AMD EPYC

AMD RYZEN

AMD RADEON

AMD GPUOpen

AMD FidelityFX Super Resolution 3

INTEGRATION OVERVIEW VERSION 3.0.3

(GPUOPEN RELEASE VERSION)

AMD together we advance_

Last revision: 14 December 2023

AMD

FidelityFX

Super Resolution 3

OVERVIEW

WHAT IS AMD FIDELITYFX™ SUPER RESOLUTION 3?

- FidelityFX Super Resolution 3 technology combines **resolution upscaling** with **frame generation**

Resolution upscaling

Image quality similar or better than Native Rendering

Support for any area scale factor between 1X and 9X (including DRS)

New FSR 3 "Native AA" mode

Frame Generation

Insert interpolated frames for smoother results

Works in both GPU and CPU-limited situations

Latency reduction

Keep latency down as much as possible

Frame pacing for regular frame rates

Variable Refresh Rate support

Ease of integration

Easy transition from FSR 2 to FSR 3

Intuitive API

DX12 and UE5 support

Open source

Source code provided on GPUOpen under an MIT license

Highly optimized

Hand-optimized for great performance across mid to high-range GPUs

Total performance increase of up to **4x** compared to Native Rendering!

AMD

FidelityFX

Super Resolution 3

API INTEGRATION GUIDE

FSR 3 INTEGRATION GUIDE - OVERVIEW

- FSR 3 includes both upscaling and frame generation
- FSR 2 is therefore **superseded** by FSR 3
- One needs only integrate FSR 3 to benefit from upscaling and frame generation
- Frame generation in FSR 3 is **optional** – may be disabled if only upscaling is desired
  - This will effectively revert to FSR 2 behavior
  - Specify if Frame Generation is not needed on Context Create: this results in memory savings
- FSR 3 API **requires** an FSR 3 quality mode to be selected for FSR 3 Frame Generation

- Games with existing FSR 2 integrations: **replace** FSR 2 with FSR 3
  - This is the simpler upgrade path
- Games without existing FSR 2 integrations: integrate FSR 3 **only** (not FSR 2)

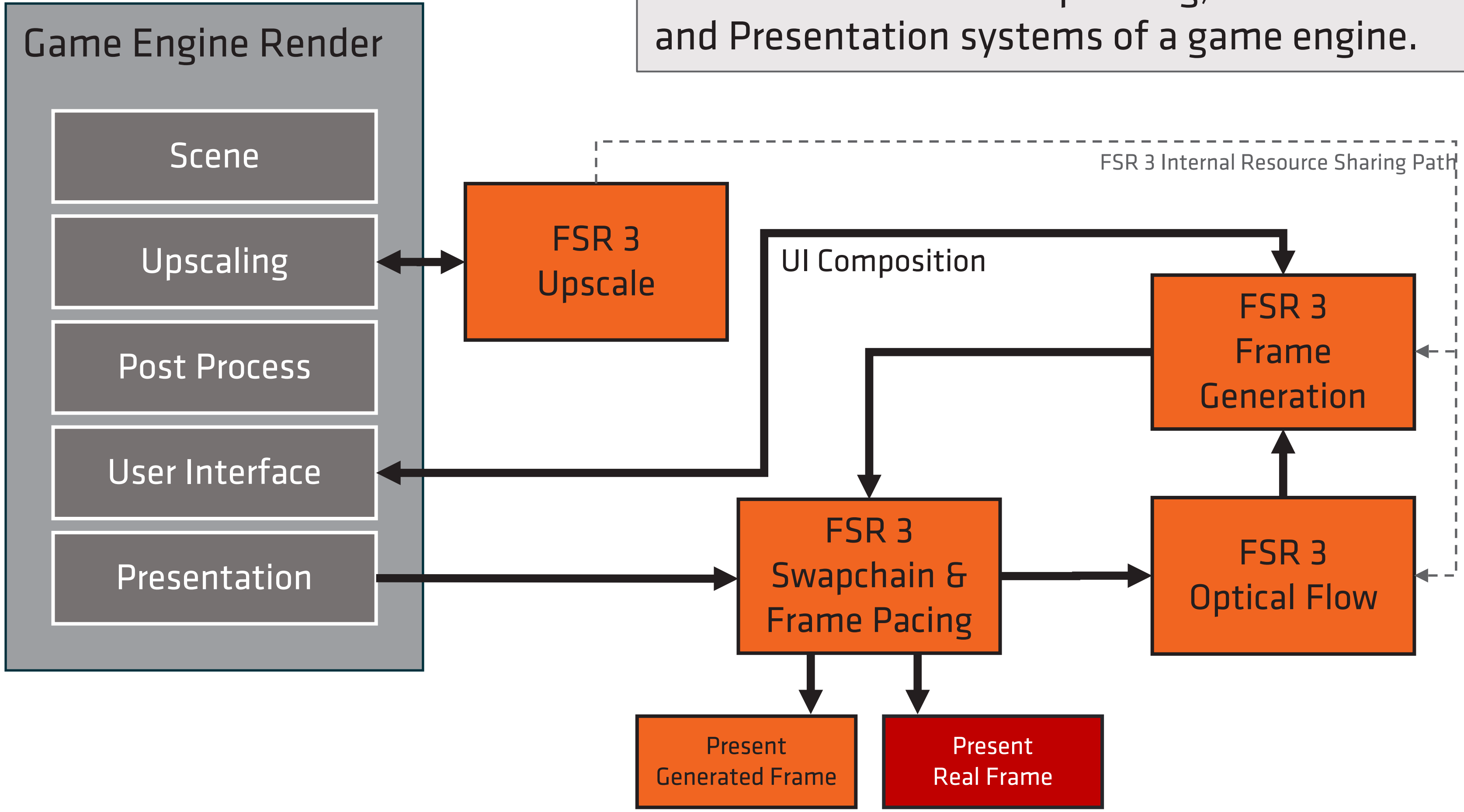
REQUISITE FOR SUCCESSFUL FSR 3 INTEGRATION

- Ensure that your game has a high-quality FSR 3 upscale-only implementation first!
  - Correct use of jittering pattern
  - Correct placement of post-process operations
  - Correct use of Reactive mask
  - Correct use of Transparency & composition mask
  - Correct setting of mip-bias for samplers
  - See FSR 2 documentation on Github
- **A sub-optimal integration of the upscale component will carry over any upscaling artefacts to interpolated frames!**

FSR 3 INTEGRATION STEPS (NATIVE DIRECTX® 12 VERSION)

FSR 3 DATA FLOW

FSR 3 Interfaces with Upscaling, User Interface and Presentation systems of a game engine.



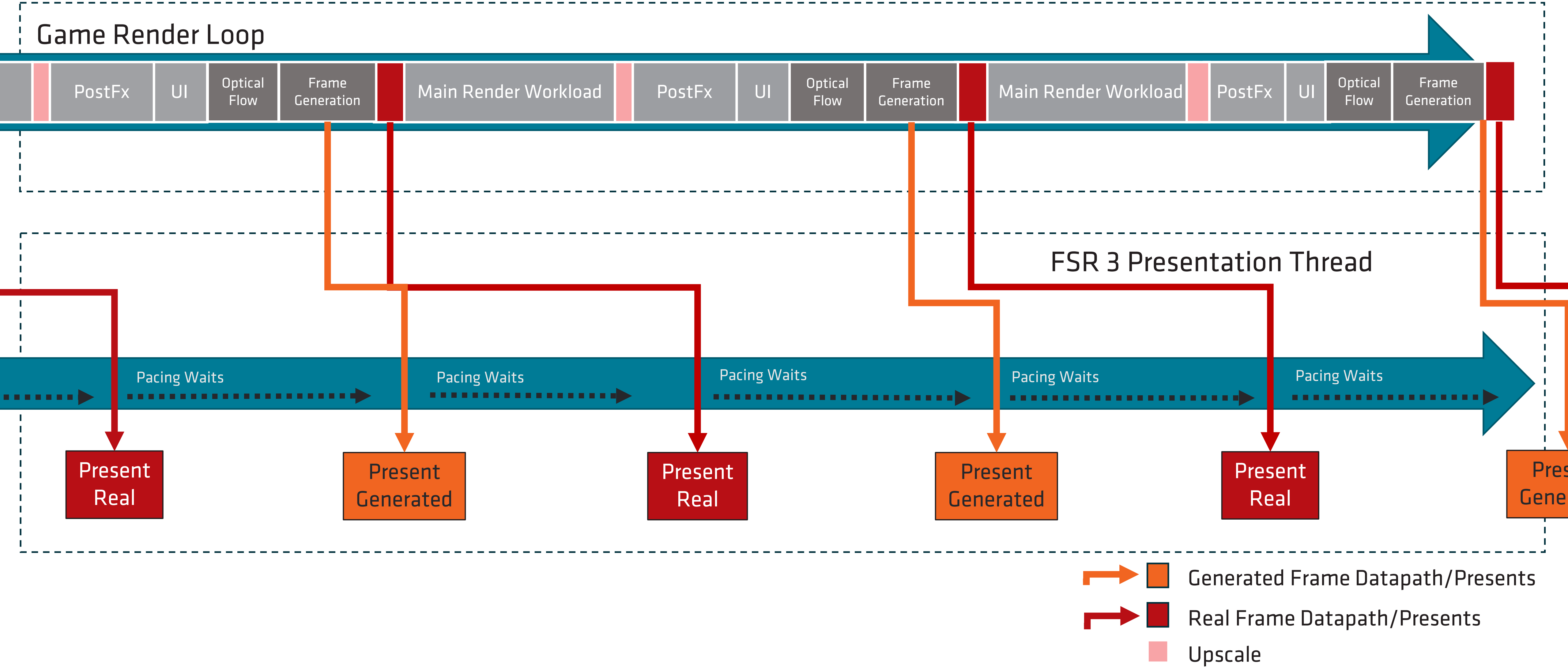
FSR 3 NATIVE DX12 INTEGRATION – BASIC STEPS

- Step 1 - Upscaling
  - FSR 3 integration is similar to FSR 2 for upscaling
    - If FSR 2 already is supported: Replace FSR 2 upscale with FSR 3 upscale
- Step 2 - Swapchain
  - Replace or create your swap chain using FrameInterpolationSwapChain
    - Implementing IDXGISwapChain interface
    - Handling frame generation, frame pacing and present
  - This will internally create
    - CPU threads for frame pacing calculations and real present/UI callbacks
    - A GPU queue to handle presents and another for interpolation
- Step 3 – Frame Generation
  - FSR 3 frame generation requires some additional input on setup and dispatch, and swapchain handling.
    - Start off with Async disabled to validate integration quality.
- Step 4 – UI Handling
  - Three different UI composition options: (note: this step is REQUIRED for integration)
    - **UI texture**: UI is stored in another RT on top of generated frames.
    - **Callback**: User calls an FSR 3 function where UI can rendered on top of the generated frame.
    - **HUDLess**: Identify UI by comparing the game frame without UI to the game frame with UI.

