A GUIDED TOUR OF BLACKREEF: RENDERING TECHNOLOGIES IN "DEATHLOOP"

GILLES MARION, ARKANE STUDIOS LYON

LOU KRAMER, AMD

GDC

AMD GPUOpen

GILLES MARION, LEAD GRAPHICS PROGRAMMER

- Since 2011 : Arkane Studios Lyon
  - Engine Programmer on Dishonored (Unreal Engine, PS3, Xbox 360, PC)
  - Lead Graphics Programmer on all Arkane Studios Lyon subsequent projects
- All developed with the VOID ENGINE™

DISHONORED 2

DISHONORED — DEATH OF THE OUTSIDER™ —

"DEATHLOOP"™

WHAT WILL BE COVERED IN THIS GDC TALK?

- Quick history of the Void Engine and supported APIs
- Evolution of Graphics Features between Dishonored 2 & Deathloop
- Raytracing implementation

[AMD Official Use Only]

WHAT IS THE VOID ENGINE ?

- Fork of idSoftware's idTech 5 which they used for
- Renamed Void Engine in 2014

VOID ENGINE GRAPHICS APIS OVER THE YEARS

- idTech 5, **Rage**, 2012 :
  - OpenGL on PC, DX9 on Xbox 360 and Sony's proprietary API on PS3
  - Editor & game merged in a single exe

- Void Engine 1.0, **Dishonored 2** & **Death of The Outsider**, 2016-2017
  - Shipped versions : DX11 on PC and Xbox One, Sony's proprietary API on PS4
  - Experimental support for Vulkan & DX12 on PC, DX12 on Xbox One, but not in a shippable state
  - We even had an AMD Mantle version working during development

VOID ENGINE GRAPHICS APIS OVER THE YEARS

- Void Engine 1.5, Deathloop, 2021
- Shipped versions : DX12 on PC, Sony's proprietary API on PS5
- Early during dev, engine & game in a functional state on 7 platforms, 6 APIs
- Most quickly abandoned
- DX11 still supported for PC internally for a long time for stability reasons
- DX12 version eventually ran better than the DX11 one, DX11 support abandoned

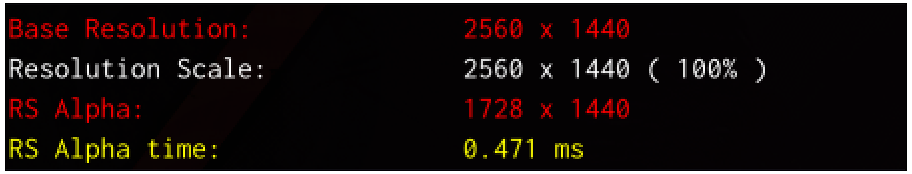
VOID ENGINE 1.0 VS 1.5 GRAPHICS FEATURES

VoidEngine 1.0	VoidEngine 1.5
Forward rendering	Deferred rendering
Ward BRDF	GGX BRDF

SDR Rendering
  Filmic tonemapping
  Color correction with cubemaps
	SDR & HDR rendering
	  ACES tonemapping
	  Parametric color correction

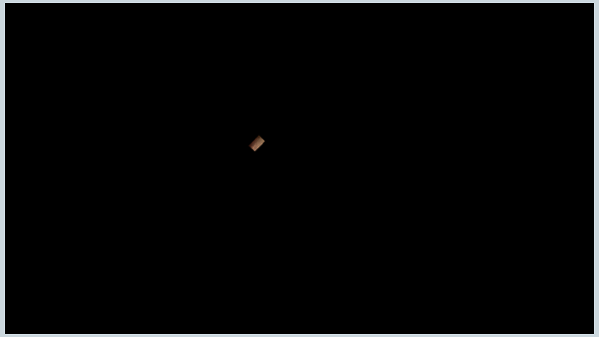
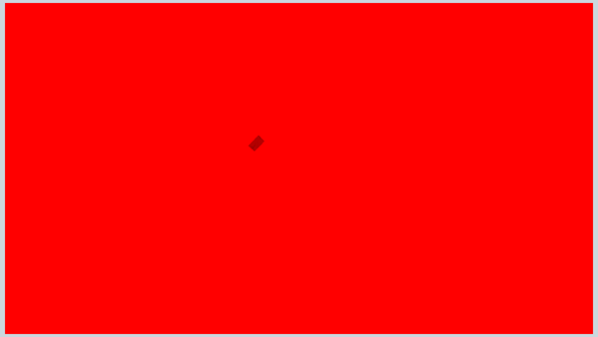
VOID ENGINE 1.0 VS 1.5 GRAPHICS FEATURES

VoidEngine 1.5
Lit Particles Decoupled transparent surfaces resolution



VOID ENGINE 1.0 VS 1.5 GRAPHICS FEATURES

VoidEngine 1.5 : Dual Resolution Order-Independent Translucency

	Accumulation Buffer	Opacity Buffer	Composited result
Low-resolution RTs			
Hi-resolution RTs			

VOID ENGINE 1.0 VS 1.5 GRAPHICS FEATURES

VoidEngine 1.5 : Deferred Decals	
With Decals	Without Decals
	

VOID ENGINE 1.0 VS 1.5 GRAPHICS FEATURES

VoidEngine 1.5 : Deferred Decals	
Dynamic Decals	Static Decals

[AMD Official Use Only]

VOID ENGINE 1.0 VS 1.5 GRAPHICS FEATURES

VoidEngine 1.5 : Deferred Decals

Snow Meshes only	Procedural snow layer applied
	

[AMD Official Use Only]

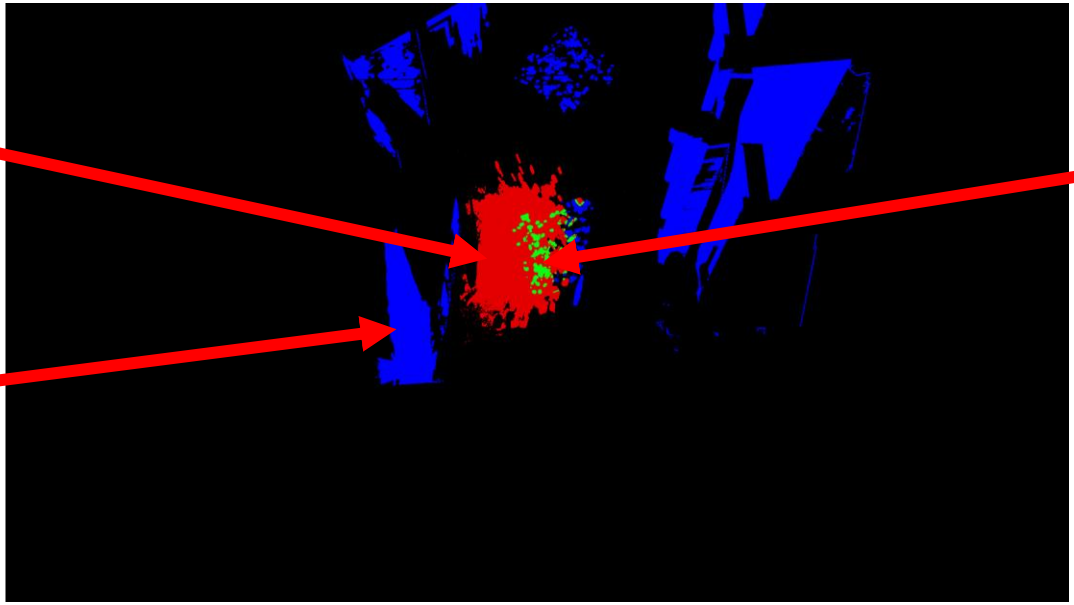
VOID ENGINE 1.0 VS 1.5 GRAPHICS FEATURES

Decals interactions with snow

- Need to store decal type
- Gbuffer : read 32bpp to use 2 bits, bad choice
- Stencil is better, writes are faster, and we only read 8 bpp

Dynamic Decal

Static Decal



Both types of Decal

VOID ENGINE 1.0 VS 1.5 GRAPHICS FEATURES

Decals interactions with snow

Gbuffer Shader pseudo-code :

```
read stencil
if decal present : read decal gbuffer
if snow not fully opaque : ApplyDeferredDecalStatic
ApplySnow
ApplyDeferredDecalDynamic
```

[AMD Official Use Only]

VOID ENGINE 1.0 VS 1.5 GRAPHICS FEATURES

VoidEngine 1.0	VoidEngine 1.5
Reflection probes	Reflections probes + SSR


SSR OFF
KILL
KILL
KILL
x2
MACO
IMPERCEPTIBLE
12 15
GHOST MODE


SSR ON
KILL
KILL
KILL
MACO
x2
IMPERCEPTIBLE
12
15
GHOST MODE

VOID ENGINE 1.0 VS 1.5 GRAPHICS FEATURES

VoidEngine 1.0	VoidEngine 1.5
Static Skybox	Procedural Sky
Ambient Occlusion • Custom AO • NVIDIA HBAO	Ambient Occlusion • NVIDIA HBAO • AMD FidelityFX CACAO • Raytraced AO
Cascaded Shadow Maps	Raytraced sun shadows

VOID ENGINE 1.0 VS 1.5 GRAPHICS FEATURES

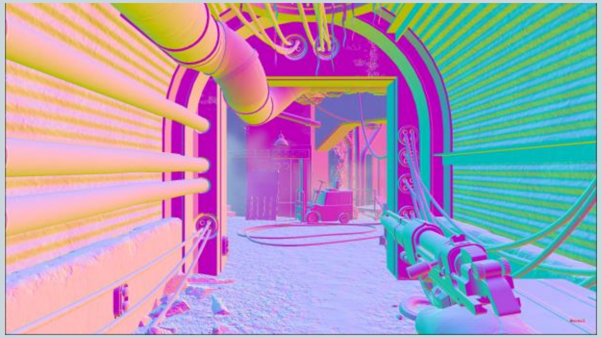
VoidEngine 1.0	VoidEngine 1.5
Anti-Aliasing • FXAA • Temporal AA	Anti-Aliasing • FXAA • Improved Temporal AA • NVIDIA DLSS • AMD FidelityFX Super Resolution 2.0
Fake sharpening through TAA	Sharpening : • Custom (cheap) • AMD FidelityFX CAS / RCAS

WHY SWITCHING TO DEFERRED RENDERING ?

Forward Rendering	Deferred Rendering
Perf issues • Main shader overly complicated, up to 128 VGPRs • Memory bandwidth • VS/PS interpolants	Simpler shaders (split computations) • Less VGPRs • Less interpolants • Still memory bandwidth bound • Easier to optimize • Usage of Async Compute
Maintenance • Tens of thousands of shader permutations	• A lot less shader permutations
No normal buffer	Full Gbuffer (3 RTs + 1 optional) Visual debugging for artists Raytracing easier to implement

WHY SWITCHING TO DEFERRED RENDERING ?

