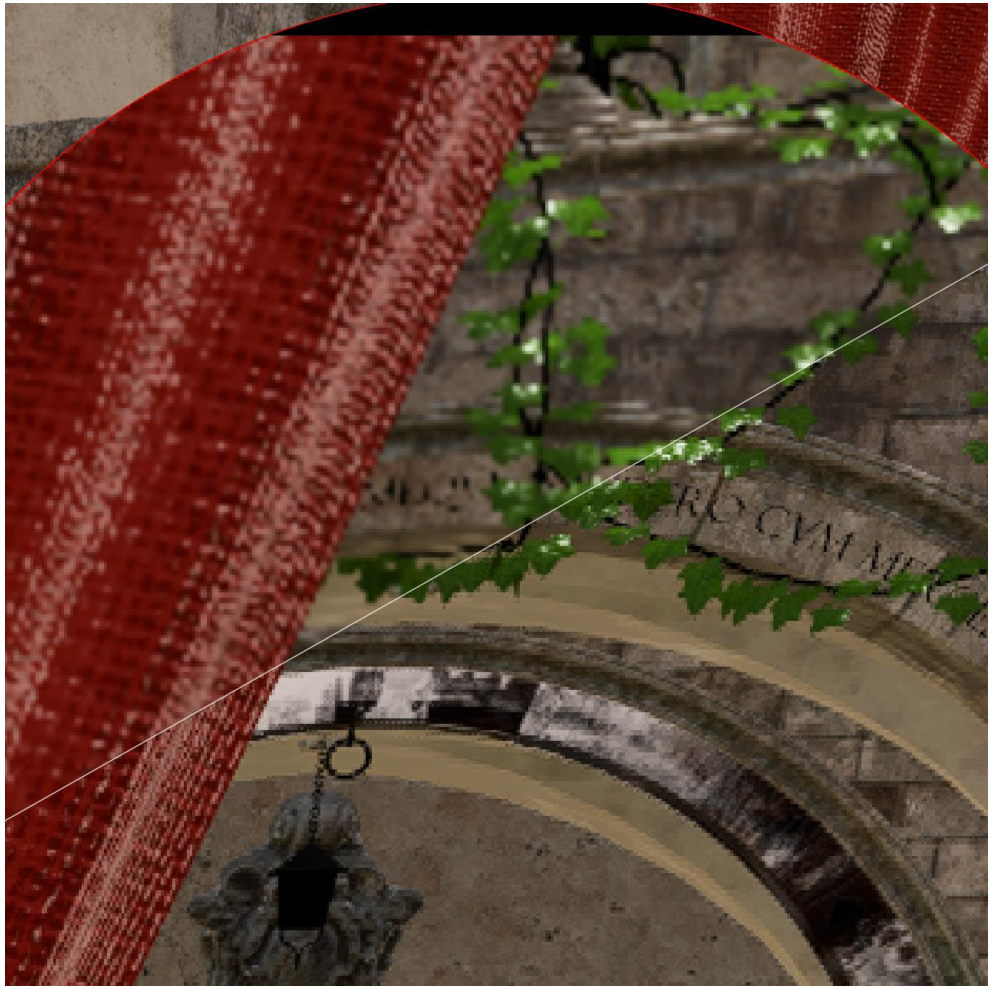
TEMPORAL UPSCALING
PAST, PRESENT, AND FUTURE

STEPHAN HODES – DEVELOPER TECHNOLOGY ENGINEER

together we advance_

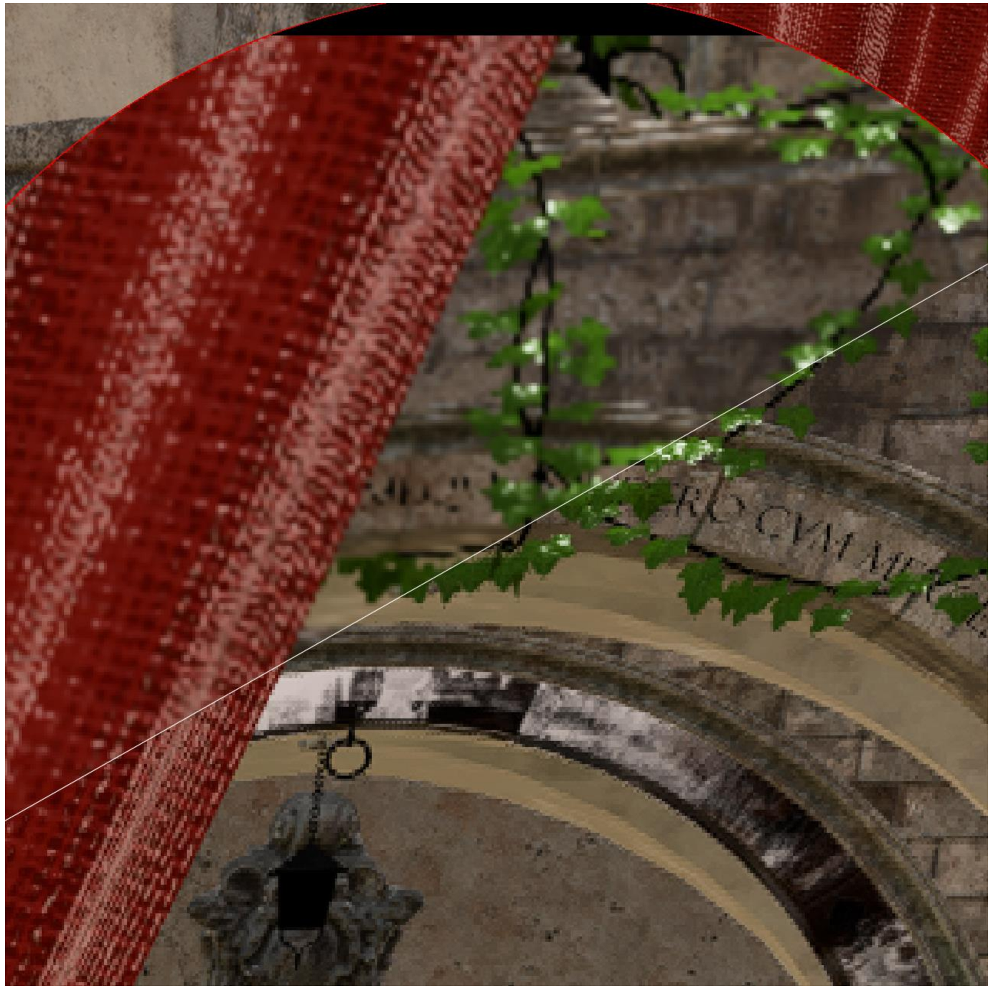
MOTIVATION OF THE PRESENTATION

- FSR 2 was released last year
  - Since release we developed several improvements to the FSR 2 core algorithm
  - Development still ongoing: FSR 2.2 was released in February
  - This presentation will provide details on the workings of the FSR 2.2 algorithm
- FSR 3 development going strong
  - Still too early to share details



TEMPORAL UPSCALING – PAST, PRESENT, AND FUTURE

- Spatial Upscaling – FSR 1
  - Problem description
  - How it works
  - The need for (T)AA
- Temporal Upscaling – FSR 2
  - Problem description
  - Corners everywhere
  - FSR 2.2 algorithm explained
- Work in progress: FSR 3
  - Adding frame interpolation
  - Need for motion estimation
  - Other challenges