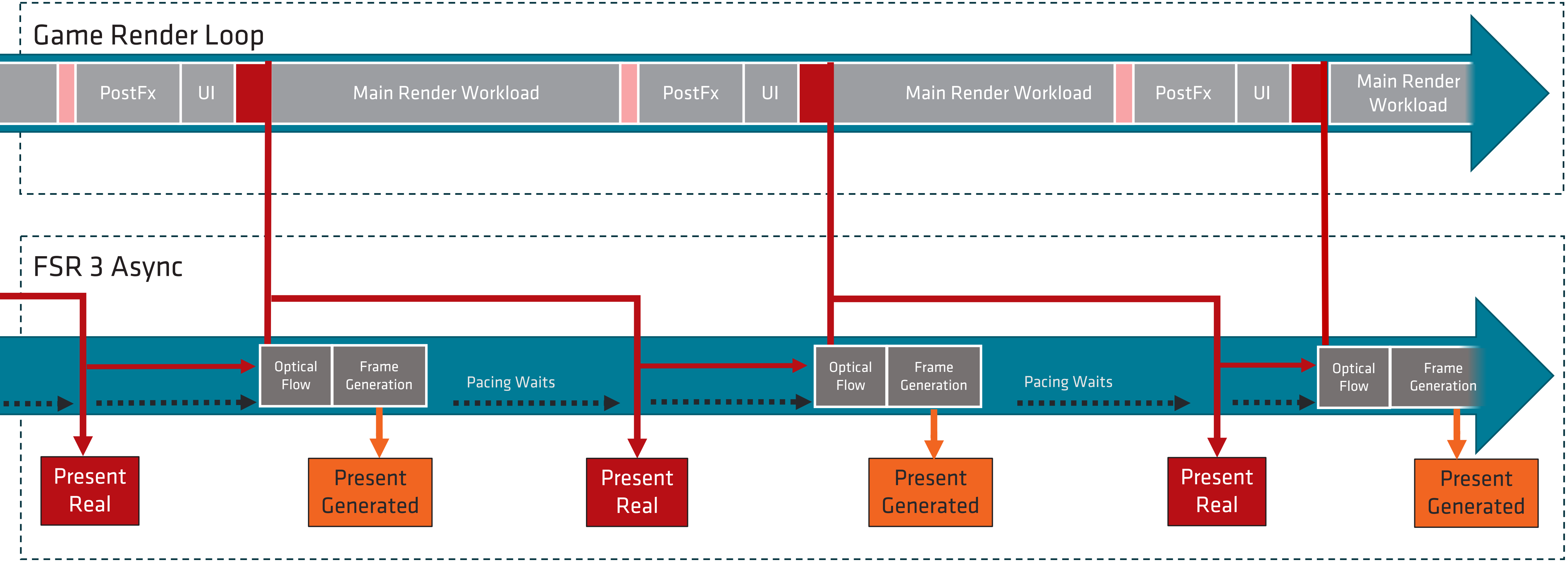
FSR 3 OPTICAL FLOW AND FRAME GENERATION WORKLOADS

- The Optical Flow and Frame Generation workloads can be either run on the **Presentation** Queue provided by the game, or an **Async** Queue provided by the FrameInterpolationSwapchain.
  - To enable Async queue the FSR 3 context must be created with flag FFX_FSR3_ENABLE_ASYNC_WORKLOAD_SUPPORT, and ffxConfigureFrameGeneration called with allowAsyncWorkloads = TRUE
- We recommend integrating with Presentation Queue use first, and then testing Async Queue use.
  - Async can introduce sync issues with frames that call Upscale, but do not Present(), which means accurate tracking and disabling/enabling frame generation is essential. Presentation Queue use can minimise these issues.
  - If a fence is required to indicate any HUDLess UI resource is consumed, this can be done on the Presentation Queue, but not on the Async Queue.
- Async Queue use can increase performance if the workload can be overlapped with early next frame rendering.
- Async Queue use will increase memory usage, as some resources need to get double-buffered so Frame Generation can execute in parallel to the next frame being rendered

FSR 3 – PRESENT QUEUE UPSCALING AND FRAME GENERATION PIPELINE



FSR 3 – ASYNC UPSCALING AND FRAME GENERATION PIPELINE



🟧 Generated Frame Datapath/Presents

🟥 Real Frame Datapath/Presents

Upscale

FSR 3 NATIVE DX12 INTEGRATION - SWAPCHAIN

- Swapchain handling has multiple possibilities for integration.
- The Cauldron sample shows a situation where there is already a DXGI swapchain present, and this swapchain is **removed and replaced** with the FrameInterpolation swapchain object FSR 3 creates
  - This requires games are **not** using Exclusive Fullscreen.
  - It recommended to replace swapchain creation with creation of a FrameInterpolation swapchain in the engine and always use the proxy swapchain, even when frame generation is disabled
- For some titles this may not be ideal. Another option is to always use the FrameInterpolationSwapchain when FSR 3 Upscaling is enabled.
  - It is acceptable to require a game restart for this to take effect.
- The FrameInterpolation swapchain will not perform frame generation until it's registered using **ffxFsr3ConfigureFrameGeneration**.
  - **When frame generation is disabled the FrameInterpolation swapchain will still handle UI composition.**
- If another system is in place which also hooks into swap chain, undefined behaviors may mean a game restart is required to enable FSR 3.

FSR 3 NATIVE DX12 INTEGRATION – SWAPCHAIN CREATION OPTIONS

Create the FSR 3 Frame Generation Swapchain using one of the following functions

To create a swap chain, when current integration uses `IDXGIFactory::CreateSwapChain`

```
        ffxCreateFrameinterpolationSwapchainDX12(const DXGI_SWAP_CHAIN_DESC* desc, ID3D12CommandQueue* queue,
                                                 IDXGIFactory* dxgiFactory, FfxSwapchain& outGameSwapChain);
```

ffxCommandQueue – an object containing the Graphics command queue to use. When Async Compute workloads are disabled, this queue is used for Frame Generation Workloads.

To create a swap chain, when current integration uses `IDXGIFactory2::CreateSwapChainForHwnd`

```
        ffxCreateFrameinterpolationSwapchainForHwndDX12(HWND hWnd, const DXGI_SWAP_CHAIN_DESC1* desc1,
                                                 const    DXGI_SWAP_CHAIN_FULLSCREEN_DESC* fullscreenDesc, ID3D12CommandQueue* queue,
                                                 IDXGIFactory* dxgiFactory, FfxSwapchain& outGameSwapChain);
```

ffxCommandQueue – an object containing the Graphics command queue to use.

To replace an existing IDXGI swap chain, use:

```
        ffxReplaceSwapchainForFrameinterpolationDX12(FfxCommandQueue gameQueue, FfxSwapchain& gameSwapChain);
```

ffxSwapChain – an object containing the current DXGI Swapchain, which will be released and replaced with a new Frame Generation swapchain, using same Descriptors.

Note: If the game supports Exclusive Fullscreen then this path is ***not recommended.***

FSR 3 NATIVE DX12 INTEGRATION - INPLACE SWAPCHAIN CAULDRON SUPPLEMENTAL

```
// Take control over the engine swapchain: get the swapchain and then set to NULL in engine
IDXGISwapChain4* dxgiSwapchainEngine = cauldron::GetFramework()->GetSwapChain()->GetImpl()->DX12SwapChain();
dxgiSwapchainEngine->AddRef();

// Save desc state if we need to return to non FSR3 swapchain later
dxgiSwapchainEngine->GetDesc(&gameSwapChainDesc);
dxgiSwapchainEngine->GetDesc1(&gameSwapChainDesc1);
dxgiSwapchainEngine->GetFullscreenDesc(&gameFullscreenDesc);

// Create the FFX FrameInterpolationSwapchain
FfxSwapchain         ffxSwapChain = ffxGetSwapchainDX12(dxgiSwapchainEngine);
// make sure engine pEngineSwapchain is not holding a ref to real swapchain
FrameworkPtr()->GetSwapChain()->GetImpl()->SetDXGISwapChain(nullptr);


FfxCommandQueue ffxGameQueue = ffxGetCommandQueueDX12(DevicePtr()->DX12CmdQueue(CommandQueue::Graphics));
ffxReplaceSwapchainForFrameinterpolation(ffxGameQueue, ffxSwapChain);

// Set frameinterpolation swapchain to engine
IDXGISwapChain4* frameinterpolationSwapchain = ffxGetDX12SwapchainPtr(ffxSwapChain);
FrameworkPtr()->GetSwapChain()->GetImpl()->SetDXGISwapChain(frameinterpolationSwapchain);
// Framework swapchain adds to the refcount, so we need to release the swapchain here
frameinterpolationSwapchain->Release();

// If app is handling Alt-Enter manually we need to update the window association after creating a different swapchain
IDXGIFactory7* factory = nullptr;
if (SUCCEEDED(frameinterpolationSwapchain->GetParent(IID_PPV_ARGS(&factory))))
{
   factory->MakeWindowAssociation(FrameworkPtr()->GetHWND(), DXGI_MWA_NO_WINDOW_CHANGES);
   factory->Release();
}
GetFramework()->GetSwapChain()->SetHDRMetadataAndColorspace();
```

FSR 3 NATIVE DX12 INTEGRATION – FSR 3 CONTEXT CREATION

- Internally, FSR 3 is implemented as multiple effects with separate backends. This is due to them occupying multiple areas of the render pipeline, and this ensures thread and data lifetime safety.

- In the sample we hold the backends and provide them in the FSR 3 context creation parameters as required.

```
// Setup Cauldron FidelityFX interface.
if (!ffxBackendInitialized_)
{
    FfxErrorCode errorCode = 0;

    int effectCounts[] = {1, 1, 2};
    for (auto i = 0; i < FSR3_BACKEND_COUNT; i++)
    {
        const size_t scratchBufferSize = ffxGetScratchMemorySize(effectCounts[i]);
        void*        scratchBuffer     = calloc(scratchBufferSize, 1);
        errorCode |= ffxGetInterface(&ffxFsr3Backends_[i], GetDevice(), scratchBuffer, scratchBufferSize, FFX_FSR3_CONTEXT_COUNT);
    }

    ffxBackendInitialized_ = (errorCode == FFX_OK);
    FFX_ASSERT(ffxBackendInitialized_);

    m_InitializationParameters.backendInterfaceSharedResources    = ffxFsr3Backends_[FSR3_BACKEND_SHARED_RESOURCES];
    m_InitializationParameters.backendInterfaceUpscaling          = ffxFsr3Backends_[FSR3_BACKEND_UPSCALING];
    m_InitializationParameters.backendInterfaceFrameInterpolation = ffxFsr3Backends_[FSR3_BACKEND_FRAME_INTERPOLATION];
}
```

