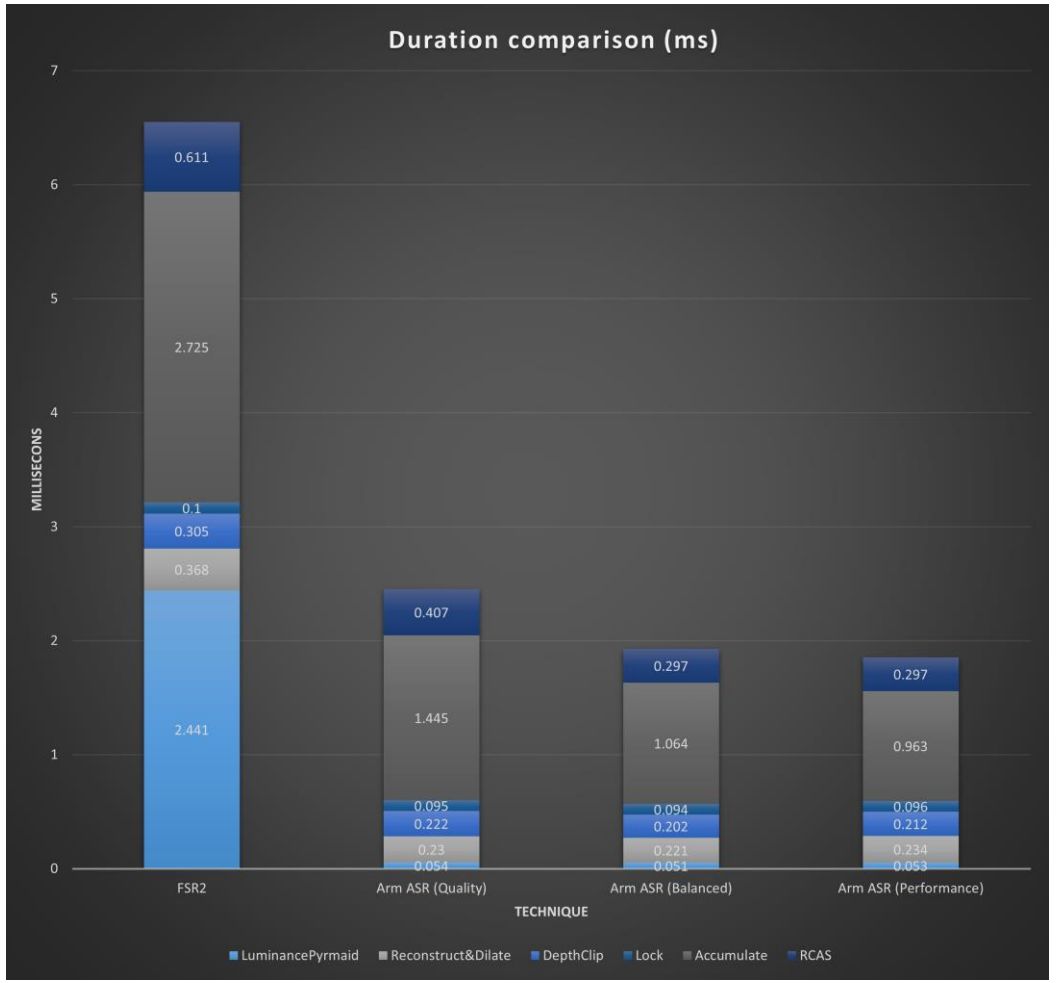
Performance Preset - Results

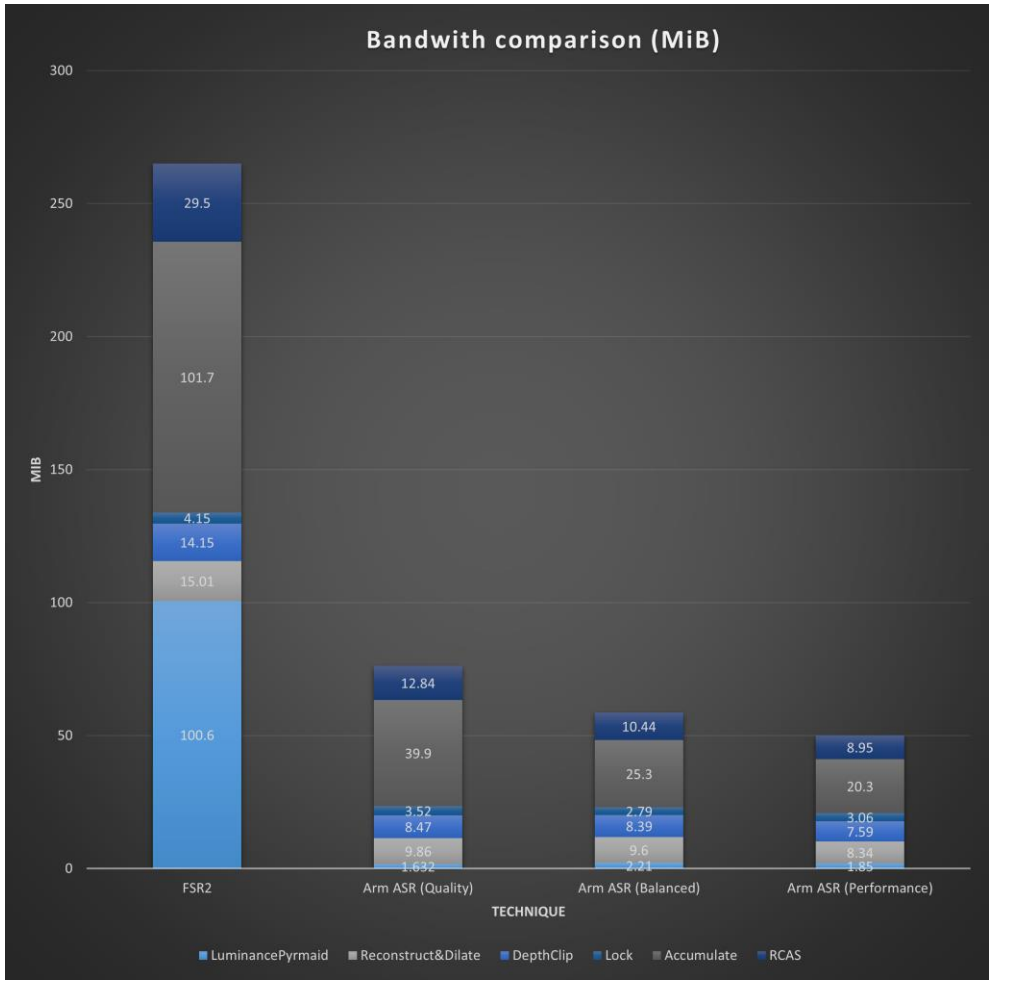
Performance



TOTAL	Arm ASR (Balanced)		Arm ASR (Performance)			
	Duration (ms)	Total BW (MiB)	Duration (ms)	Relative (%)	Total BW (MiB)	Relative (%)
x2 (1200x540 -> 2400x1080)	1.929	58.73	1.855	-4%	50.09	-15%
No RCAS, x2 (1200x540 -> 2400x1080)	1.632	48.29	1.558	-5%	41.14	-15%
x1.7 (1412x635 -> 2400x1080)	2.2	72.26	2.044	-7%	61.07	-15%
No RCAS, x1.7 (1412x635 -> 2400x1080)	1.9	61.71	1.744	-8%	52.42	-15%
x1.5 (1600x720 -> 2400x1080)	2.459	80.26	2.306	-6%	71.29	-11%
No RCAS, x1.5 (1600x720 -> 2400x1080)	2.162	69.15	2.007	-7%	62.59	-9%

FSR2 vs Arm ASR (All presets) – 2x (1200x540 -> 2400x1080)





Arm ASR at GPC 24

- For more details, watch our GPC24 talk for a deep-dive of the optimizations we did in Arm ASR.