Normals

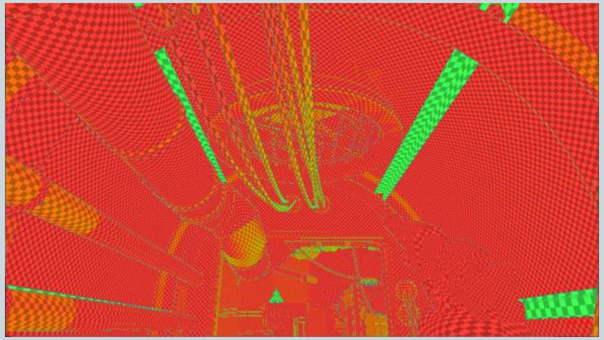


Roughness

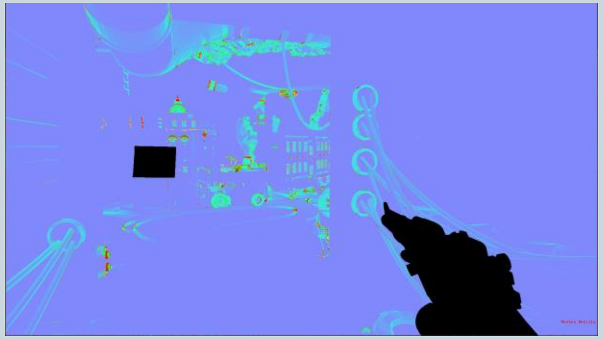
Albedo

Metallicity

Texel Ratio



Vertex Density



RAYTRACING IMPLEMENTATION

Why did we do it ?

- Availability on next-gen consoles
- Wider support on PC

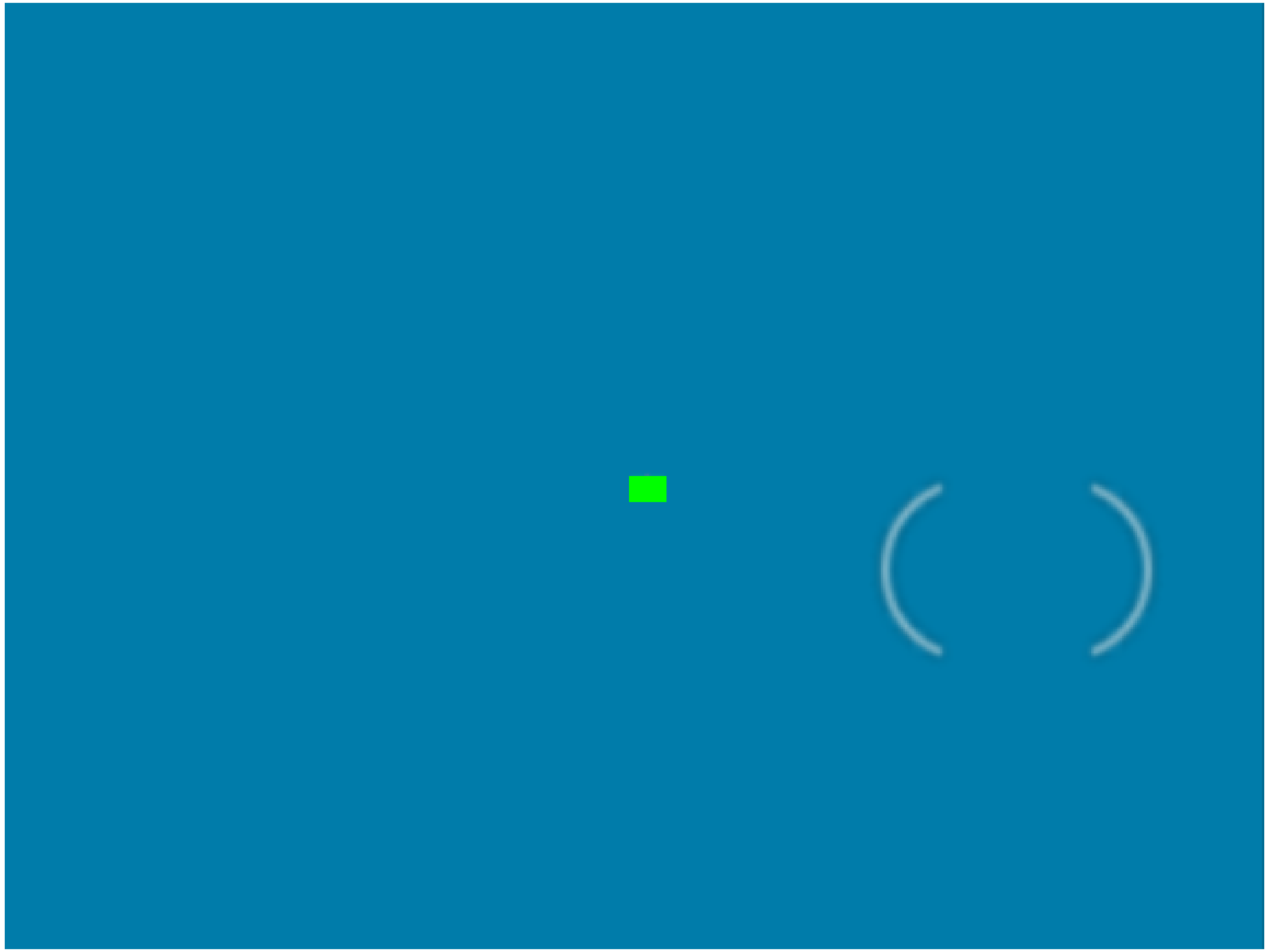
RAYTRACING IMPLEMENTATION

First prototype

- PC only
- Build acceleration structures : only static opaque geometry
- No BLAS & TLAS per-frame updates
- No texture & material management
- Cast our first rays

[AMD Official Use Only]

RAYTRACING IMPLEMENTATION



RAYTRACING IMPLEMENTATION

Step 2

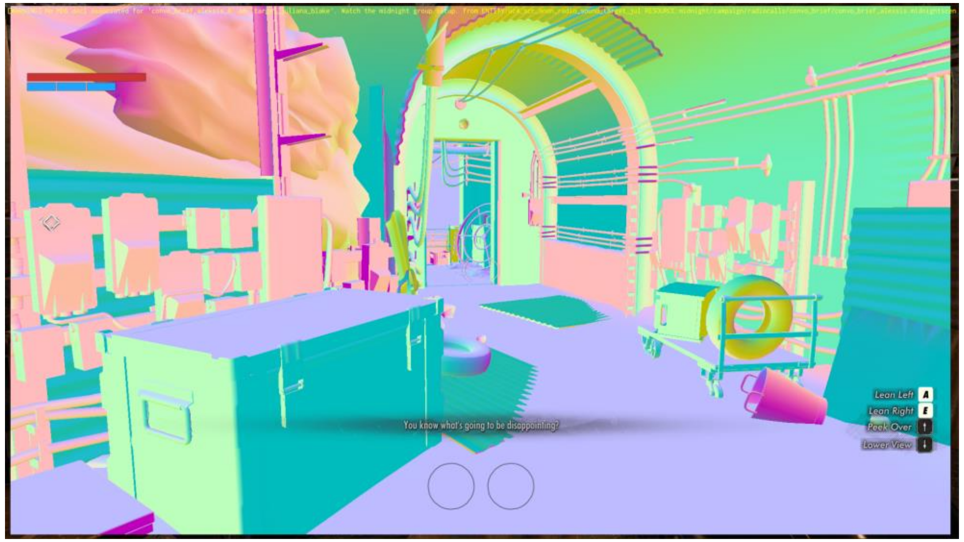
- Same scene, viewed from the player camera

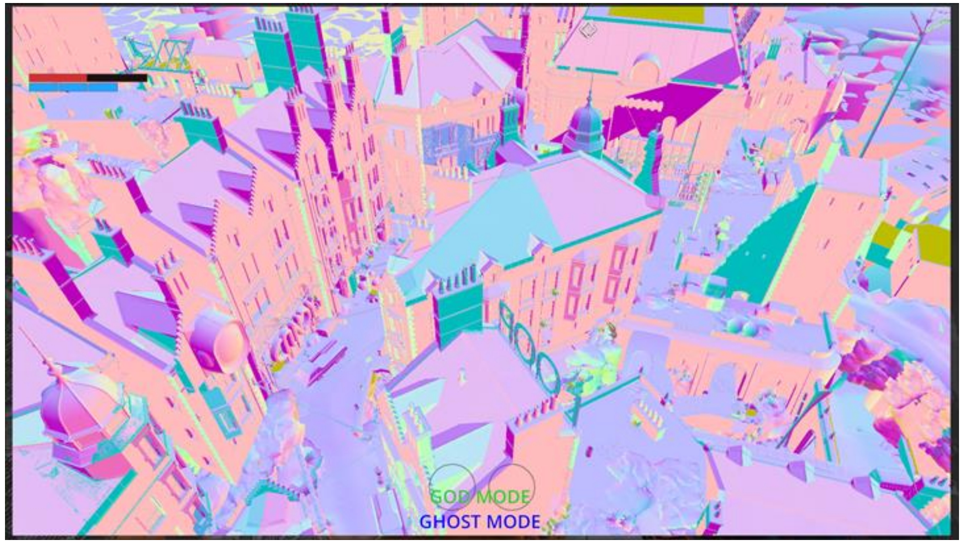
[AMD Official Use Only]

RAYTRACING IMPLEMENTATION

Step 3

- A real game level, flat shaded with the triangles normal in object-space





RAYTRACING IMPLEMENTATION

First difficulties

- Texture access
- Not a bindless renderer, we had to hack it

RAYTRACING IMPLEMENTATION

Reachable targets

- Ambient Occlusion
- Sun Shadows

RAYTRACING IMPLEMENTATION

Denoiser

- We tried a few ones, not good enough
- Main issue : ghosting
- The right one : AMD FidelityFX Denoiser
- Open source, usable on consoles

RAYTRACING IMPLEMENTATION

Last missing features

- Console port
- Moving objects
- Skinned meshes
- Alphatest, 2 issues
- Renderer still not bindless
- Texture streaming

RAYTRACING IMPLEMENTATION

Optimizations

- Multi-thread BLAS update code
- Optional half-res AO
- More usage of async compute
  - BLAS & TLAS updates (rebuild/refit)
  - DispatchRays

[AMD Official Use Only]

ARKANE LYON IS HIRING

We still have a lot of various positions opened for our next project.

It is exciting !