FSR 3 NATIVE DX12 INTEGRATION – FSR 3 CONTEXT CREATION

```
// Setup FidelityFX backend
const ResolutionInfo& resInfo                             = GetFramework()->GetResolutionInfo();
m_InitializationParameters.maxRenderSize.width            = resInfo.RenderWidth;
m_InitializationParameters.maxRenderSize.height           = resInfo.RenderHeight;
m_InitializationParameters.upscaleOutputSize.width        = resInfo.DisplayWidth;
m_InitializationParameters.upscaleOutputSize.height       = resInfo.DisplayHeight;
m_InitializationParameters.displaySize.width              = resInfo.DisplayWidth;
m_InitializationParameters.displaySize.height             = resInfo.DisplayHeight;
m_InitializationParameters.flags                          = FFX_FSR3_ENABLE_DEPTH_INVERTED | FFX_FSR3_ENABLE_DEPTH_INFINITE |
                                                            FFX_FSR3_ENABLE_HIGH_DYNAMIC_RANGE | FFX_FSR3_ENABLE_AUTO_EXPOSURE;
#if defined(_DEBUG)
 m_InitializationParameters.flags |= FFX_FSR3_ENABLE_DEBUG_CHECKING;
 m_InitializationParameters.fpMessage = &FSR3RenderModule::FfxMsgCallback;
#endif  // #if defined(_DEBUG)

// Async OF+FI can allow for more performance, however, requires careful state management. Validate without Async first.
if (m_EnableAsyncCompute) {
  m_InitializationParameters.flags |= FFX_FSR3_ENABLE_ASYNC_WORKLOAD_SUPPORT;
}
m_InitializationParameters.backBufferFormat = GetFfxSurfaceFormat(GetFramework()->GetSwapChain()->GetSwapChainFormat());

// create the context.
FfxErrorCode errorCode = ffxFsr3ContextCreate(&m_FSR3Context, &m_InitializationParameters);
```

FSR 3 NATIVE DX12 INTEGRATION – FSR 3 CONFIGURE

- **To be called once a frame, before Frame Generation dispatch.**
- Includes enable bit for frame Generation
- Provide the swapchain
- Provide the Frame Generation Callback, if using it (manually dispatch frame generation if not)
  - **`frameGenerationCallback`**
- Provide any UI composition resources/functions
  - “Hudless Scene” resource **`HUDLessColor`**
  - UI Composition callback function **`presentCallback`**

```
// configure frame generation
m_FrameGenerationConfig.frameGenerationEnabled  = m_FrameGenEnabled;
m_FrameGenerationConfig.frameGenerationCallback = ffxFsr3DispatchFrameGeneration;
m_FrameGenerationConfig.swapChain               = ffxSwapChain;
m_FrameGenerationConfig.presentCallback         = (s_uiRenderMode == UI_CALLBACK) ? UiCompositionCallback : nullptr;
m_FrameGenerationConfig.HUDLessColor            = (s_uiRenderMode == UI_HUDLESSCOLOR) ? hudLessResource : FfxResource({});
m_FrameGenerationConfig.flags      |= m_DrawDebugTearLines ? FFX_FSR3_FRAME_GENERATION_FLAG_DRAW_DEBUG_TEAR_LINES : 0;
m_FrameGenerationConfig.flags      |= m_DrawDebugView ? FFX_FSR3_FRAME_GENERATION_FLAG_DRAW_DEBUG_VIEW : 0;
m_FrameGenerationConfig.allowAsyncWorkloads     = m_AllowAsyncCompute && m_EnableAsyncCompute;

ffxFsr3ConfigureFrameGeneration(&m_FSR3Context, &m_FrameGenerationConfig);
```

FSR 3 NATIVE DX12 INTEGRATION – FSR 3 CONFIGURE CONTINUED

- To use “UI Generic” mode a resource is provided to the swapchain helper that is blended onto all frames.
- Can be set to a null resource to disable this mode.

```
// configure UI resource
FfxResource uiColor = (s_uiRenderMode == UI_TEXTURE)
                   ? ffxGetResource(m_pUiTexture[m_curUiTextureIndex]->GetResource(), L"FSR3_UiTexture", FFX_RESOURCE_STATE_PIXEL_COMPUTE_READ)
                   : FfxResource({});
cauldron::SwapChain* pSwapchain   = GetFramework()->GetSwapChain();
FfxSwapchain ffxSwapChain = ffxGetSwapchainDX12(pSwapchain->GetImpl()->DX12SwapChain());

ffxRegisterFrameinterpolationUiResource(ffxSwapChain, uiColor);
```

FSR 3 NATIVE DX12 INTEGRATION – UPSCALE + AA

```
FfxFsr3DispatchUpscaleDescription dispatchParameters = {};
dispatchParameters.commandList   = ffxGetCommandList(pCmdList);
dispatchParameters.color         = ffxGetResource(m_pColorTarget->GetResource(), L"FSR3_Input_OutputColor", FFX_RESOURCE_STATE_PIXEL_COMPUTE_READ);
dispatchParameters.depth         = ffxGetResource(m_pDepthTarget->GetResource(), L"FSR3_InputDepth", FFX_RESOURCE_STATE_PIXEL_COMPUTE_READ);
dispatchParameters.motionVectors = ffxGetResource(m_pMotionVectors->GetResource(), L"FSR3_InputMotionVectors", FFX_RESOURCE_STATE_PIXEL_COMPUTE_READ);
dispatchParameters.exposure      = ffxGetResource(nullptr, L"FSR3_InputExposure", FFX_RESOURCE_STATE_PIXEL_COMPUTE_READ);
dispatchParameters.upscaleOutput = dispatchParameters.color;
dispatchParameters.reactive = ffxGetResource(m_pReactiveMask->GetResource(), L"FSR3_InputReactiveMap", FFX_RESOURCE_STATE_PIXEL_COMPUTE_READ);
dispatchParameters.transparencyAndComposition = ffxGetResource(m_pCompositionMask->GetResource(), L"FSR3_TransparencyAndCompositionMap",
                                                               FFX_RESOURCE_STATE_PIXEL_COMPUTE_READ);

dispatchParameters.jitterOffset.x      = -jitterX;
dispatchParameters.jitterOffset.y      = -jitterY;
dispatchParameters.motionVectorScale.x = resInfo.fRenderWidth();
dispatchParameters.motionVectorScale.y = resInfo.fRenderHeight();
dispatchParameters.reset               = m_shouldReset;
dispatchParameters.enableSharpening    = m_RCASSharpen;
dispatchParameters.sharpness           = m_Sharpness;
dispatchParameters.frameTimeDelta      = (float)deltaTime * 1000.f; // Milliseconds!

dispatchParameters.preExposure         = GetScene()->GetSceneExposure();
dispatchParameters.renderSize.width    = resInfo.RenderWidth;
dispatchParameters.renderSize.height   = resInfo.RenderHeight;

dispatchParameters.cameraFovAngleVertical = pCamera->GetFovY();
dispatchParameters.cameraFar  = pCamera->GetFarPlane();
dispatchParameters.cameraNear = pCamera->GetNearPlane();

FfxErrorCode errorCode = ffxFsr3ContextDispatchUpscale(&m_FSR3Context, &dispatchParameters);
```

FSR 3 NATIVE DX12 INTEGRATION – FRAME GENERATION CALLBACK

- When using the frame generation callback, the swapchain will queue the frame generation dispatch automatically. There will be no need to manually call **`ffxFsr3DispatchFrameGeneration`**

- **This is the recommended path when integrating.**

- **See configure :**

```
m_FrameGenerationConfig.frameGenerationCallback = ffxFsr3DispatchFrameGeneration;
```

FSR 3 NATIVE DX12 INTEGRATION – MANUAL FRAME GENERATION

```
FfxFrameGenerationDispatchDescription fgDesc = {};

IDXGISwapChain4* dxgiSwapchain = GetFramework()->GetSwapChain()->GetImpl()->DX12SwapChain();
            ffxGetInterpolationCommandlist(ffxGetSwapchainDX12(dxgiSwapchain), fgDesc.commandList);

fgDesc.presentColor          = backbuffer;
fgDesc.numInterpolatedFrames = 1;
fgDesc.outputs[0]            = ffxGetFrameinterpolationTextureDX12(ffxGetSwapchainDX12(dxgiSwapchain));

FfxErrorCode errorCode = ffxFsr3DispatchFrameGeneration(&fgDesc);
```

The backbuffer resource that will be presented and form the basis for frame generation.

Resource from swapchain

- This is only required if not using the Frame Generation callback
- Note that Frame Generation does not have a context associated with it
- **This implementation only supports a single Frame Generation context**
  - The context provided to **`ffxFsr3ConfigureFrameGeneration`** with Frame Generation enabled will be the context used

FSR 3 NATIVE DX12 INTEGRATION: UI COMPOSITION

- UI composition and Present happen asynchronously to a game's Frame Generation
  - Note: the game needs to use **multiple command lists** to allow UI composition and presents to be injected on the GPUs graphics queue

- UI composition is a **required** step for FSR 3 integration
  - Failure to implement this step will result in major UI artefacts when Frame Generation is enabled!

- FSR 3 supports three composition modes of UI (User Interface) rendering:
  - **Callback** (recommended)
  - **UI texture**
  - **HUDLess** (compatibility mode, not recommended)

- Pick the best one according to your needs (see next slides)
  - One of them **must** be used. We recommend Callback for quality purposes.

FSR 3 NATIVE DX12 INTEGRATION - UI CALLBACK COMPOSITION MODE

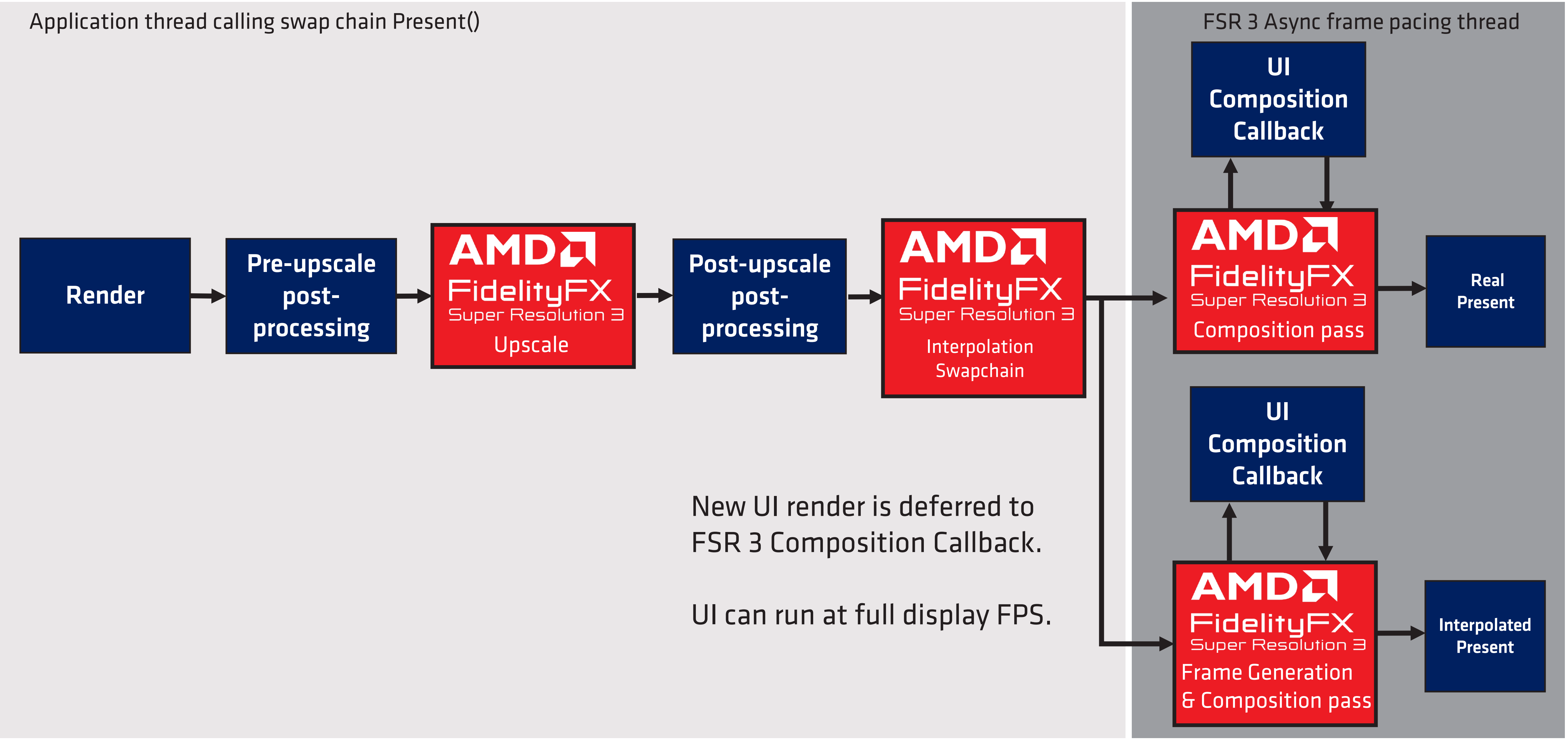
- Overload the UI composition by passing a callback function pointer in the **FfxFrameInterpolationContextDescription** which is part of the **FfxFsrContextDescription** on initialization
- Then call **FSR3RenderModule::UiCompositionCallback**

```
FfxErrorCode FSR3RenderModule::UiCompositionCallback(FfxInterface* backendInterface,
                                                     const FfxSwapChainPresentDescription* params)
```