arm

Accuracy Super Resolution

ON

FPS: 39.056

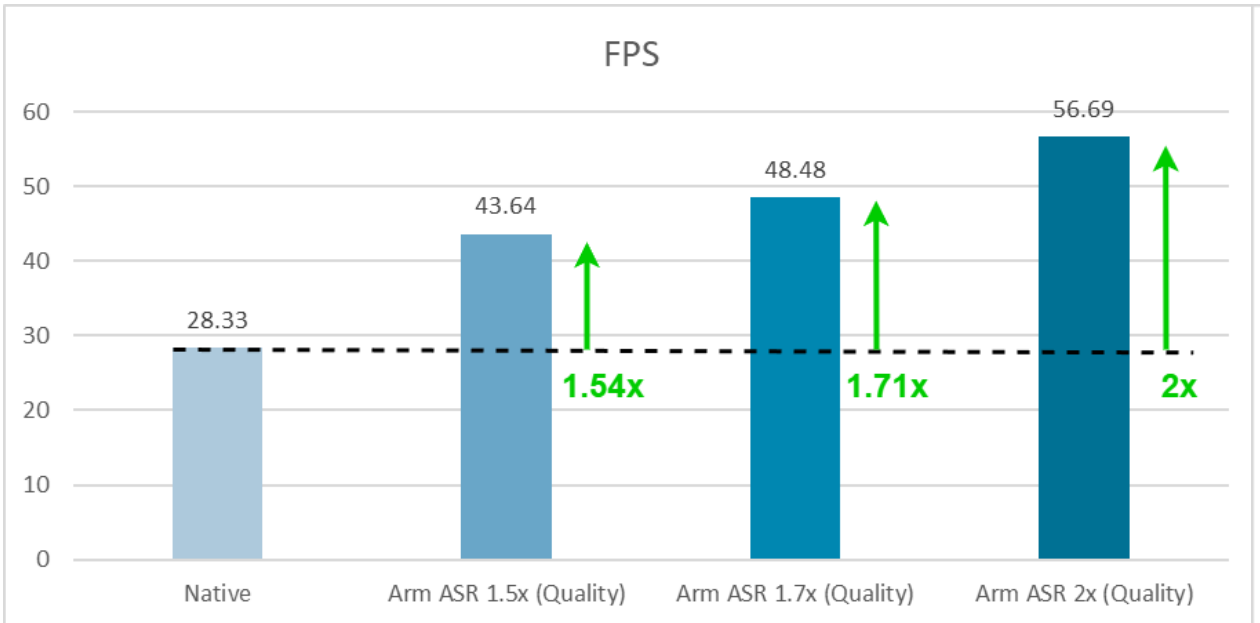
OFF

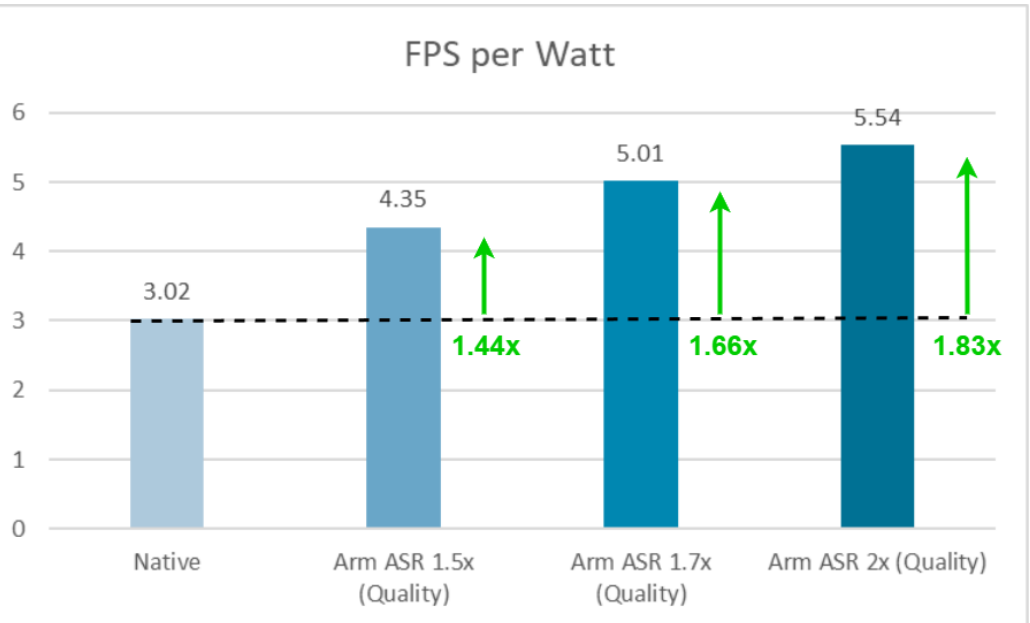
FPS: 20.274

arm

Some more data (internal benchmarking)

- Running on an Immortalis-G720 device at target-res 1600x720
  - Average data gathered after 3 runs, each run was 900 frames long
- Used the Quality preset, plenty of savings in this content to offset for the upscaler





Integrating Arm ASR

arm

Where can I run Arm ASR?