BIBLIOGRAPHY

- The Devil is in the details, Tiago Sousa, Jean geffroy, Siggraph 2016
  https://advances.realtimerendering.com/s2016/Siggraph2016_idTech6.pdf

- Weighted Blended Order-Independent Transparency, Morgan McGuire, Louis Bavoil, JCGT 2013
  https://jcgt.org/published/0002/02/09/

A GUIDED TOUR OF BLACKREEF: RENDERING TECHNOLOGIES IN "DEATHLOOP"

GILLES MARION, ARKANE STUDIOS LYON

LOU KRAMER, AMD

GDC

OPTIMIZATIONS & FEATURE INTEGRATION

[AMD Official Use Only]

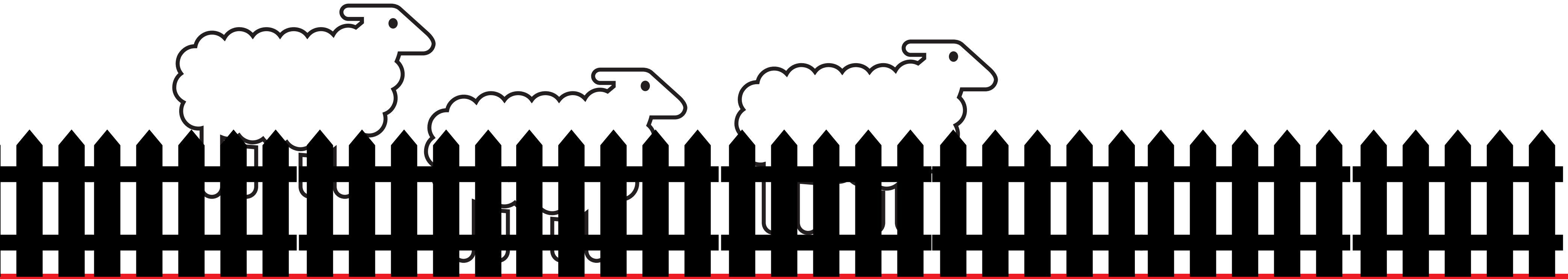
OPTIMIZATIONS

Barriers

- General performance recommendations.
- Analyzing barriers.
- Barrier optimization example in Deathloop.

Optimizing output write patterns

- Breakdown of a one-line optimization done in Arkane‘s Scattering Light Fog shader.

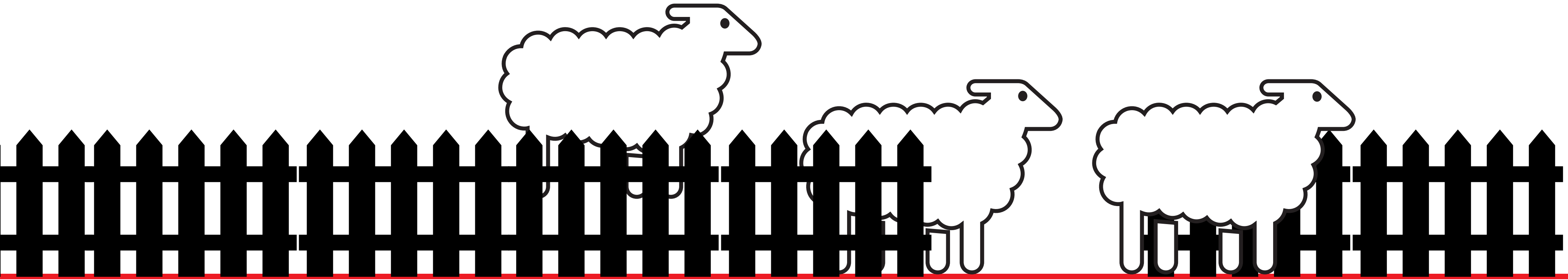


[AMD Official Use Only]

BARRIERS

- Barriers are not a new topic in regards to performance and optimizations.
- The reason is simple:
  - Poorly placed and/or non-optimal configured barriers can hurt performance a lot.
  - Missing barriers or incorrectly configured barriers can cause severe stability and correctness issues.

- Stability and Correctness comes before performance, so usually developers start with a more conservative approach to get things right and running.
- This also means that there might be room for improvements when looking at the barriers.

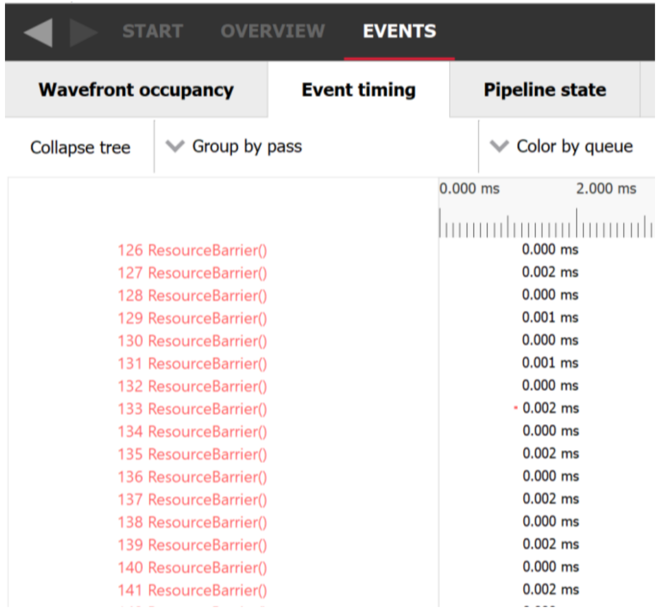


BARRIERS – AMD RDNA™2 TECHNOLOGY PERFORMANCE GUIDE

- Minimize the number of barriers used per frame.
  - Barriers can drain the GPU of work.
  - Don’t issue read to read barriers. Transition the resource into the correct state the first time.

- Batch groups of barriers into a single call to reduce overhead of barriers.
  - Creates fewer calls into the driver.
  - Allows the driver to remove redundant operations.

- Avoid GENERAL / COMMON layouts unless required.
  - Always use the optimized state for your usage.



Easy to spot with our Radeon™ GPU Profiler (RGP). Also gets you rid of redundant operations ☺.

BARRIERS – AMD RDNA™2 TECHNOLOGY PERFORMANCE GUIDE

- Minimize the number of barriers used per frame.
  - Barriers can drain the GPU of work.
  - Don’t issue read to read barriers. Transition the resource into the correct state the first time.

- Batch groups of barriers into a single call to reduce overhead of barriers.
  - Creates fewer calls into the driver.
  - Allows the driver to remove redundant operations.

- Avoid GENERAL / COMMON layouts unless required.
  - Always use the optimized state for your usage.

Transitions to an un-optimized state can cause unnecessary cache invalidations, cache flushes or even decompressions. These are all visible in RGP.

[AMD Official Use Only]

BARRIERS – WHAT TO SEE IN RGP

Overview → Barriers

- List of all barriers in the frame:

Duration	Drain Time	Stalls	Depth/Stencil Decompress	HiZ Range Resummarize	DCC Decompress	FMask Decompress	Fast Clear Eliminate	Init Mask RAM	Invalidated	Flushed	Type
0.633 ms	0.564 ms	FULL				✓			K L0 L1 CB DB	CB DB	APP
0.608 ms	0.449 ms	FULL				✓			K L0 L1 CB DB	CB DB	APP
1.534 ms	0.409 ms	FULL				✓			K L0 L1 CB DB	CB DB	APP
0.475 ms	0.465 ms	FULL				✓			K L0 L1 CB DB	CB DB	APP
0.549 ms	0.546 ms	FULL				✓			K L0 L1 CB DB	CB DB	APP
0.468 ms	0.468 ms	FULL				✓			K L0 L1 CB DB	CB DB	APP
0.061 ms	0.005 ms	FULL			✓				K L0 L1 CB DB	CB DB	APP
0.012 ms	0.007 ms	FULL			✓				K L0 L1 CB DB	CB DB	APP
0.014 ms	0.005 ms	FULL			✓				K L0 L1 CB DB	CB DB	APP
0.007 ms	0.005 ms	FULL			✓				K L0 L1 CB DB	CB DB	APP
0.025 ms	0.004 ms	FULL			✓				K L0 L1 CB DB	CB DB	APP
0.053 ms	0.048 ms	FULL			✓				K L0 L1 CB DB	CB DB	APP
0.232 ms	0.120 ms	FULL			✓				K L0 L1 CB DB	CB DB	APP
0.045 ms	0.040 ms	FULL			✓				K L0 L1 CB DB	CB DB	APP
0.096 ms	0.035 ms	FULL			✓				K L0 L1 CB DB	CB DB	APP
0.001 ms	0.001 ms	VS PS CS							K L0 L1		APP
0.001 ms	0.000 ms	VS PS CS							K L0 L1		APP
0.005 ms	0.001 ms	FULL							K L0 L1 L2 CB DB	L2 CB DB	APP