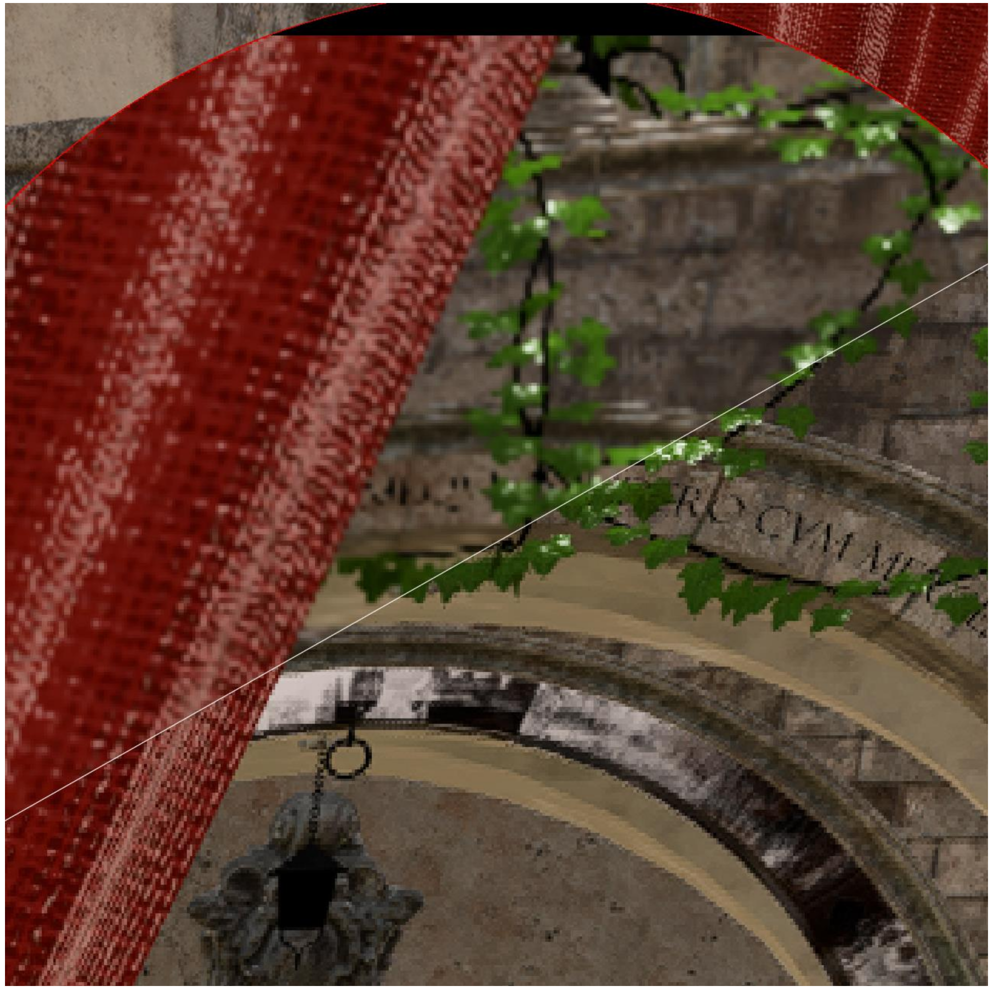


MOTIVATION BEHIND UPSCALING

- Rendering time increases with number of visible pixels
  - 720p resolution has ~1 million pixels
  - 1080p resolution has ~2 million pixels
  - 4k resolution has ~8.3 million pixels
  - 8k resolution has ~33 million pixels

- Effects with fixed screen size at higher resolution => larger pixel area
  - Heavy on memory bandwidth and cache

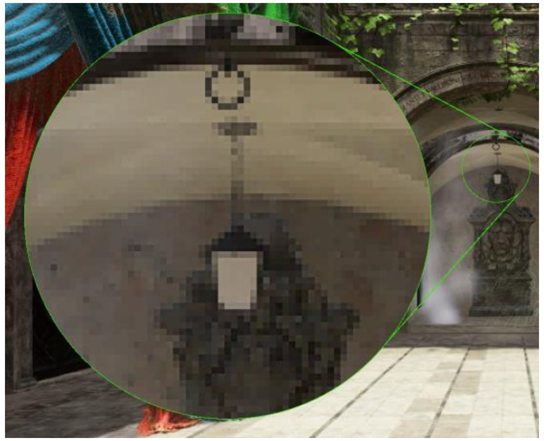
Spend time on **quality** of pixels, instead of quantity!
(Especially if we can reconstruct higher resolution detail)



SPATIAL UPSCALING - GOALS

- Higher perceived visual quality compared to commonly used filters
- Provide overall performance gain
- Easy to integrate
  - Open source
  - Support multiple quality modes

- Support a wide range of hardware
  - Not limited to a certain API
  - Not limited to AMD
    - Not require driver support
  - Not limited to most recent generation
    - Not require special hardware features



Point sampling

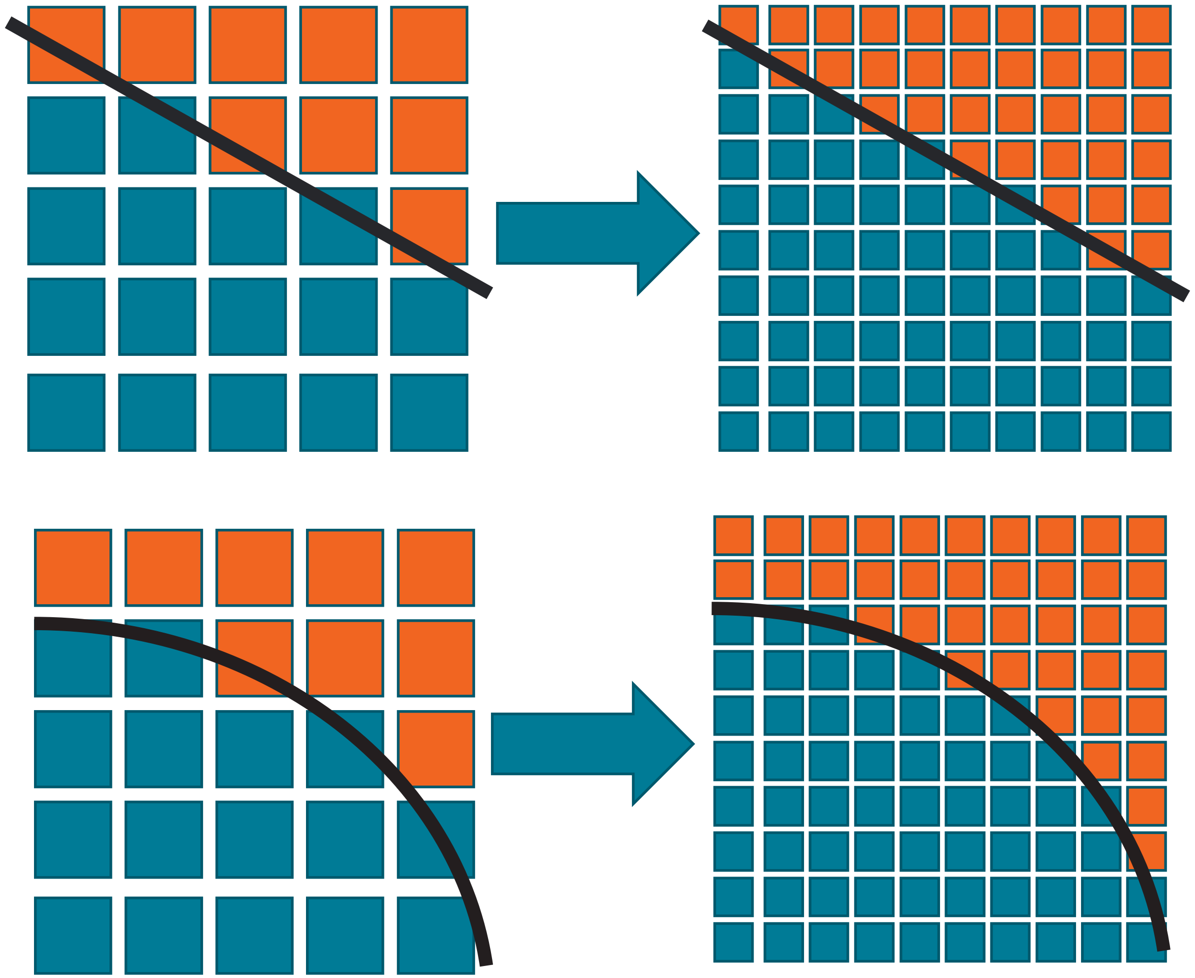
Bilinear sampling

Bicubic sampling

Lanczos sampling

SPATIAL UPSCALING – PROBLEM DESCRIPTION

- Input: any 2D image
- Output: Same image in higher resolution
  - Improve sharpness
  - Retain straight edges
  - Retain color information
    - Don’t add false colors e.g. ringing
- Problem
  - Hard to estimate “real” edge
    - Especially from aliased input
  - Output needs to be temporally stable

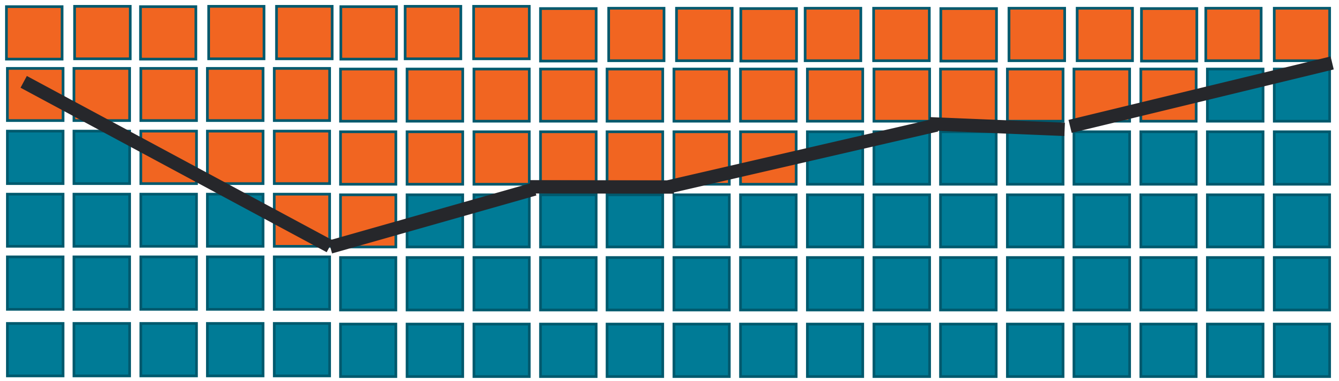


SPATIAL UPSCALING IN A NUTSHELL

- For every pixel:
  - Analyze a fixed sized region surrounding it
  - Detect local features
    - Edge strength and direction
    - Common shapes like corners
  - Adjust filter based on features
  - Filter input data to generate output

Sidenote:

During the final phase of development, Machine Learning was used to assist with identifying relevant features and suggest filters.
We then hand-optimized an algorithmic filter to produce a similar output to make FSR faster and usable on low end hardware.

