HIGH PERFORMANCE RENDERING IN SNOWDROP USING AMD FSR 3

HAMPUS SIVERSSON

COLIN RILEY

together we advance_

AMD FIDELITYFX™ SUPER RESOLUTION 3

COLIN RILEY - AMD

AMD

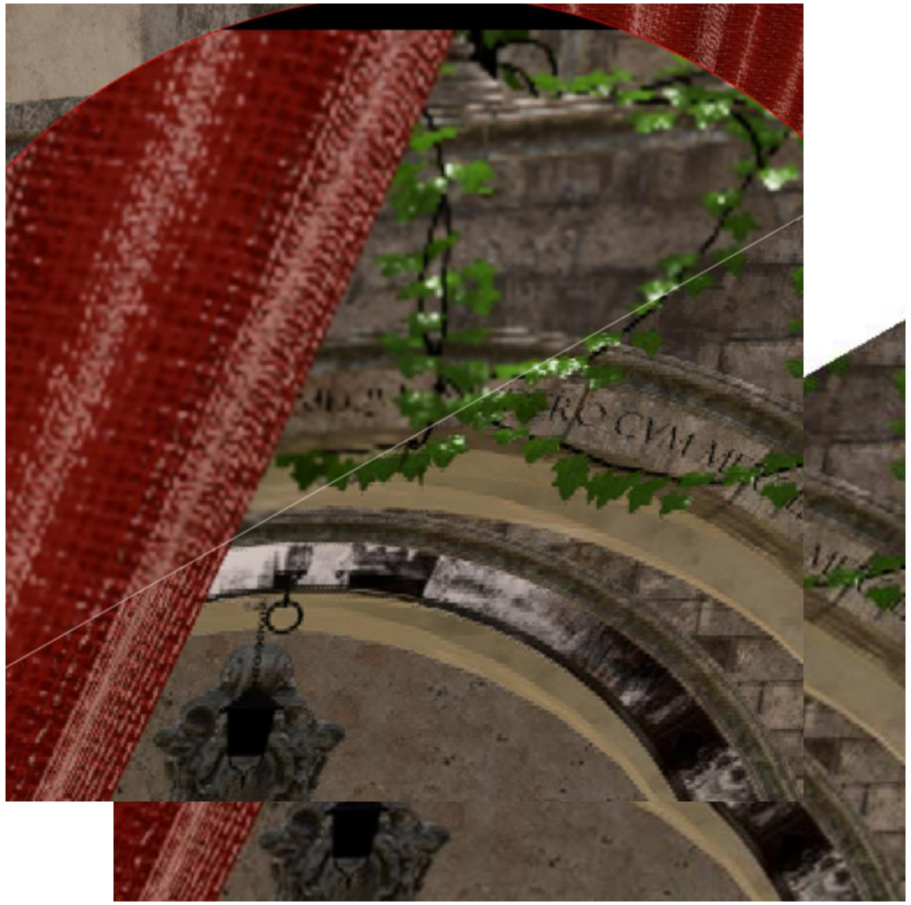
together we advance_

AMD FSR*: THE ROAD TO TODAY

- Spatial Upscaling: **FSR 1**
  - Placed late in the graphics pipeline.
  - Basic integrations.
  - Requires good TAA implementation.
  - Analytical Solution, cross platform, MIT license.

