



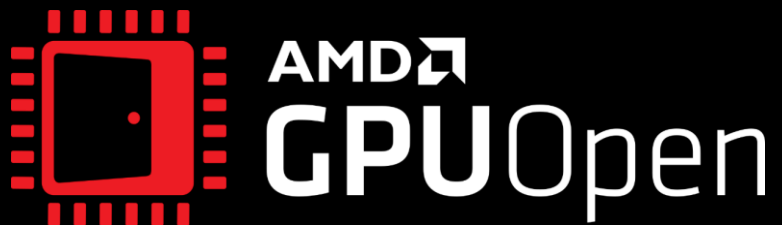
FIDELITYFX SUPER RESOLUTION 2.0

COLIN RILEY

CORE TECHNOLOGY GROUP

THOMAS ARCILA

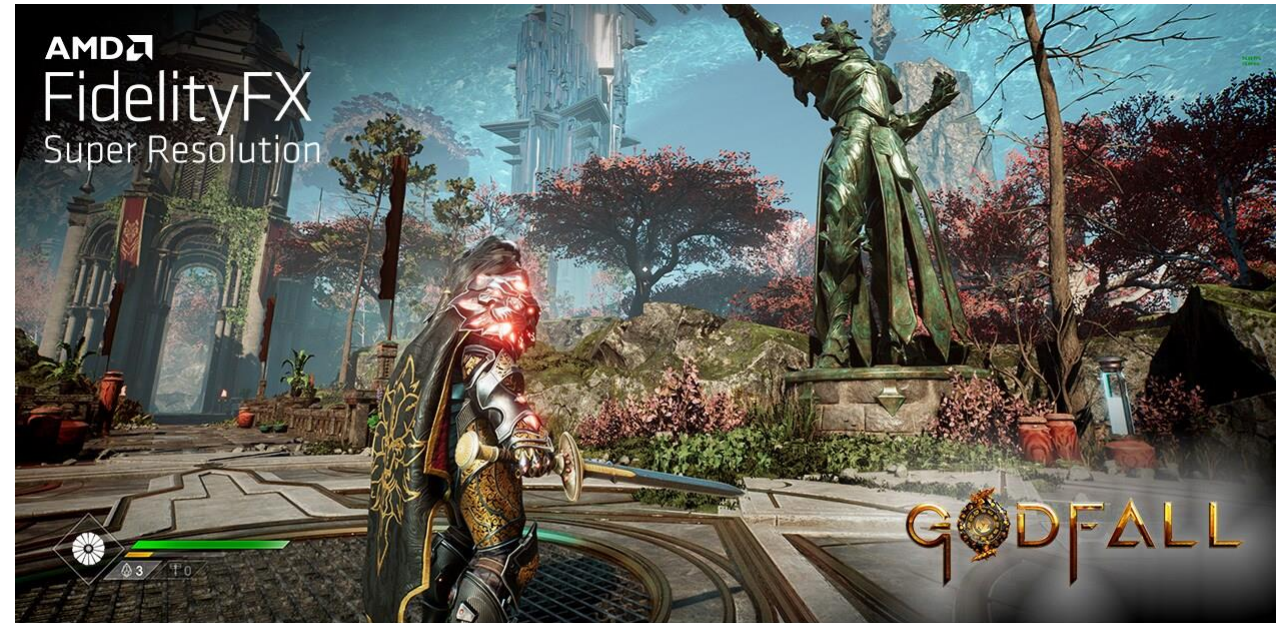
ADVANCED RENDERING RESEARCH GROUP



GDC

LOOKING BACK AT FSR 1.0

- Our best-in-class Spatial Upscaling solution.
- Designed for high performance.
 - Easy to integrate into games.
- Open sourced with a permissive MIT license.
- FSR 1.0 has enjoyed great developer adoption
 - Available and coming soon in over 70 games.
 - Xbox® Game Development Kit sample available from day-0.
 - Cross-platform codebase has seen the technology utilized for PC, Consoles and mobile gaming.
- Radeon™ Super Resolution driver integration bringing the benefits of FSR to more games.



FSR Ultra Quality Mode **ON** - 87 FPS

LOOKING BACK AT FSR 1.0 - LIMITATIONS

- Spatial upscalers, whilst being easy to integrate, can have certain shortcomings.
- FSR 1.0 **requires** a high quality anti-aliased source image.
 - This is hard problem to solve - even before considering upscaling.
 - FSR1.0 attached to low-quality TAA implementations produces inferior upscaled output.
 - Games without anti-aliasing had to implement it, which made integrating FSR 1.0 more time consuming.
- Upscaling quality is a function of the source resolution input.
 - When the source resolution is very low resolution, there is not enough information to draw upon in order to regenerate thin detail.
 - Additional artefacts which can be seen include shimmering and poor edge reconstruction.
 - This was more apparent with the Performance upscaling preset.

INTRODUCING FIDELITYFX SUPER RESOLUTION 2.0

- Our next-generation upscaling solution.
 - The product of years of research from AMD, developed from the ground up.
 - Utilizes cutting-edge temporal algorithms to reconstruct fine detail.
- FSR* 2.0 is a new technique and not built upon FSR 1.0.
 - Requires different inputs.
 - FSR 2.0 produces an anti-aliased output from aliased input.
- Significantly higher image quality than FSR 1.0.
 - Different quality modes available.
 - Dynamic Resolution Scaling supported.
- Open source, cross-platform, and highly optimized.
 - Does not require Machine Learning hardware to achieve high-quality output.
 - Provided via library, with full C++ and HLSL source with API documentation to enable custom integrations.