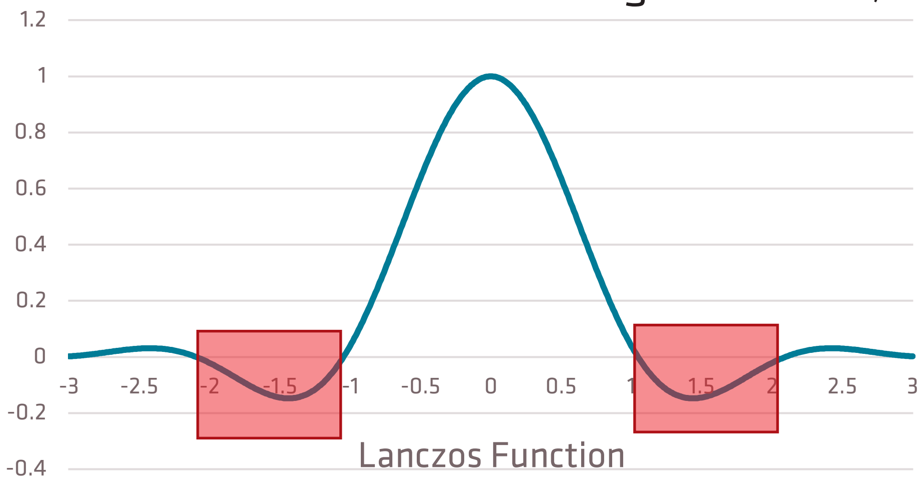


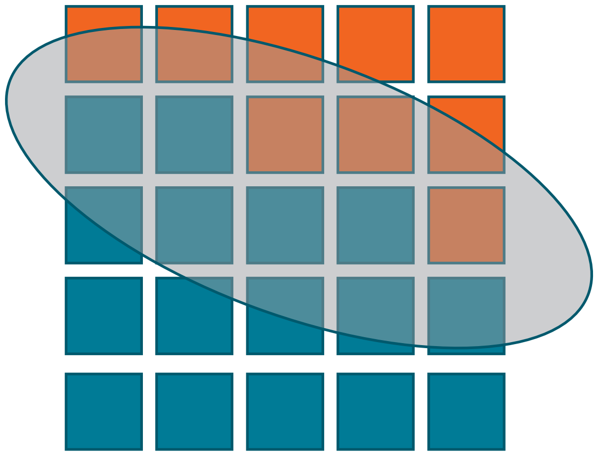
Local view of 5x5 pixels resulting in steps instead of straight line

FSR 1.0 IN A NUTSHELL

- It’s not “just” Lanczos ☺
  - FSR applies a directionally and anisotropically adaptive radial Lanczos

- Filter dimensions adjusted and rotated to match features
- Adjust filter to reduce ringing while keeping sharpness
- Color clamp to ensure output color range matches input color range
  - Reversible tone mapper assumes all values [0:1[
    Values outside caused negative colors, NaN or Inf





FINAL RESULT

Basic Lanczos upscale

FSR 1.0 upscale

FSR 1.0 – RELEASED 2021

- Our best-in-class Spatial Upscaling solution

- Designed for high performance
  - Easy to integrate into games

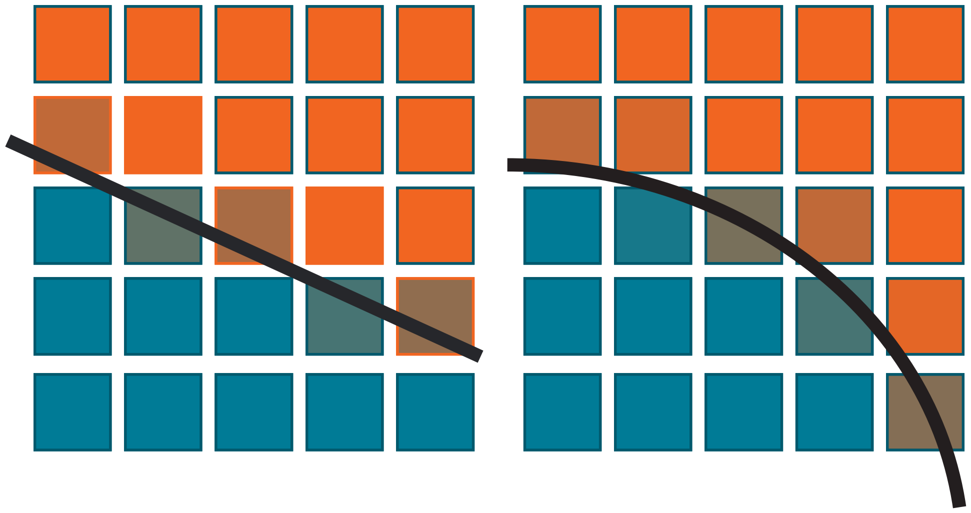
- Open sourced with a permissive MIT license

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

- FSR 1.0 has enjoyed great developer adoption
  - 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

SPATIAL UPSCALING – THE NEED FOR GOOD AA



- Rastered output is undersampled for upscaling
  - Local information not enough to distinguish all cases
    - Moiré patterns – virtually impossible to reconstruct pattern
    - Curves vs corners
    - Step artifacts on straight lines due to limited view
  - Temporal instability on motion
    - Thin objects flickering
    - Slow movement not smooth in high resolution

- AA provides much needed detail information
  - Preferably TAA to keep number of computed samples each frame low
  - Some games successfully use MSAA with FSR 1,
    but it would seem a custom resolve to target resolution could provide even better quality
  - Combining SSAA with spatial upscaling seems a strange choice ☺

- Up next: Combine TAA and upscaling for FSR 2!

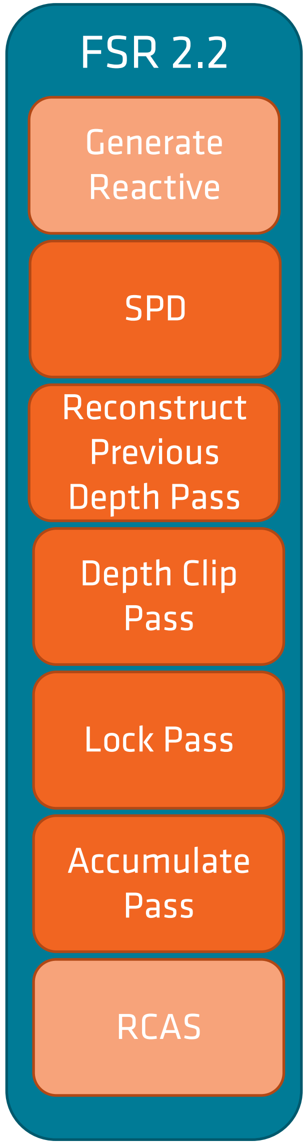
TEMPORAL UPSCALING – GOALS

- Higher perceived visual quality than native without AA
- Overall performance gain

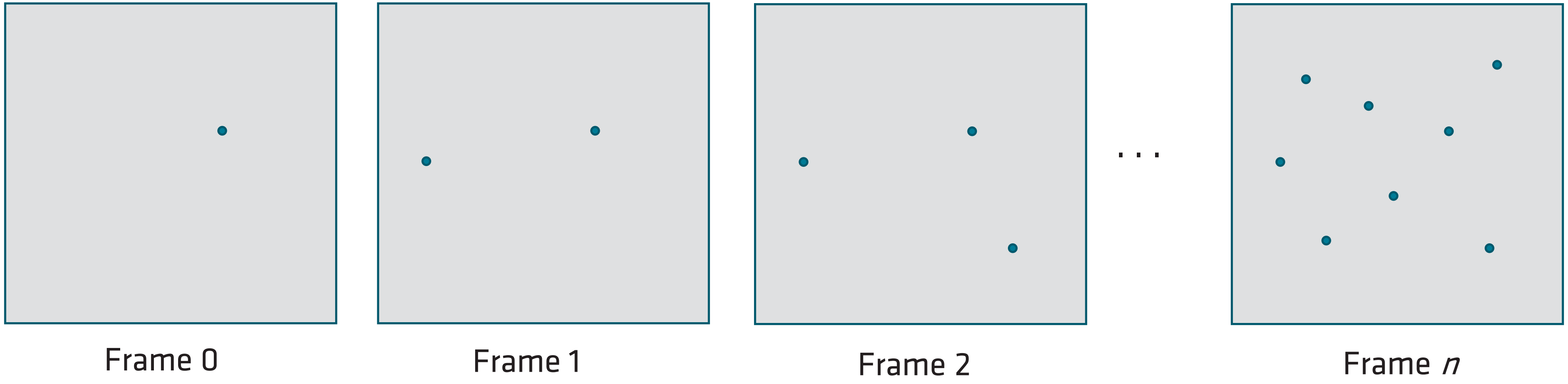
- Ease of integration is still a primary consideration
  - Open source
  - Camera Jitter + Dispatch
  - Requires readable depth buffer and game motion vector data
  - FFX_SDK backend to allow full control over resources and allocations

- Support multiple quality modes
  - Including dynamic resolution scaling

- Working on wide range of hardware
  - Not limited to current GPU generation
    - Not require special hardware features
  - Not limited to AMD
    - Not require driver support
  - Not limited to a specific API