- Use this for more complex effects needed for the UI (e.g. blurring the UI background) or when low latency UI is desired
- Using this method the UI will be displayed at **interpolated** framerate
  - We recommend only basic rendering is performed in this callback!
- UI callback will still be called even when FSR 3 Frame Generation is OFF
  - Will be composited on top of the real frame in this case

- *Note: UI callback composition mode is the* ***recommended*** *mode for UI composition*

FSR 3 NATIVE DX12 INTEGRATION - UI CALLBACK COMPOSITION MODE

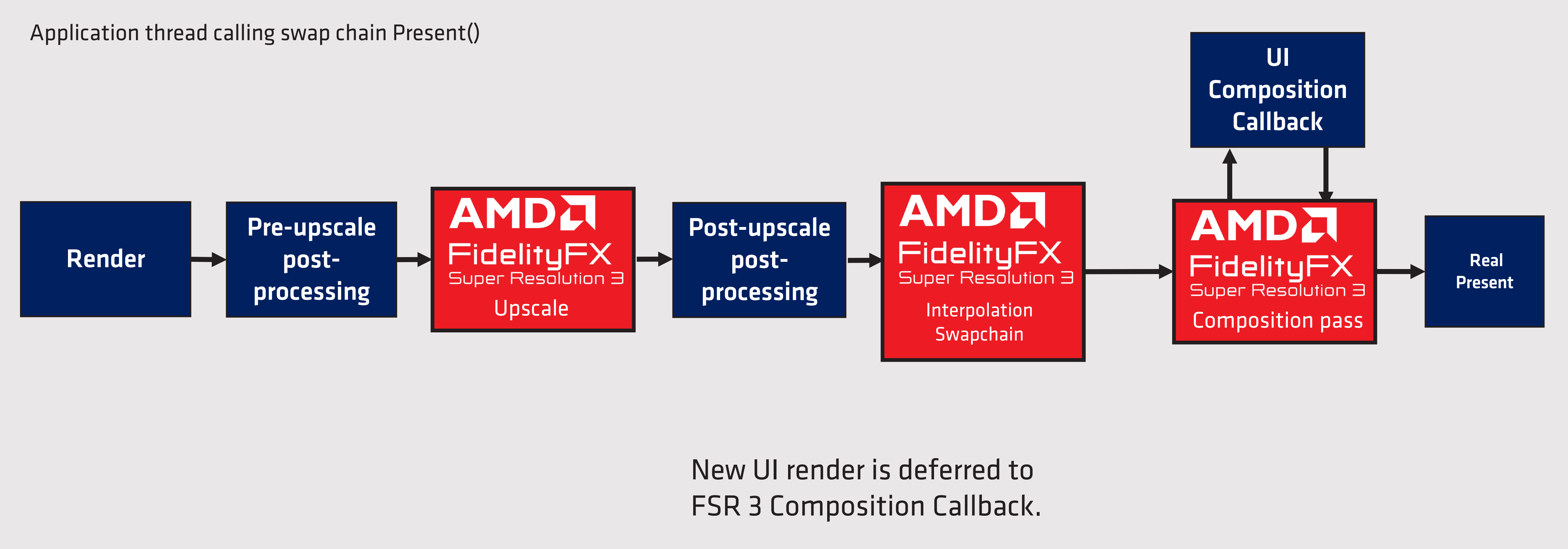

Application thread calling swap chain Present()
FSR 3 Async frame pacing thread
Render
Pre-upscale post-processing
AMD FidelityFX Super Resolution 3
Upscale
Post-upscale post-processing
AMD FidelityFX Super Resolution 3
Interpolation Swapchain
UI Composition Callback
AMD FidelityFX Super Resolution 3
Composition pass
Real Present
UI Composition Callback
AMD FidelityFX Super Resolution 3
Frame Generation & Composition pass
Interpolated Present

New UI render is deferred to FSR 3 Composition Callback.

UI can run at full display FPS.

FSR 3 NATIVE DX12 INTEGRATION - UI CALLBACK COMPOSITION MODE

WHEN FRAME GENERATION IS DISABLED


Application thread calling swap chain Present()
Render
Pre-upscale post-processing
AMD FidelityFX Super Resolution 3
Upscale
Post-upscale post-processing
AMD FidelityFX Super Resolution 3
Interpolation Swapchain
UI Composition Callback
AMD FidelityFX Super Resolution 3
Composition pass
Real Present

New UI render is deferred to FSR 3 Composition Callback.

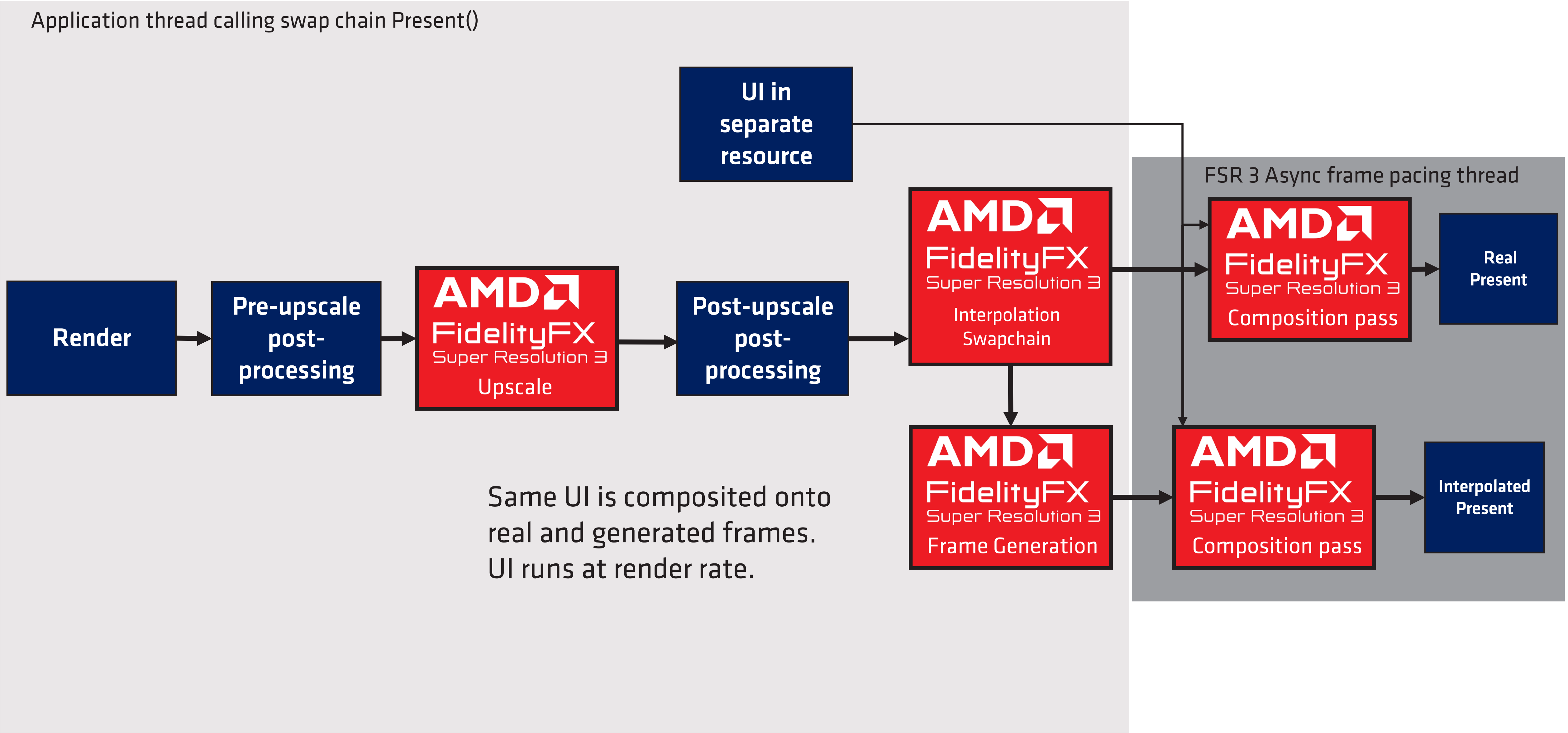
FSR 3 NATIVE DX12 INTEGRATION – UI TEXTURE COMPOSITION MODE

- Render UI into a dedicated texture
- Call `ffxRegisterFrameInterpolationUiResource`
  - Provide FSR 3 swapchain and UI resource in NON_PIXEL_SHADER_RESOURCE
- FSR 3 will blend the UI onto the back buffer of **both** the interpolated and the real frame using the alpha channel of the UI texture

- Using this method the UI will be displayed at **render** framerate (not interpolated)
  - i.e. the same UI will be displayed for two consecutive frames if Frame Generation is enabled

- *Note: If UI includes part of the currently-rendered frame (e.g. blurry background) then the Callback option is required otherwise artefacts will ensue*