AMD
FidelityFX
Super Resolution 2.0

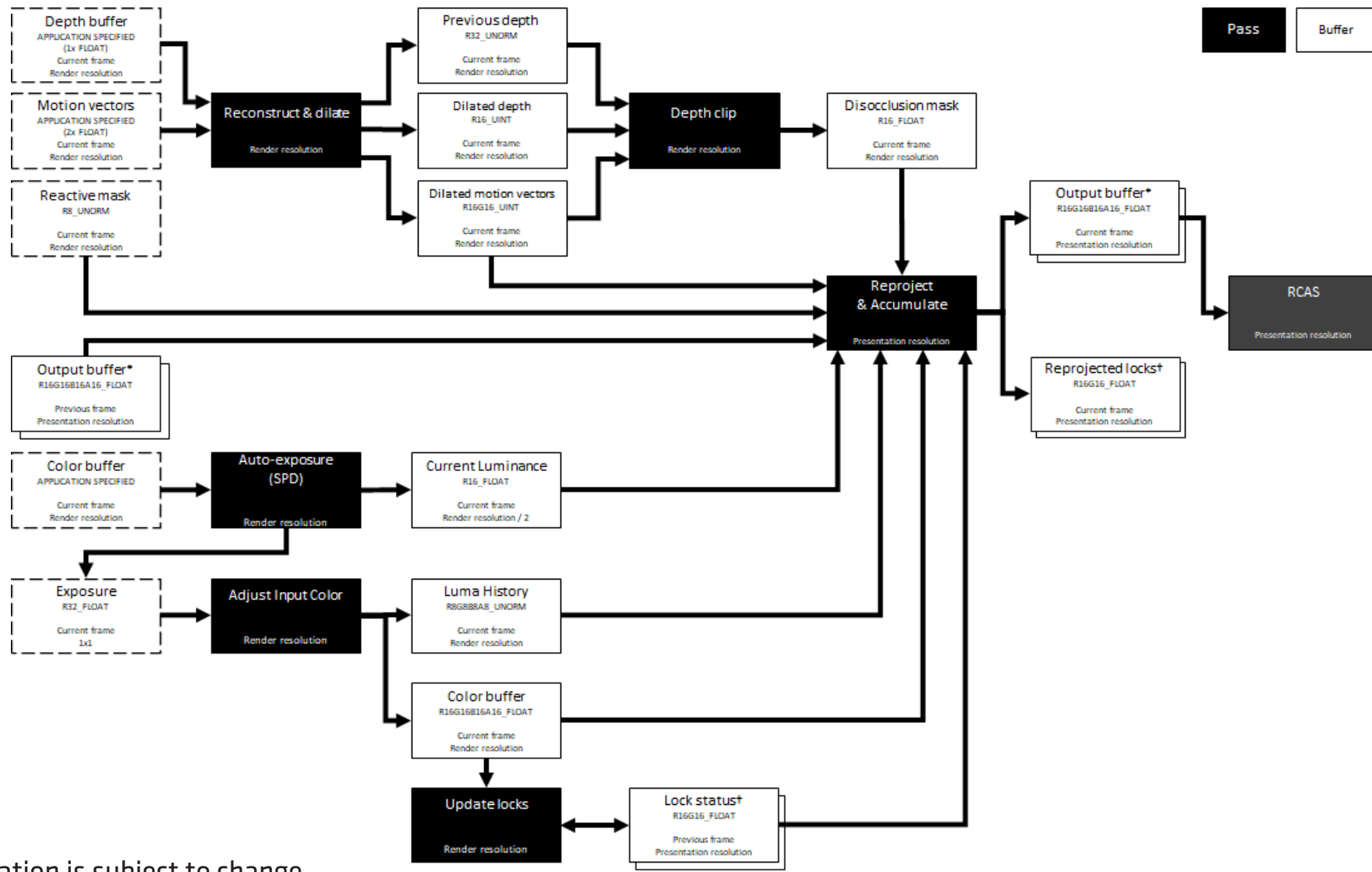
*See end note GD-187

ALGORITHM PRESENTATION

FSR 2.0 AT A GLIMPSE



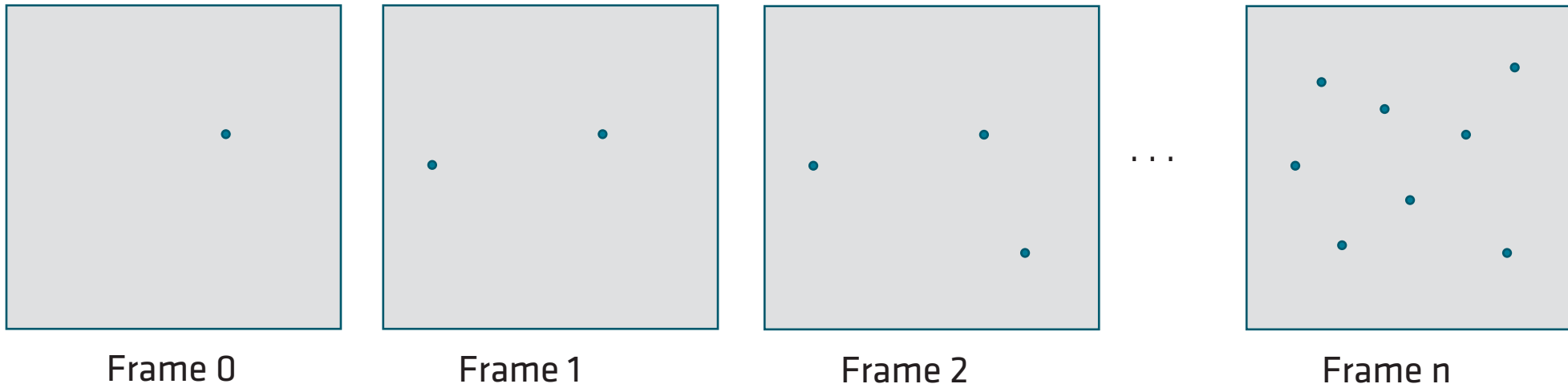
INSIDE THE BOX



* workload organization is subject to change

BUILDING UP ON TEMPORAL ANTIALIASING

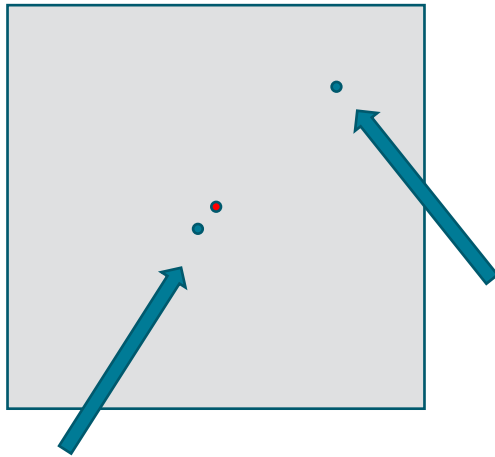
- TAA consists of sample aggregation at the rendered pixel level. Aggregation of more samples will increase antialiasing quality.



- Render resolution jittered samples. By jittered, we mean offset samples within the pixels.
 - Jitter sequence quality is key to achieve good antialiasing/upscaling.
 - Good distribution over space and time.
 - FSR 2.0 has some consideration on the jitter sequence, when dealing with thin features. More on that later.

PER PIXEL SAMPLE CONTRIBUTION

- Each sample will have a different contribution to the new frame, depending on:
 - Its spatial relevance to the actual target pixel.
 - The already accumulated amount of information.
- Closer and more recent samples are more important.



- Center of the result pixel
- New incoming samples

Frame N - 1: close to center,
high contribution

Frame N: more recent,
high contribution

PER PIXEL SAMPLE CONTRIBUTION

- Blending new sample with the existing accumulated pixel color.
 - S is the new contributing sample.
 - H is the already accumulated history samples.
- α is the sample spatial weight ω relative to the total accumulated weight τ at this pixel.
- Given previous equations, all samples have an equivalent contribution over time (still differing in their spatial contribution).

$$\alpha S + (1 - \alpha)H$$

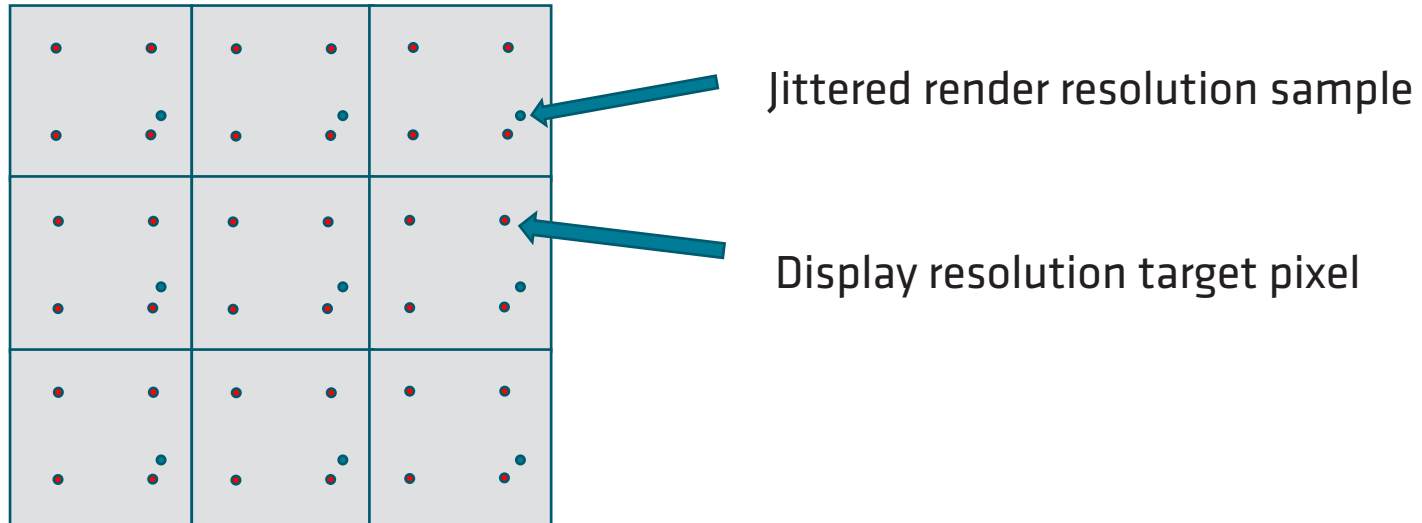
$$\alpha = \frac{\omega}{\tau}$$

$$\tau = \omega + \omega_{n-1} + \omega_{n-2} + \dots + \omega_0$$

Artificially capping τ to the number of expected contributing frames.
Emphasizes new incoming samples.

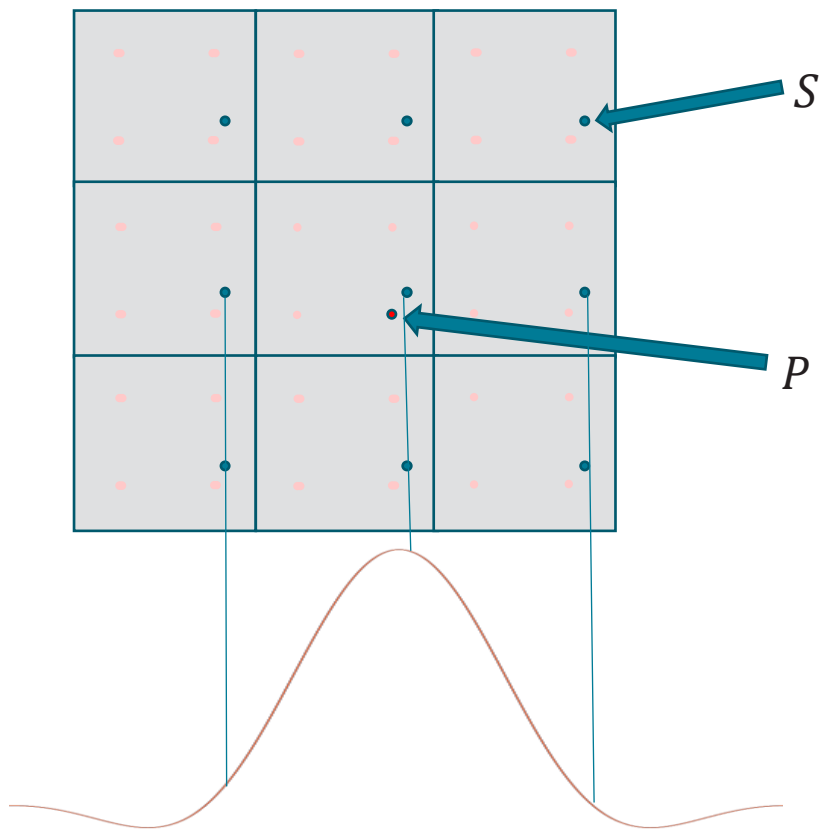
ADDING UPSAMPLING TO THE PICTURE

- Render resolution is resampled using Lanczos.



ADDING UPSAMPLING TO THE PICTURE

- Render resolution is resampled using Lanczos.
- Ringing is avoided clamping.