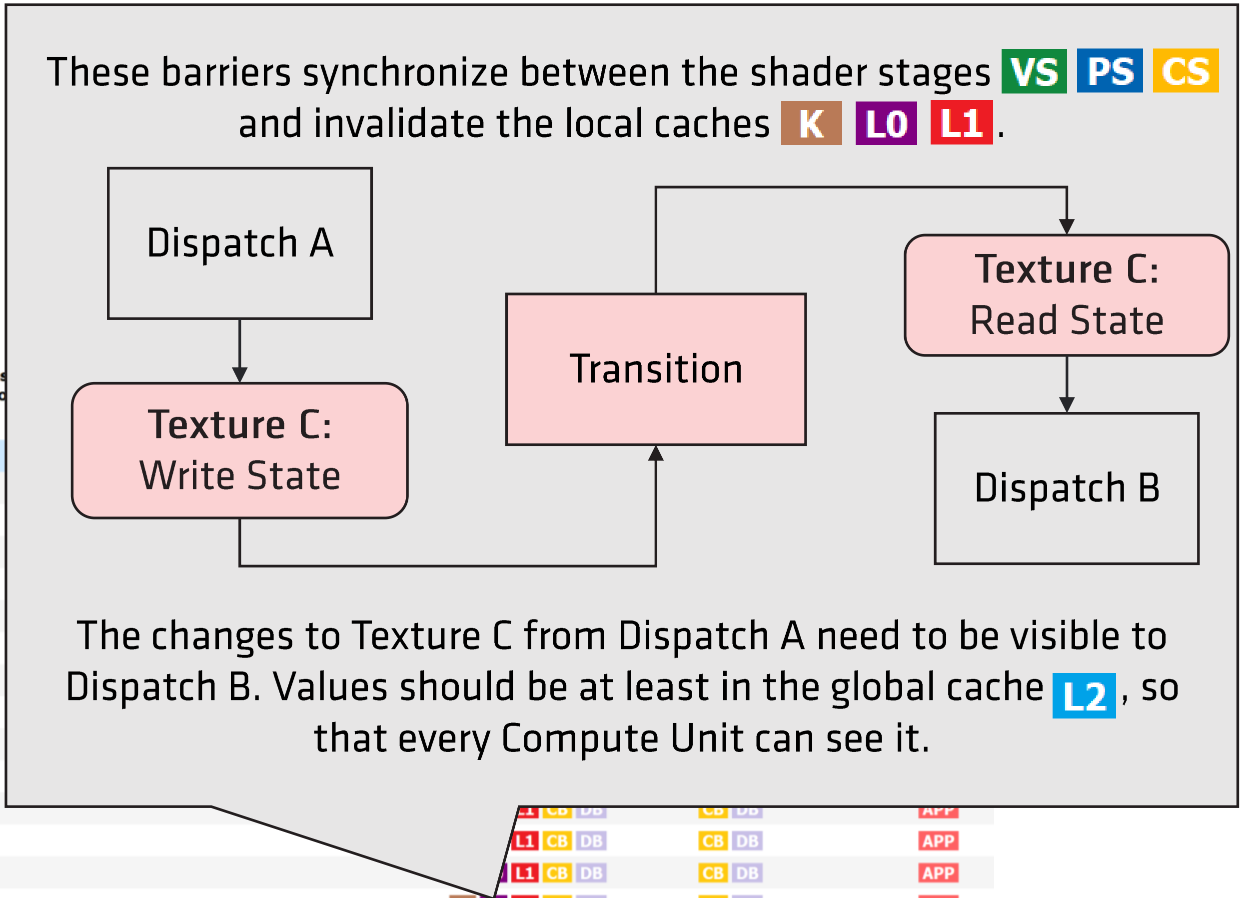
- Indicates the effect of the barrier besides waiting for the previous commands to finish.

BARRIERS – WHAT TO SEE IN RGP

Overview → Barriers

- List of all barriers in the frame:

Duration	Drain Time	Stalls	Depth/Stencil Decompress	HiZ Range Resummarize	DCC Decompress	FMas Deco			
0.633 ms	0.564 ms	FULL							
0.608 ms	0.449 ms	FULL							
1.534 ms	0.409 ms	FULL							
0.475 ms	0.465 ms	FULL							
0.549 ms	0.546 ms	FULL							
0.468 ms	0.468 ms	FULL							
0.061 ms	0.005 ms	FULL			✓				
0.012 ms	0.007 ms	FULL			✓				
0.014 ms	0.005 ms	FULL			✓				
0.007 ms	0.005 ms	FULL			✓				
0.025 ms	0.004 ms	FULL			✓				
0.053 ms	0.048 ms	FULL			✓				
0.232 ms	0.120 ms	FULL			✓		L1 CB DB	CB DB	APP
0.045 ms	0.040 ms	FULL			✓		L1 CB DB	CB DB	APP
0.096 ms	0.035 ms	FULL			✓		K L0 L1 CB DB	CB DB	APP
0.001 ms	0.001 ms	VS PS CS					K L0 L1		APP
0.001 ms	0.000 ms	VS PS CS					K L0 L1		APP
0.005 ms	0.001 ms	FULL					K L0 L1 L2 CB DB	L2 CB DB	APP



- Indicates the effect of the barrier besides waiting for the previous commands to finish.

[AMD Official Use Only]

BARRIERS – WHAT TO SEE IN RGP

Overview → Barriers

- List of all barriers in the frame:

Duration	Drain Time	Stalls	Depth/Stencil Decompress	HiZ Range Resummarize	DCC Decompress	FMask Decompress	Fast Clear Eliminate	Init Mask RAM	Invalidated	Flushed	Type
0.633 ms	0.564 ms	FULL				✓			K L0 L1 CB DB	CB DB	APP
0.608 ms	0.449 ms	FULL				✓			K L0 L1 CB DB	CB DB	APP
1.534 ms	0.409 ms	FULL				✓			K L0 L1 CB DB	CB DB	APP
0.475 ms	0.465 ms	FULL				✓			K L0 L1 CB DB	CB DB	APP
0.549 ms	0.546 ms	FULL				✓			K L0 L1 CB DB	CB DB	APP
0.468 ms	0.468 ms	FULL				✓			K L0 L1 CB DB	CB DB	APP
0.061 ms	0.005 ms	FULL			✓				K L0 L1 CB DB	CB DB	APP
0.012 ms	0.007 ms	FULL			✓				K L0 L1 CB DB	CB DB	APP
0.014 ms	0.005 ms	FULL			✓						
0.007 ms	0.005 ms	FULL			✓						
0.025 ms	0.004 ms	FULL			✓						
0.053 ms	0.048 ms	FULL			✓						
0.232 ms	0.120 ms	FULL			✓						
0.045 ms	0.040 ms	FULL			✓						
0.096 ms	0.035 ms	FULL			✓						
0.001 ms	0.001 ms	VS PS CS							K L0 L1		APP
0.001 ms	0.000 ms	VS PS CS							K L0 L1		APP
0.005 ms	0.001 ms	FULL							K L0 L1 L2 CB DB	L2 CB DB	APP

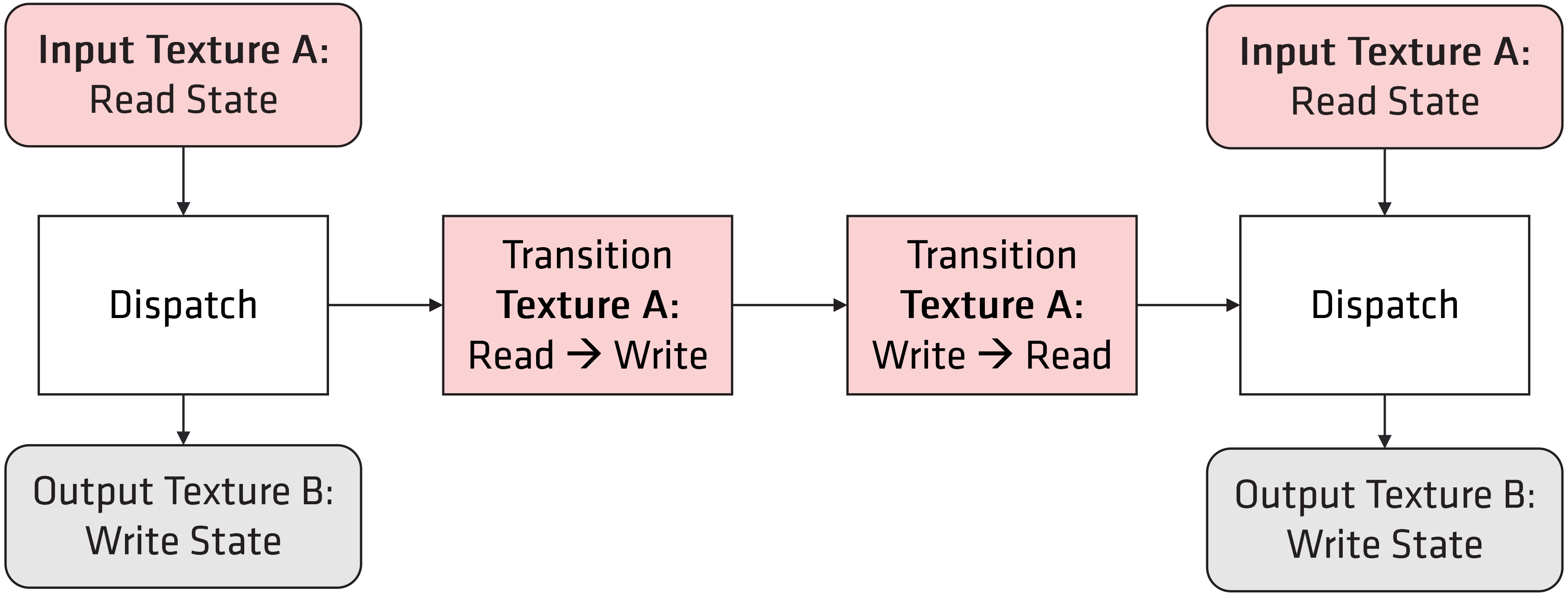
Depending on the next possible command, invalidating K L0 L1 might not be enough.

- Indicates the effect of the barrier besides waiting for the previous commands to finish.

[AMD Official Use Only]

BARRIERS IN DEATHLOOP

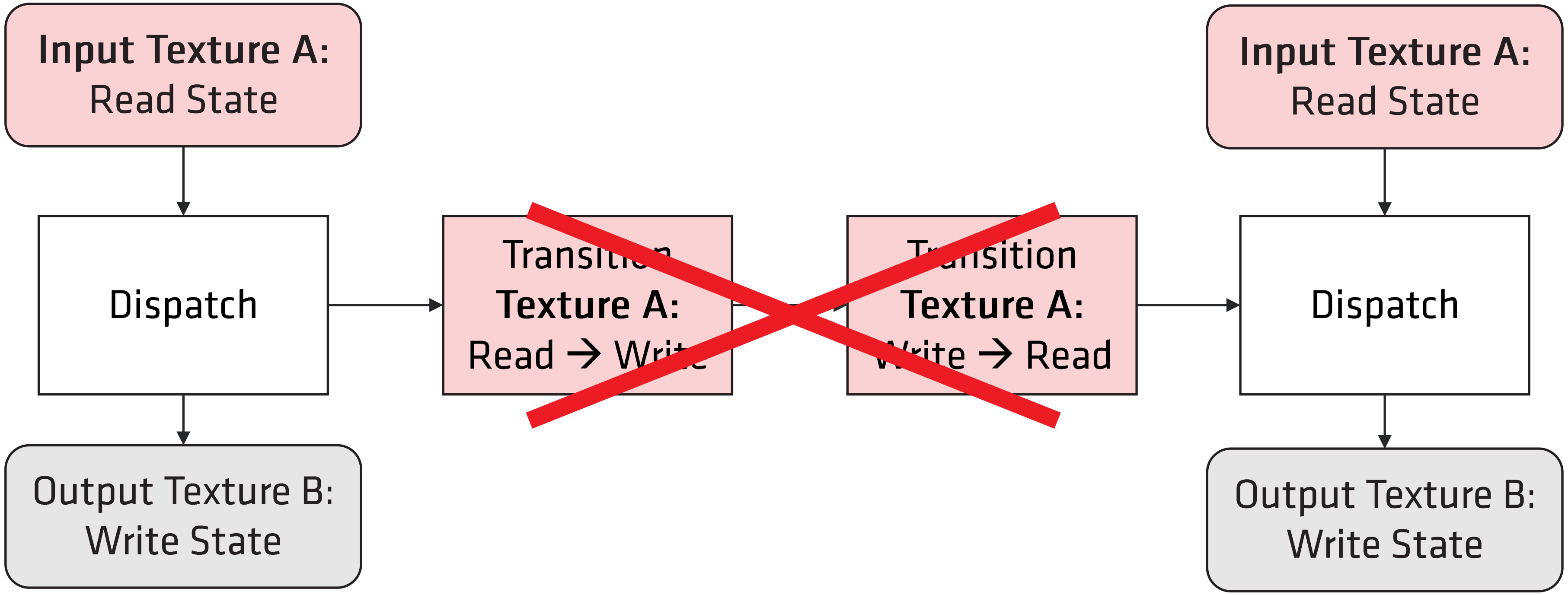
- Deathloop uses a conservative automatic barrier generation system.
  - It's robust: All necessary transitions are automatically generated.
  - It's convenient: No manual placement and configuration of barriers is required.
- However, due to the conservative approach, unnecessary barriers can be issued.
- In fact, sometimes it issues back-to-back barriers that transition the resource to a state back and forth:



[AMD Official Use Only]

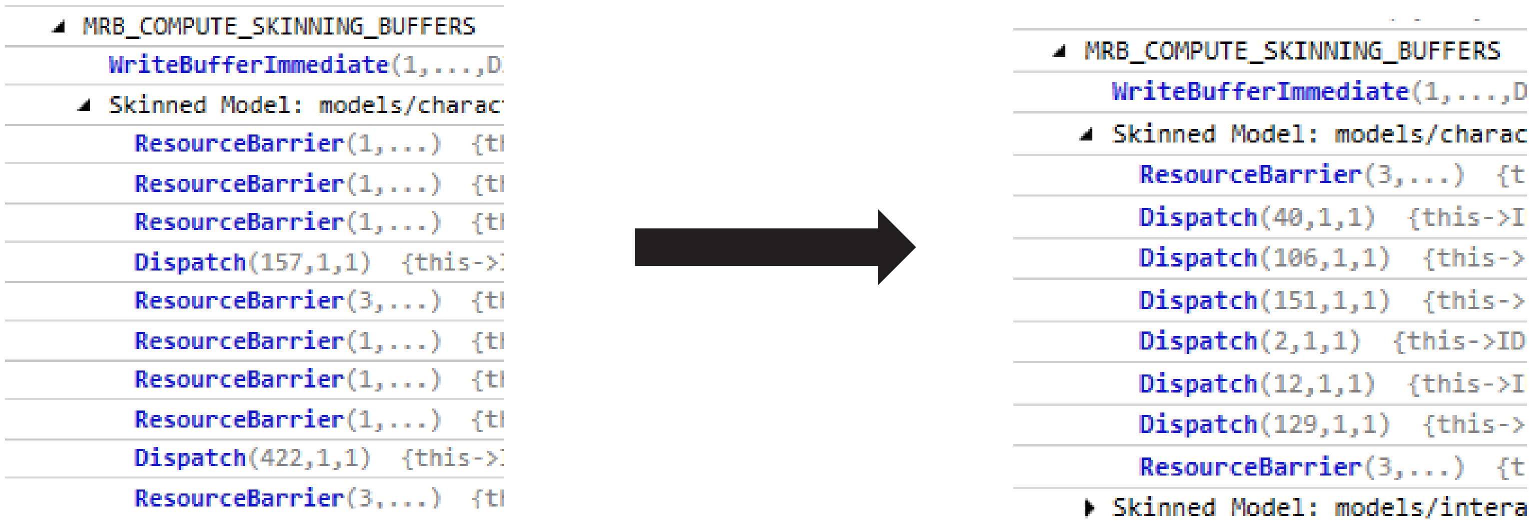
BARRIERS IN DEATHLOOP

- Deathloop uses a conservative automatic barrier generation system.
  - It's robust: All necessary transitions are automatically generated.
  - It's convenient: No manual placement and configuration of barriers is required.
- However, due to the conservative approach, unnecessary barriers can be issued.
- In fact, sometimes it issues back-to-back barriers that transition the resource to a state back and forth:



BARRIERS IN DEATHLOOP

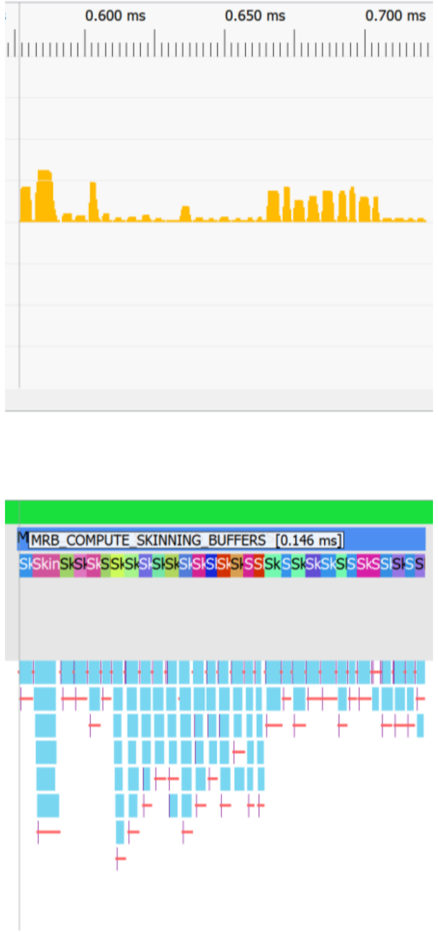
- One case where this happened is the Compute Skinning Buffers pass.
- There were barriers around every dispatch call, even if the next dispatch was not dependent on the previous work.
- The barriers caused the buffers to switch states back and forth for no good reason.
- A typical frame had 100-200 barriers in this function!
- A manual barrier management codepath solved the issue.



[AMD Official Use Only]

BARRIERS IN DEATHLOOP

This change improved the performance of the Compute Skinning Buffers pass up to ~60%![1]



Independent dispatches can overlap.

[1]RGP traces, before and after, captured on AMD Radeon™ RX 6900 XT, driver 22.2.1. See backup slide for full system specs.

OPTIMIZING OUTPUT WRITE PATTERN

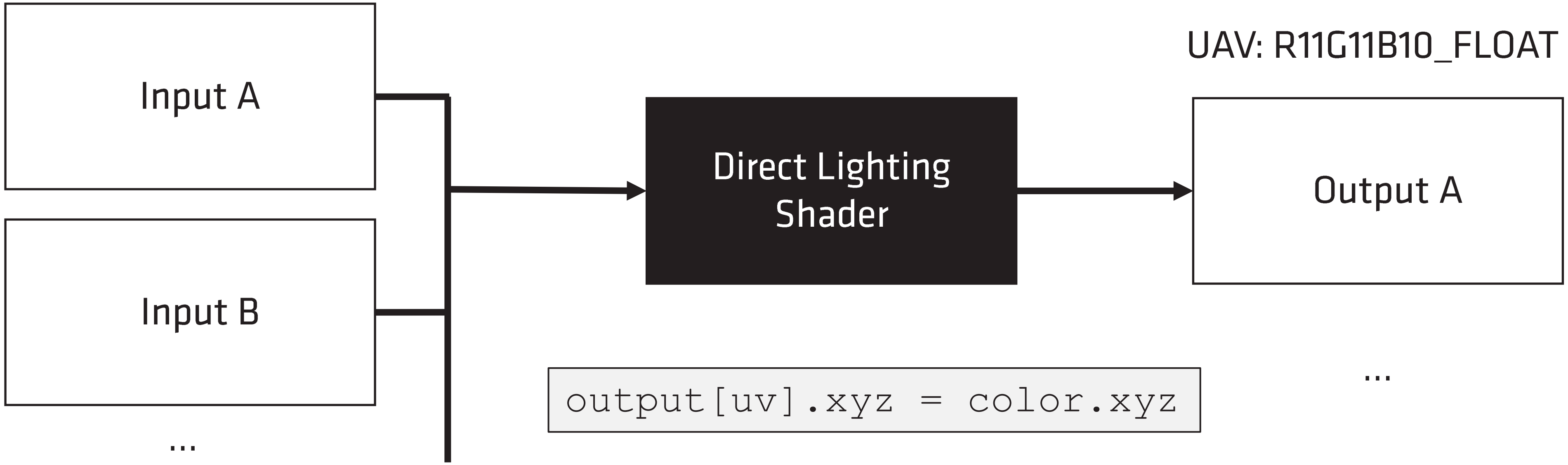
- One line shader change in the Scattering Light Fog shader.
- Affected the pattern the output was written.

[AMD Official Use Only]

SCATTERING LIGHT FOG SHADER

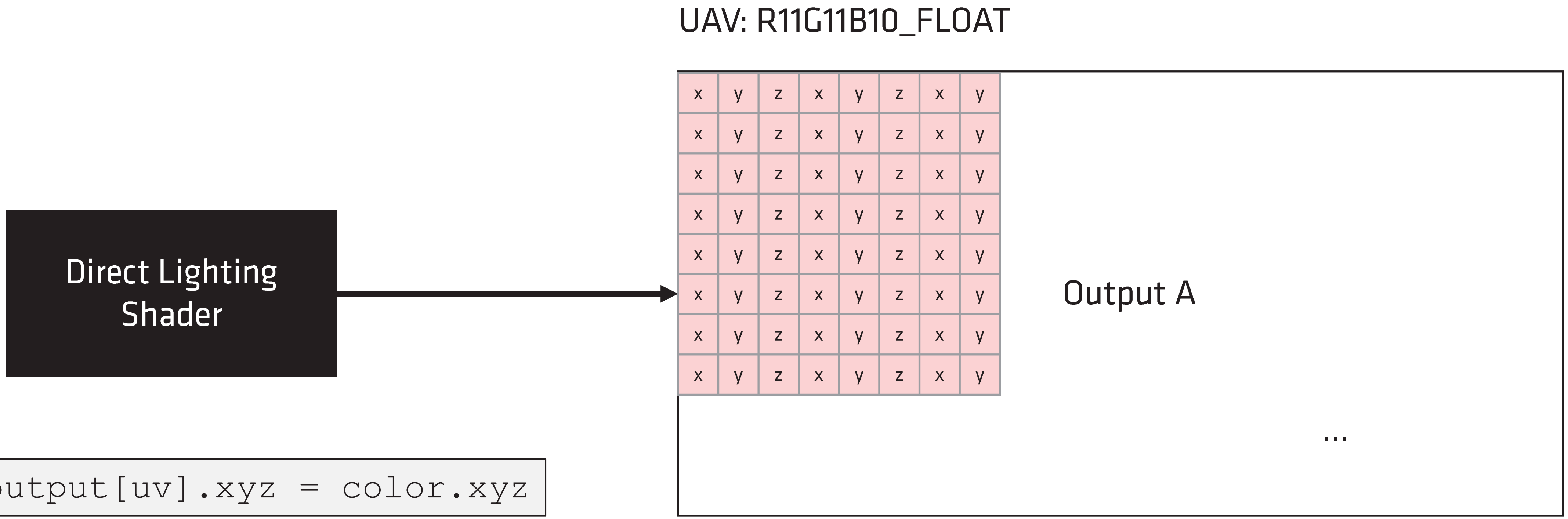
- Computes the light that gets scattered by fog.
- Impact of this shader depends on scene.
- The shader has a lot to compute in scenes with a lot of fog ...
- ... but not so much in scenes with little or no fog ☺.