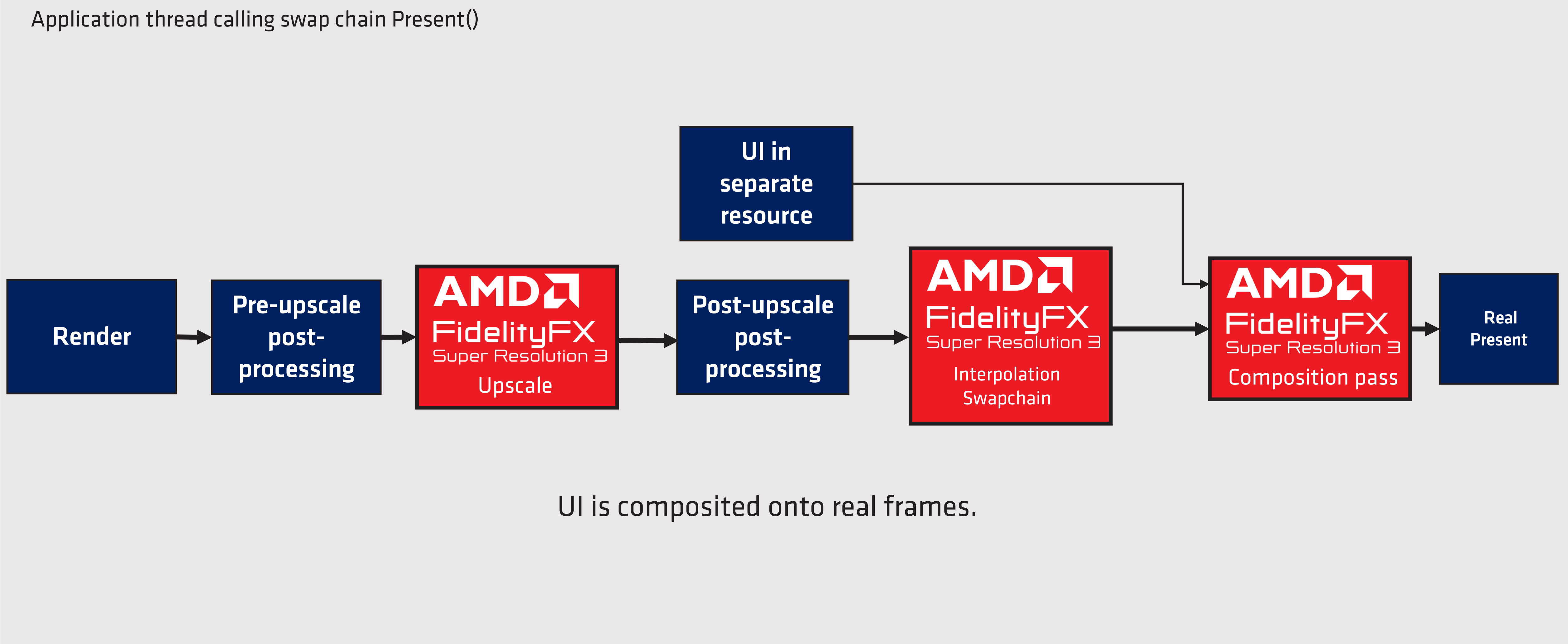
FSR 3 NATIVE DX12 INTEGRATION - UI TEXTURE COMPOSITION MODE


Application thread calling swap chain Present()
UI in separate resource
Render
Pre-upscale post-processing
AMD FidelityFX Super Resolution 3 Upscale
Post-upscale post-processing
AMD FidelityFX Super Resolution 3 Interpolation Swapchain
AMD FidelityFX Super Resolution 3 Frame Generation
FSR 3 Async frame pacing thread
AMD FidelityFX Super Resolution 3 Composition pass
Real Present
AMD FidelityFX Super Resolution 3 Composition pass
Interpolated Present

Same UI is composited onto real and generated frames.
UI runs at render rate.

FSR 3 NATIVE DX12 INTEGRATION - UI TEXTURE COMPOSITION MODE

WHEN FRAME GENERATION IS DISABLED


Application thread calling swap chain Present()
UI in separate resource
Render
Pre-upscale post-processing
AMD FidelityFX Super Resolution 3
Upscale
Post-upscale post-processing
AMD FidelityFX Super Resolution 3
Interpolation Swapchain
AMD FidelityFX Super Resolution 3
Composition pass
Real Present

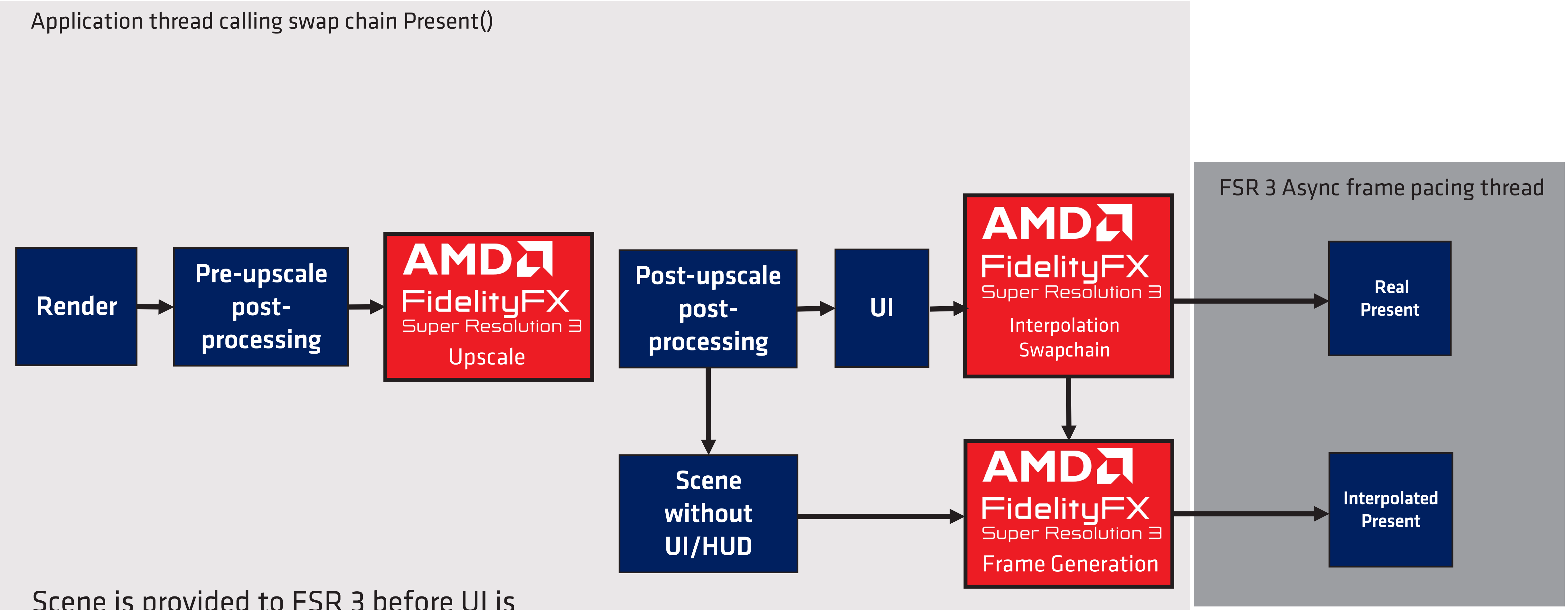
UI is composited onto real frames.

FSR 3 NATIVE DX12 INTEGRATION - HUDLESS COMPOSITION MODE

- Provide FSR 3 with scene resource pre UI/HUD render
  - Resource must be of same color and resource format as the swapchain present buffer
- Provide the resource in **HUDLessColor** as part of the **FfxFrameGenerationConfig** struct of **ffxFsr3ConfigureFrameGeneration**.
  - NON_PIXEL_SHADER_RESOURCE
- FSR 3 will blend the UI onto the back buffer of **both** the interpolated and the real frame using the alpha channel of the UI surface
- Using this method the UI will be displayed at **render** framerate (not interpolated)
  - i.e. the same UI will be displayed for two consecutive frames if Frame Generation is enabled

- *Note: this mode is considered a compatibility mode and is **not recommended** as it will likely produce visual artefacts in the UI*

FSR 3 NATIVE DX12 INTEGRATION - HUDLESS COMPOSITION MODE


Application thread calling swap chain Present()
Render
Pre-upscale post-processing
AMD FidelityFX Super Resolution 3 Upscale
Post-upscale post-processing
UI
AMD FidelityFX Super Resolution 3 Interpolation Swapchain
Scene without UI/HUD
AMD FidelityFX Super Resolution 3 Frame Generation
FSR 3 Async frame pacing thread
Real Present
Interpolated Present

Scene is provided to FSR 3 before UI is rendered. FSR 3 uses this data to generate UI masks. There is a possibility for artefacts on transparent elements.

FSR 3 INTEGRATION STEPS (UNREAL ENGINE 5)

FSR 3 UE5 INTEGRATION - INSTALL

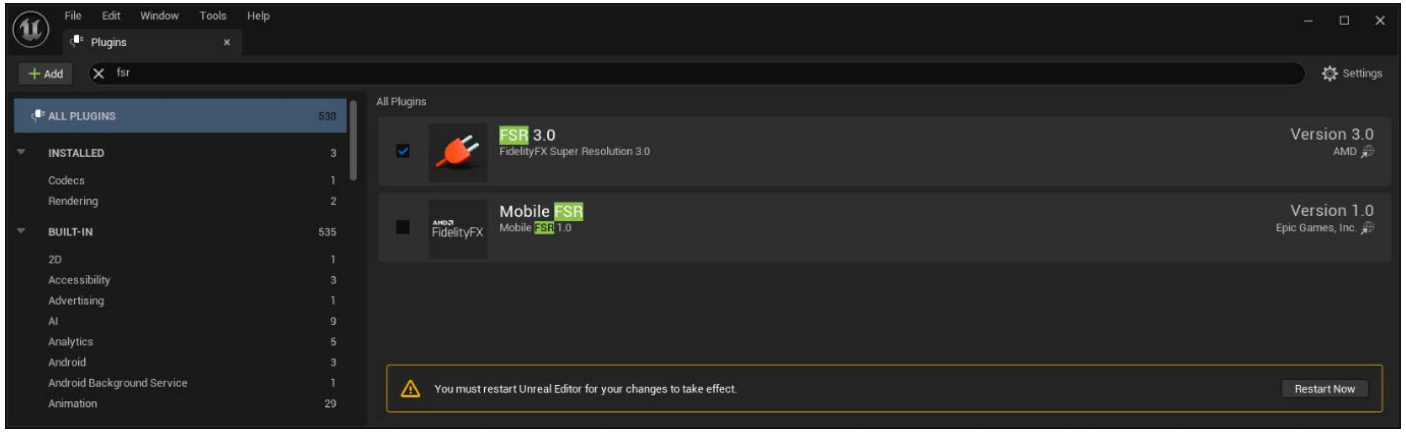
- The FSR 3 and FSR3MovieRenderPipeline plugins are intended for Unreal Engine 5.1.1* or later. If you are not a registered Unreal Engine developer, you will need to follow these instructions and register for access to this link.

- For optimal quality it is necessary to use Unreal Engine from source code and apply source code patches.
  - To improve FSR 3's handling of animated opaque materials:
    - Use: git apply <VERS>-LitReactiveShadingModel.patch
    - Where <VERS> should be the engine-version in use.

- Install the plugin into the Unreal Engine:
  - Locate the *Engine/Plugins* directory of your Unreal Engine installation.
  - Extract the contents of the FSR3.zip file.
  - Select the sub-folder that corresponds to the Unreal Engine version to be used.
  - Place the **FSR3** folder within your Unreal Engine source tree at: *Engine/Plugins/Marketplace*
    - (Optional) Place the **FSR3MovieRenderPipeline** folder within your Unreal Engine source tree at: *Engine/Plugins/Marketplace*.

