Vulkanised 2023

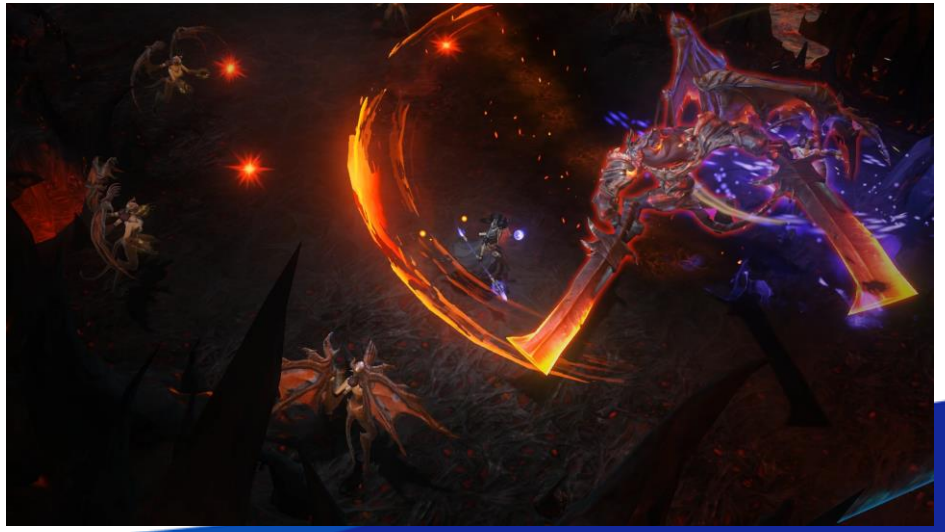
The 5th Vulkan Developer Conference
Munich, Germany / February 7–9