- **Not vendor specific.**
  - Can run in any platform.

- **Gfx API agnostic**
  - Arm ASR is a shader-based technique
  - Can run with every API out there: Vulkan, GLES, DX11, DX12, Metal …

- **Advise to run on late 2022 devices (i.e Immortalis-G715) and onward**
  - Hard to run efficiently temporal techniques in older devices without sacrificing too much IQ
  - For older devices you might want to fallback to spatial upscalers

User experience: The challenge!

Technique's complexity

- Multiple stages/shaders, intermediate resources …
- Multiple shader quality presets (variants)
- Providing the correct uniform data

Integration time

- Vulkan is our main target, and it is not a trivial API
- Minimize how long it takes to do a first working integration
- Leave the door open for developers to connect things tightly with their renderers

Maintenance & Scalability

- Minimize user's API fragmentation with existing techniques
- Abstract algorithm details, API & platform
- Think about the future and minimize impact to developers updating to newer versions

What we did … follow AMD’s lead with FidelityFX SDK

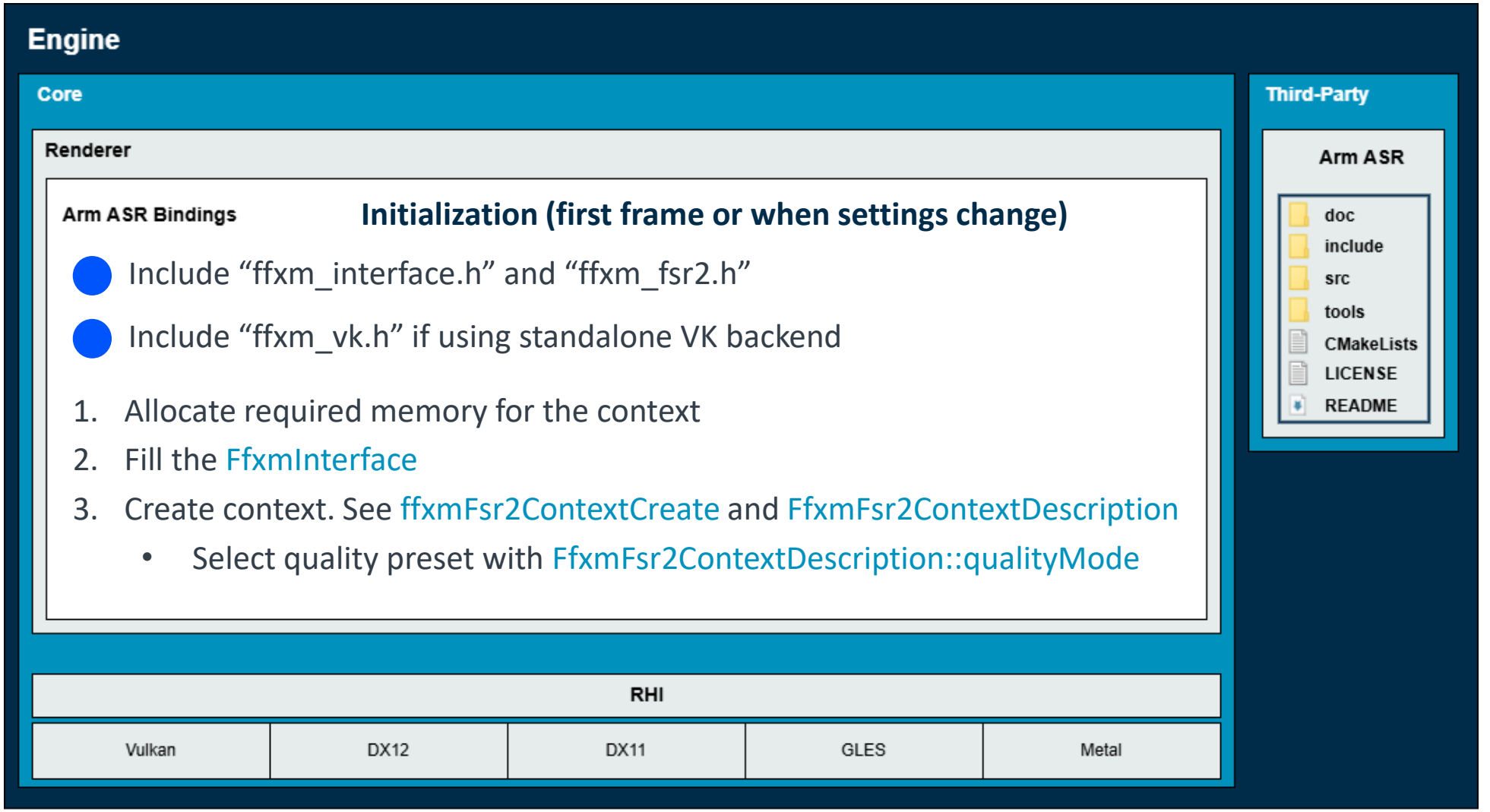
- Reference implementation comes with everything needed for ArmASR to be almost standalone (modified FidelityFX-SDK and the technique)