P is the high-resolution sample to compute
 S is the set of low-resolution samples to resample from

$$\omega_s = \text{Lanczos}(|PS|)$$

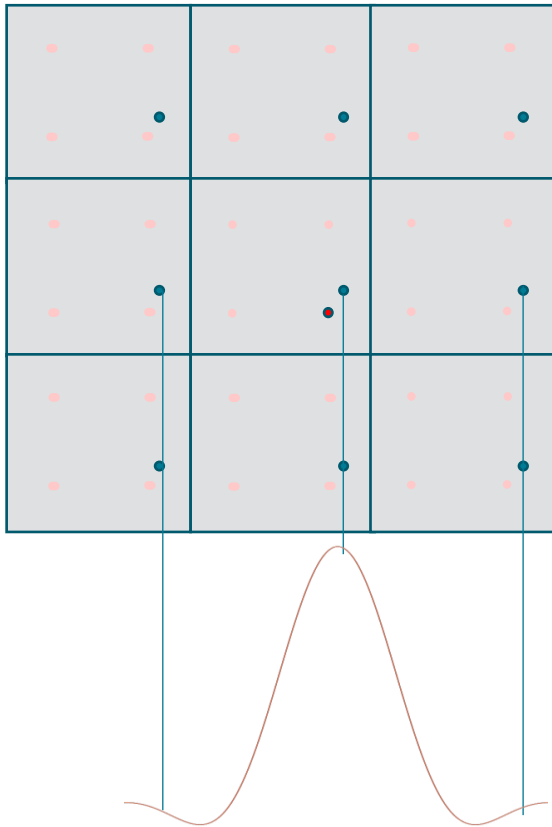
$$P = \sum_s S \cdot \omega_s$$

Ω is the weight associated to P

$$\Omega = \sum_s \omega_s$$

FROM LOW RES TO HIGH RES – GETTING SHARPER

- Render resolution is resampled using Lanczos.
- Getting sharper images through increasing the weight of the inner samples when doing Lanczos resampling.
 - Depending on local temporal luminance stability.



- On high luminance stability we will sharpen the resampling.

$$\omega_s = \text{Lanczos}(\beta |PS|)$$

Where β is a bias computed from averaged input luminance.

MOTION VECTORS

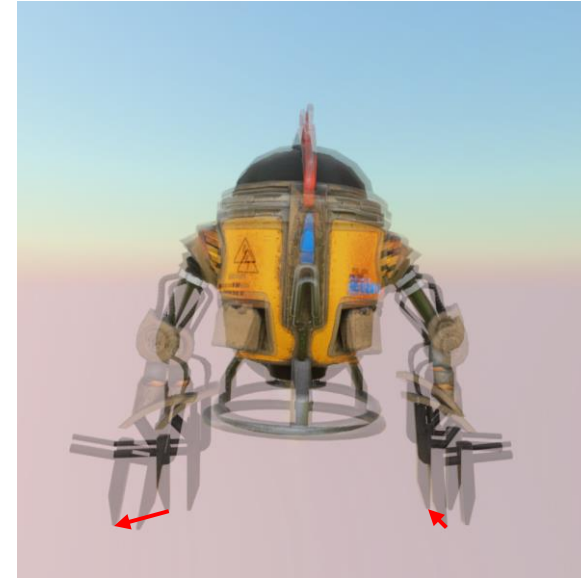
- A motion vector describes how a sample moved from the previous frame to the current one.
- Motion vectors must have jitter cancelled out as null vectors are expected on still.
- In order to correctly follow edges, the nearest vector in a 3x3 neighborhood is used. This is referred to as dilation.



Frame n - 1



Frame n



Motion vectors

HISTORY REPROJECTION

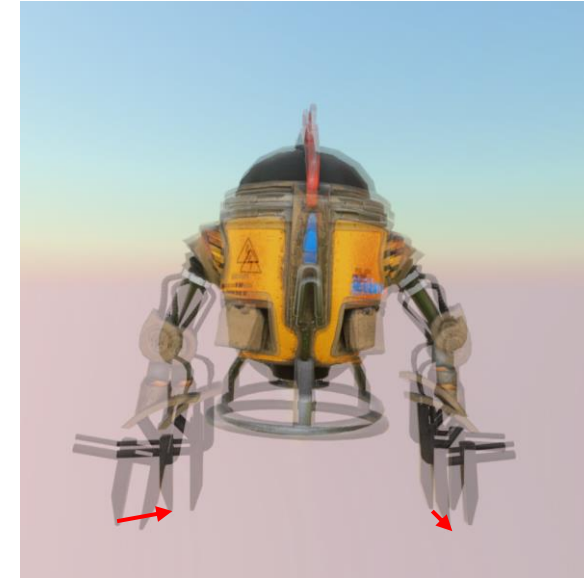
- Previous frame color information needs to be reprojected into the actual incoming frame.
- Again using Lanczos, ringing is handled as in the upsample stage.



Frame n - 1



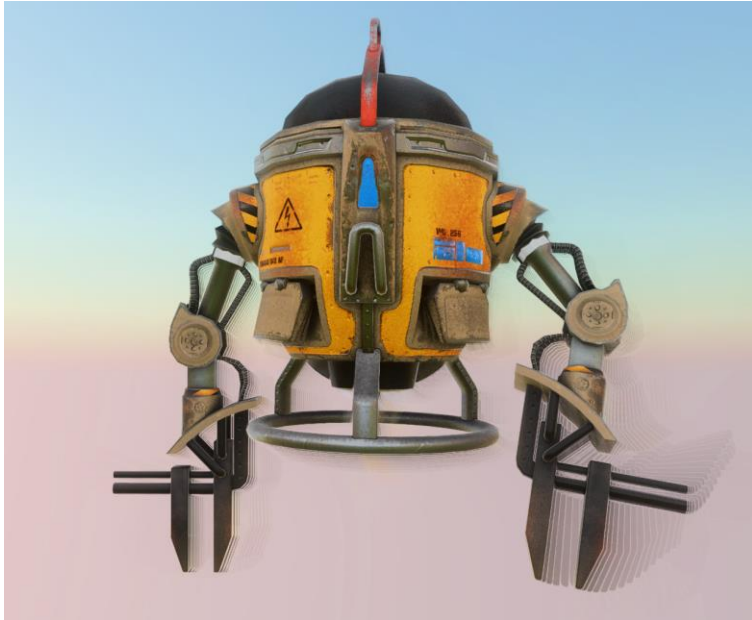
Frame n



Motion vectors

TEMPORAL GHOSTING

- There are cases where history data is not correlated anymore with the new input.
 - Disocclusion.
 - Shading change.
- Ghosting is seen when we don't correctly handle those cases.



DETECTING DISOCCLUSIONS

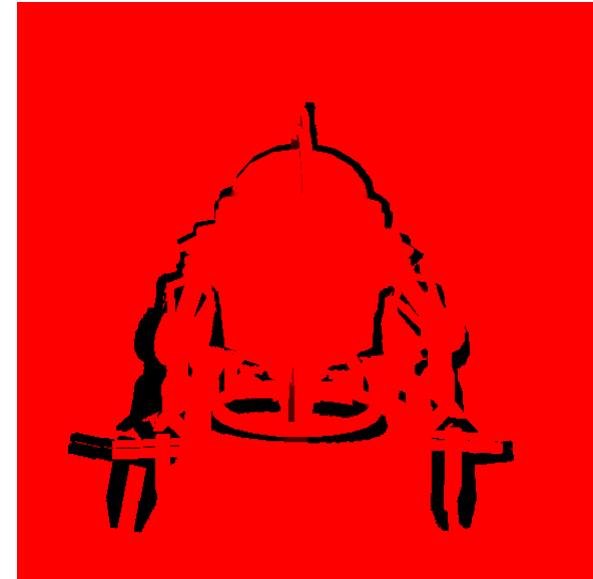
- A disocclusion mask is created using:
 - The actual render resolution depth.
 - The reconstructed previous frame depth.