Battle-tested Optimisations for Mobile

Ralph Potter

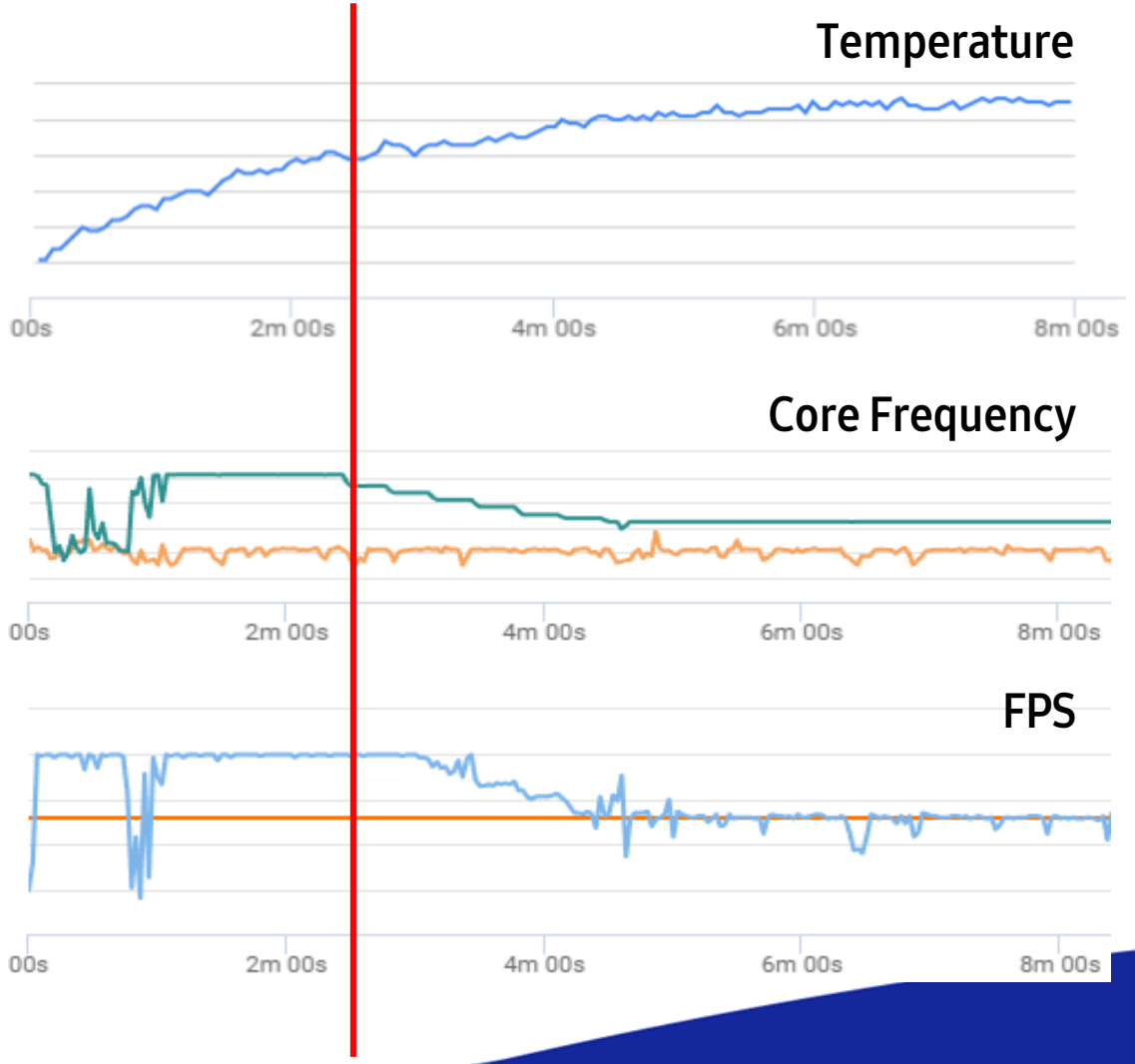
Khronos Standards, Samsung

Introduction



Constraints on Mobile

Mobile	Desktop
5W	300-500W
Passively Cooling	Active Cooling
Battery Power	Tethered Power
Shared/unified memory (30 GB/s)	Dedicated GPU memory (300 GB/s)



Methodology/Expectations

- All measurements are produced using locked, sustainable frequencies both for the CPU/GPU
- Individual optimizations in this session may only show small improvements
- Many examples will feature Unreal Engine 4.21~4.24, because the source is freely accessible.
- We've contributed similar techniques to Unity and custom engines

Best Practices and Optimisations

SAMSUNG
GPU DevTech

Only Enable What You Use

We Recommend:

- Restricting both extension and feature enables to the functionality you actually need
- Separating production and development feature sets
- Robustness (out-of-bounds) behaviour is a well-documented example
- Pipeline info and performance counter extensions can carry surprising costs

Redundant API Calls

We Recommend:

- Avoiding redundant state setting, barriers or layout transitions
- Prefer array usage of API entrypoints over multiple API calls
  - i.e. vkUpdateDescriptorSets
- Caching objects rather than creating duplicates

Choose your Formats Wisely

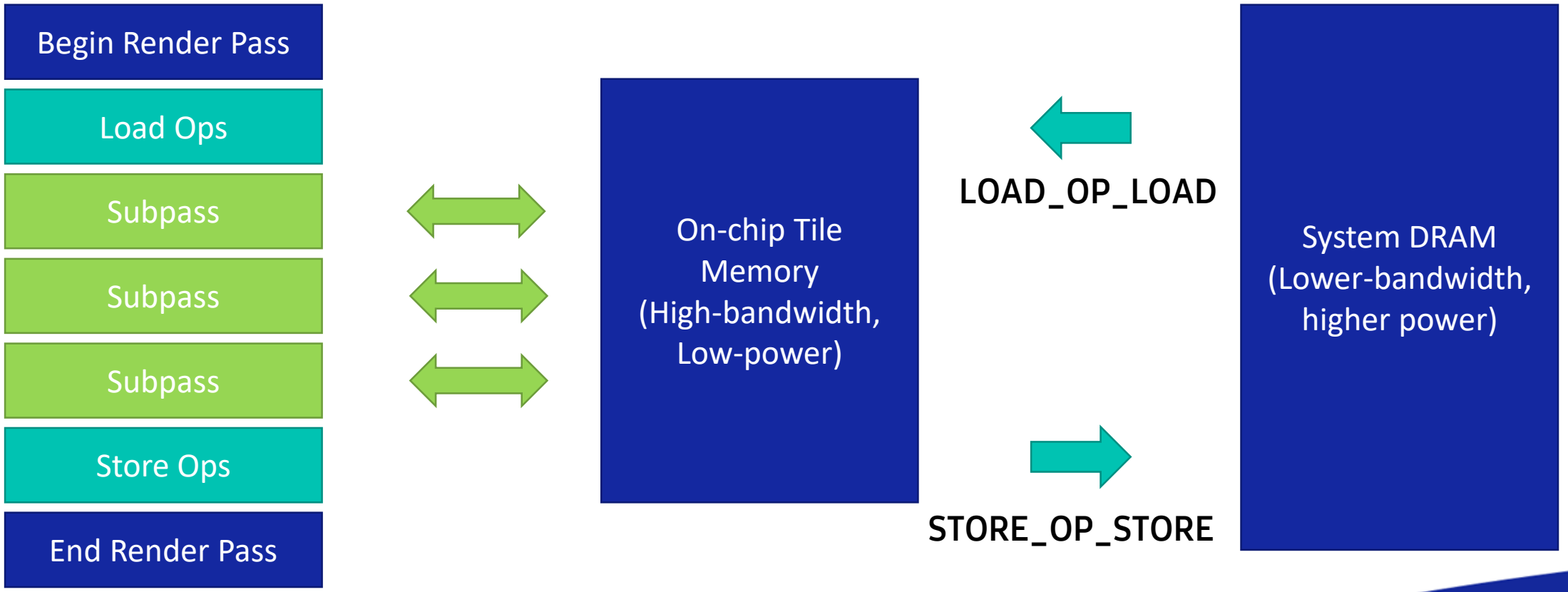
- How much precision do you really need?
- Do you actually use the all of the channels?
- Smaller image formats will consume less bandwidth, less power and generate less heat
- Consider R11G11B10 instead of R16G16B16A16 for render targets (32-bit vs. 64-bit)

FORZA STREET © TURN10 & MICROSOFT All Right Reserved.