FSR 3 UE5 INTEGRATION – ENABLE PLUGIN

- To enable the plugin once installed:
  - Open your Unreal Engine project in the Editor.
  - Navigate to **Edit > Plugins** in the Unreal Engine toolbar.
  - Within the plugin dialog:
    - Ensure that All is selected on the left side.
    - Type fsr into the search box in the top right corner.
    - Select the **Enabled** checkbox for the **FSR 3.0** plugin.
      - i. (Optional) Select the **Enabled** checkbox for the **FSR3MovieRenderPipeline** plugin.
    - When prompted, click **Restart Now** to apply changes, and restart Unreal Engine.



fsr

ALL PLUGINS	538
INSTALLED	3
Codecs	1
Rendering	2
BUILT-IN	535
2D	1
Accessibility	3
Advertising	1
AI	9
Analytics	5
Android	3
Android Background Service	1
Animation	29

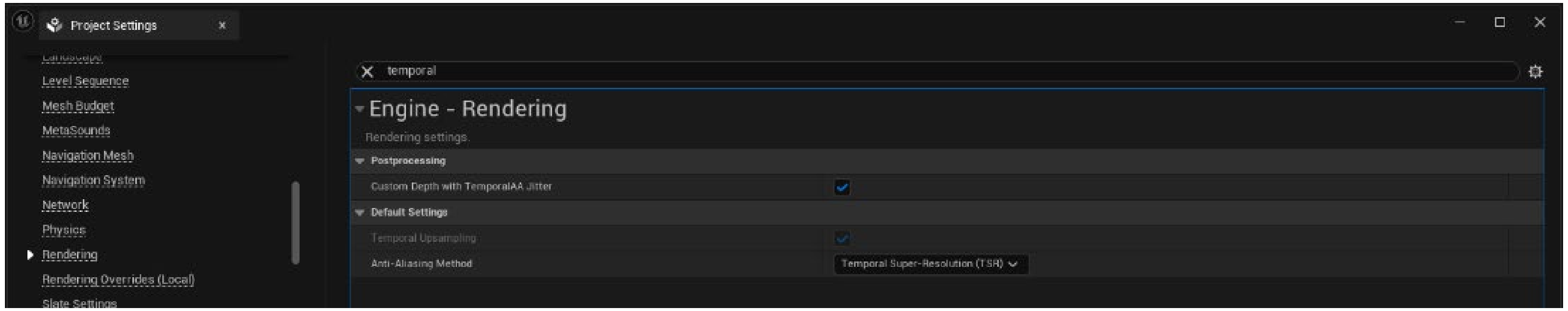
☑ FSR 3.0 — FidelityFX Super Resolution 3.0 — Version 3.0 — AMD

□ Mobile FSR — Mobile FSR 1.0 — Version 1.0 — Epic Games, Inc.

⚠ You must restart Unreal Editor for your changes to take effect. Restart Now

FSR 3 UE5 INTEGRATION – BASIC USAGE

- **Temporal Upsampling** must be enabled in the **Project Settings > Rendering** window, accessed via **Edit > Project** Settings in the Unreal Engine toolbar or via the Console Variable *`r.TemporalAA.Upsampling`*.



temporal

Engine - Rendering

Rendering settings.

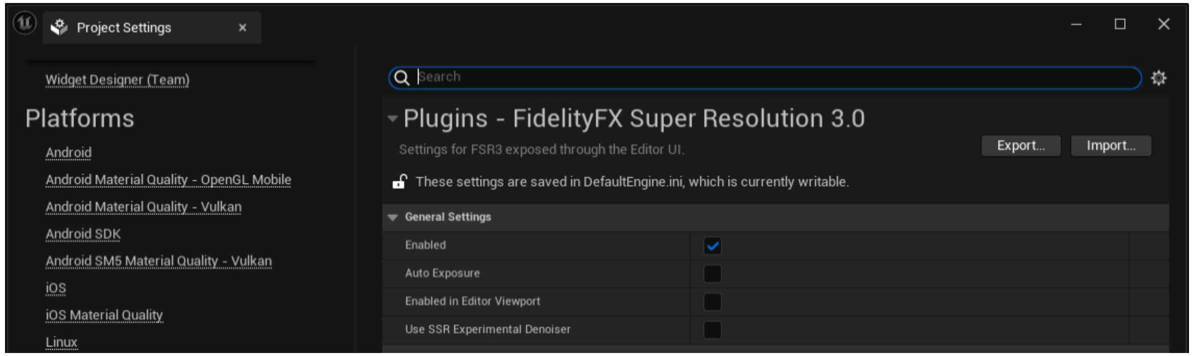
Postprocessing

Custom Depth with TemporalAA Jitter	☑

Default Settings

Temporal Upsampling	☑
Anti-Aliasing Method	Temporal Super-Resolution (TSR)

- FSR 3.0 can be enabled or disabled via the **Enabled** option in the **Project Settings > FidelityFX Super Resolution 3.0** settings window, or with the console variable *`r.FidelityFX.FSR3.Enabled`* in the configuration files. The variable can be modified at runtime ***however*** this is not guaranteed to be safe when other third-party upscalers are also enabled.



Plugins - FidelityFX Super Resolution 3.0

Settings for FSR3 exposed through the Editor UI.

These settings are saved in DefaultEngine.ini, which is currently writable.

General Settings

Enabled	☑
Auto Exposure	□
Enabled in Editor Viewport	□
Use SSR Experimental Denoiser	□

FSR 3 UE5 INTEGRATION – QUALITY MODES

- The plugin will use specific quality modes specified via *`r.FidelityFX.FSR3.QualityMode`* overriding *`r.ScreenPercentage`*. The exposed modes are:
  - **Native AA (1.0x)**: *`r.FidelityFX.FSR3.QualityMode 0`*
    - Provides an image quality superior to native rendering with a modest performance cost.
  - **Quality (1.5x)**: *`r.FidelityFX.FSR3.QualityMode 1`*
    - Provides an image quality equal or superior to native rendering with a significant performance gain.
  - **Balanced (1.7x)**: *`r.FidelityFX.FSR3.QualityMode 2`*
    - Offers an ideal compromise between image quality and performance gains.
  - **Performance (2.0x)**: *`r.FidelityFX.FSR3.QualityMode 3`*
    - Provides an image quality similar to native rendering with a major performance gain.
  - **Ultra Performance (3.0x)**: *`r.FidelityFX.FSR3.QualityMode 4`*
    - Provides the highest performance gain while still maintaining an image quality representative of native rendering.

FSR 3 UE5 INTEGRATION – RECOMMENDATIONS

- **Optimizing translucency appearance**:

  While the default settings for the FSR 3 Reactive Mask should generate reasonable results, it is important that developers are aware that the appearance can be altered via the *`r.FidelityFX.FSR3.ReactiveMask`* console-variables. Tuning these variables to suit the content may be necessary to optimize visual results.

- **Translucent skyboxes and background planes**:

  When using a skybox or a distant background plane it is beneficial for this to be rendered with the Opaque or Masked shading model when using FSR 3. If these are rendered with the Translucent shading model they can result in unnecessary artefacts.

  To reduce artefacts when opaque objects occlude a translucent skybox the FSR 3 plugin will fade out translucency contribution based on reconstructed distance from the camera, the distance at which this occurs can be controlled with the *`r.FidelityFX.FSR3.ReactiveMaskTranslucencyMaxDistance`* console variable.

  When using an opaque skybox or backplane, adjust the *`r.FidelityFX.FSR3.ReactiveMaskTranslucencyMaxDistance`* console variable to avoid translucency cut-outs.

- **Hair and dither effects**:

  FSR 3 does not smooth dither effects in the way other upscalers do. They are retained as thin features which may not be intentional. To avoid this, especially with hair, enable the *`r.FidelityFX.FSR3.DeDither`* console variable which attempts to smooth dither effects prior to FSR 3 upscaling.

FSR 3 UE5 INTEGRATION – RECOMMENDATIONS

- **World-Position-Offset on static objects**:

  In specific circumstances static objects that use a material with World-Position-Offset do not always generate motion vectors which may result in blurring/ghosting of the affected objects.

  If it is necessary *`r.Velocity.ForceOutput`* can be used to force all primitives to emit velocity.

- **Animated opaque materials**:

  Animated opaque materials which do not generate motion vectors for the animated content, such as in-world video screens, may also blur or ghost when obscured by static geometry. This can be reduced by ensuring such materials write into the Reactive Mask generated in the plugin.

  - For the Deferred Renderer:
    - Select a Shading Model for the 'Reactive Shading Model' option in the FSR 3 section of the project settings or using the `r.FidelityFX.FSR3.ReactiveMaskForceReactiveMaterialValue` console variable.
    - The 'LitReactiveShadingModel' engine patch which adds a new 'LitReactive' shading model that can be used specifically for this purpose.
    - Specify the reactive value to write in `r.FidelityFX.FSR3.ReactiveMaskForceReactiveMaterialValue` console variable or via the material's CustomData0.x channel.
  - Substrate materials (Strata in Unreal 5.1) and the Forward Renderer are not supported.

FSR 3 UE5 INTEGRATION – FRAME GENERATION

- Frame Generation can be enabled or disabled via the **Project Settings > FidelityFX Super Resolution 3.0** settings panel by changing the setting of ‘Frame Generation Enabled’.
  - It does not affect the Editor and requires the FSR3 temporal upscaler to be enabled as it requires inputs generated by the upscaling component.

- Frame Generation supports both an RHI and Direct3D 12 backend.
  - The default is the native Direct3D 12 backend as it provides better performance and frame pacing via the FSR3 proxy IDXGISwapChain implementation.
    - **D3D12 Async. Present**: *`r.FidelityFX.FI.OverrideSwapChainDX12`*
      - Enables the FSR3 proxy IDXGISwapChain implementation. This is a read-only setting, a restart is required to modify.
    - **D3D12 Async. Interpolation**: *`r.FidelityFX.FI.AllowAsyncWorkloads`*
      - Executes frame interpolation on Async. Present queue for concurrent execution with following frame to improve performance.

  - The RHI backend can be used on other platforms to provide basic frame generation support running serially after the end of the game frame and using the in-built Slate and RHI presentation framework.
    - If using the RHI Backend then V-Sync should be enabled as this backend does not provide any form of frame pacing.