- We rely on **ffx** (**ffxm** to avoid symbols duplication issues) to provide a simple wrapping API abstracting algorithm details

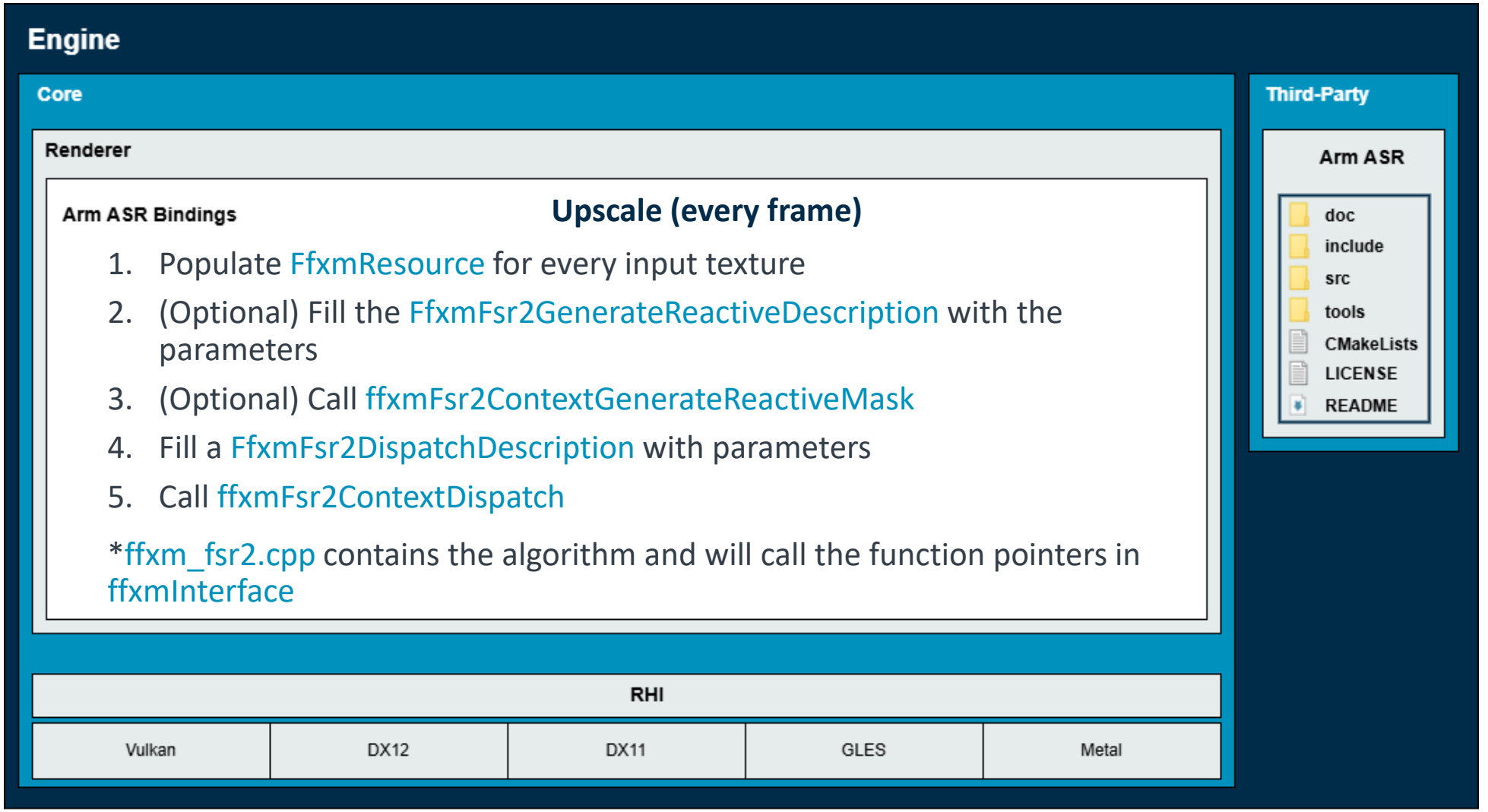
- Provide an **extended standalone VK backend** with precompiled shaders for quick integrations in Vulkan environments