Device	SM-N976B (Note10)	Format	R16G16B16A16F	R11G11B10F
Chipset	Mali-G76	FPS	53	60
Game Resolution	2048x1080p	Stability (%)	100	100
Game Setting	Ultra, Max 60 FPS	CPU Util. (%)	25.30	23.24
Clock Setting	Locked	GPU Util. (%)	98.65	97.23

RenderPasses and Subpasses

RenderPasses on Tile-based GPUs



Minimize the Number of RenderPasses

We Recommend:

- Using multiple subpasses over separate RenderPasses

```
vkCmdBeginRenderPass(C=Don't Care, D=Clear, S=Don't Care)
vkCmdDrawIndexed(4227, 1)
vkCmdDrawIndexed(4227, 1)

vkCmdDrawIndexed(36, 1)
vkCmdDrawIndexed(36, 1)
vkCmdEndRenderPass(C=Store, D=Store, S=Don't Care)
=> vkQueueSubmit(1)[0]: vkEndCommandBuffer( Baked Command Buffer 14072 )
=> vkQueueSubmit(1)[0]: vkBeginCommandBuffer( Baked Command Buffer 14073 )
vkCmdBeginRenderPass(C=Load, DS=Load)
vkCmdDrawIndexed(144, 1)
vkCmdDrawIndexed(144, 1)
vkCmdDrawIndexed(6, 241)
vkCmdDrawIndexed(144, 1)
vkCmdDrawIndexed(144, 1)
```

```
vkCmdBeginRenderPass(C=Don't Care, D=Clear, S=Don't Care)
vkCmdDrawIndexed(4227, 1)
vkCmdDrawIndexed(4227, 1)

vkCmdDrawIndexed(36, 1)
vkCmdDrawIndexed(36, 1)
vkCmdNextSubpass() => 1
vkCmdDrawIndexed(144, 1)
vkCmdDrawIndexed(6, 3)
vkCmdDrawIndexed(18, 16)
vkCmdDrawIndexed(6, 304)
vkCmdDrawIndexed(6, 103)
vkCmdDrawIndexed(96, 1)
vkCmdDrawIndexed(252, 1)
vkCmdDrawIndexed(96, 1)
```

Variant	FPS	FPS Stabillity	Power	CPU	GPU	Memory
Subpasses	49 (31-56)	97%	725 mA	36.46%	93.24%	787 MB
Separate Render Passes	41 (25-45)	98%	872 mA	32.62%	96.37%	785 MB

Avoid Empty RenderPasses

RHI Command	Original VK Calls		VK Calls with Pending Renderpass	
RHISetRenderTargets	Renderpass #01	vkCmdBeginRenderpass	Skipped	
No draw call				Pending..
RHISetRenderTargets		vkCmdEndRenderpass		
	Renderpass #02	vkCmdBeginRenderpass	Renderpass #01	vkCmdBeginRenderpass
RHIDrawIndexedPrimitive		vkCmdDrawIndexed		vkCmdDrawIndexed
RHIDrawIndexedPrimitive		vkCmdDrawIndexed		vkCmdDrawIndexed
RHIDrawIndexedPrimitive		vkCmdDrawIndexed		vkCmdDrawIndexed
RHIDrawIndexedPrimitive		vkCmdDrawIndexed		vkCmdDrawIndexed
RHISetRenderTargets		vkCmdEndRenderpass		vkCmdEndRenderpass
	Renderpass #03	vkCmdBeginRenderpass	Renderpass #02	vkCmdBeginRenderpass
RHIDrawIndexedPrimitive		vkCmdDrawIndexed		vkCmdDrawIndexed
RHIDrawIndexedPrimitive		vkCmdDrawIndexed		vkCmdDrawIndexed
RHIDrawIndexedPrimitive		vkCmdDrawIndexed		vkCmdDrawIndexed
		vkCmdEndRenderpass		vkCmdEndRenderpass

Clearing Framebuffer Attachments

We Recommend:

- When clearing attachments at the start of a render pass, use VK_ATTACHMENT_LOAD_OP_CLEAR.
- When clearing attachments within a sub-pass, use vkCmdClearAttachments.
- vkCmdClearColorImage and vkCmdClearDepthStencilImage can be used to clear outside of a render pass. These functions are the least efficient mechanism on tile-based GPU

Avoid Empty RenderPasses – cont.