TAAU CAMERA JITTER



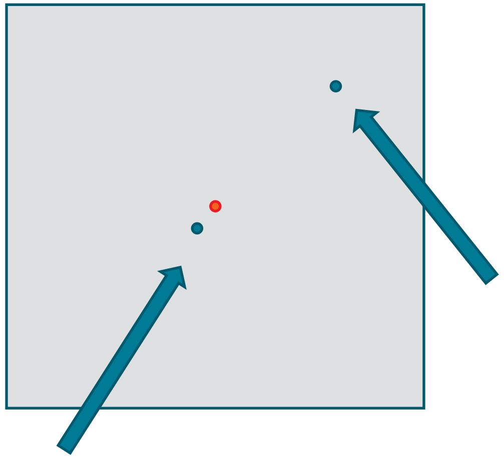
- Camera matrix needs to be modified before rendering to apply jitter
  - Render a different sample within render resolution pixels every frame
  - Upscaling requires increased cycle length
- Jitter sequence quality is key to achieve good quality
  - Good distribution over space and time
  - Halton(2,3) sequence is recommended

FSR 2.2

- Generate Reactive
- SPD
- Reconstruct Previous Depth Pass
- Depth Clip Pass
- Lock Pass
- Accumulate Pass
- RCAS

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 information
- Closer and more recent samples are more important



- Center of the result pixel
- New incoming samples

Frame n : more recent,
high contribution

Frame $n - 1$: close to center,
high contribution

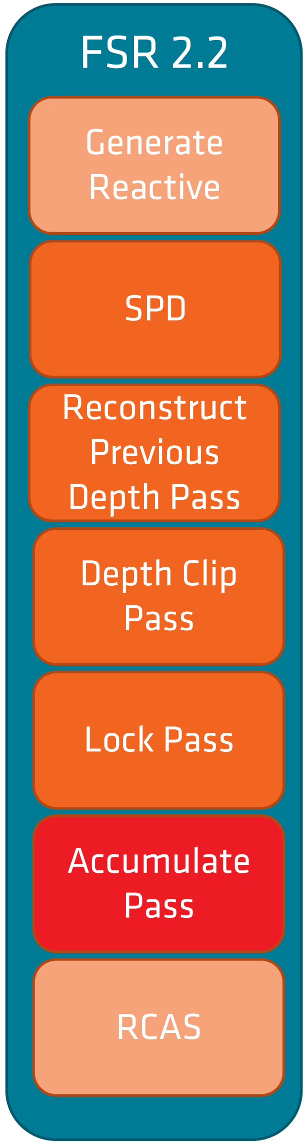
FSR 2.2

- Generate Reactive
- SPD
- Reconstruct Previous Depth Pass
- Depth Clip Pass
- Lock Pass
- Accumulate Pass
- RCAS

LANCZOS, LANCZOS EVERYWHERE

$$Lanczos(x) = \begin{cases} 1 & \text{if } x = 0 \\ (a * \sin(\pi * x) * \sin(\pi * x/a)/(\pi^2 * x^2) & \text{if } -a <= x < a \text{ and } x \mathrel{!=} 0 \\ 0 & \text{otherwise} \end{cases}$$

- Benefits of using Lanczos:
  - Good quality for resampling
  - Flexibility: a can be used to control sharpness
  - While ALU heavy, it can easily be implemented as LUT
    - Depends if shader is memory bandwidth or ALU limited
    - Can't use approximation in reprojection step: this causes noticeable artifacts



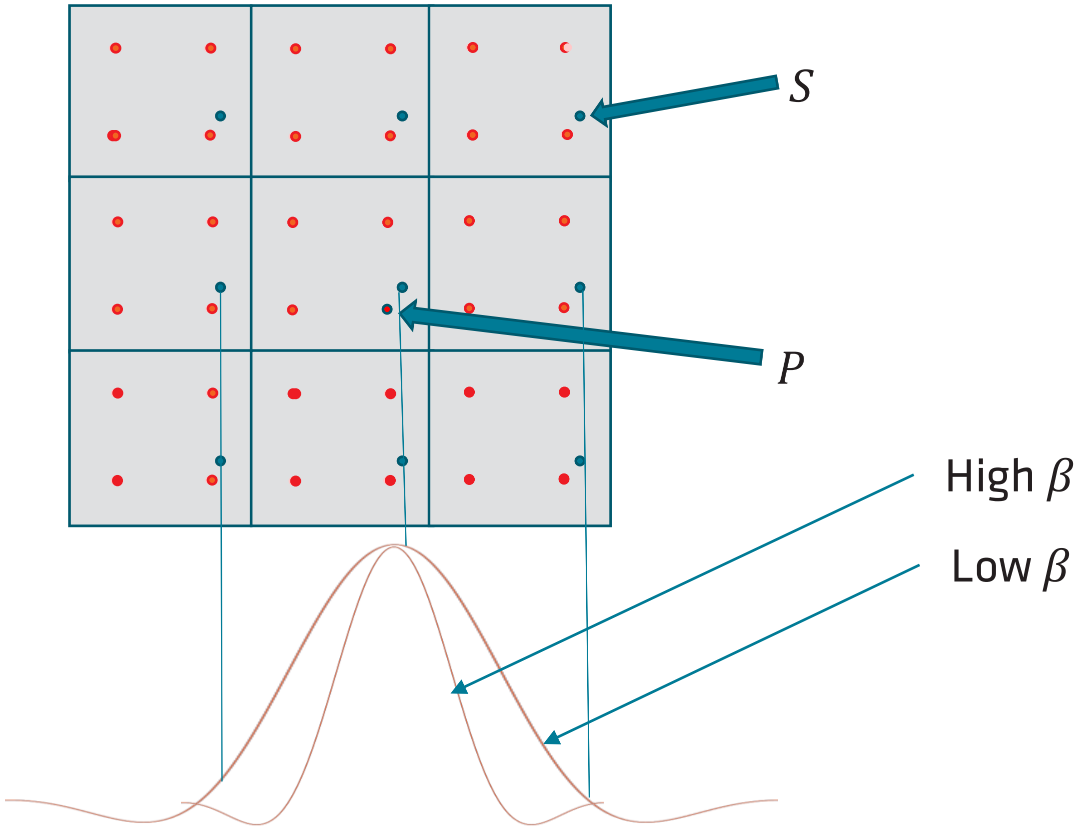
ADDING UPSAMPLING TO THE PICTURE

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

P: high-resolution sample to compute
S: source set of low-resolution samples

$$\omega_s = Lanczos(\beta|PS|)$$

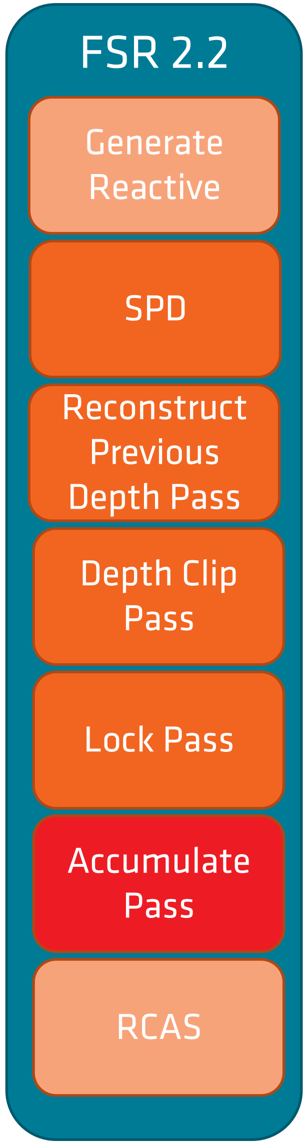
Where β is a bias to affect output sharpness based on temporal stability.



Ω: total weight associated to P

$$\Omega = \sum_s \omega_s$$

$$P = \frac{1}{\Omega} \cdot \sum_s (S \cdot \omega_s)$$

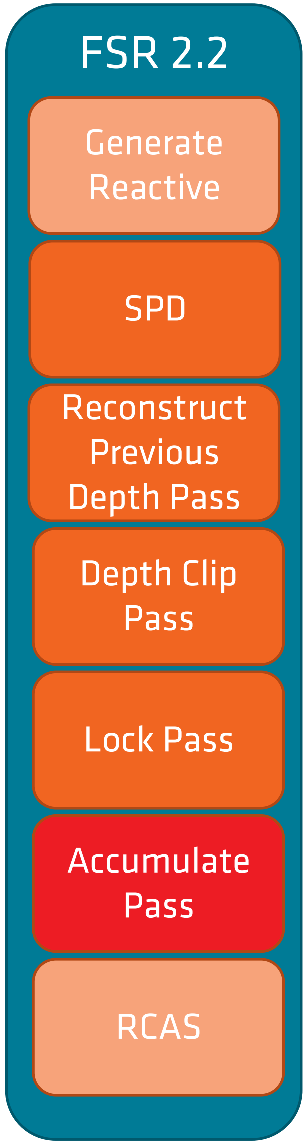


PER PIXEL SAMPLE CONTRIBUTION

- Blending new sample P with the existing accumulated pixel color H.

$$Final\ Color = \alpha P + (1 - \alpha)H$$

- Where $\alpha = \frac{\max(\Omega, 0)}{\tau}$
- And τ is a value estimated each frame based on
  - Disocclusion
  - Velocity
  - Shading change



TEMPORAL UPSCALING – CORNERS EVERYWHERE

- So all is well… Except for snowflakes!
  - Or translucent objects!
  - Or reflections!
  - Or animated textures!

  - And as we don't have changing light conditions!
  - Or moving objects or a moving camera!