- **FfxmInterface** can be overridden by the engine to connect the technique tightly with the renderer (i.e RenderGraph)

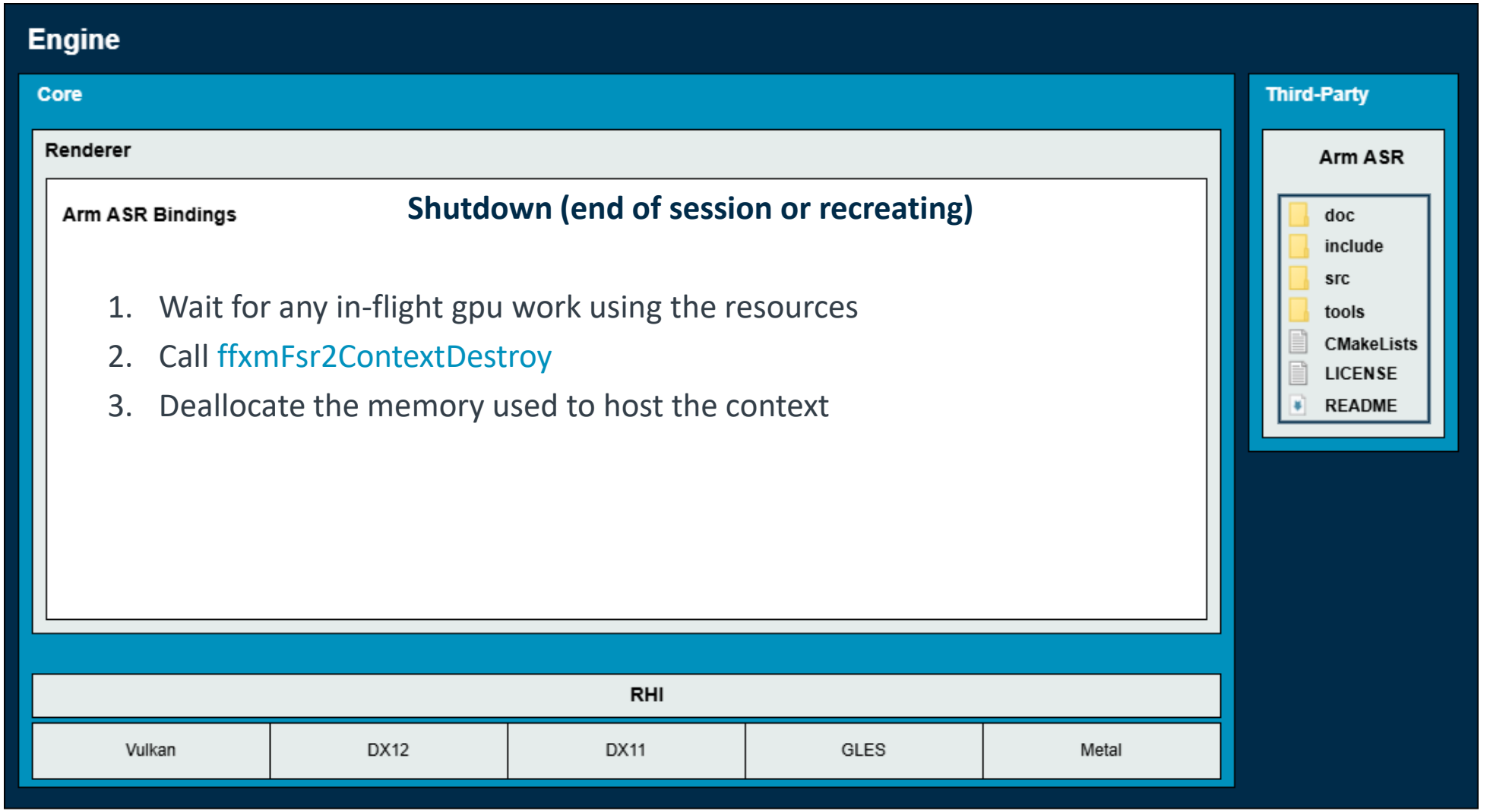
How to … initialize?