RHI Command	Original VK Calls		VK Calls with Pending Clear / Renderpass	
	Renderpass #01	.....	Renderpass #01	.....
RHISetRenderTargetsAndClear		vkCmdEndRenderpass		vkCmdEndRenderpass
		vkCmdClearImageXXX		Save vkCmdClearImageXXX into PendingClearList
	Renderpass #02	vkCmdBeginRenderpass		
RHISetRenderTargetsAndClear		vkCmdEndRenderpass		
		vkCmdClearImageXXX		Save vkCmdClearImageXXX into PendingClearList
RHISetRenderTargets	Renderpass #03	vkCmdBeginRenderpass	Renderpass #02	
		vkCmdEndRenderpass		Flush PendingList
RHISetRenderTargets	Renderpass #04	vkCmdBeginRenderpass		vkCmdBeginRenderpass (**LOAD-CLEAR**)
RHIDrawIndexedPrimitive		vkCmdDrawIndexed		vkCmdDrawIndexed
RHIDrawIndexedPrimitive		vkCmdDrawIndexed		vkCmdDrawIndexed
RHIDrawIndexedPrimitive		vkCmdDrawIndexed		vkCmdDrawIndexed
RHIDrawIndexedPrimitive		vkCmdDrawIndexed		vkCmdDrawIndexed
RHIDrawIndexedPrimitive		vkCmdDrawIndexed		vkCmdDrawIndexed
		vkCmdEndRenderpass		vkCmdEndRenderpass

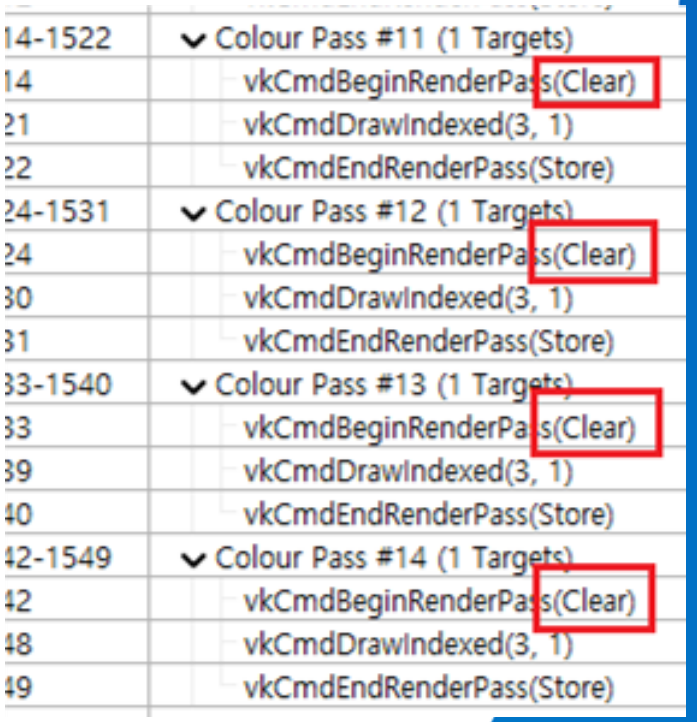
RenderPasses and Load OPs

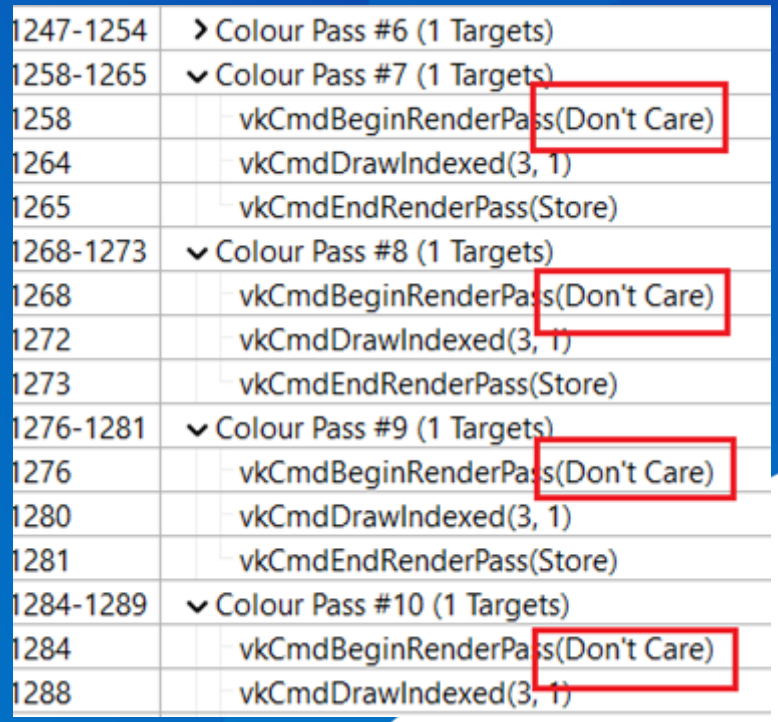
LOAD_OP_DONT_CARE	Fastest option, but attachment content undefined and may be modified. Implementation dependent
LOAD_OP_CLEAR	Clears to a uniform value. Fast on TBDRs, slower on Xclipse (IMR). Prefer this over shader-based clears
LOAD_OP_LOAD	Preserves attachment content, accessible in renderpass. Slowest option, but unavoidable if you need the load
LOAD_OP_NONE	Preserves attachment content, inaccessible in renderpass - passthrough