Frame n - 1



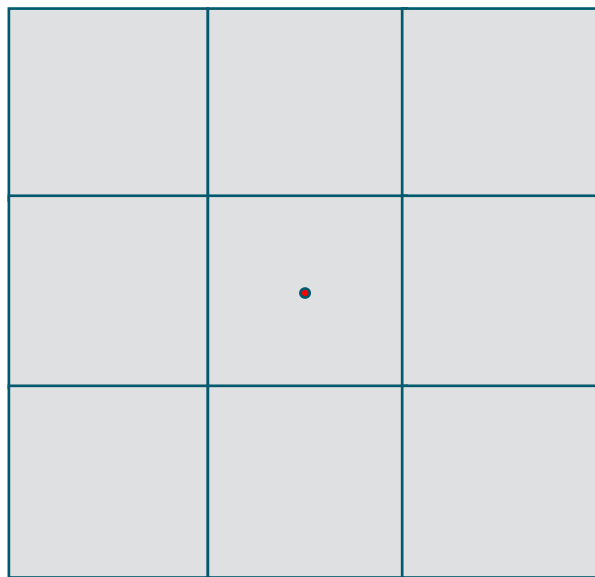
Frame n



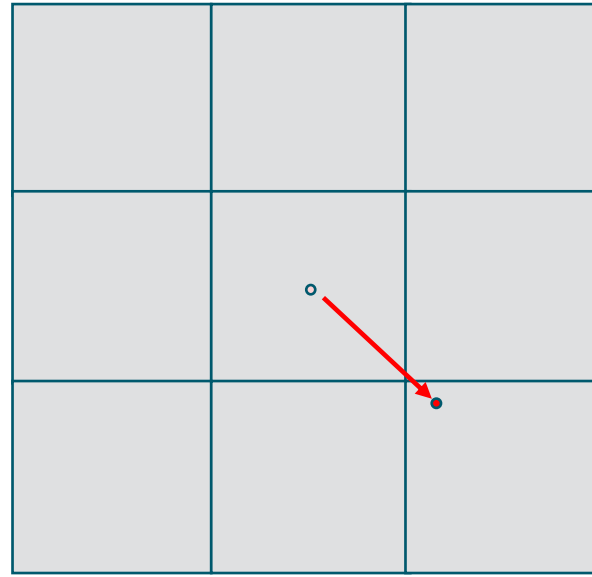
Disocclusion mask

RECONSTRUCTING PREVIOUS FRAME DEPTH

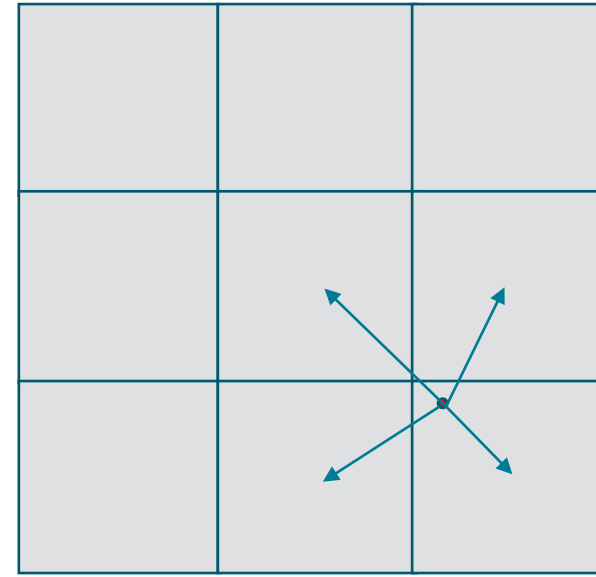
- Each depth sample is reprojected to its location into the previous frame.
- The reprojected sample is scattered among all impacted samples (backward or reverse reprojection*).
- The nearest depth is kept when multiple samples hit the same location.



Current frame depth



Reprojected and scattered depth



* Systematic Evaluation of the Quality Benefits of Spatiotemporal Sample Reprojection in Real-time Stereoscopic Path Tracing

DISOCCLUSION MASK

- For each sample, its current depth D is compared to the reconstructed previous depth at its previous location D_p (forward reprojection).
- Minimum depth separation ($MinDepthSep$) factor
 - Based on a K_{sep} constant*.
 - Computed in view space.
- Disocclusion is assumed per pixel ($D_p - D > MinDepthSep$)

*Minimum Triangle Separation for Correct Z-Buffer Occlusion

HISTORY RECTIFICATION

- All rectifications are based on current frame information only .
Accumulated, reprojected information would already incorporate ghosting, defeating the purpose of rectification.

HISTORY RECTIFICATION – DEPTH BASED

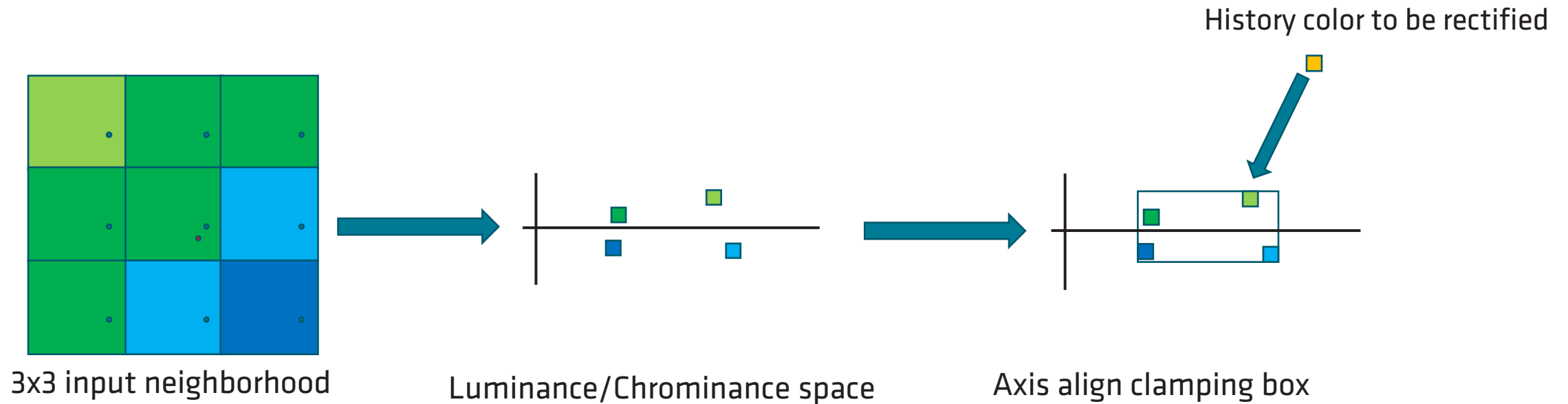
When a sample is disoccluded:

- Most of the previously accumulated color is dropped.
 - Creates some unnoticeable ghosting but makes the disocclusion smoother.
- New blurred sample is being pushed.
 - To increase image quality, the new incoming sample is blurred.
 - The blurred sample is computed at the upsampling stage, by decreasing β on the Lanczos curve.

$$\omega_s = \text{Lanczos}(\beta|PS|)$$

HISTORY RECTIFICATION - COLOR BASED

- Still valuable information that needs to be refit from the incoming frame.
- Compute a validity range that the history color will be clamped to.



THIN FEATURES

- Thin features are possibly only occurring for a few jittered samples over the overall sequence.
- Color rectification will take them down as soon as it is not part of the input image.
- This result in an unstable image, or some very dimmed features.
- We need to address this problem.



Almost invisible
and flickering
thin feature

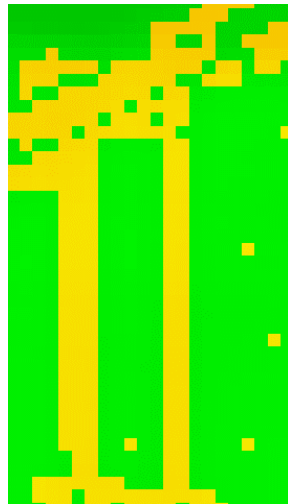


THIN FEATURES - LOCKING

- We detect pixel wide ridges as thin features and lock them.
- Next time history color goes through rectification, its weight will be emphasized.
- It will not get clipped.



Almost invisible
and flickering
thin feature



Locked pixels



Stabilized thin
feature



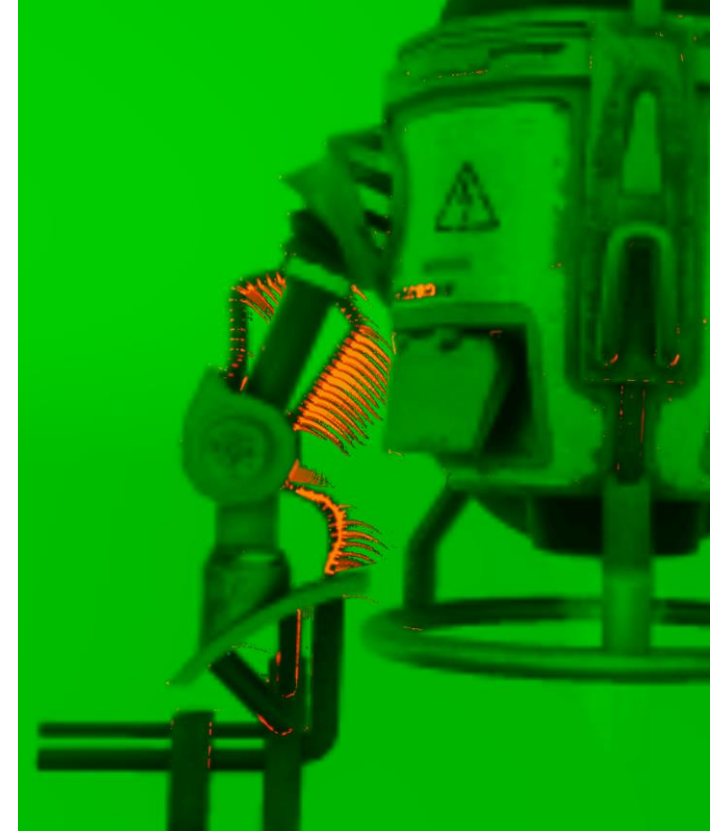
THIN FEATURES – WHEN TO UNLOCK

- Once locked, a thin feature stays for the length of the jitter sequence, when it will eventually be retriggered.
- After that, the lock is implicitly killed.
⇒ The jitter sequence needs to be the shortest possible, so a stall lock gets killed fast.

THIN FEATURES – WHEN TO UNLOCK

- When the scene moves, locks need to be updated:
 - Moved with the rest of the scene.
 - Kill if the thin feature disappear.
Implicit lock killing will most probably not happen fast enough.
⇒ Ghosting

Those locks needs to be detected and killed.



THIN FEATURES – WHEN TO UNLOCK

- Locks are killed on disocclusion, using the disocclusion mask.
- A shading change detection algorithm helps detecting stalled locks on shading change:
 - A local, low frequency, luminance comparison of this new candidate accumulation and locked one.
⇒ Passed some threshold, the lock is killed

HIGH DYNAMIC RANGE

- To avoid firefly artifacts, HDR data are internally tonemapped during the process and reverse tonemapped when output.
- The tonemapper is local and reversible.
- See <https://gpuopen.com/learn/optimized-reversible-tonemapper-for-resolve/>



INPUT COLOR PRE EXPOSURE AND EXPOSURE

- Engine can integrate a pre-exposure factor in their render images.
 - This factor will be different from frame to frame and is being cancelled by the game engine when blitting the final frame, usually when tonemapping.
 - Given that FSR 2.0 is composing previous images possibly with different pre-exposure factors, this parameter needs to be passed to FSR 2.0.
- For FSR 2.0 to achieve its optimal quality upscaling when dealing with HDR data, it needs to know the actual image exposure value.
 - If the engine is not exposing this value, an additional auto exposure computation stage can be enabled to compute per frame exposure value.