How to … upscale?



How to … shutdown?



User’s responsibilities (tight integration)

- Shader compilation and loading as part of your engine’s workflow
- Patch shader bindings (UAV, SRV, UBOs, RTs) when **fpCreateComputePipeline** or **fpCreateGraphicsPipeline** are called. See FfxmPipelineState.
- Provide custom hooks for function pointers in FfxmInterface.

```
typedef struct FfxmInterface {

    FfxmGetSDKVersionFunc            fpGetSDKVersion;           ///< A callback function to query the SDK version.
    FfxmCreateBackendContextFunc     fpCreateBackendContext;    ///< A callback function to create and initialize the backend context.
    FfxmGetDeviceCapabilitiesFunc    fpGetDeviceCapabilities;   ///< A callback function to query device capabilities.
    FfxmDestroyBackendContextFunc    fpDestroyBackendContext;   ///< A callback function to destroy the backend context. This also dereferences the device.
    FfxmCreateResourceFunc           fpCreateResource;          ///< A callback function to create a resource.
    FfxmRegisterResourceFunc         fpRegisterResource;        ///< A callback function to register an external resource.
    FfxmGetResourceFunc              fpGetResource;             ///< A callback function to convert an internal resource to external resource type
    FfxmUnregisterResourcesFunc      fpUnregisterResources;     ///< A callback function to unregister external resource.
    FfxmGetResourceDescriptionFunc   fpGetResourceDescription;  ///< A callback function to retrieve a resource description.
    FfxmDestroyResourceFunc          fpDestroyResource;         ///< A callback function to destroy a resource.
    FfxmCreatePipelineFunc           fpCreateComputePipeline;   ///< A callback function to create a compute pipeline.
    FfxmCreatePipelineFunc           fpCreateGraphicsPipeline;  ///< A callback function to create a render pipeline.
    FfxmDestroyPipelineFunc          fpDestroyPipeline;         ///< A callback function to destroy a render or compute pipeline.
    FfxmScheduleGpuJobFunc           fpScheduleGpuJob;          ///< A callback function to schedule a render job.
    FfxmExecuteGpuJobsFunc           fpExecuteGpuJobs;          ///< A callback function to execute all queued render jobs.

    void*                           scratchBuffer;             ///< A preallocated buffer for memory utilized internally by the backend.
    size_t                          scratchBufferSize;         ///< Size of the buffer pointed to by <c><i>scratchBuffer</i></c>.
    FfxmDevice                       device;                    ///< A backend specific device

} FfxmInterface;
```

Arm ASR meets Vulkan

arm

State of things

- Dealing with low level Vulkan details takes time
  - Synchronization, resource management, RenderPass …
- A tight integration with Vulkan might take longer if your engine is not well prepared
- We wanted to help developers to speedup their integrations when targeting Vulkan
  - Reused and extended the standalone Vulkan backend

So, how does the integration look like?

- Shaders are precompiled (See “tools/generate_prebuilt_shaders.py”) and serialized in headers with reflection information.
  - Either hlsl or glsl can be used in this workflow. For hlsl, we use [[vk::binding(…)]] when compiling with DXC.
  - “ffxm_vk.cpp” will retrieve the correct shader variants at pipeline creation.

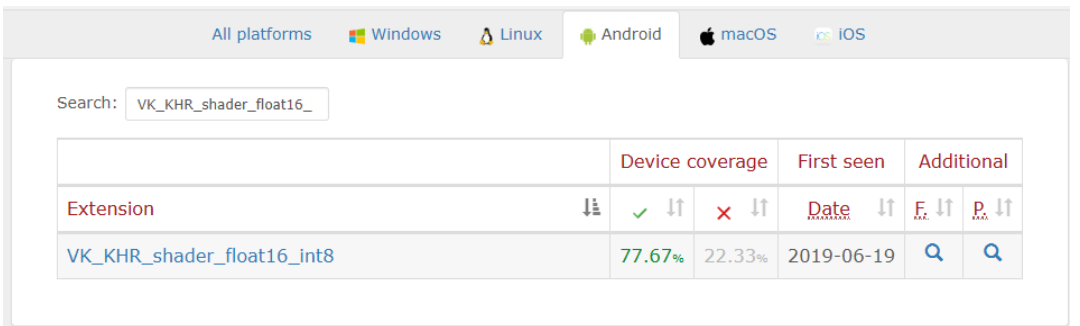
- When allocating memory and filling the FfxmInterface
  - Use ffxmGetScratchMemorySizeVK & ffxmGetInterfaceVK.