Subpasses and Load OPs

Fullscreen Post-processing passes:

FRCPassPostProcessBloomSetupES2
FRCPassPostProcessBloomDownES2
FRCPassPostProcessBloomUpES2
FRCPassPostProcessSunMaskES2
FRCPassPostProcessSunAlphaES2
FRCPassPostProcessSunBlurES2
FRCPassPostProcessSunMergeES2
FRCPassPostProcessTonemap





TRAHA, © NEXON Korea Corporation & MOAI GAMES All Rights Reserved.

Device	SM-G970F (S10)	Format	GLES	Vulkan
Chipset	Mali-G76	FPS	30	30
Game Resolution	1440x720p	Stability (%)	100	100
Game Setting	High, Capped 30 FPS	CPU Util. (%)	8.69	8.95
Clock Setting	Locked	GPU Util. (%)	80.32	77.46

RenderPases and Store OPs

Store Operation	Effect
STORE_OP_STORE	Render area content written to memory
STORE_OP_DONT_CARE	Destructive discard, even if you didn't write to the attachment
STORE_OP_NONE	Non-destructive

Shader Precision

Shader Precision

We Recommend:

- Use reduced precision to improve performance and reduce power consumption.
- Beware of compilers promoting precision. Test on lots of devices to catch shader precision artefacts early.
- Beware of rendering errors that are hidden on some devices by reduced precision.

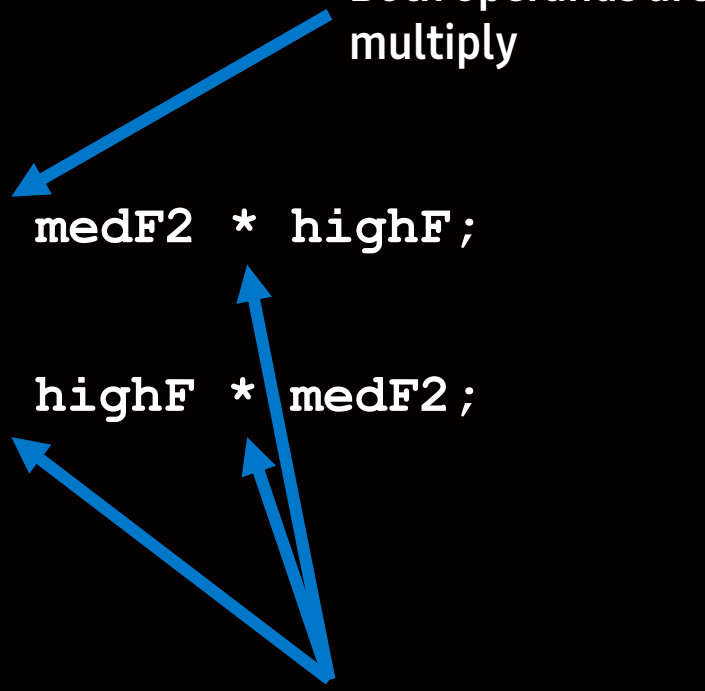
https://developer.samsung.com/remotetestlab

Shader Precision

GLSL	SPIR-V	Native ISA
mediump/lowp qualifiers	OpDecorate RelaxedPrecision	Float16 native instructions (hopefully)
Complex rules on implicit precision promotion	Compiler optimisations removing qualifiers	Compiler may chose to ignore the reduced precision