FSR 3 TECHNICAL DETAILS AND GUIDELINES

FSR 3 FRAME GENERATION – MINIMUM FRAME RATE

- FSR 3 Frame Generation runs best when interpolating from a minimum of 60 fps **pre-interpolation** (e.g. after upscale)

- Whilst FSR 3 can roughly double any input frame rate, going below 60 is **not recommended**
  - This is due to interpolation visual artefacts being more prominent at lower frame rates
  - Any fps below 30 fps pre-interpolation should be absolutely avoided

- Altering FSR 3 behaviour based on sub-60 fps detection is **not recommended**
  - Better to educate users about this and let them adjust their own graphics settings as needed

FSR 3 FRAME GENERATION –FRAME RATE DISPLAY

- To calculate fps when FSR 3 frame generation is on, you can use the following function:

  FrameInterpolationSwapChainDX12::GetLastPresentCount()

- Please refer to how the sample outputs “Display FPS” to calculate fps

FSR 3 VARIABLE REFRESH RATE CONSIDERATIONS

- FreeSync, G-Sync and Adaptive Sync are all forms of Variable Refresh Rate technologies
  - With VRR, refresh rate is dictated by frames sent by the GPU to the monitor
- FSR 3 Frame Generation behaves according to the following table

	VRR OFF	VRR ON
V-Sync OFF	Tearing will be seen at all frame rates.	**Recommended setting if frame times are highly variable** Some tearing may appear in some circumstances (e.g. fps close to or above monitor's max refresh). Hardware Accelerated GPU Scheduling disabled may result in more tearing.
V-Sync ON	Tearing-free experience at all frame rates. FPS limited to integer multiple of max refresh when fps is under max refresh. This may cause "judder" due to uneven sync intervals.	**Recommended setting if frame times are stable (e.g. via limiter)** Tearing-free experience at all FPS Frame rate will be capped at monitor's maximum refresh rate. Render rate is implicitly limited to half of the monitor's max refresh rate

- It is highly recommended that games implement a frame limiter to provide options to players who want a steady framerate

VISUAL DEBUG MARKERS

- FSR 3 has a debug view for Frame Generation and screen tearing
    - To enable it, configure with the FFX_FSR3_FRAME_GENERATION_FLAG_DRAW_DEBUG_TEAR_LINES flag

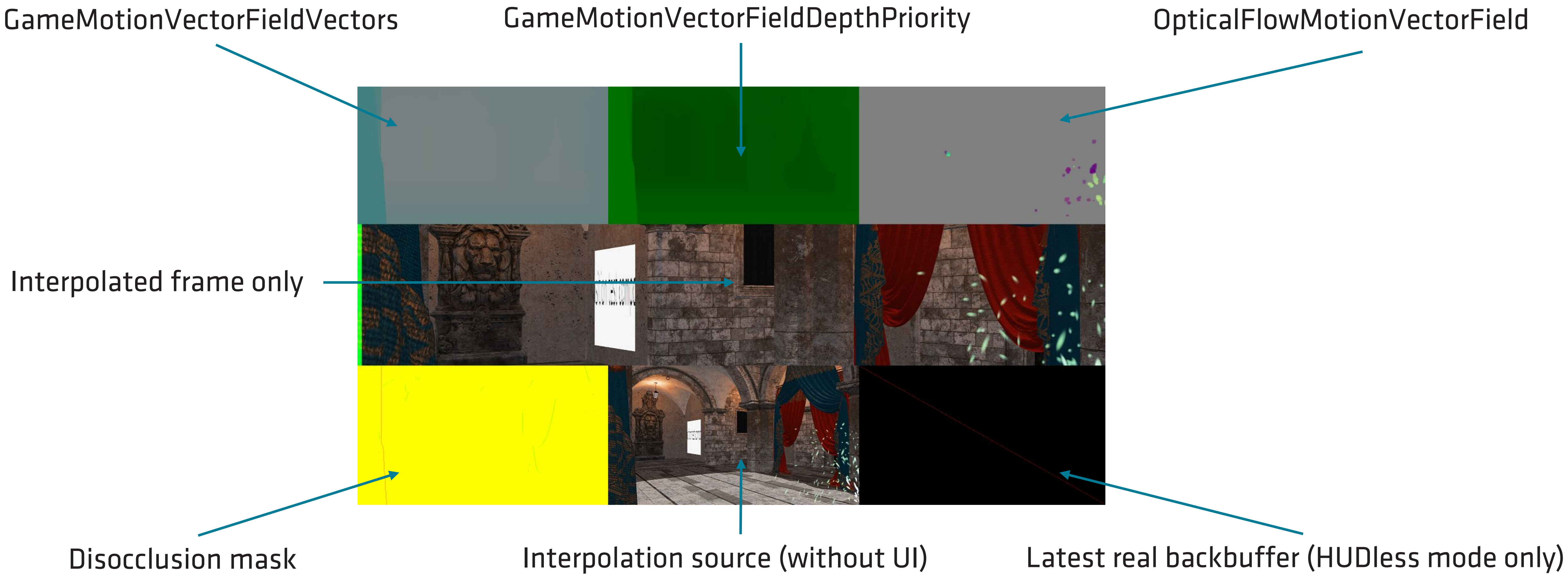
- FSR 3 will then render a solid green bar on the left, and a bar on the right which cycles through colors
    - A large red square in top left indicates Optical Flow has detected the scene has changed significantly.
    - A blue square in top left indicates the RESET passed to FSR 3 Frame Generation is TRUE

- With V-Sync enabled these bars should flicker entirely to show alternative frames being generated in perfect circumstances

- With FPS > Monitor Refresh Rate, and tearing enabled, these bars allow developers to see the tear regions clearly and tweak any particular frame pacing issue that may require per-game tuning

VISUAL DEBUG MARKERS

- FSR 3 has a debug view for integration testing
  - To enable it, configure with the FFX_FSR3_FRAME_GENERATION_FLAG_DRAW_DEBUG_VIEW flag



GameMotionVectorFieldVectors

GameMotionVectorFieldDepthPriority

OpticalFlowMotionVectorField

Interpolated frame only

Disocclusion mask

Interpolation source (without UI)

Latest real backbuffer (HUDless mode only)

MOTION VECTORS

- Motion vectors must be provided to the FSR 3 upscale component in the same way as FSR 2
- Motion vectors should be of **minimum 16-bit precision** for quality purposes
- All opaque elements and elements writing depth should have motion vectors:
  - Foliage and all other alpha tested materials
  - Playable characters & NPCs
  - Vehicles and other deformable geometry
- If elements have incorrect or missing motion vectors, ghosting will occur when upscaling
- **If upscaling has poor motion vector inputs, interpolated frames will amplify artefacts!**

FSR 3 CONSIDERATIONS

- Using FSR 3 **FrameinterpolationSwapchain** has a slight performance impact even if frame generation is disabled (one extra back buffer copy)
  - When switching the swapchain, the game **must not** be in Exclusive Fullscreen mode

- FSR 3 and DLSS 3 **should not** be enabled at the same time
  - When creating the FSR 3 **FrameInterpolationSwapChain** DLSS 3 hooking should already be disabled

- It is **strongly recommended** to integrate FSR 3 via the DLL method
  - It allows better debugging from AMD if the need arises
  - It allows better future compatibility with newer versions of the technology

RESOURCE LIFETIMES

- FSR 3 Frame Generation shares some resources generated in upscaling
  Compared to FSR 2, the following resources have extended lifetimes and can’t be aliased:
  - DilatedDepth
  - DilatedMotionVectors
  - ReconstructedPrevNearestDepth
- When Async Queue usage is enabled, the shared resources will get double buffered
  - This prevents issues when frame interpolation overlaps with next frames upscaling
- When UITexture composition mode is used:
  - The UITexture will get used in composition (or the UI callback)
  - The application is responsible to ensure it persists until composition of the real frame is finished
- When HUDLess composition mode is used:
  - The HUDLess texture will be used during FrameInterpolation
  - The application is responsible to ensure it persists until FrameInterpolation is complete

NEW FSR 3 QUALITY MODE: NATIVE AA

- FSR 3 introduces a new Quality mode: **Native AA**
- In this mode FSR 3 Upscaling is a pure AA option (no actual upscaling takes place)
- "Native AA" with "Frame Generation" enabled together essentially provide frame generation without upscaling

- Note: Native AA quality mode has the largest performance overhead!

- Note: Native AA quality mode **still requires Reactive and Transparency & Composition masks** to work correctly!

VERSIONS OF FSR

FSR version	Technology description	Quality modes supported	API support	Hardware support[1]
FSR 1	Spatial upscale of input frames	Ultra Quality Quality Balanced Performance	DX12, DX11, Vulkan® UE4, UE5, Unity HDRP & URP	RX 460 and above Xbox Series X and S
FSR 2	Temporal upscale of input frames	Quality Balanced Performance Ultra performance	DX12, DX11[2], Vulkan UE4, UE5, Unity URP	RX Vega Series and above Xbox Series X and S[3]
FSR 3	Temporal upscale of input frames with frame generation	Native AA Quality Balanced Performance Ultra performance	DX12, Vulkan (in development) UE5	AMD RDNA Series and above Xbox Series X and S[3]

[1] AMD FidelityFX Super Resolution is "game dependent" and is only supported if the minimum requirements of the game are met.

[2] On demand only

[3] FSR 3 memory requirements may require special considerations on Xbox Series S

FSR 3 PERFORMANCE OVERHEAD

- Performance taken from FidelityFX SDK sample and Microsoft® PIX
  - Sharpening disabled, async compute disabled
  - Integrating FSR 3 with async compute can help reduce the cost of frame generation by having it scheduled asynchronously

FSR 3 Target resolution	Quality mode	Enthusiast GPUs (AMD RADEON™ RX 7900 XTX)		Performance GPUs (AMD RADEON™ RX 6800 XT)		Mainstream GPUs (AMD RADEON™ RX 5700 XT)	
		Upscale (ms)	Frame Generation (ms, up to)	Upscale (ms)	Frame Generation (ms, up to)	Upscale (ms)	Frame Generation (ms, up to)
4K	Native AA	1.4	2.4	2.3	4.2	4.7	8.0
	Quality	0.9	1.6	1.5	2.7	3.4	5.9
	Performance	0.7	1.4	1.3	2.3	3.0	5.3
1440p	Native AA	0.6	1.0	0.9	1.6	2.1	3.6
	Quality	0.4	0.8	0.7	1.3	1.6	2.9
	Performance	0.3	0.7	0.6	1.0	1.3	2.5
1080p	Native AA	0.3	0.7	0.6	1.0	1.2	2.2
	Quality	0.2	0.5	0.4	0.8	0.9	1.6
	Performance	0.2	0.5	0.4	0.7	0.8	1.5

Configuration: MSI X570-A Pro, AMD Ryzen™ 9 5900X @ 3.70 Ghz, System RAM: 16GB G.Skill DDR4-3600 CL16-16-16-36, Windows® 10 Pro 64-bit, AMD Software: Adrenalin Edition 23.11.1

FSR 3 MEMORY USAGE

- FSR 3 Upscaling + Frame Generation, async compute off, UI Callback composition mode
  - Note: FSR 3 memory usage is higher in async compute mode due to double buffering shared resources

Output resolution	Upscaling quality mode	Input resolution	Memory usage (upscaling only)	Memory usage (upscaling + FG)
4K	Native AA	3840 x 2160	628 MB	984 MB
	Quality	2560 x 1440	437 MB	648 MB
	Balanced	2258 x 1270	396 MB	576 MB
	Performance	1920 x 1080	365 MB	512 MB
1440p	Native AA	2560 x 1440	293 MB	460 MB
	Quality	1706 x 960	204 MB	303 MB
	Balanced	1505 x 847	186 MB	270 MB
	Performance	1280 x 720	169 MB	241 MB
1080p	Native AA	1920 x 1080	169 MB	267 MB
	Quality	1280 x 720	116 MB	174 MB
	Balanced	960 x 540	107 MB	158 MB
	Performance	640 x 360	102 MB	149 MB

AMD

FidelityFX

Super Resolution 3

USER INTERFACE GUIDE

FSR 3 UI GUIDE - OVERVIEW

- The FSR 3 UI should be composed of two elements labelled as such:
  - **AMD FSR 3**: a selection between all supported **FSR 3 quality modes**
  - **AMD FSR 3 Frame Generation**: either **On** or **Off**

- Default FSR 3 API usage requires:
  - **AMD FSR 3** to be set to one of the **FSR 3 quality modes** when **AMD FSR 3 Frame Generation** is on
  - If **AMD FSR 3** is set to **Off** then **AMD FSR 3 Frame Generation** must be automatically set to **Off** and **grayed out**