- We're golden as long as your game looks like this:

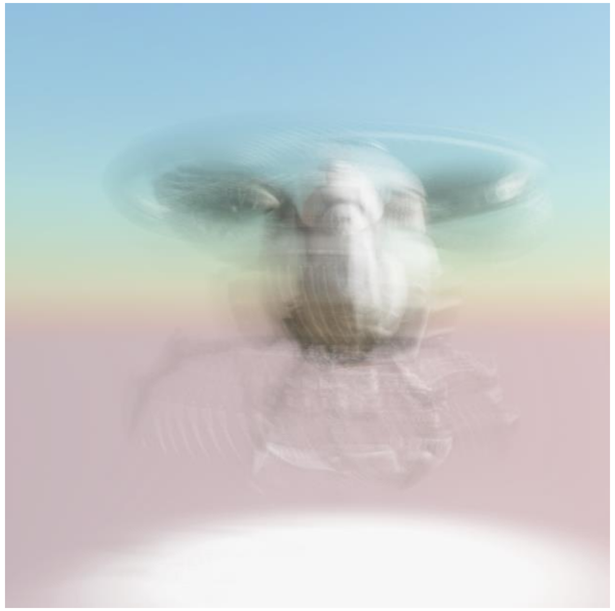
☺

(Snow ghosting initially caused headache for FSR 2 team!)
Screenshot from Deathloop by Arkane Studios

<u>We need to figure out if history data is trustworthy</u>

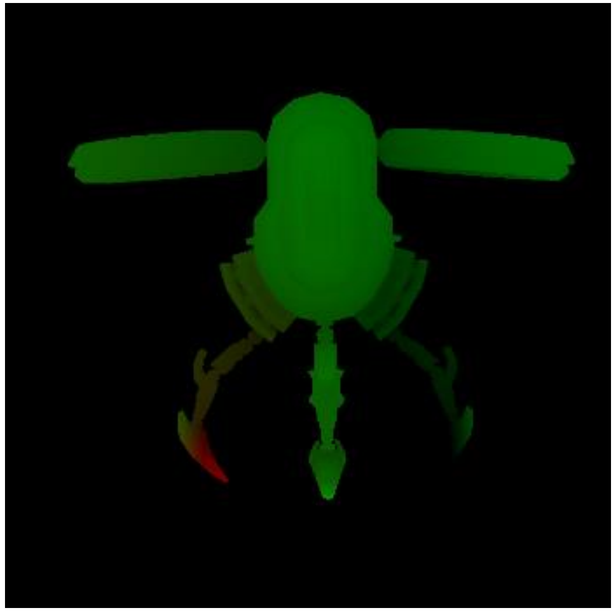
MOTION VECTORS

- A 2D motion vector in screen space that describes how a geometrical sample moves 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 vector of the closest pixel in a 3x3 neighbourhood is used. This is referred to as *dilation*.



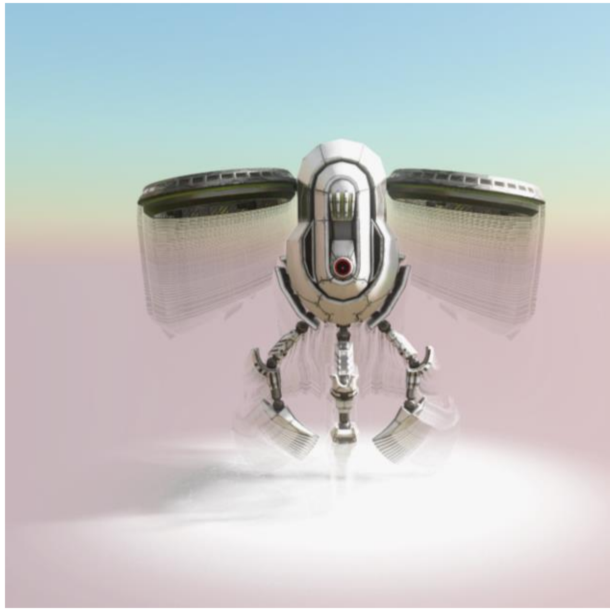
Basic accumulation

+



2D motion vectors

=



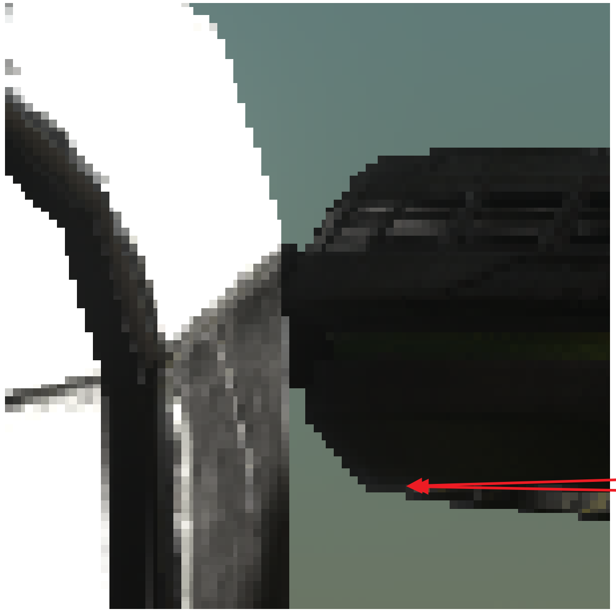
Using motion vectors

FSR 2.2

- Generate Reactive
- SPD
- Reconstruct Previous Depth Pass
- Depth Clip Pass
- Lock Pass
- Accumulate Pass
- RCAS

TEMPORAL GHOSTING

- There are cases where history data is no longer correlated with the new input:
  - Disocclusion
  - Shading change
- Ghosting is seen when we don’t correctly handle those cases.
  - Depth clip and Color clamp are used to fix this



Previous frame

Current frame
(with MVs and disocclusion overlaid)

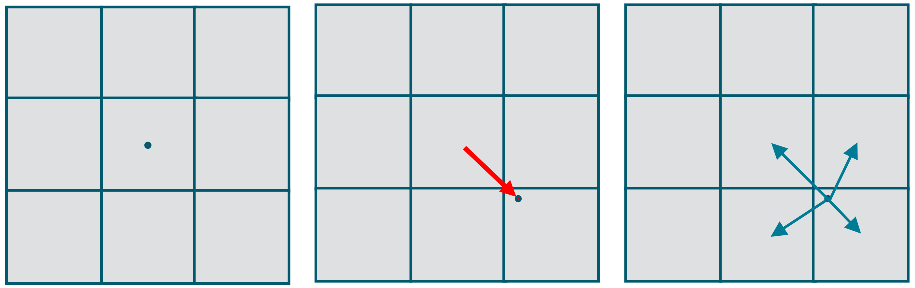
Ghosting

FSR 2.2

- Generate Reactive
- SPD
- Reconstruct Previous Depth Pass
- Depth Clip Pass
- Lock Pass
- Accumulate Pass
- RCAS

RECONSTRUCT PREVIOUS DEPTH

- Reproject depth buffer to its location in last frame
  - The reprojected sample is scattered among all impacted samples (reverse reprojection)
  - The nearest depth is kept when multiple samples hit the same location

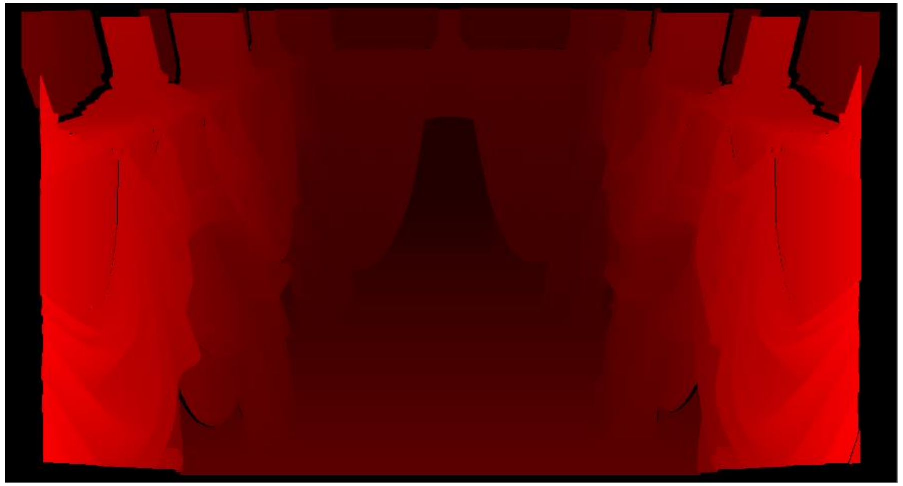


Current frame depth

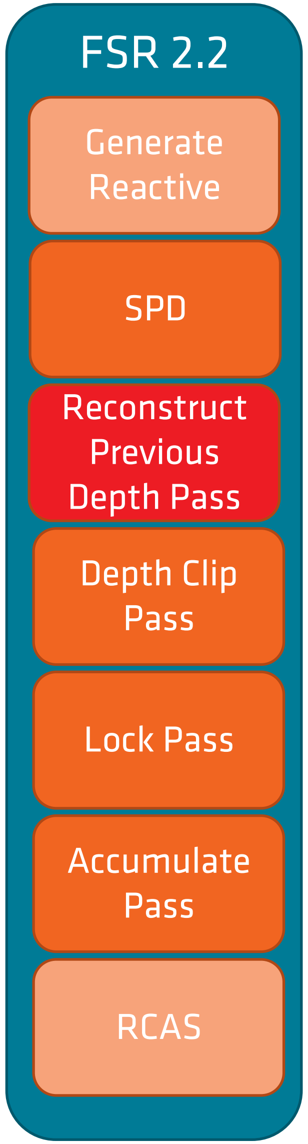
Reproject and scatter depth

- Dilate depth and motion vectors
  - Being conservative since sample locations are not at pixel center
- Compute lock input luma