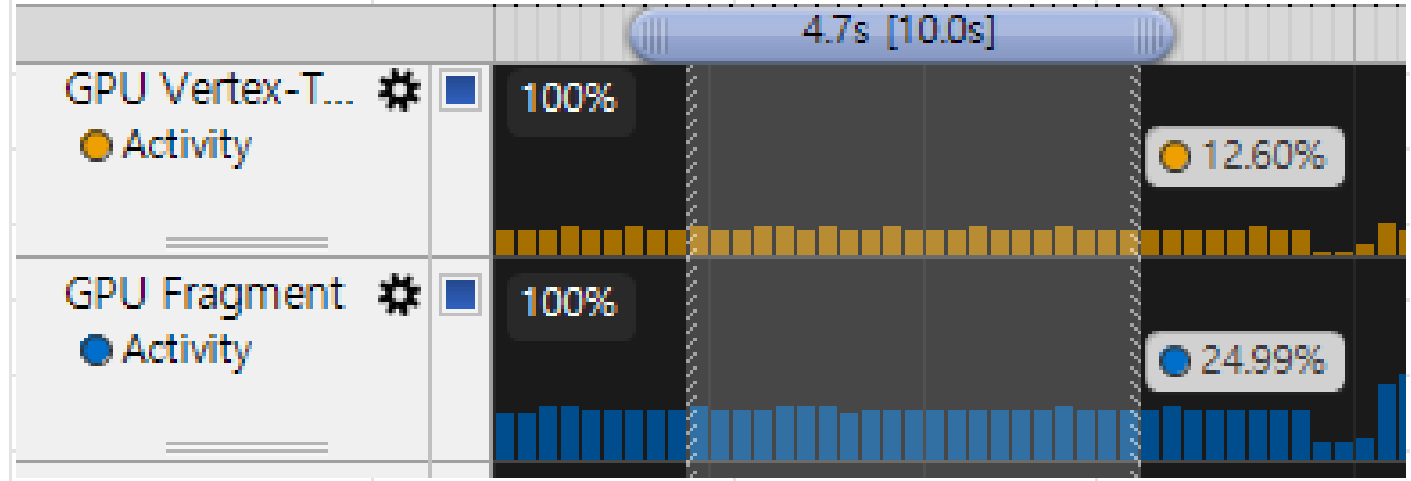
Shader Precision - GLSL

```
highp float highF;
mediump float medF1;
mediump float medF2;

float res1 = medF1 * medF2 * highF;

float res1 = medF1 * highF * medF2;
```