- Not too different from the Direct Lighting shader:



[AMD Official Use Only]

SCATTERING LIGHT FOG SHADER

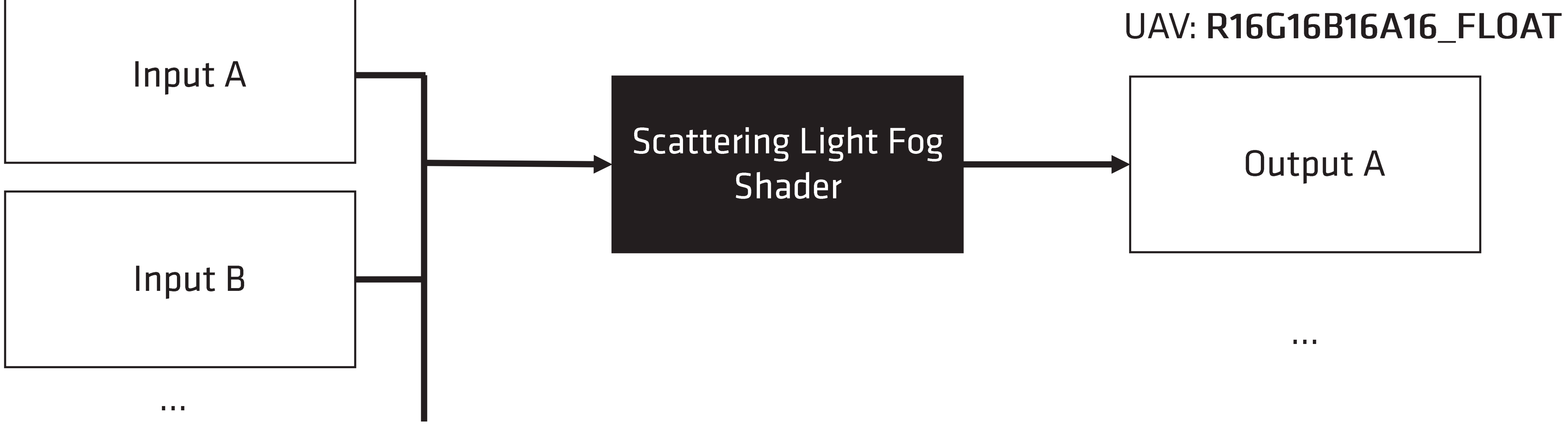



```
output[uv].xyz = color.xyz
```

SCATTERING LIGHT FOG SHADER

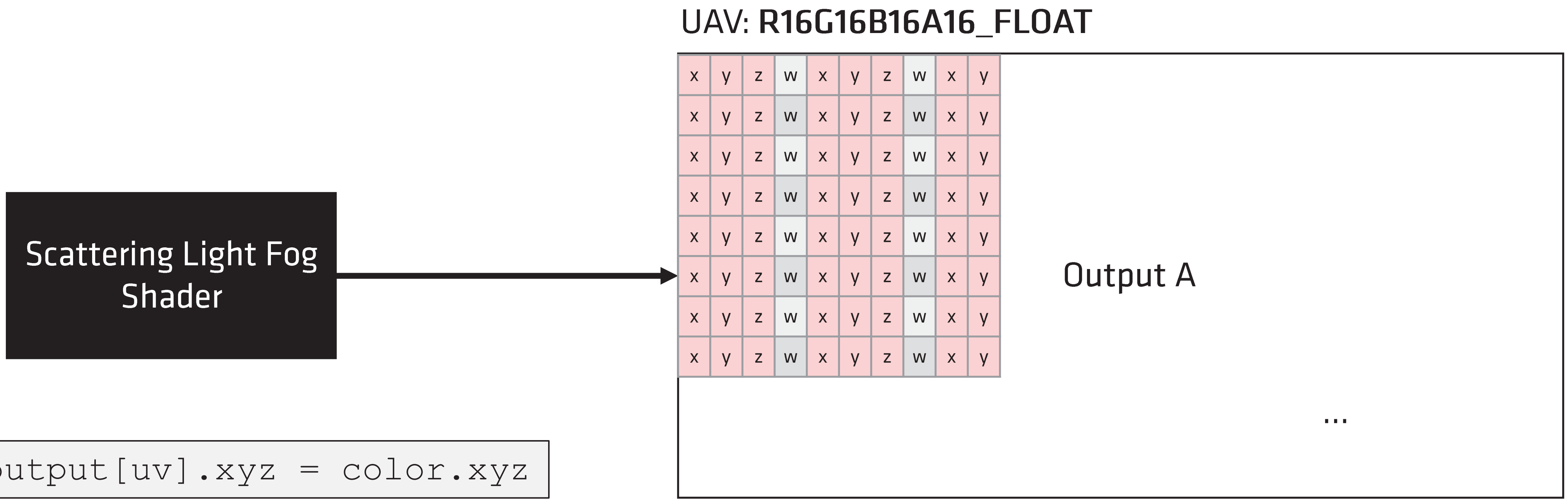
- The fog shader has a different format for the output image though.
- Instead of only 3 channels, it has now 4 channels.
- But the Scattering Light Fog shader also just computes RGB values, as the Direct Lighting shader does.
- The output was still written to only 3 channels:

```
output[uv].xyz = color.xyz
```



[AMD Official Use Only]

SCATTERING LIGHT FOG SHADER




```
output[uv].xyz = color.xyz
```

This goes against our recommendation:

To help maximize bandwidth in compute shaders,
write to images in coalesced 256-byte blocks per wave.

PARTIAL WRITES AND COMPRESSION

- If the data is uncompressed, the writes can be simply masked for partial writes.
- This does not work for compressed data.
- If the data is compressed, a coalesced 256-byte block needs to be first decompressed, and then compressed again to preserve the untouched channels.

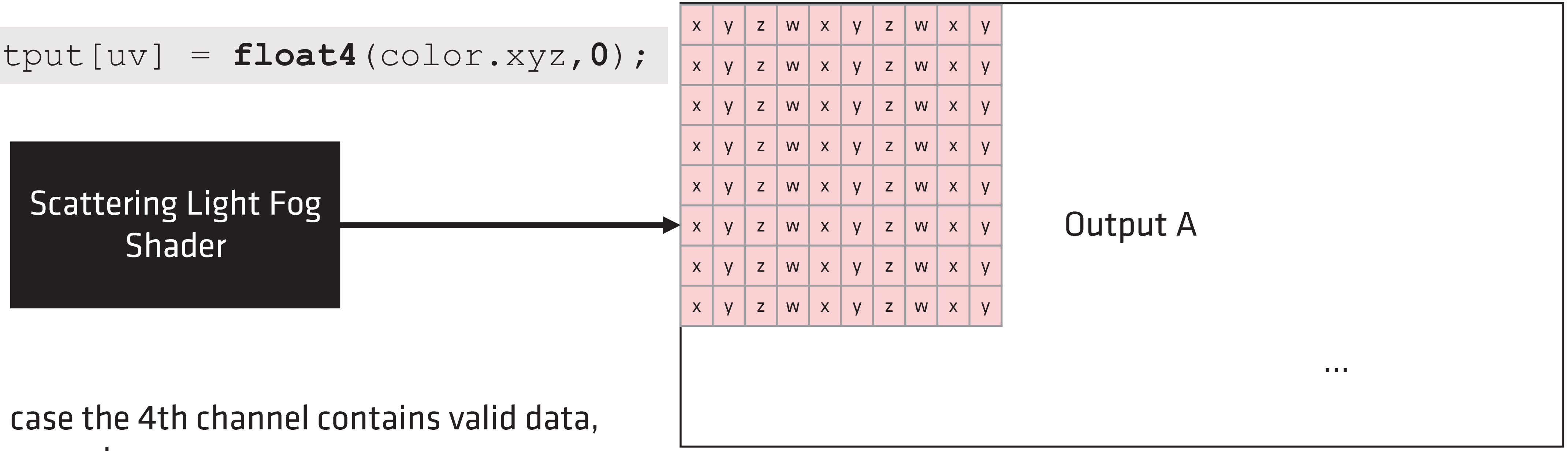
- Therefore, to efficiently use compression, it’s best to fully overwrite the underlying data.
- If a coalesced 256-byte block is written, it will be written directly as compressed block.
- There is no decompression step, because no channels need to be preserved.

[AMD Official Use Only]

SCATTERING LIGHT FOG SHADER

Fortunately, the 4th channel was unused!
So, we could just do:

```
output[uv] = float4(color.xyz,0);
```



In case the 4th channel contains valid data, you can do:

```
output[uv] = float4(color.xyz,output[uv].w);
```

OPTIMIZING OUTPUT WRITE PATTERNS

- The observed speed up of the Scattering Light Fog shader was up to ~30%.[1]
- In scenes with a lot of fog, this was quite significant.

- Take aways when writing to UAVs in compute shaders:
  - If possible, write to images in coalesced 256-byte blocks per wave.
  - Rule of thumb is to have 8x8 thread group write 8x8 blocks of pixels.
  - Write to all channels.

- If one channel needs to be preserved, test if it‘s more performant to just read and write it again:

```
output[uv] = float4(color.xyz,output[uv].w);
```

[1]RX 6900 XT driver 22.2.1: shader execution time was compared with original `output[uv].xyz = color.xyz` and modified `output[uv] = float4(color.xyz,0);`
See backup slide for full system specs.

AMD

FidelityFX
Super Resolution 2.0

Attend the following session for more details!

This presentation will focus on the integration part specific to Deathloop.