Current depth



Previous depth reconstructed from current depth

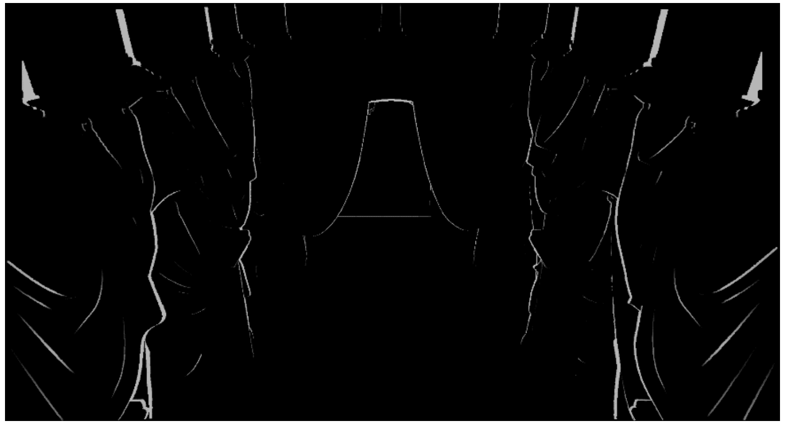


DEPTH CLIP PASS

- Re-Reproject estimated/dilated previous depth buffer to its original location
  - Using 4 bilinear samples
  - Compare current depth to re-reprojected
  - If we detect a significant difference the sample is tagged as disocclusion
  - Disocclusion mask is conservative to ensure stable edges

- Compute `PreparedInputColor`
  - Apply exposure
  - Convert from RGB to YCoCg
  - Store disocclusion mask in Alpha

- Dilate reactive masks
  - Combine them in single 2-channel texture



Disocclusion mask in `preparedInputColor.a`

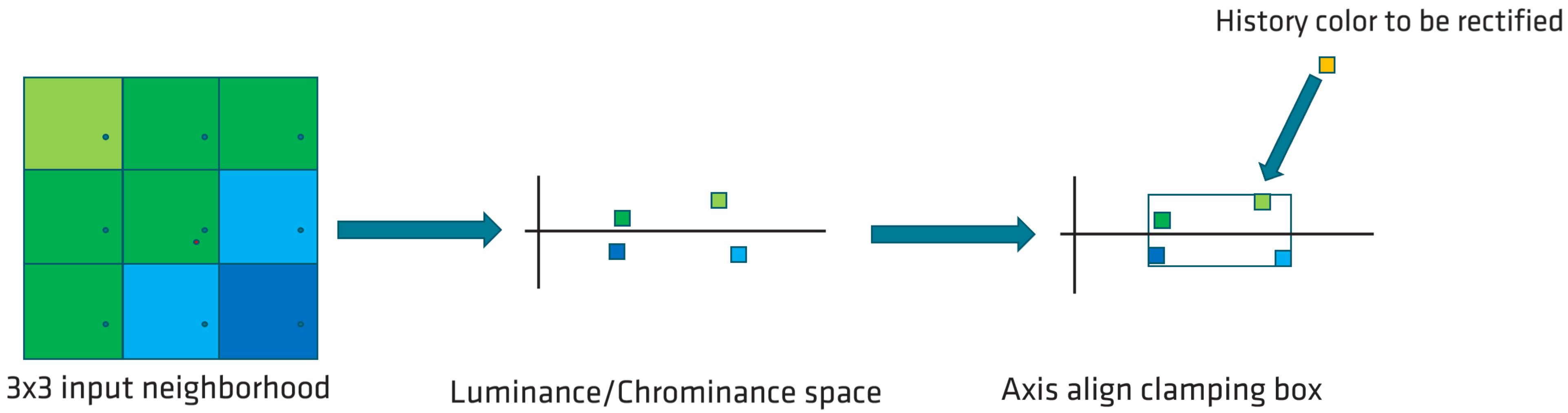
YCoCg backbuffer in `preparedInputColor.rgb`

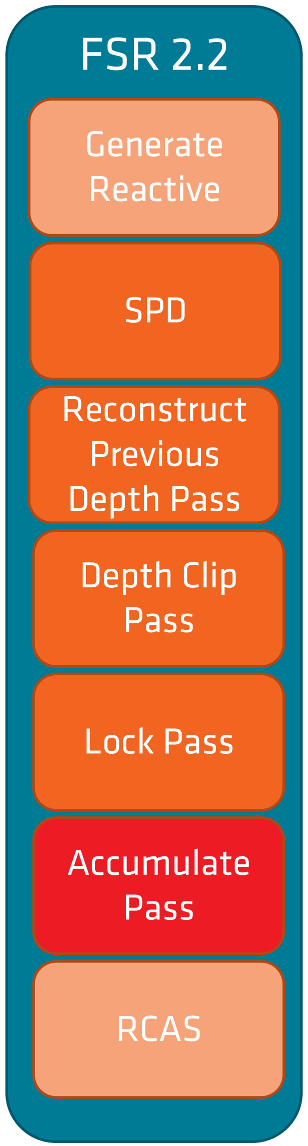
FSR 2.2

- Generate Reactive
- SPD
- Reconstruct Previous Depth Pass
- Depth Clip Pass
- Lock Pass
- Accumulate Pass
- RCAS

ADDING COLOR RECTIFICATION

- History color is clamped to a range based on adjusted and spatially weighted input color values
- Color rectification box is based on current frame information only

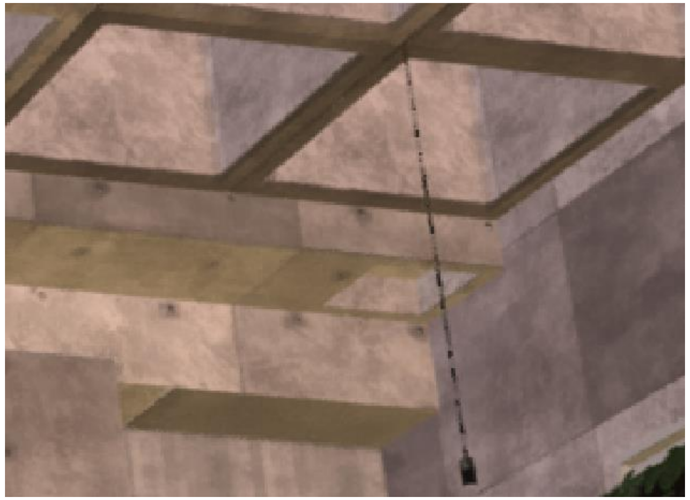




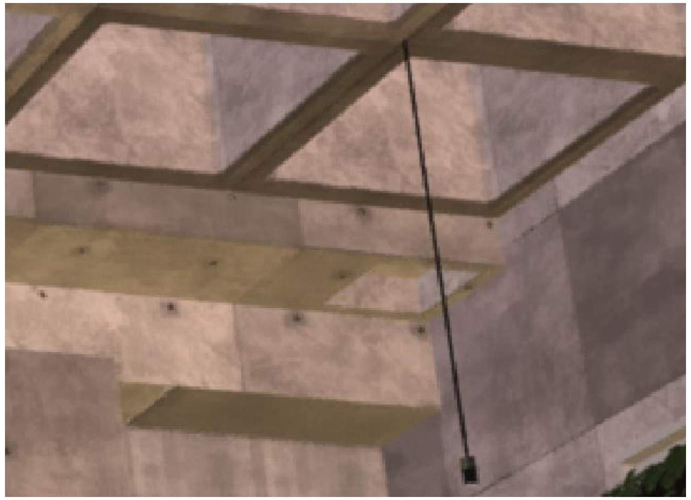
THIN FEATURES

- Thin features are only visible for a few jittered samples over the overall sequence
- Color rectification will take them down as soon as they are not part of the input image
- This results in an unstable image, or very dimmed features

Thin object only partly visible due to undersampling



Unstable and dimmed due to color clamp interfering with accumulation



Applying locks to prevent color clamp on thin features

FSR 2.2

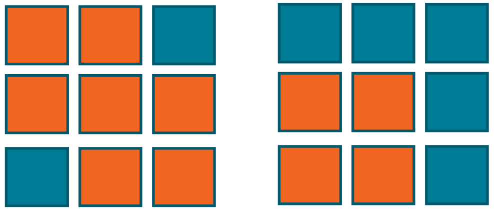
- Generate Reactive
- SPD
- Reconstruct Previous Depth Pass
- Depth Clip Pass
- Lock Pass
- Accumulate Pass
- RCAS