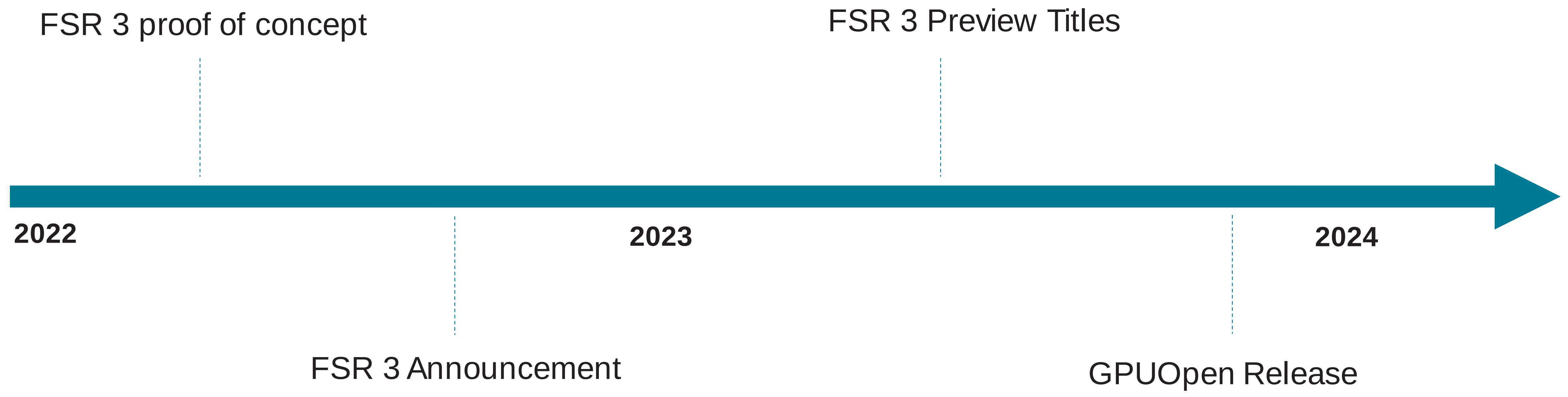
- Temporal Upscaling: **FSR 2**
  - Placed early in the post-processing graphics pipeline.
  - TAA + Upscaling.
  - Analytical Solution, cross platform, MIT license.

- Frame Generation: **FSR 3**
  - Adding frame generation.
  - Touches many more areas of graphics pipeline.
  - TAA + Upscaling, Frame Generation, Optical Flow, Frame Pacing.
  - Analytical Solution, cross platform, MIT license.