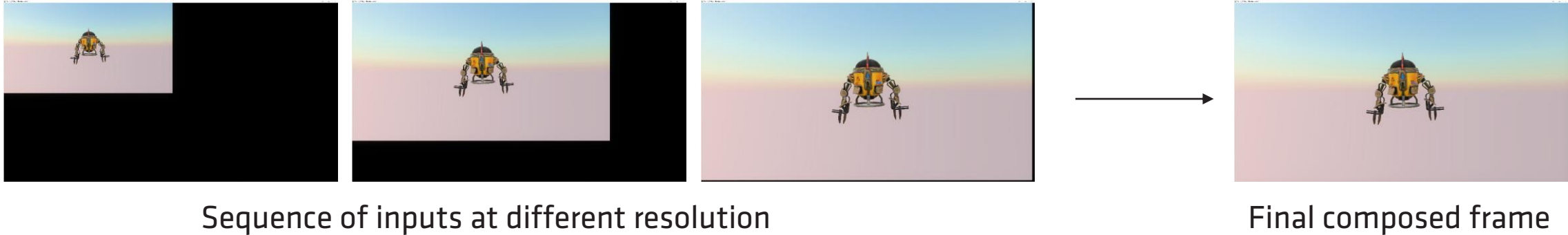
DYNAMIC RESOLUTION SCALING

Dynamic Resolution Scaling give the ability to the engine to decrease its render resolution on heavy workload still outputting an high resolution target. Once the load decrease, the render size can be increased again.

FSR 2.0 supports DRS:

- FSR 2.0 only relies on render resolution inputs.
- All persistent data are kept at display resolution.

This makes FSR 2.0 DRS compliant.



SHARPENING

- FSR 2.0 is also providing a post-process sharpening feature to help emphasizing features that have been degraded by the low-resolution input.
- This relies on RCAS (Robust Contrast-Adaptive Sharpening), from FSR 1.0.



Upscaled output



Sharpened output

OPTIMIZATIONS

OPTIMIZATION – TONEMAP OPERATIONS

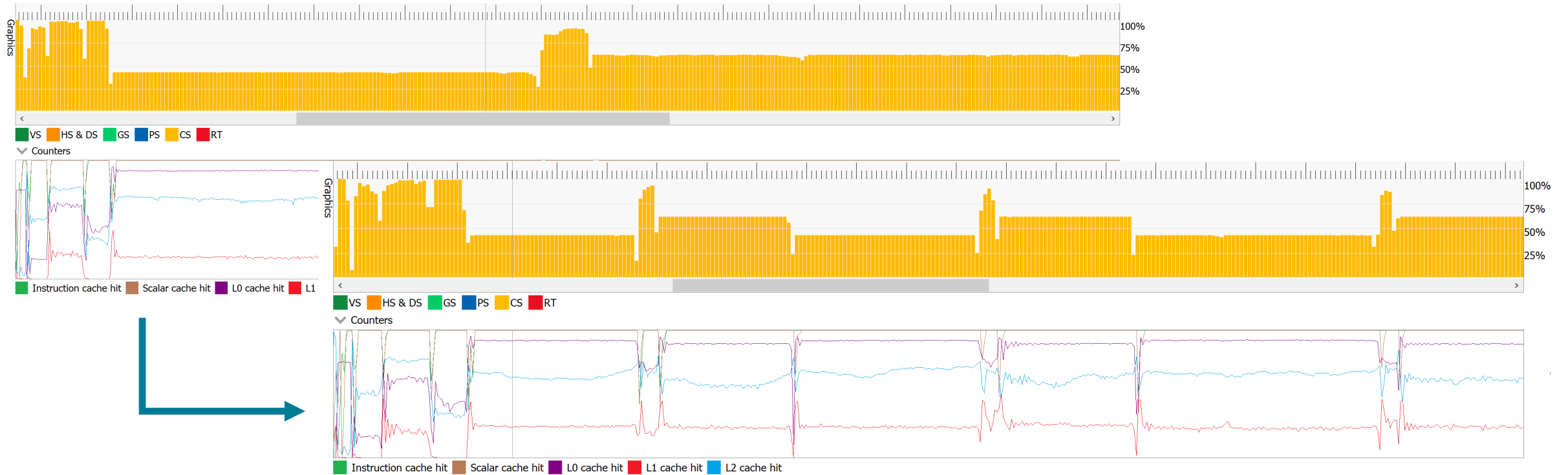
- At various points in the datapath of FSR 2.0, reversible tonemap operations are performed on input data in order to help improve output quality.
- Initial implementation pass had some tonemap operations occurring per-sample. With many neighborhood sample operations, significant ALU resources of the GPU were occupied.
- The optimization aims to apply tonemap per-pixel, rather than per-sample.
- This freed up a significant amount of ALU resources.

OPTIMIZATION – CACHE BLOCKING

- The FSR 2.0 algorithm can be very dependant on memory bandwidth, so keeping as much data in caches is essential.
- For a 4K scene, the amount of data read in a pass outstrips the caches. In RDNA™ 2 with AMD Infinity Cache™, there can be some spilling.
- To help alleviate this, compute dispatches can be split into multiple workloads, ensuring sampling operations remain within a generally known window.
 - This can increase cache hit rates, and therefore allow for faster execution of the workload.
 - Similar to how tiled GPU architectures operate, but manually implemented in software.



OPTIMIZATION – CACHE BLOCKING

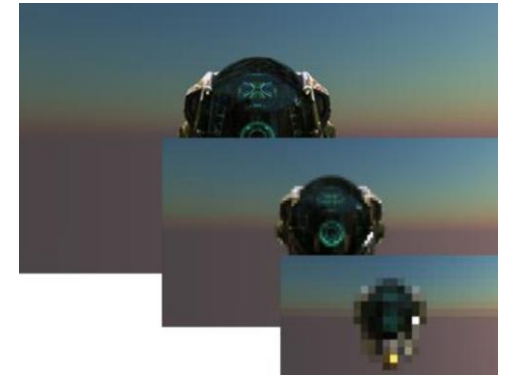


- L0 cache hit rates increased 37% on AMD Radeon™ RX 6800XT when splitting workloads into 3 blocks.

OPTIMIZATION – USE SPD FOR EXPOSURE CALCULATION

- FSR 2.0 has an option for “Auto Exposure”
 - This will calculate exposure for the input automatically, with a small amount of overhead.
- Exposure workloads in the past have been inefficient.
 - Single thread dispatch, poor hardware utilization.
- SPD – FidelityFX™ Single Pass Downsampler.
 - Optimized compute shader for outputting multiple downsamples of an image.
 - Generally used for mip generation.
- SPD has a configurable kernel.
 - Therefore, a reduction to a single pixel exposure value can be performed.
- Using SPD, the overhead for Auto Exposure in FSR 2.0 is only 17us for a 1080p input on AMD Radeon™ RX 6800XT

AMD
FidelityFX
Downsampler



OPTIMIZATION – LANCZOS APPROXIMATIONS AND LUT

- A lot of ALU cycles are spent calculating Lanczos weights.
 - Many transcendental operations – slower to execute, especially on older hardware.
- Approach: utilize the Lanczos approximation equation from FSR 1.0
 - Output quality was found not to be impacted using this approximation.

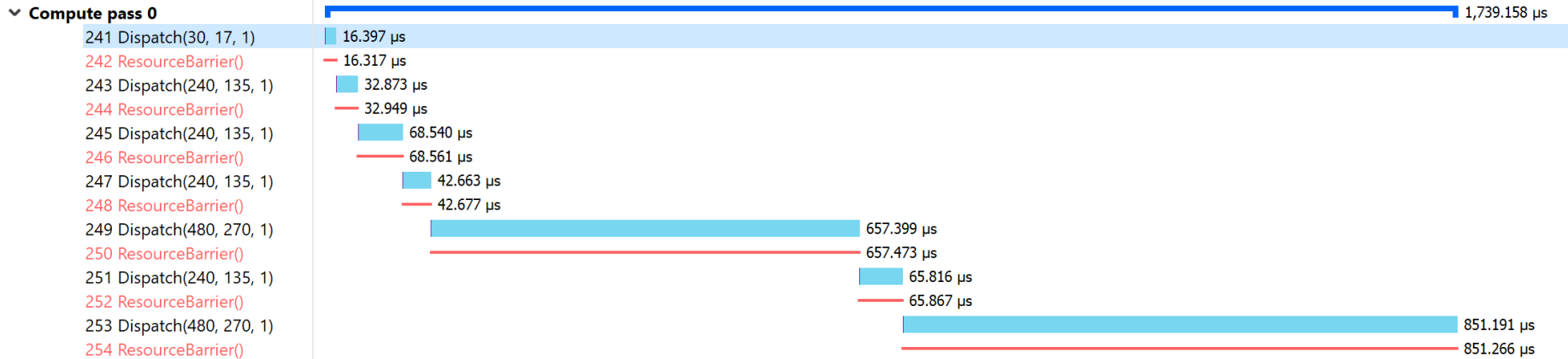
```
// FSR1 Lanczos approximation. Input is x*x and must be <= 4.  
float Lanczos2ApproxSqNoClamp(float x2)  
{  
    float a = (2.0f / 5.0f) * x2 - 1;  
    float b = (1.0f / 4.0f) * x2 - 1;  
    return ((25.0f / 16.0f) * a * a - (25.0f / 16.0f - 1)) * (b * b);  
}
```

- AMD Radeon™ RX 6800XT VALU hardware utilization reduced in the Reproject workload, of which this approximate Lanczos sampling is part of.
- Another optimization is to place a Lanczos look up table in a texture, and sample it.
 - This was discovered to be slightly faster on some RDNA™ 2 architectures.