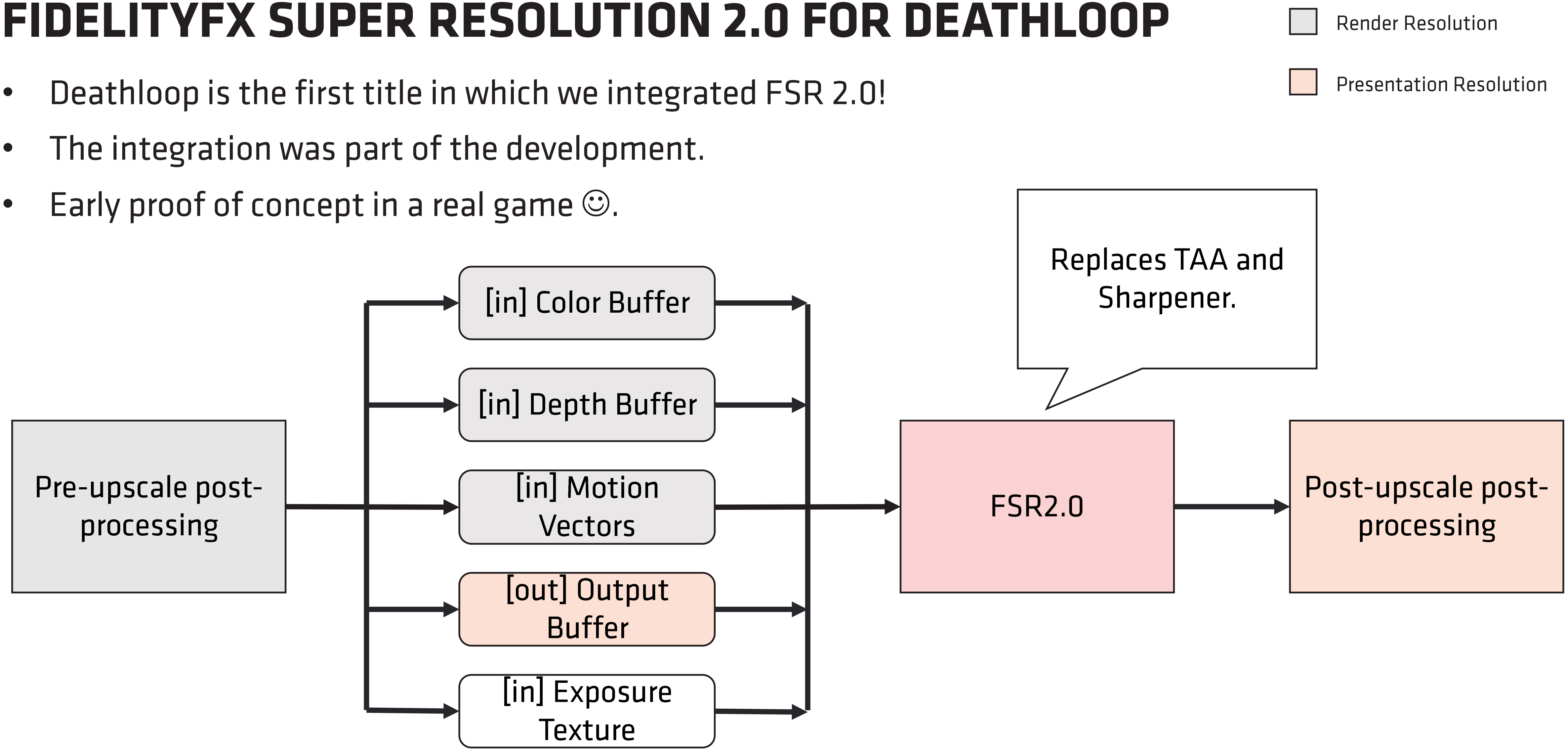
FSR 2.0 IN ACTION

Native 4K, TAA + Sharpener enabled

FidelityFX Super Resolution 2.0
Quality
1440p → 4K

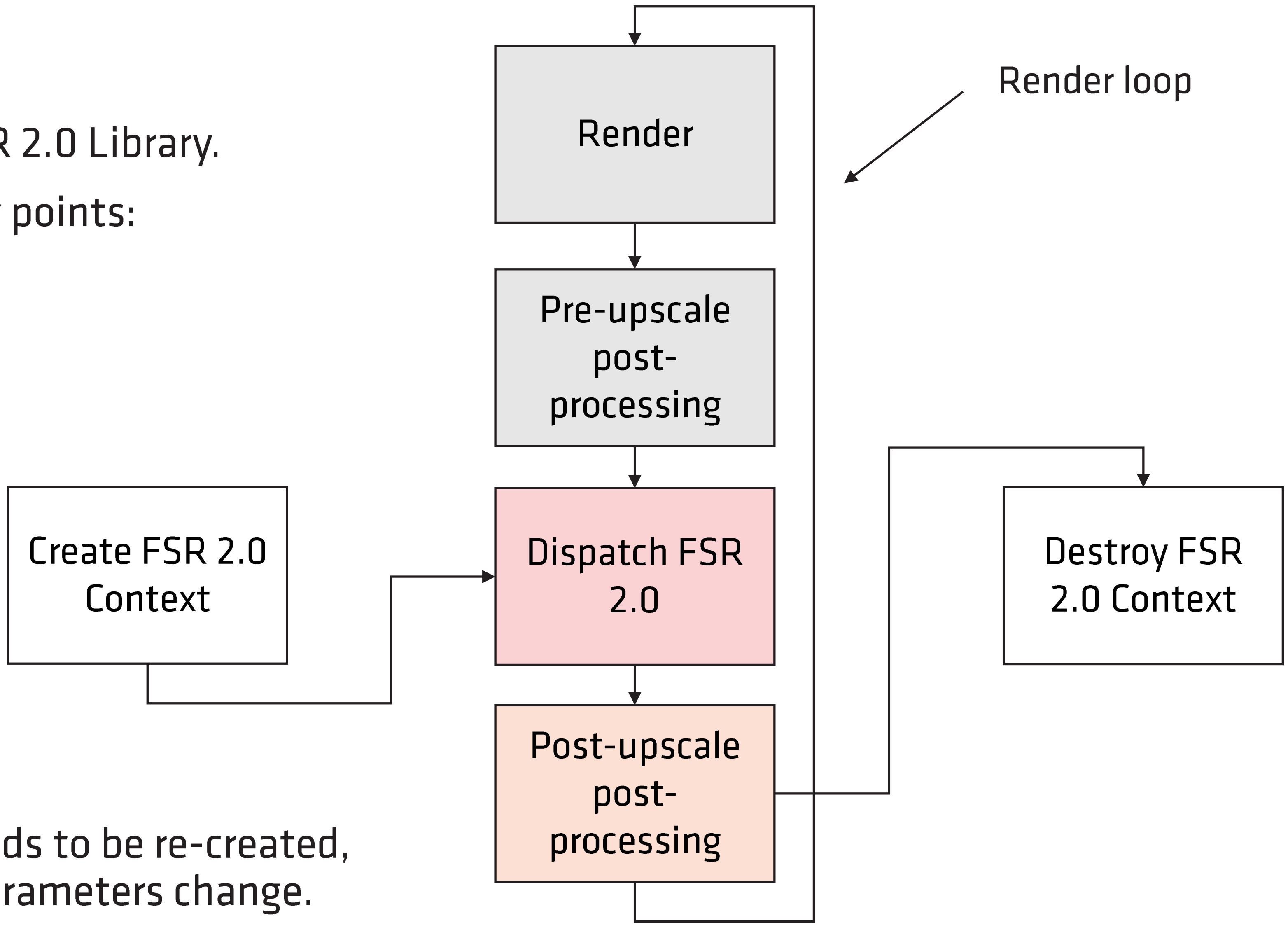
FIDELITYFX SUPER RESOLUTION 2.0 FOR DEATHLOOP

- Deathloop is the first title in which we integrated FSR 2.0!
- The integration was part of the development.
- Early proof of concept in a real game ☺.



FSR 2.0 - SETUP

- Deathloop integrates the FSR 2.0 Library.
- It makes use of the API entry points:
  - ffxFsr2ContextCreate
  - ffxFsr2ContextDestroy
  - ffxFsr2ContextDispatch

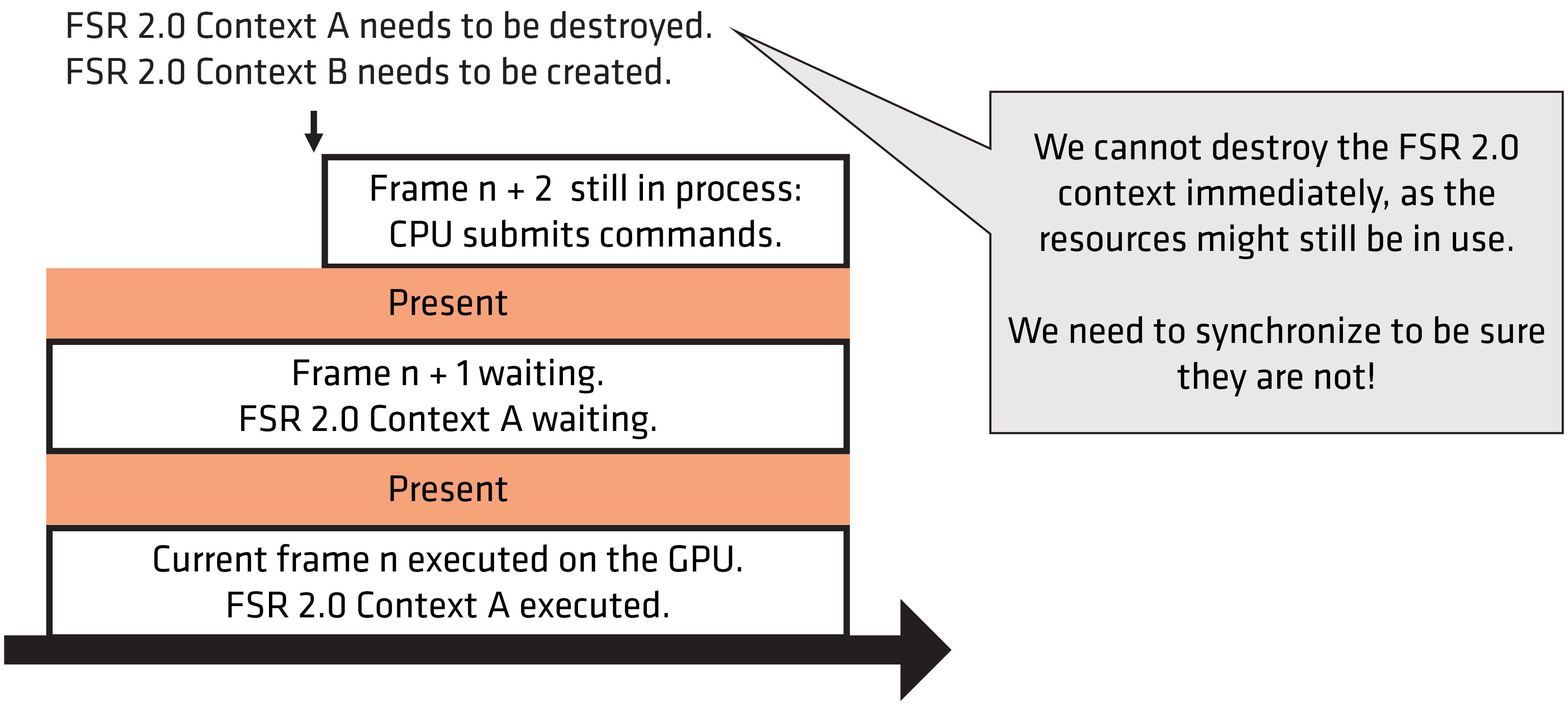


- The FSR 2.0 context only needs to be re-created, when the context creation parameters change.
- One such change is a change in presentation resolution.