- When filling FfxmResource
  - The “resource” field must contain the ***VkImage*** handle.

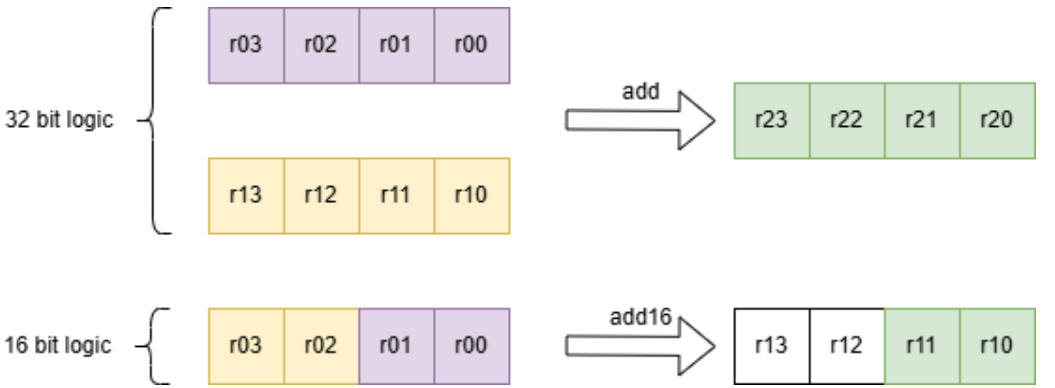
- When filling FfxmFsr2GenerateReactiveDescription or FfxmFsr2DispatchDescription
  - The “**commandList**” field should be the ***VkCommandBuffer*** handle.

Modern Vulkan: 16-bit types

- Use explicit types (wide coverage 77.67%):
  - GL_EXT_shader_explicit_arithmetic_types
  - VK_KHR_shader_float16_int8



**Use 16b arithmetic to reduce register pressure*



- Micro-optimized shader logic to tell the compiler what needed to be half precision.
  - This work happened together with us using our internal shader disassembly for full visibility
  - Thanks to doing this we removed all the stack spilling from the shaders
- We also provide fallback to plain **RelaxedPrecision** for shaders (i.e compiling for older devices)

Modern Vulkan: Subgroup ops

- AMD's SPD pass can work with subgroup quad ops for the reduction. See:
  - KHR_shader_subgroup
  - GL_KHR_shader_subgroup_quad

```
FfxFloat32x4 SpdReduceQuad(FfxFloat32x4 v)
{
#if defined(FFXM_GLSL) && !defined(FFXM_SPD_NO_WAVE_OPERATIONS)
    FfxFloat32x4 v0 = v;
    FfxFloat32x4 v1 = subgroupQuadSwapHorizontal(v);
    FfxFloat32x4 v2 = subgroupQuadSwapVertical(v);
    FfxFloat32x4 v3 = subgroupQuadSwapDiagonal(v);
    return SpdReduce4(v0, v1, v2, v3);
#else
    //...
#endif
}
```

- Arm ASR uses subgroup-based workflow by default
  - Can use FFXM_SPD_NO_WAVE_OPERATIONS to disable them

- ⚠ **Wasn't the case for FSR2/Arm ASR** but (for awareness):
  - Different platforms have different warp size (Current Mali -> 16 threads per subgroup).
  - Wrongly assuming a particular warp size could lead to buggy results in certain platforms

Conclusions

Conclusions

- Simplified learning curve by reusing most of FSR2's integration process (FidelityFX way).

- Arm ASR became a platform and API agnostic technique abstracting the user from these details. Also, provided mechanisms to target older APIs like GLES_3_2, DX11 …

- Provided 2 paths for integrations:
  - Quick integrations targeting Vulkan
  - Slower but more flexible integrations connecting the technique tightly to the engine's renderer.

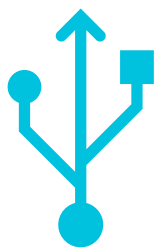
- The user is abstracted from algorithmic details.

Arm ASR Experience Kit

Coming soon

Source code

Tutorials



Plugins

arm

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