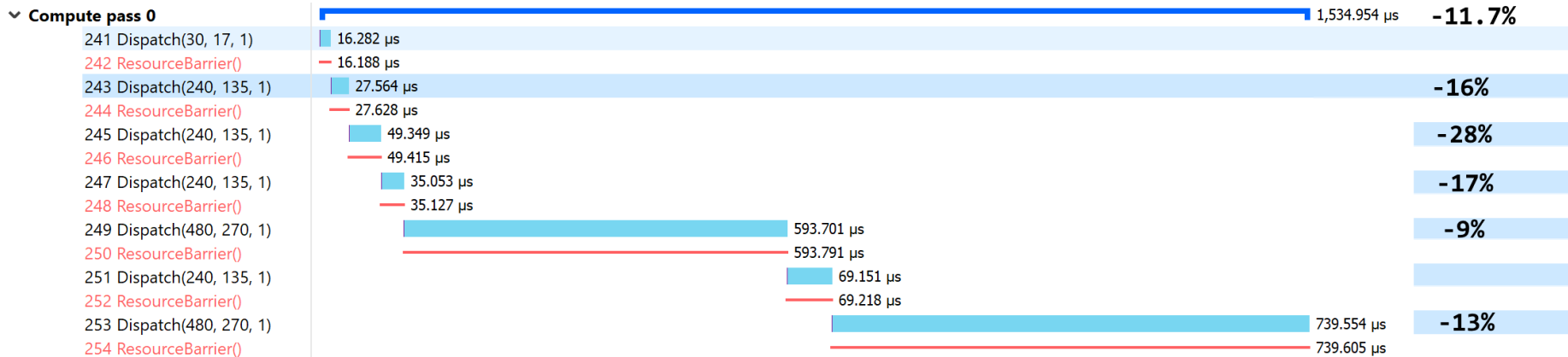
OPTIMIZATION – WAVE SIZE

- With RDNA™, Radeon™ GPUs are able to run in either Wave 32 or 64 mode.
- Generally, wave 32 can be faster due to reduced latencies but 64-wave mode can be faster in certain circumstances:
 - When using wave atomics.
 - To exploit better data access patterns.
- By default, the FSR 2.0 shaders ran in Wave32 mode.
- However, some workloads were identified to work better with Wave64.

OPTIMIZATION – WAVE SIZE



Forcing Wave64 mode



- Forcing Wave64 mode on all CS can reduce runtime by nearly 12% on AMD Radeon™ RX 6800XT.
- With Shader Model 6.6, there is an ability to specify at the source level what wave mode a developer would like their shader to run at on the GPU.

OPTIMIZATION - FALLBACKS

- Some RDNA™ 2 optimizations actually run slower on some GPUs (including previous Radeon™ products).
- Optimizations we have identified in this category have fallbacks implemented to improve performance on those GPUs.
 - As an example, Wave64 is only enabled on certain AMD GPUs where a performance increase is seen.
- If integration of FSR 2.0 is performed using our recommendations, the best permutation of possible optimizations will be used automatically.
 - There are hundreds of permutations possible.
- FSR 2.0 is incredibly competitive when it comes to performance comparisons.

PERFORMANCE – QUALITY MODE

- Includes auto-exposure, no sharpening.

FSR 2.0 Target resolution	Enthusiast GPUs (AMD RADEON™ RX 6800XT)	Performance GPUs (AMD RADEON™ RX 6700XT)	Mainstream GPUs (AMD RADEON™ RX 5700XT)
4K	< 1.1 ms		
1440p		< 0.8ms	
1080p			< 0.6ms

Performance taken from FSR2.0 Beta3 provided to partners March 2022. Subject to change.

PERFORMANCE – PERFORMANCE MODE

- Includes auto-exposure, no sharpening.

FSR 2.0 Target resolution	Enthusiast GPUs (AMD RADEON™ RX 6800XT)	Performance GPUs (AMD RADEON™ RX 6700XT)	Mainstream GPUs (AMD RADEON™ RX 5700XT)
4K	< 1ms		
1440p		< 0.7ms	
1080p			< 0.5ms

Performance taken from FSR2.0 Beta3 provided to partners March 2022. Subject to change.

PERFORMANCE – PERFORMANCE MODE

- Includes auto-exposure, no sharpening.

FSR 2.0 Target resolution	Enthusiast GPUs (AMD RADEON™ RX 6800XT)	Performance GPUs (AMD RADEON™ RX 6700XT)	Mainstream GPUs (AMD RADEON™ RX 5700XT)
4K	< 1ms	< 1.5ms	
1440p		< 0.7ms	< 0.9ms
1080p			< 0.5ms

Performance taken from FSR2.0 Beta3 provided to partners March 2022. Subject to change.

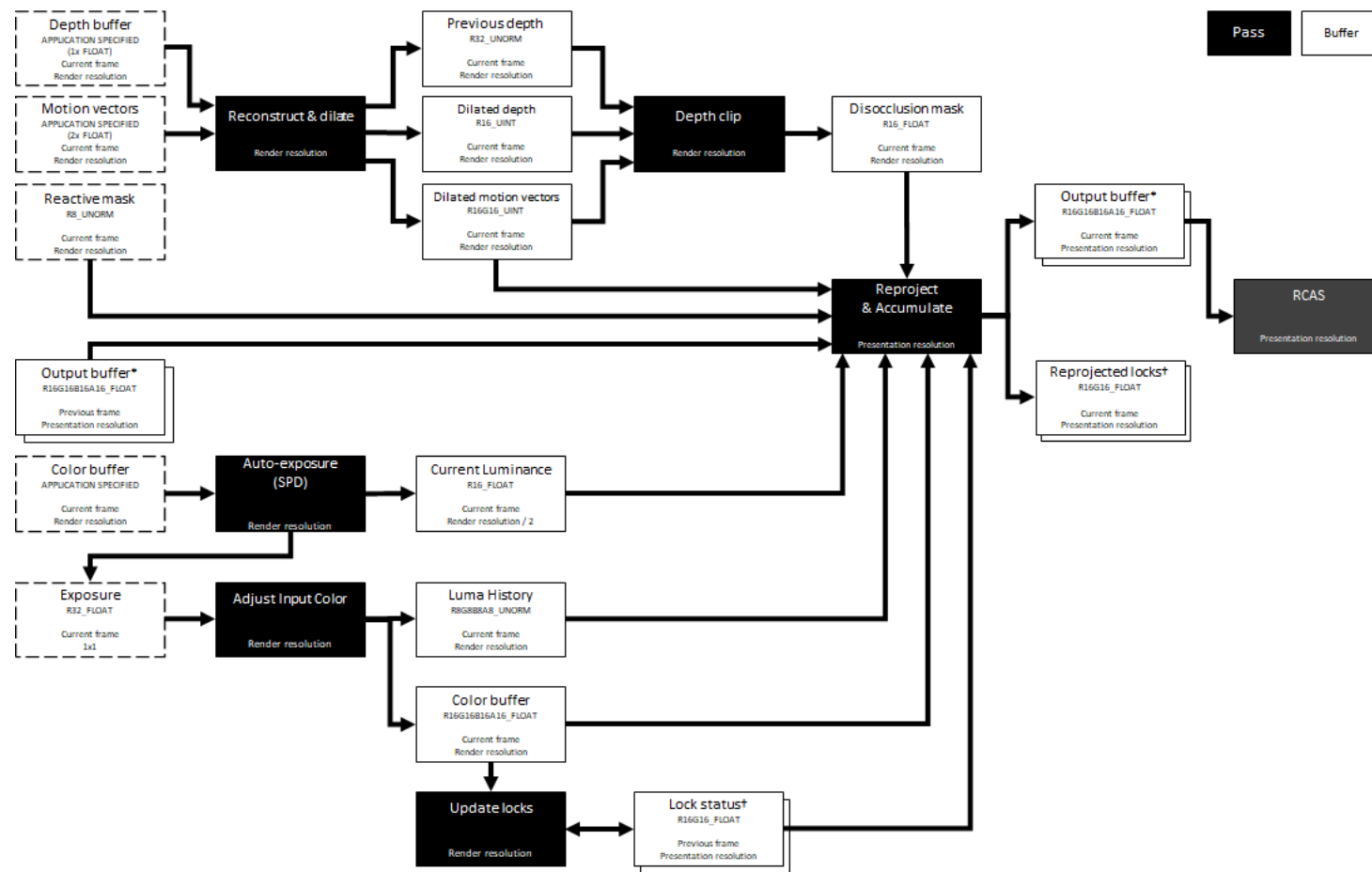
INTEGRATION

INTEGRATION

- FidelityFX™ Super Resolution 2.0 is designed to run on a wide range of hardware.
 - Designed to run on all GPU vendors.
 - No reliance on dedicated Machine Learning hardware.
 - Previous architectures supported.
 - Samples provided for DirectX® 12 (and later Vulkan®).
 - UE4 plugin will be provided (supports UE4.26 and UE4.27).
 - A GDKX sample will be made available to registered Xbox® developers.

INTEGRATION

- The FSR 2.0 dataflow is complex.
- Due to this, we have worked on an API that allows easier integration into game titles.