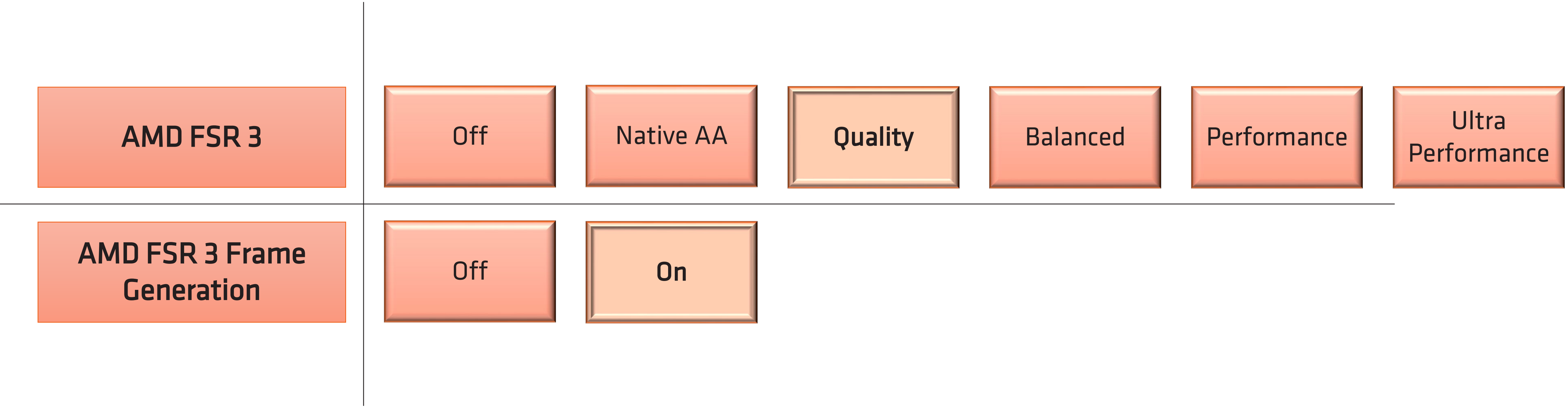
- **AMD FSR 3 Frame generation** can be enabled with **any FSR 3 quality modes**
  - Native AA, Quality, Balanced, Performance (and optionally Ultra Performance)

- Ensure your game exposes an option to limit fps in order to provide steadier frame inputs to FSR 3

- Due to interactions with similar technologies a game restart may be an acceptable solution to enable FSR 3
  - However the toggling of the Frame Generation feature of FSR 3 should not require a restart

FSR 3 UI GUIDE – VISUAL EXAMPLE 1

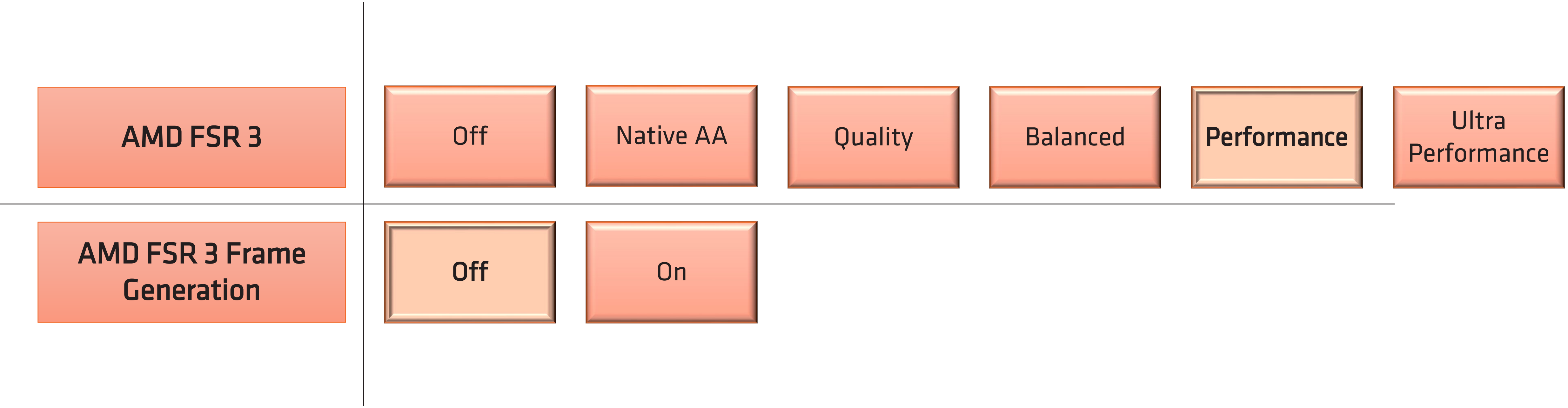
AMD FSR 3 is set to **Quality** with Frame Generation **on**
(1.5x upscaling with frame interpolation enabled)



AMD FSR 3	Off	Native AA	[✓ **Quality**]	Balanced	Performance	Ultra Performance
AMD FSR 3 Frame Generation	Off	[✓ **On**]				

FSR 3 UI GUIDE – VISUAL EXAMPLE 2

AMD FSR 3 is set to **Performance** with Frame Generation **off**
(2.0x upscaling with Frame Generation disabled)



AMD FSR 3	Off	Native AA	Quality	Balanced	[✓ **Performance**]	Ultra Performance
AMD FSR 3 Frame Generation	[✓ **Off**]	On				

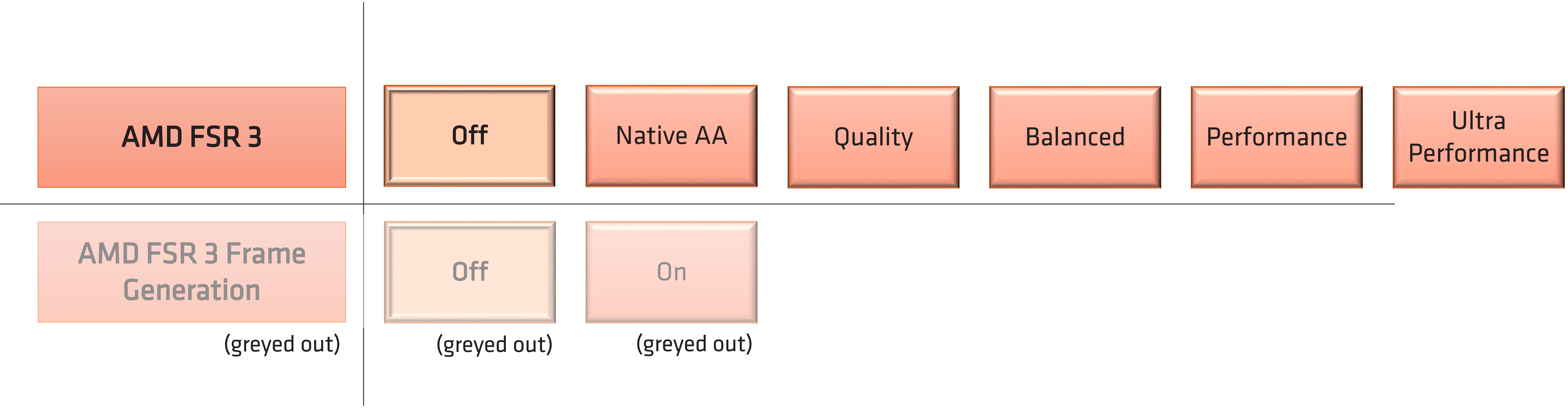
FSR 3 UI GUIDE – VISUAL EXAMPLE 3

AMD FSR 3 is set to **Native AA** with Frame Generation **on**
(No upscaling with Frame Generation enabled)

AMD FSR 3	Off	[✓ **Native AA**]	Quality	Balanced	Performance	Ultra Performance
AMD FSR 3 Frame Generation	Off	[✓ **On**]				

FSR 3 UI GUIDE – VISUAL EXAMPLE 4

AMD FSR 3 is completely off



AMD FSR 3

Off

Native AA

Quality

Balanced

Performance

Ultra Performance

AMD FSR 3 Frame Generation

(greyed out)

Off

(greyed out)

On

(greyed out)

FSR 3 QUALITY MODES

FSR 3 quality mode	Description	Scale factor	Input resolution	Output resolution
Native AA	**Native AA** mode provides an image quality superior to native rendering with a modest performance cost.	**1.0x** per dimension (1.0x area scale) (100% screen resolution)	1920 x 1080 2560 x 1440 3440 x 1440 3840 x 2160	1920 x 1080 2560 x 1440 3440 x 1440 3840 x 2160
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 mode to expose.*

FSR 3 UI REQUIREMENTS – DESCRIPTION OF UI ELEMENTS

- Desired UI description for **AMD FSR 3**:
  - "AMD FidelityFX Super Resolution 3 combines high-quality image upscaling with frame generation technologies to generate high resolution frames at fast frame rates"
- Desired UI description for **AMD FSR 3 Frame Generation**:
  - "AMD FidelityFX Super Resolution 3 Frame Generation increases frame rate by creating additional frames computed from existing inputs."
- Desired UI descriptions for **FSR 3 quality modes**:
  - Please use the descriptions mentioned on the previous slide.

- Localization strings are available on the GPUOpen website at https://gpuopen.com/fidelityfx-naming-guidelines/

SHARPENING

- FSR 3 comes with its own optional sharpening pass

- It is strongly recommended that the game exposes a sharpening slider
  - This is a common request from players

- If your game already supports a sharpening option in the UI please connect the value to the FSR 3 parameter
  - This is to avoid a clash with the sharpening feature of FSR 3
  - Depending on the existing sharpening range, this may require some trivial remapping

COMPATIBILITY WITH THIRD-PARTY SOFTWARE OR CODE

- FSR 3 requires unencumbered access to the swap chain for best frame pacing results

- Software libraries that intercept DXGI calls may cause frame pacing issues with FSR 3

- Third-party software that intercept DXGI calls to display an on-screen overlay may be incompatible with FSR 3 Frame Generation
- It is recommended to disable those for best frame pacing FSR 3 results
  - AMD OCAT has been validated to work correctly with FSR 3 Frame Generation

HARDWARE ACCELERATED GPU SCHEDULING

- For best results with FSR 3 Frame Generation it is recommended that Hardware Accelerated GPU Scheduling be enabled in your Windows OS
- At the moment Hardware Accelerated GPU scheduling is supported on the following AMD GPUs under Windows® 11:
  - Radeon RX 7900XTX
  - Radeon RX 7900XT
  - Radeon RX 7800XT
  - Radeon RX 7700XT

SUMMARY

- FSR 3 combines upscaling and frame generation
  - Compatible with VRR monitors
- A successful integration requires a quality FSR 3 upscaling implementation first (see slide 6)
- Default FSR 3 API behaviour requires an FSR 3 Quality mode to be enabled to use FSR 3 Frame Generation
- Integrate FSR 3 Frame Generation using the Presentation queue first to ensure correctness
  - Then move to Async queue for best results
- A DLL integration is strongly recommended for debug and future compatibility purposes
- Use the optimal method for UI composition (Callback) – avoid HUDLess mode
- Follow recommendations for UI requirements
- Localization strings can be found on GPUOpen at https://gpuopen.com/fidelityfx-naming-guidelines/

- Please share pre-release FSR 3 integrations with your AMD Alliance Manager - AMD may be able to run it through our labs and give you feedback