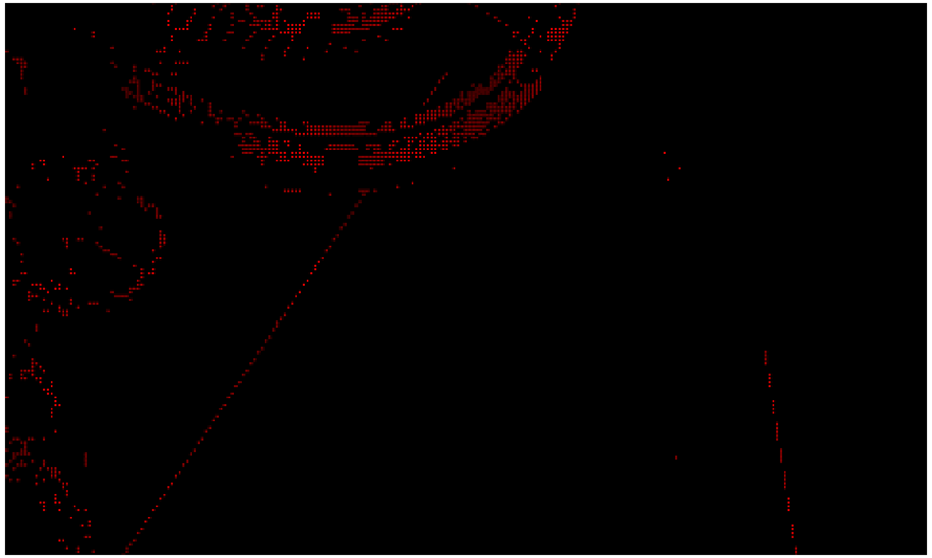
LOCK PASS

- Analyze 3x3 region around each pixel to identify thin features
  - Mark middle and surrounding pixels where luma is similar to middle pixel
  - Create new lock if no 2x2 square of marked pixels exists
  - Detect in render resolution, create lock in display resolution
  - Locked pixels will not get clamped during color rectification

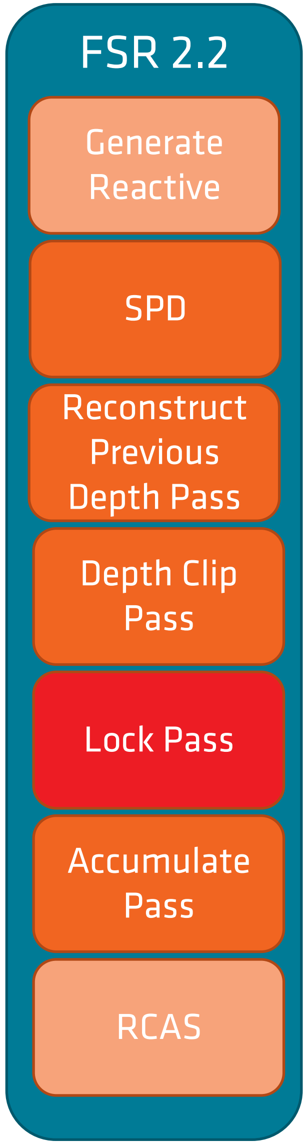
Example patterns creating locks:

Example patterns not creating locks:



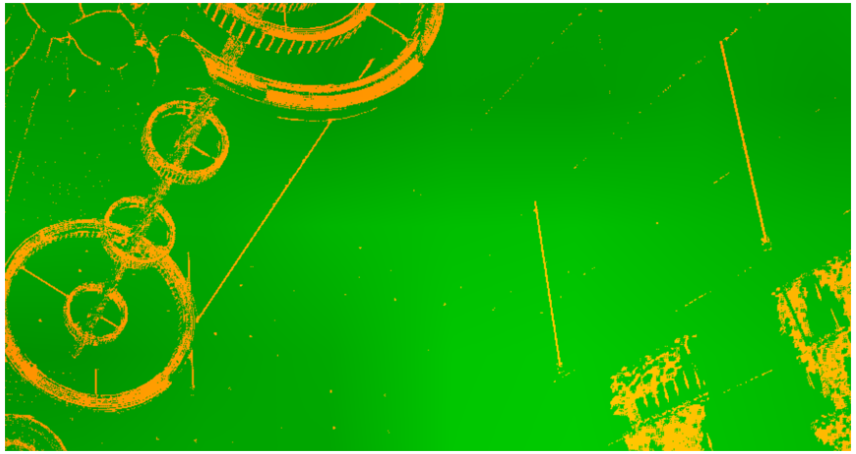


New locks texture

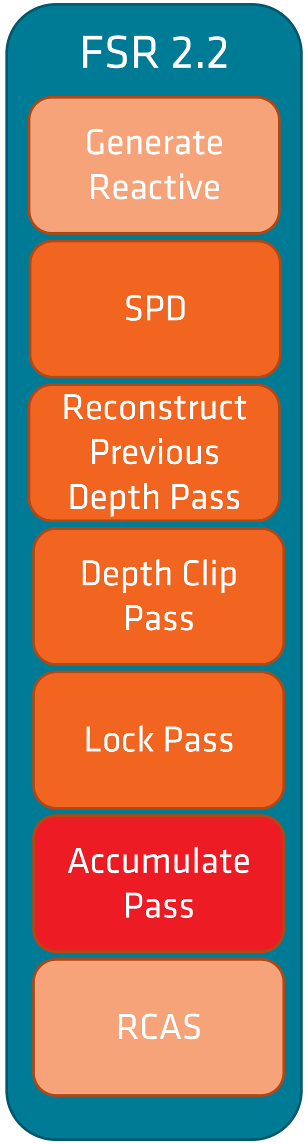


WHEN TO UNLOCK?

- Locks get reprojected every frame
  - Get killed if the thin feature disappears
- Locks decay over time if not updated
  - This can lead to ghosting
- Locks are killed or not trusted using:
  - Disocclusion/depth clip mask
  - Shading change detection (local, low frequency, luminance comparison)
  - Reactive masks
  - Spatial and temporal motion divergence

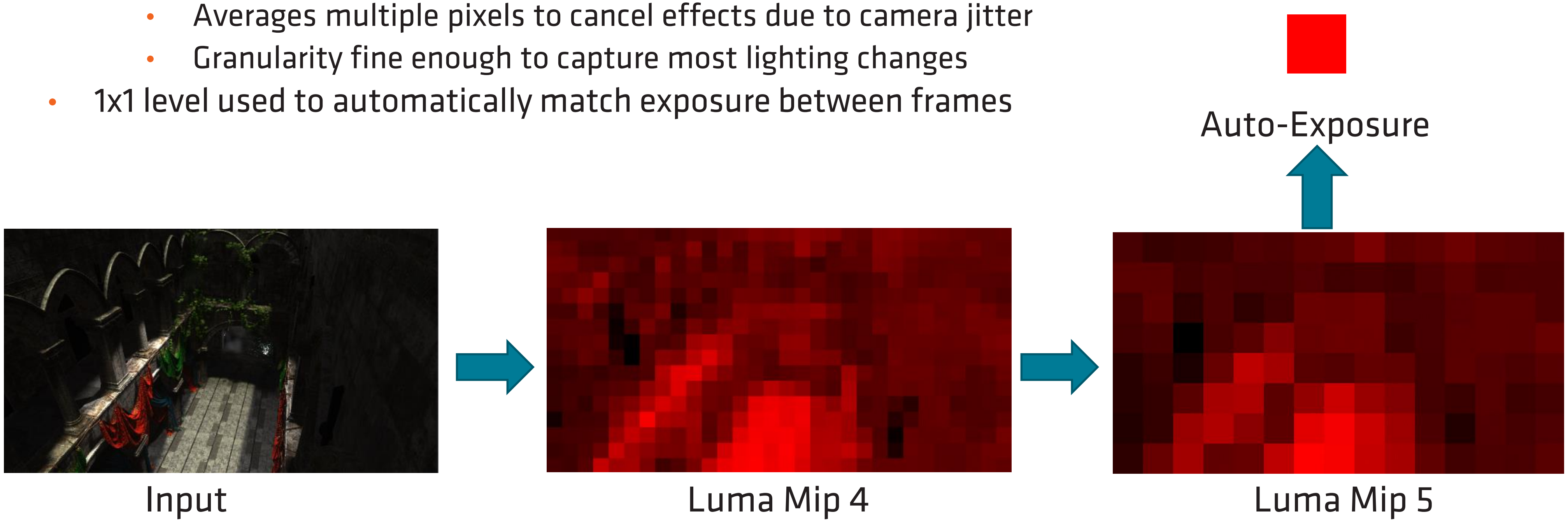


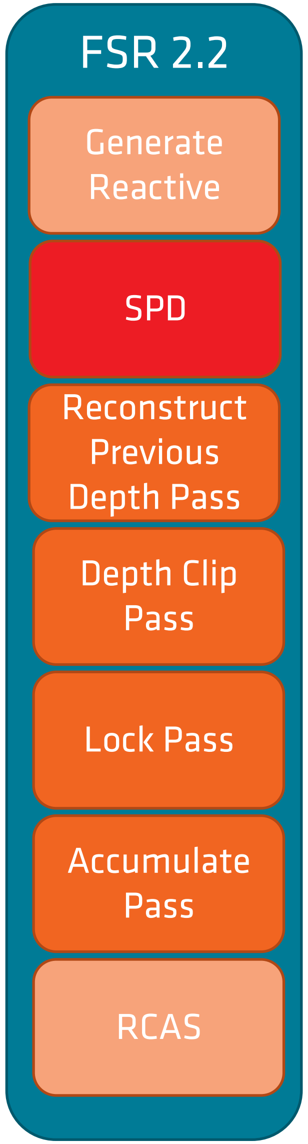
Lock status texture



FSR2 ALGORITHM – BUILD LUMINANCE PYRAMID

- SPD – FidelityFX Single Pass Downsampler
  - Compute luminance and downsample to 4th, 5th and 1x1 level
  - Used to identify lighting changes averaged over larger region
    - Averages multiple pixels to cancel effects due to camera jitter
    - Granularity fine enough to capture most lighting changes
  - 1x1 level used to automatically match exposure between frames