*See endnote GD-187A

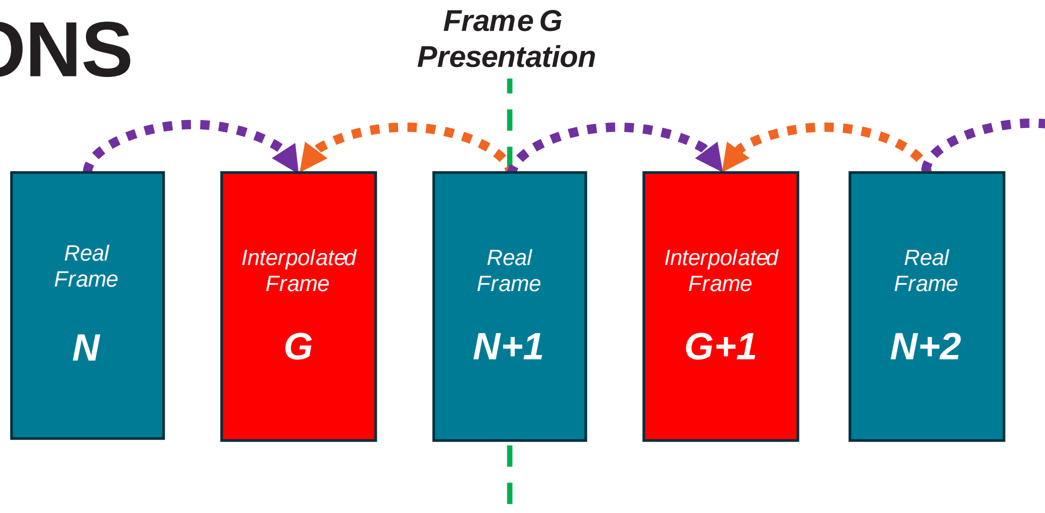
AMD FSR 3: THE ROAD TO TODAY



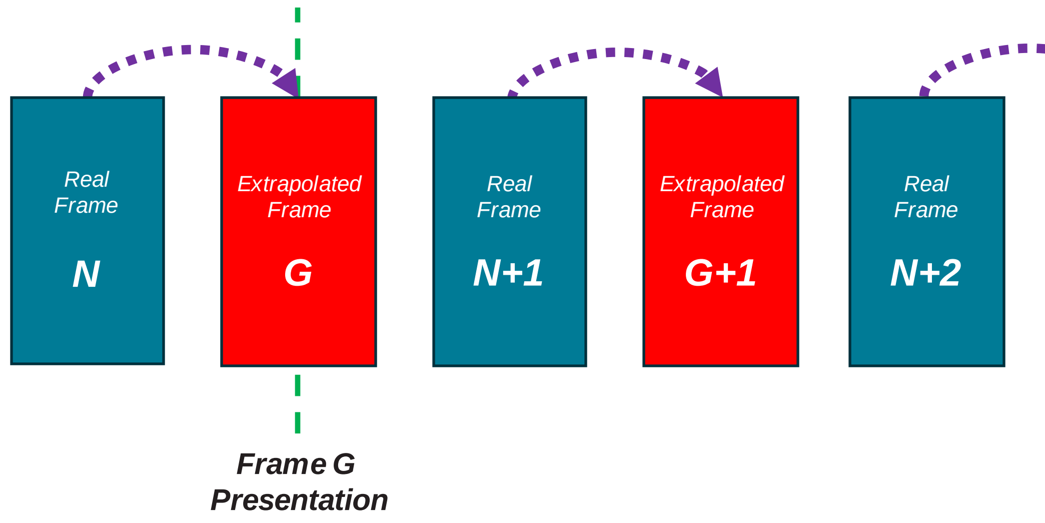
- Small team started work on FSR 3 in 2022.
- Algorithmic Solution.
- No changes to upscaling, except explicitly supporting NativeAA mode and a bugfix related to it.
- Wide adoption strategy.

COMMON FRAME GENERATION OPTIONS

- Interpolation
  - Generate data between two datapoints.
  - Adds latency – Frame N+1 required to present Frame G.
  - Two confident datapoints provides more data to infer upon.



- Extrapolation
  - Generate data beyond a datapoint.
  - Can be lower latency than Interpolation.
  - Less data, lower confidence.

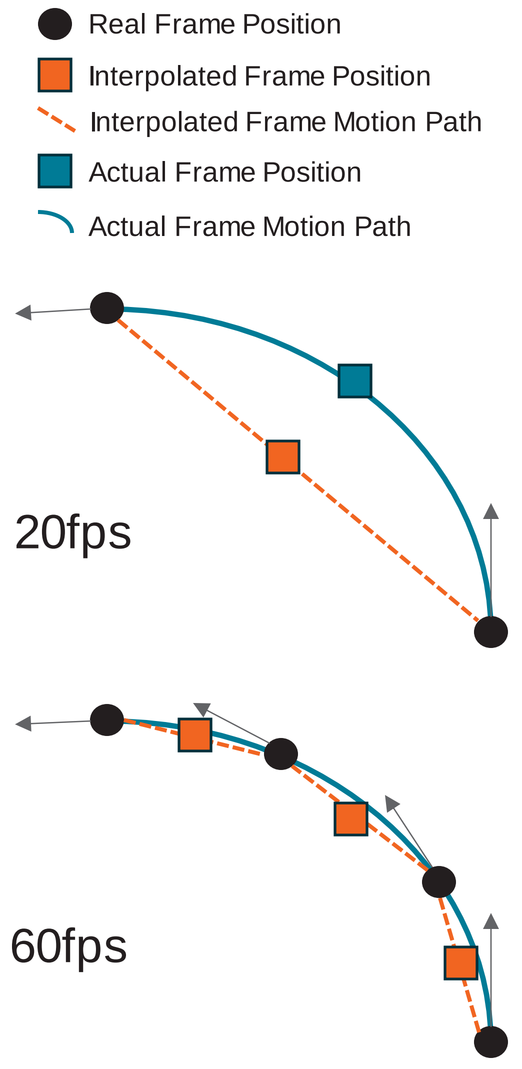


- Both options have complex frame-pacing challenges.