[AMD Official Use Only]

FSR 2.0 - LIFECYCLE

- When destroying a FSR 2.0 context, we need to be sure it's not in use anymore.



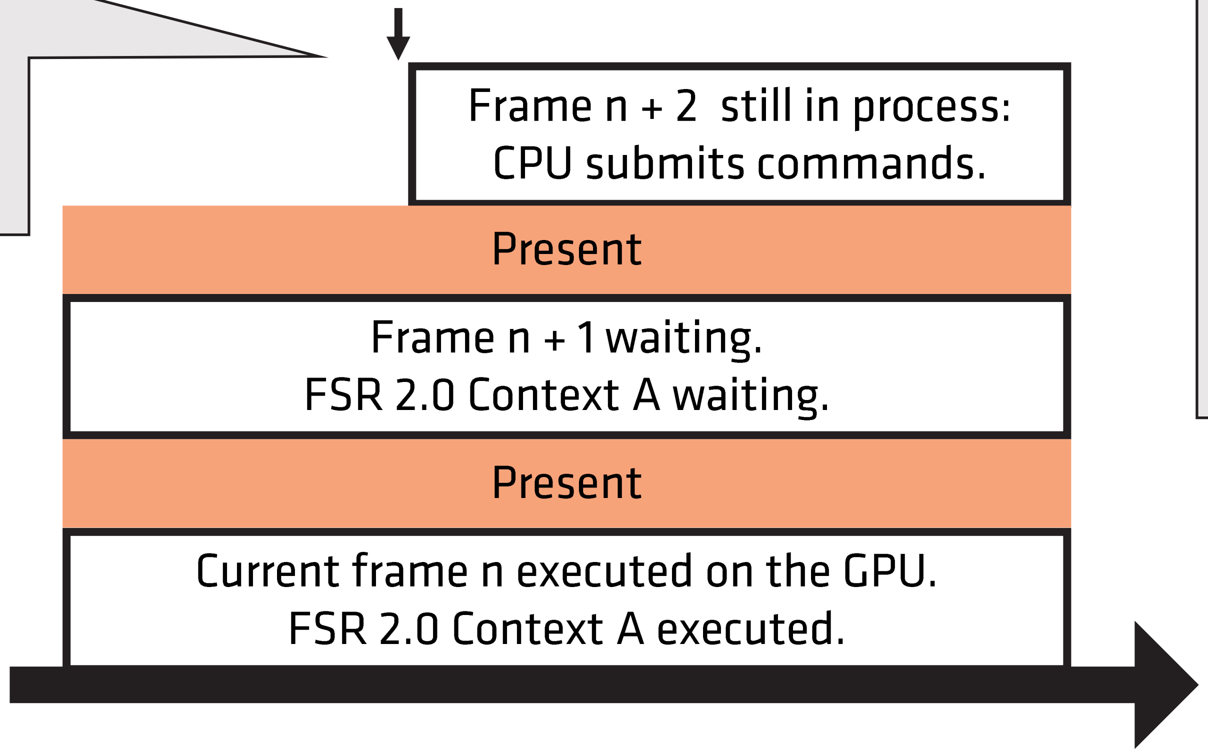
[AMD Official Use Only]

FSR 2.0 - LIFECYCLE

- When destroying a FSR 2.0 context, we need to be sure it's not in use anymore.

FSR 2.0 Context A needs to be destroyed.
FSR 2.0 Context B needs to be created.

We register the FSR 2.0 Context A destruction and execute it by the end of Frame n + 2.



However, we can create the FSR 2.0 Context B immediately.

As a result, there might be two FSR 2.0 context for a short time.

[AMD Official Use Only]

FSR 2.0 - INPUT

Color Buffer

- The Color buffer is in Linear Color Space, the image format is R11G11B10_FLOAT.
- To improve the precision in the R11G11B10_FLOAT target, the values are multiplied with a pre-exposure value.
- FSR 2.0 needs to do this as well.
- The pre-exposure value is passed to FSR 2.0 as a per-frame parameter.

Exposure Texture

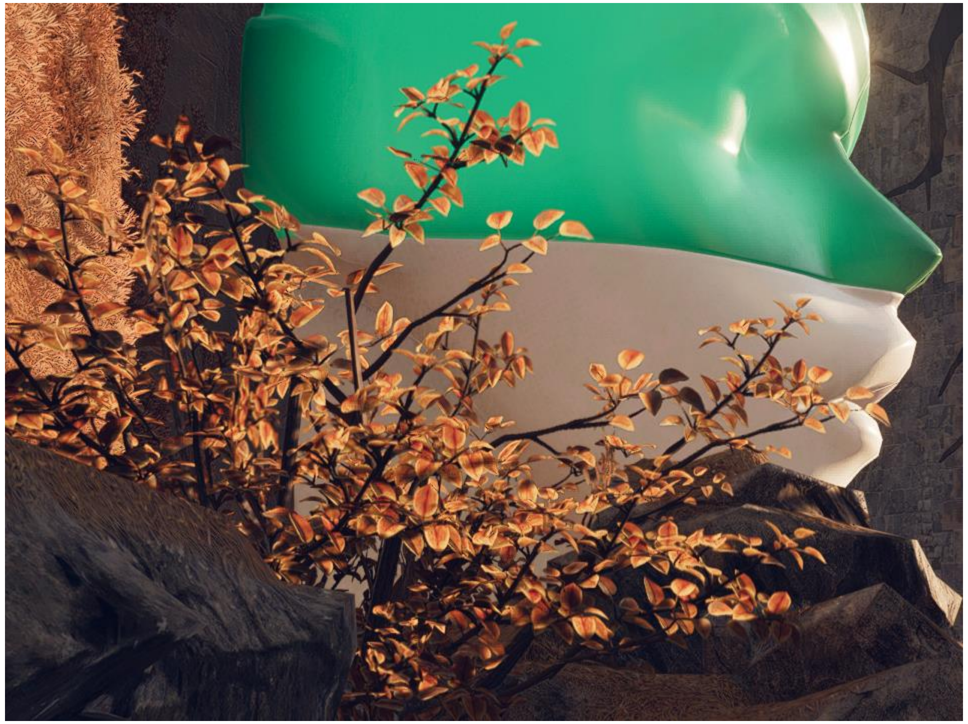
- Deathloop provides its own exposure texture to FSR 2.0.
- This exposure texture is optional though – FSR 2.0 can compute one of its own if needed.

FSR 2.0 - INPUT

Motion Vectors

- Deathloop provides Motion Vectors for nearly all scene elements, including Vegetation, in R16G16_Float.
- This is great, as Motion Vectors help to ensure a high-quality upscaled image ☺.

Native 4K, TAA + Sharpener enabled



FSR 2.0 Quality, 1440p → 4K

FSR 2.0 - OUTPUT

Sharpening

- Deathloop‘s presentation image rendered natively – without any upscaling - is very soft.
  - There is the option to enable sharpening in the menu.
  - The sharpener is enabled in the quality presets high to ultra.
- To achieve a comparable result when upscaling, we enable the FSR 2.0 built-in sharpener:
  - Robust Contrast Adaptive Sharpener (RCAS).
- To avoid double-sharpening, Deathloop‘s own sharpener is disabled.

Deathloop supports all four FSR 2.0 quality modes:

- Quality: 1.5x scaling
- Balanced: 1.7x scaling
- Performance: 2.0x scaling
- Ultra Performance: 3.0x scaling

It also supports FSR 2.0 Dynamic Resolution Scaling.

[AMD Official Use Only]

FSR 2.0 IN ACTION

Native 4K, TAA + Sharpener enabled

FidelityFX Super Resolution 2.0
Quality
1440p → 4K

NATIVE 4K, TAA + SHARPENER ENABLED

FSR 2.0 QUALITY, 2560 X 1440 → 4K

FSR 2.0 BALANCED, 2256 X 1272 → 4K

FSR 2.0 PERFORMANCE, 1920 X 1080 → 4K

FSR 2.0 ULTRA PERFORMANCE, 1280 X 720 → 4K

FSR 2.0 PERFORMANCE IMPACT

All Performance numbers[2] shown are from FSR 2.0 beta versions.

They will most likely change for final release.

Compared to Native 4K, TAA + Sharpening + RT enabled, FSR 2.0 improves the frame time:

- FSR 2.0 Quality mode: up to ~50%.
- FSR 2.0 Balanced mode: up to ~69%.
- FSR 2.0 Performance mode: up to ~90%.
- FSR 2.0 Ultra Performance mode: up to ~147%.

[2]On an AMD Radeon™ RX 6900 XT. See backup slide for full system specs.

[AMD Official Use Only]

SUMMARY

Optimizations

- Barriers
  - Barriers can cause the GPU to wait until all the work is completed.
  - They also can cause decompression and cache invalidations/flushes.
- Write pattern
  - Try to write in coalesced 256-byte blocks.
  - Sometimes only a small change is required, with a nice performance boost ☺

Feature integration

- Deathloop was the first title that integrated FidelityFX Super Resolution 2.0!

DISCLAIMER & ATTRIBUTIONS

DISCLAIMER

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

© 2022 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, EPYC, Radeon, RDNA, Ryzen, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective owners.

AMD

HARDWARE CONFIGURATION FOR BENCHMARKS

[1]Benchmarks for optimizations, before and after:

- AMD Radeon™ RX 6900 XT
- Driver: Radeon™ Adrenalin 22.2.1
- AMD Ryzen™ Threadripper™ 3970X

[2]Benchmarks for FSR 2.0:

- AMD Radeon™ RX 6900 XT
- Driver: Radeon™ Adrenalin 21.50-220210a
- AMD Ryzen™ 9 5900X @ 3.79 GHz
- Smart Access Memory (SAM) enabled