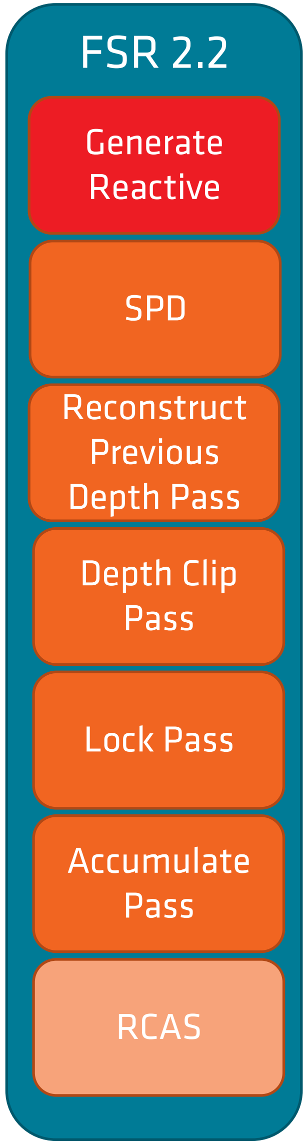


REACTIVE AND TC MASKS – FIXING REMAINING ISSUES

- Reactive and TextureAndComposition masks provide additional input to rectify ghosting
  - Both masks can be provided by the game or generated automatically
  - Automatic generation requires an additional back buffer containing opaque geometry only
  - Generating the textures during rendering is likely to produce better results than autogeneration

Ghosting test scene without masks

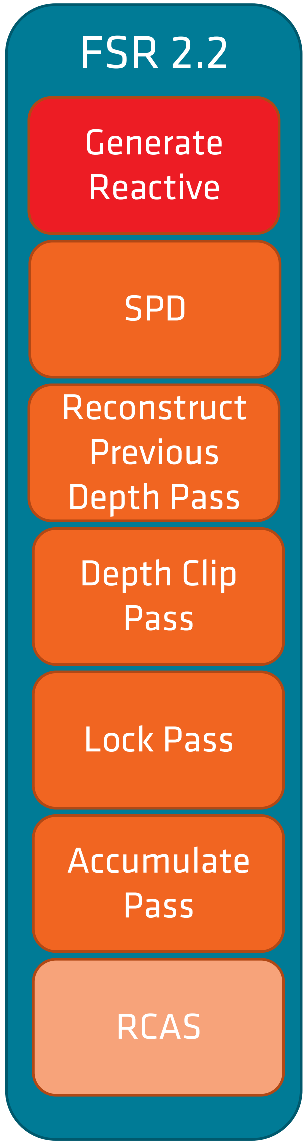
Ghosting test scene with masks applied



REACTIVE AND TC MASKS – FIXING REMAINING ISSUES

- Reactive mask:
  - Marks pixels to reduce history contribution in final blend
  - All values should be significantly below 1, otherwise this will result in instability
  - Examples of which pixel types to mark as reactive:
    - Small particles
    - Highly reflective surfaces (use low values)
    - Animated textures (use low values)

- TextureAndComposition mask:
  - Mask pixels to reduce lock lifetime
  - Examples of which pixel types to mark as reactive
    - Larger translucent objects
    - Highly reflective surfaces
    - Animated textures



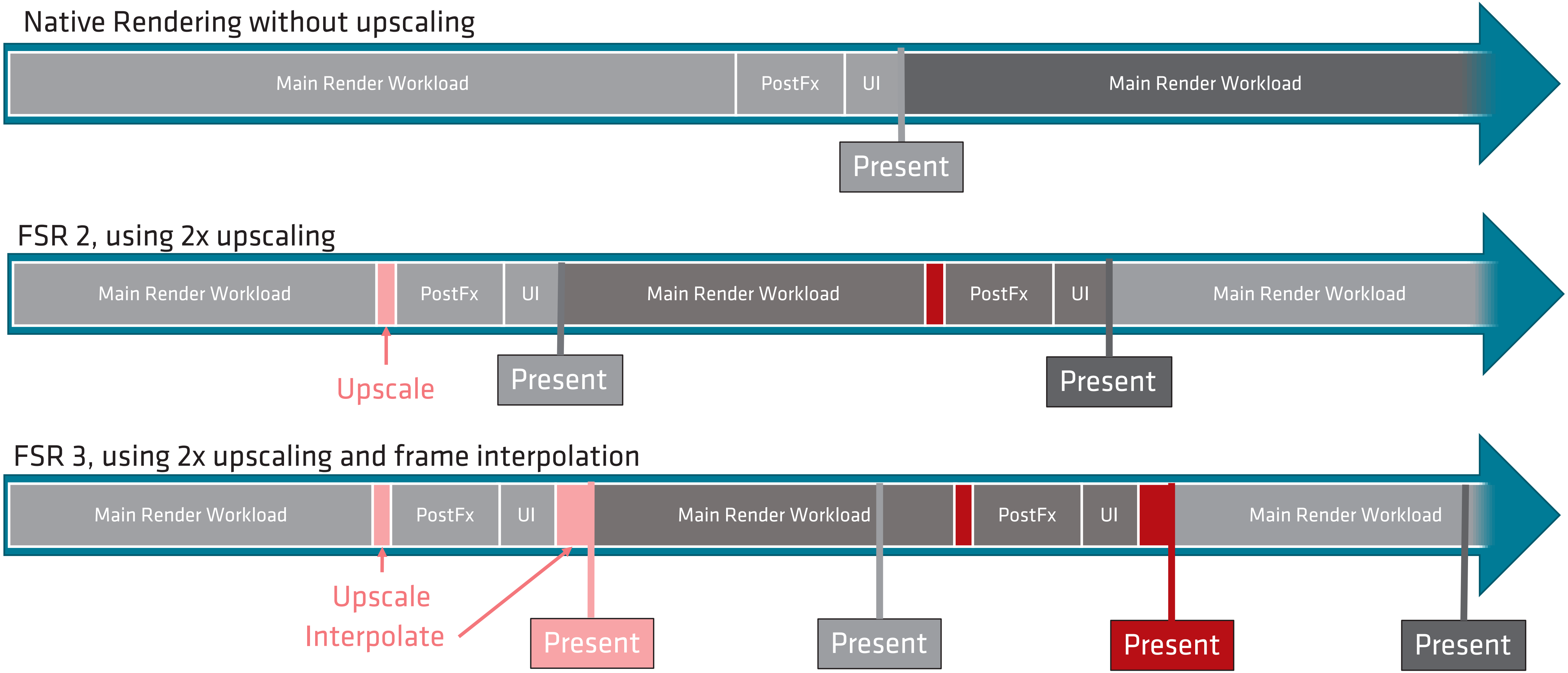
INTRODUCING FSR 3

- With FSR 2 we're already computing more pixels than we have samples in the current frame
  - We could generate even more by introducing interpolated frames!
  - Achieve up to 2x framerate boost in the process

- The good news
  - High probability there will be least one sample for every interpolated pixel
  - No feedback loop: interpolated frame will only be shown once
    - Any interpolation artifact would only remain for one frame

- Challenges
  - We can't rely on color clamping to correct color of outdated samples
  - Non linear motion interpolation is hard with 2D screen space motion vectors
  - Interpolating final frames means all postprocessing and UI needs to get interpolated

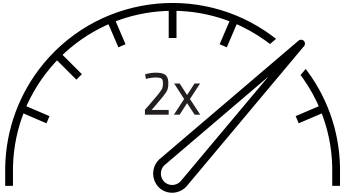
FROM NATIVE RENDERING TO FSR 3



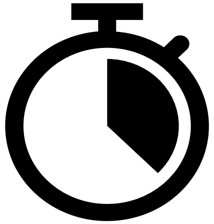
INTRODUCING FSR 3

AMD FidelityFX Super Resolution 3

- FSR 3 combines **resolution upscaling** with **frame interpolation**



Up to 2x performance compared to FSR 2



Latency reduction

FSR 2 ❯❯❯ FSR 3

Easy transition

Permissive license

EARLY LOOK AT FSR 3 TECH (SUBJECT TO CHANGE)

- FSR 3 benefits from synergies between upscaling and interpolation
  - Leverages motion vectors and AMD Fluid Motion to produce interpolated frames
  - Good motion estimation is key for interpolation
  - Additional internal information from FSR 2 can be leveraged

- Latency reduction is a focus area
  - Gamers need both high framerate and the lowest latency possible

- Existing FSR 2 integrations will more easily transition to FSR 3

- MIT license to allow optimal flexibility of integration if needed

CONCLUSION

- We’ve come a long way in the last few years!
  - From spatial upscaling, to temporal, to temporal with frame interpolation
  - Represents decades of R&D innovation
  - Previous technologies are often able to be leveraged in the next one
  - Some challenges remain

- FSR 3 is the next step in the evolution of FidelityFX Super Resolution
  - It will enable smoother gaming experience
  - And simultaneously allow developers to focus more GPU time on visual quality

THANK YOU FOR ATTENDING MY SESSION

Questions?

AMD GPUOpen

AMD FidelityFX Super Resolution

DISCLAIMERS

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's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

Use of third-party marks / products is for informational purposes only and no endorsement of or by AMD is intended or implied. GD-83