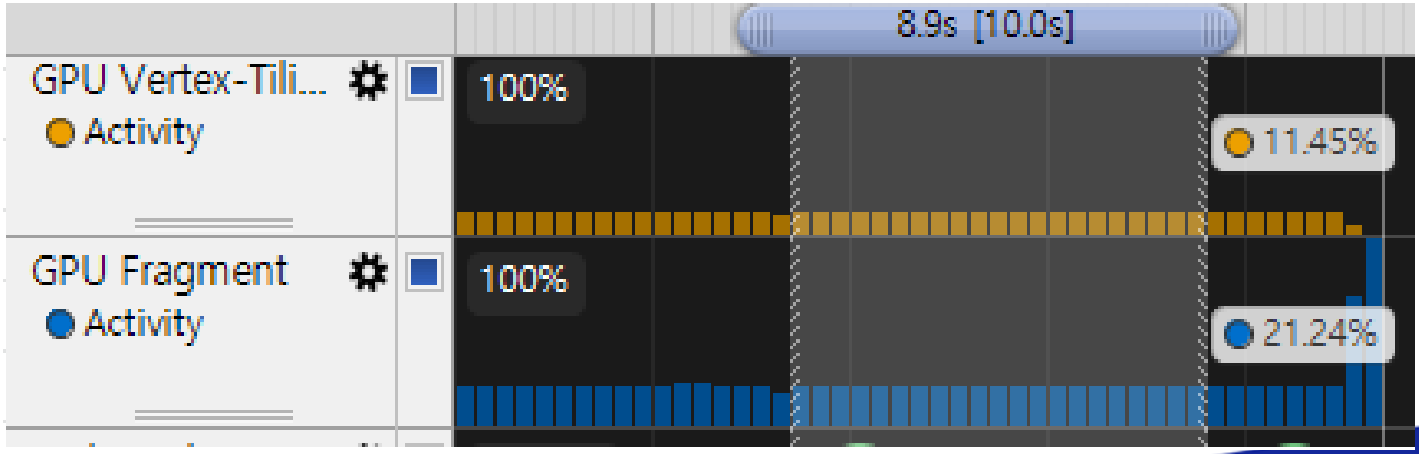


Shader Precision – Interface Mismatches

Mismatched Attributes
Vertex Shader: 12.6%
Fragment Shader: 24.99%



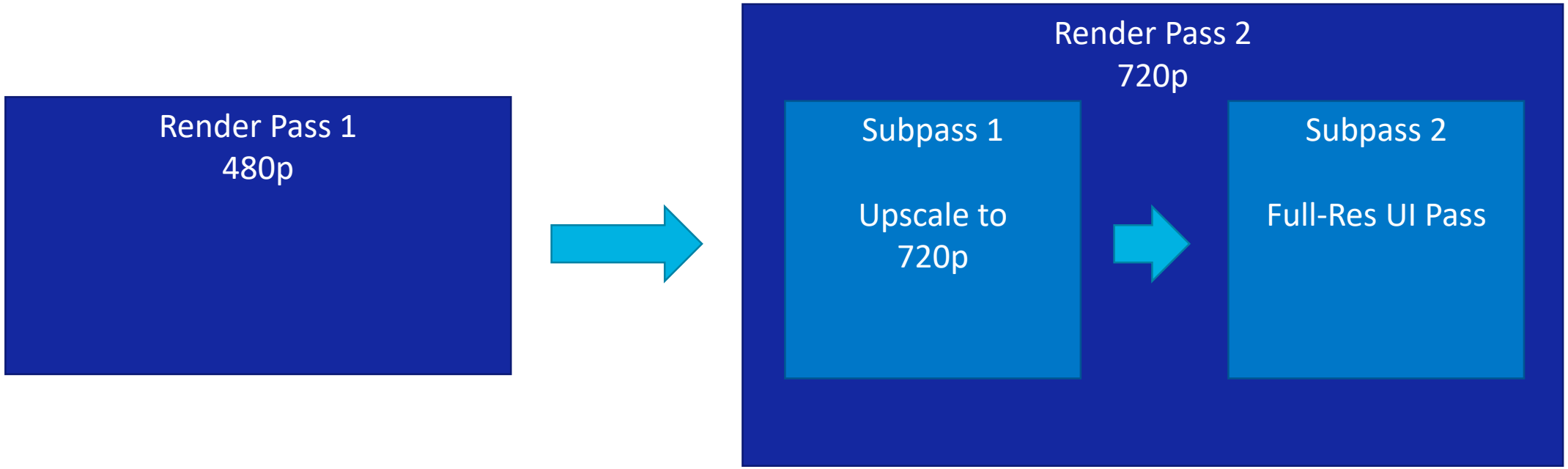
Consistent Attributes
Vertex Shader: 11.45%
Fragment Shader: 21.24%



Efficient Upscaling

SAMSUNG
GPU DevTech

Efficient render pass upscaling



- Only when limited by fragment shading throughput!
- Prefer textured full-screen quad upscale over vkCmdBlitImage
- Consider AMD FidelityFX Super Resolution

Efficient render pass upscaling

483p

720p Upscale

RCAS 720p
(AMD FidelityFX)

https://www.amd.com/en/technologies/fidelityfx-super-resolution

AMD FidelityFX FSR

Adreno *(740, 1.8GHz / 401MHz)*

	FPS (average)	FPS Stability	FPS Hitching	CPU usage: App (average)	GPU usage (average)
Performance (50%)	59.90	100%	0	21.11%	72.07%
Balanced (59%)	59.93	100%	0	21.21%	77.73%
Quality (67%)	59.93	100%	0	21.26%	81.23%
Ultra Quality (77%)	59.87	100%	0	21.61%	87.28%
Original (100%)	59.90	100%	0	21.49%	95.46%

Xclipse *(920, 1.7GHz / 500MHz)*

	FPS (average)	FPS Stability	FPS Hitching	CPU usage: App (average)	GPU usage (average)
Original (100%)	37.94	100%	0	17.32%	98.98%
Ultra Quality (77%)	49.64	100%	0	18.27%	94.51%
Quality (67%)	50.46	100%	0	19.05%	81.72%
Balanced (59%)	56.38	100%	0	21.26%	82.21%
Performance (50%)	58.21	100%	0	21.57%	74.04%

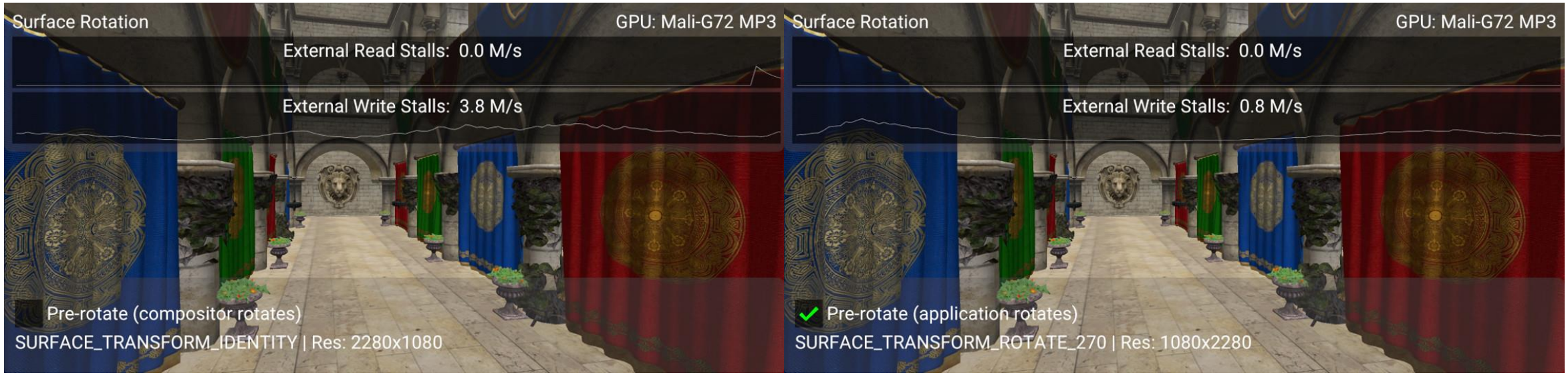
Swapchains and Presentation

Swapchains and Presentation

We Recommend:

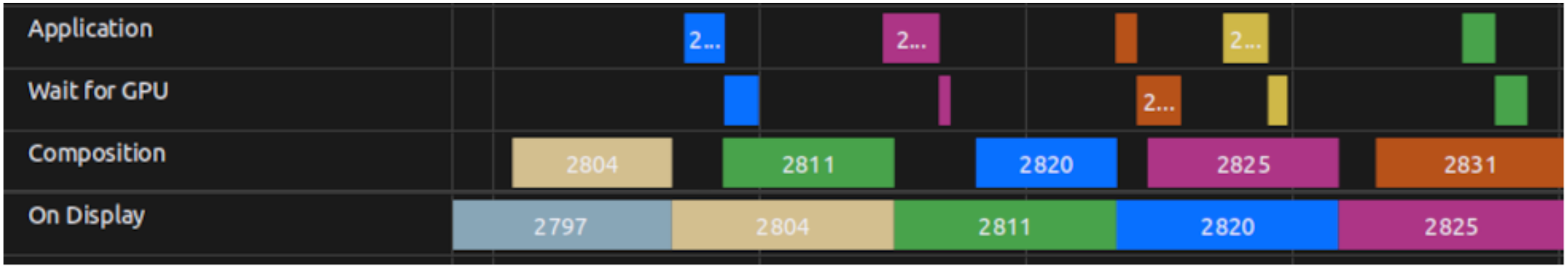
- Use the VK_PRESENT_MODE_FIFO_KHR presentation mode
- Set your swapchain's minImageCount to 3
- Use a separate thread for your vkQueuePresent calls

Swapchain Pre-Rotation



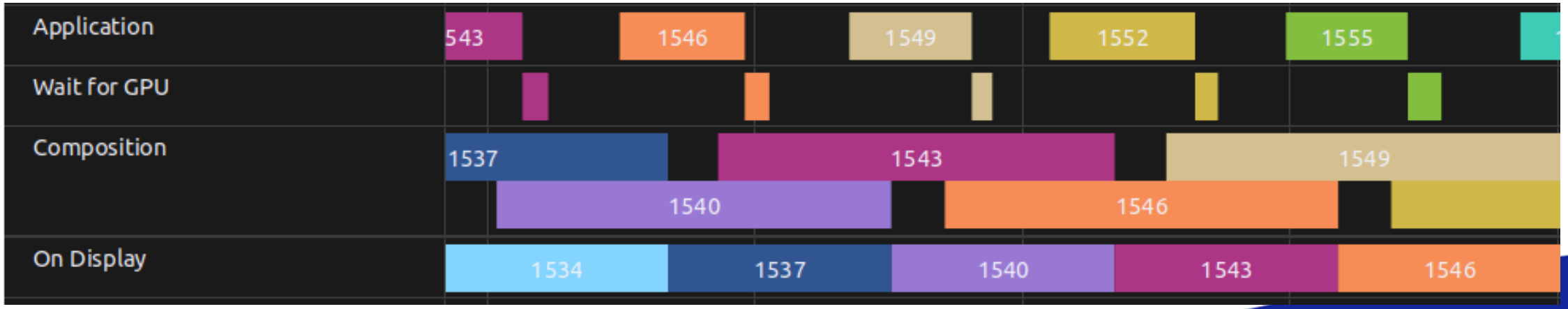
https://github.com/KhronosGroup/Vulkan-Samples/tree/main/samples/performance/surface_rotation

Swapchain Pre-Rotation



Identity/Pre-rotated

GPU Compositor Rotated



Swapchain Pre-Rotation

We recommend:

- Applications should handle pre-rotation themselves
- On rotation:
  - Recreate your swapchain
  - Update your model-view matrix
  - (Optionally) Correct your fragment shader derivatives

- https://developer.android.com/games/optimize/vulkan-prerotation
- https://registry.khronos.org/vulkan/specs/1.3-extensions/man/html/VK_QCOM_render_pass_transform.html

Takeaways

- Mobile optimization is not just about frame rates
- Reducing the work required by your application will reduce the risk of throttling
- Understand your GPU architecture, particularly around RenderPass behaviour and memory load/stores
- Limit yourself to what you actually need (minimally sized formats, reduced precision)
- Profile your app!

Resources

- Vulkan Samples
  - https://github.com/KhronosGroup/Vulkan-Samples
- Samsung Best Practice
  - https://developer.samsung.com/galaxy-gamedev/resources/articles/usage.html
- ARM Best Practice
  - https://developer.arm.com/documentation/101897/0301
- QCOM Best Practice
  - https://developer.qualcomm.com/sites/default/files/docs/adreno-gpu/snapdragon-game-toolkit/gdg/gpu/best_practices.html

Questions?