INTEGRATION

- The FSR 2.0 API is offered through a Windows[®] library to link against.
 - Full source code including shader and C++ code for the library will be made available on GPUOpen.
 - The library handles deciding the best optimizations to apply to the dispatched compute workloads.
- The API has several entry points
 - A callback system exists for operations such as resource creation.
 - These callbacks can be overridden to integrate into engine systems or rendering APIs.

```
ffxFsr2ContextCreate  
ffxFsr2ContextDestroy  
ffxFsr2ContextDispatch
```

```
ffxFsr2GetJitterPhaseCount  
ffxFsr2GetJitterOffset  
ffxFsr2CalculateRenderResolutionFromQualityMode  
ffxFsr2GetUpscaleRatioFromQualityMode
```

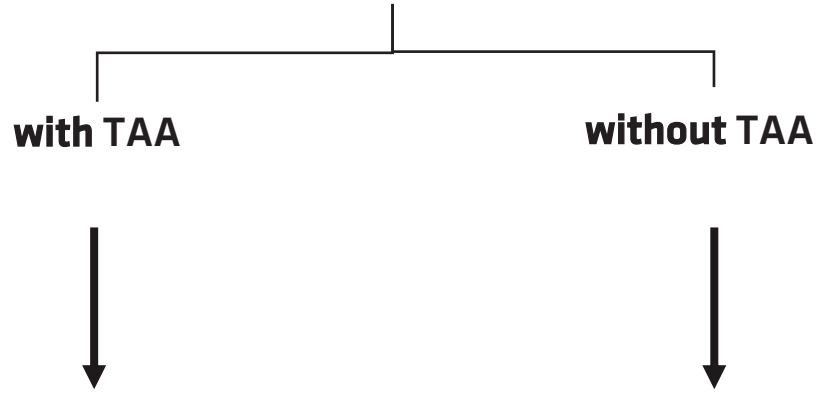
INTEGRATION - TIMEFRAME ESTIMATES

Games that already support DLSS 2.X

UE4/UE5 titles integrating FSR 2.0 plugin

Games with support for decoupled display/render resolution

Games without support for decoupled display/render resolution or Motion Vectors



Quickest
(less than 3 days)

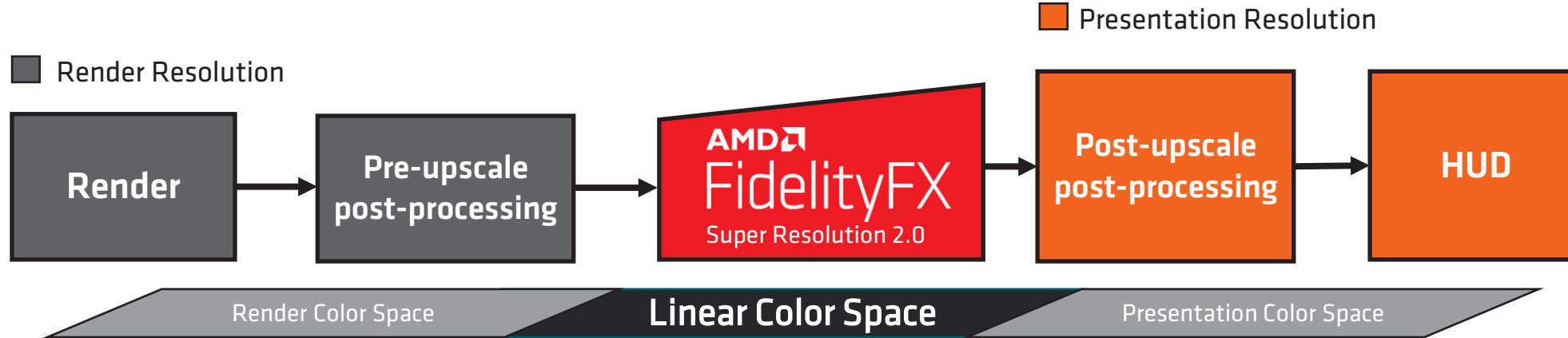
Longest
(~4 weeks+)

INTEGRATION – ANTI ALIASING

- FSR 2.0 replaces any TAA within the game frame.
 - When FSR 2.0 is enabled, TAA should not be run.
 - When FSR 2.0 is disabled, a TAA pass should be run.

- An FSR 2.0 TAA-only mode is in development.

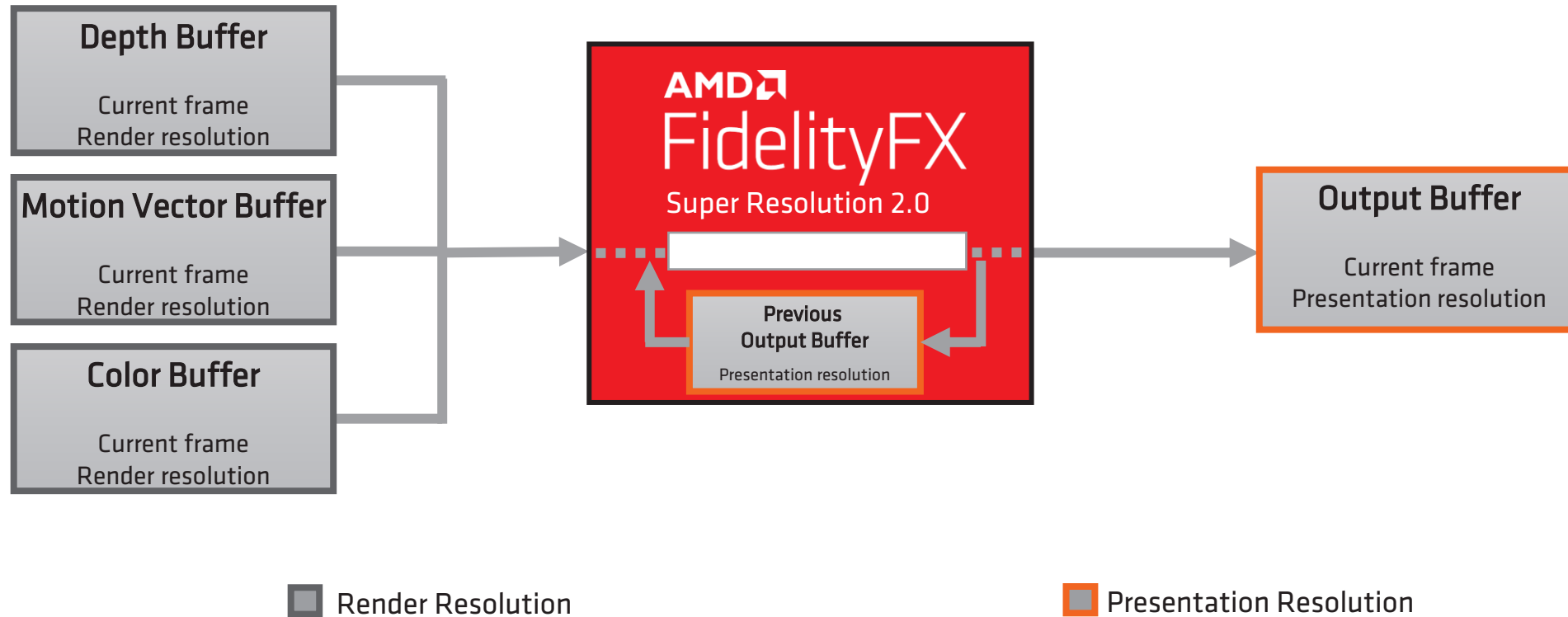
INTEGRATION – WHERE IN THE FRAME



- Compared to FSR 1.0, FSR 2.0 is earlier in the frame pipeline.
- As it replaces TAA, any post processing that requires anti-aliased input will need to be post-upscale.
- Any post-processing effect that requires the depth buffer will need to be pre-upscale.

INTEGRATION – INPUTS AND OUTPUTS

- Using the Library requires three buffers be provided at render resolution.
 - Internally, the FSR 2.0 Library retains the output buffer history for use in subsequent frames.



INTEGRATION – INPUT - DEPTH

- Recommendations for the input depth buffer:
 - Reverse depth
 - Infinite farplane
 - In R32_FLOAT format
- This will provide the best quality.
- Alternate depth buffer configurations will still work but may introduce quality degradation.
 - The API context create flags expose reverse / infinite depth configuration.
 - FFX_FSR2_ENABLE_DEPTH_INVERTED
 - FFX_FSR2_ENABLE_DEPTH_INFINITE

INTEGRATION – INPUT – MOTION VECTORS

- Requirements:
 - 2D vectors in UV space.
 - A single buffer resource.
 - At least R16G16_Float format – R8G8 will not give sufficient precision.
 - Internally the Motion Vectors are computed in FP16
 - Ensure all scene elements are included in the motion vector resource.
- There are options within the context create and dispatch parameters to configure the data within the motion vectors buffer for scaling the vectors, and whether the vectors are jittered.
 - FFX_FSR2_ENABLE_DISPLAY_RESOLUTION_MOTION_VECTORS
 - FFX_FSR2_ENABLE_MOTION_VECTORS_JITTER_CANCELLATION
- For more advanced integrations, the source can be modified in a single place to extract vectors from other formats or from packed resources.