AMD FSR 3 – INTERPOLATED FRAMES

- FSR 3 uses frame interpolation.
  - Provides more confidence in data sampling for generating frames.
    - Potentially less artefacts per given measure of GPU cost.
    - Less inpainting – more on that later.

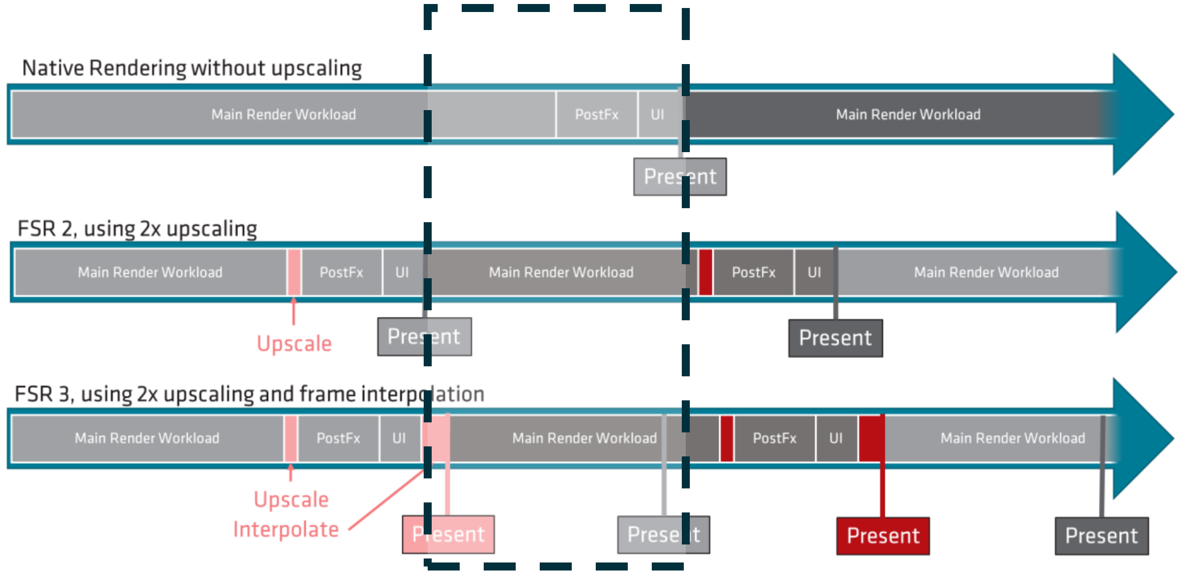
- Challenges.
  - Nonlinear motion.
    - Not trivial to figure out correct interpolated position.
    - Ensuring high fps (sample rate) can make approximations better.
  - Interpolation artifacts in high frequency regions.
  - Game motion vectors not matching the images presented to screen.
  - Small amount of additional latency.



LATENCY

- Three metrics:
  - Existing display native rendering latency.
  - Upscaled to display native rendering latency.
  - Upscaled to display + Frame Generation rendering latency.

- When FSR 3 Upscaling and FG enabled:
  - Latency should fall within the area highlighted.
  - Between upscaled-only, and native rendering.



MOTION VECTORS

- Frame Generation takes the motion vectors from upscale as input.
  - Disocclusions.
  - Reprojection.