INTEGRATION – INPUT - COLOR

- LDR Pipelines
 - Linear is not a requirement but is recommended.
- HDR Pipelines
 - Input needs to be encoded as linear RGB.
 - PQ or HLG are not suitable encodings for input.
 - Avoid negative color inputs – we will clamp to positive on load.
 - Set the `FFX_FSR2_ENABLE_HIGH_DYNAMIC_RANGE` flag in the creation params.
- Auto Exposure
 - `FFX_FSR2_ENABLE_AUTO_EXPOSURE` flag enables automatic exposure application to input color
 - Alternatively, a resource can be provided if this is already performed in your pipeline

INTEGRATION – JITTER PATTERNS

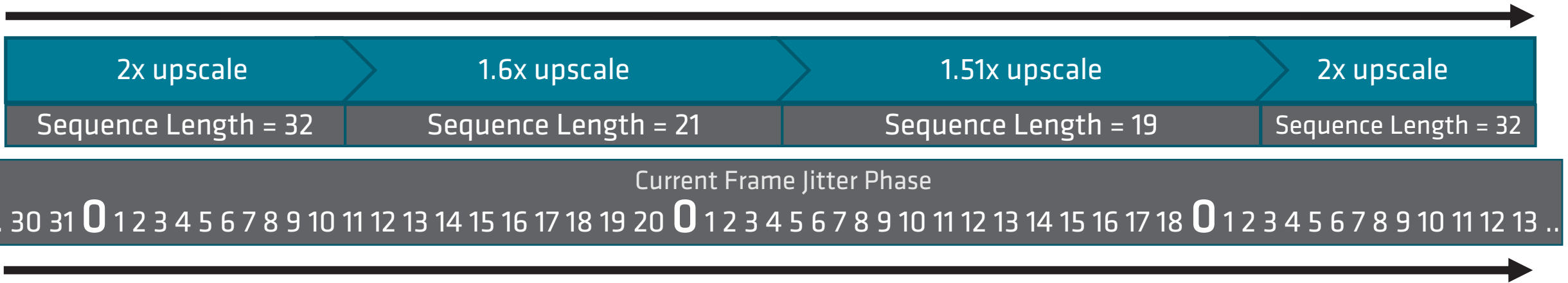
- Our recommended perspective jitter is Halton (2,3).
- There are helper functions included in the FSR 2.0 API which can assist in generating the correct jitter for application to the projection matrix.

```
m_index++;  
const int32_t jitterPhaseCount = ffxFsr2GetJitterPhaseCount(renderWidth, displayWidth);  
ffxFsr2GetJitterOffset(&m_JitterX, &m_JitterY, m_index, jitterPhaseCount);  
SetProjectionJitter(2.0f * m_JitterX / (float)renderWidth, -2.0f * m_JitterY / (float)renderHeight);
```

Quality mode	Scaling ratio (per dimension)	Halton sequence length
ULTRA_PERFORMANCE	3x	72
PERFORMANCE	2x	32
BALANCED	1.7x	23
QUALITY	1.5x	18
DRS	[1..n]x where n <= 3	ceil(8 * [scaling ratio] ²)

INTEGRATION – DYNAMIC RESOLUTION

- FFX_FSR2_ENABLE_DYNAMIC_RESOLUTION
 - We recommend Dynamic Resolution Scale scenarios have a maximum upscale of 1.5x.
- Take care with the jitter pattern sequence length
 - *Sequence Length* = `ffxFsr2GetJitterPhaseCount(renderWidth, displayWidth)`



Each time the render scale factor changes:

- the sequence length should update immediately.
- the current phase should continue through the scale change.

INTEGRATION – MIP BIAS

- A negative mip bias is required to obtain the most quality from FSR 2.0 upscaling.
 - $MIP\ bias = \log_2(RenderResolution/DisplayResolution) - 1.0$
- Textures with high-frequency detail should have their mip bias tuned towards 0 accordingly.
 - This is to prevent temporal aliasing artefacts.

FSR 2.0 quality mode	Scale factor	MIP bias
Quality	1.5x per dimension	-1.58
Balanced	1.7x per dimension	-1.77
Performance	2.0x per dimension	-2.0
Ultra Performance	3.0x per dimension	-2.58

This is an example of a texture that should have mip bias tuned.



INTEGRATION - SHARPENING

- FSR 2.0 includes a configurable sharpening pass using RCAS.
 - By default, sharpening is disabled.
 - This is performed as part of FSR 2.0, and should be decoupled from any existing pass for FSR 1.0.
- The sharpness factor goes from 0.0 to 1.0, with 1.0 being most sharp.
 - This is different from the scale used in FSR 1.0.
 - Specified in the arguments to `ffxFsr2ContextDispatch`.



INTEGRATION – HISTORY RESET

- Harsh transitions between scenes can result in ghosting artefacts.
- To prevent this, there is a reset boolean within `FfxFsr2DispatchParams` which will indicate that any previous history must be discarded.
- It is advised to use this when transitioning between gameplay and cinematics, or other element that results in a sudden change of visuals.

INTEGRATION – CUSTOM INTEGRATIONS

- Due to FSR 2.0 being open source, it is possible to bypass/rebuild the API and integrate it manually into your game engine.
 - This may be particularly useful if you use some form of packed motion vectors, as to integrate the decode step into FSR 2.0.
- It's also possible to build a custom version of this API library incorporating changes.
- Please let us know of any feature requests!

INTEGRATIONS – UI, OPTIONS AND LANGUAGE

- The latest guidance for how we would appreciate FSR 2.0 be exposed in options menus, along with technique descriptions and language, is available on GPUOpen.
 - <https://gpuopen.com/fsr2>

FSR 2.0 quality mode	Description	Scale factor	Input resolution	Output resolution
Quality	Quality mode provides an image quality equal or superior to native rendering with a significant performance gain.	1.5x per dimension (2.25x area scale) (67% screen resolution)	1280 x 720 1706 x 960 2293 x 960 2560 x 1440	1920 x 1080 2560 x 1440 3440 x 1440 3840 x 2160
Balanced	Balanced mode offers an ideal compromise between image quality and performance gains.	1.7x per dimension (2.89x area scale) (59% screen resolution)	1129 x 635 1506 x 847 2024 x 847 2259 x 1270	1920 x 1080 2560 x 1440 3440 x 1440 3840 x 2160
Performance	Performance mode provides an image quality similar to native rendering with a major performance gain.	2.0x per dimension (4x area scale) (50% screen resolution)	960 x 540 1280 x 720 1720 x 720 1920 x 1080	1920 x 1080 2560 x 1440 3440 x 1440 3840 x 2160
Ultra Performance*	Ultra Performance mode provides the highest performance gain while still maintaining an image quality representative of native rendering.	3.0x per dimension (9x area scale) (33% screen resolution)	640 x 360 854 x 480 1147 x 480 1280 x 720	1920 x 1080 2560 x 1440 3440 x 1440 3840 x 2160

*optional

CONCLUSION

- FSR 2.0 is a significant quality improvement over FSR1.
 - Encompasses a new temporal algorithm which provides excellent quality at high performance.
- FSR 2.0 API library allows for easier integration into games that already have a TAA or DLSS render path.
 - FSR 2.0 is open source and compatible with a wide set of graphics hardware.
- To keep up to date, ensure GPUOpen is followed on Twitter!
 - @GPUOpen



FSR 1 – Performance Mode



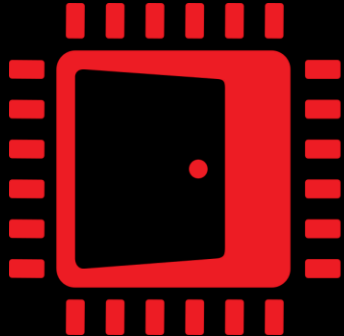
FSR 2.0 – Performance Mode




Deathloop running at 4K using FSR 2.0 Performance mode

ACKNOWLEDGEMENTS AND SPECIAL THANKS

- Stefan Petersson
- Zhuo Chen
- Stephan Hodes
- Lou Kramer
- Dihara Wijetunga
- Mark Satterthwaite
- Tim Mcquaig
- Mark Simpson
- Rys Sommefeldt
- Brian Bowman
- Sylvain Meunier
- Nicolas Thibieroz
- Steven Tovey



AMD 
GPU Open



RADEON



AMD 

ATTRIBUTION

- “Mech Drone” 3D Model- unmodified.
 - "Mech Drone" (<https://skfb.ly/LMro>) by Willy Decarpentrie is licensed under Creative Commons Attribution (<http://creativecommons.org/licenses/by/4.0/>).

BENCHMARK HARDWARE CONFIGURATION

- Optimization Slides system:
 - AMD Radeon™ RX 6800XT
 - AMD Ryzen™ 9 3900X
 - Graphics Driver: Radeon™ Adrenalin 2020 22.2.1

- Performance Slides system – FSR2.0 Beta:
 - AMD Ryzen™ 9 5900X
 - Radeon™ Adrenalin 2020 22.2.3
 - GPU as described on slides

DISCLAIMER

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS.' AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD FidelityFX Super Resolution is available on select games and requires developer integration. See <https://www.amd.com/en/technologies/radeon-software-fidelityfx-super-resolution> for a list of supported games. AMD FidelityFX Super Resolution is “game dependent” and is supported on the following AMD products: AMD Radeon™ RX 6000, RX 5000, RX 500, RX Vega series graphics cards, RX 480, RX 470, RX 460, and all AMD Ryzen™ processors with Radeon™ graphics if the minimum requirements of the game are met. AMD does not provide technical or warranty support for AMD FidelityFX Super Resolution enablement on other vendor's graphics cards. GD-187.

© 2022 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, EPYC, FidelityFX, Infinity Cache, 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. DirectX is either a registered trademark or trademark of Microsoft Corporation in the US and/or other countries. Vulkan and the Vulkan logo are registered trademarks of the Khronos Group Inc. Xbox is a registered trademark of Microsoft Corporation in the US and/or Other countries.

AMD 