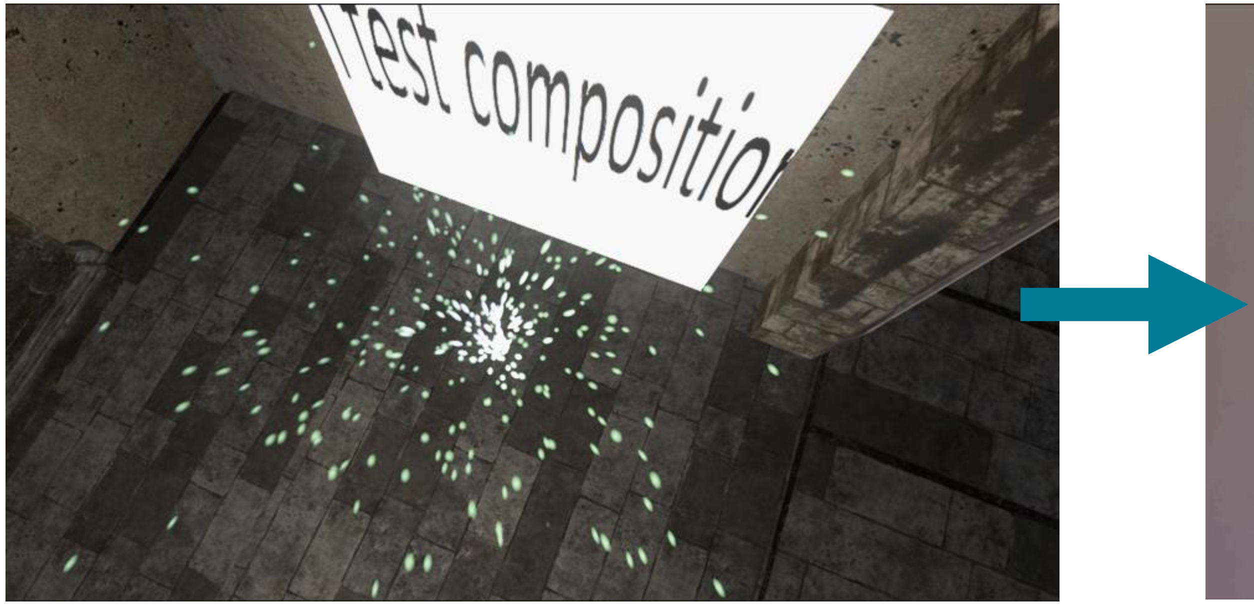
- Motion vectors for upscaling will not map directly to the final presented image.
  - User Interface.
  - Post-Processing effects – motion blur, noise effects.
  - Transparent objects and particles which did not have valid MVs.
  - Shadows of moving objects or static objects with moving light sources.

- Frame Generation requires more information to help reduce artefacts when game motion vectors are not enough.

- Optical flow can assist in these instances.

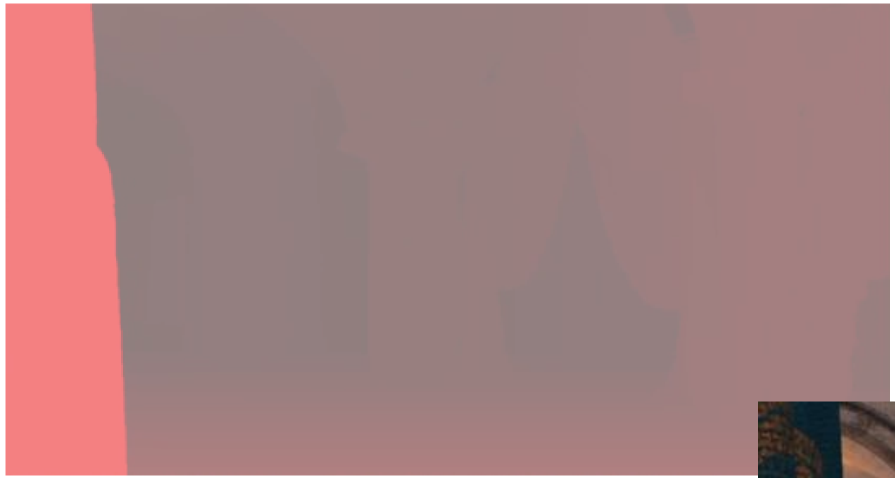
FSR 3 OPTICAL FLOW – OPTIMIZED FLUID MOTION FRAMES (FMF)

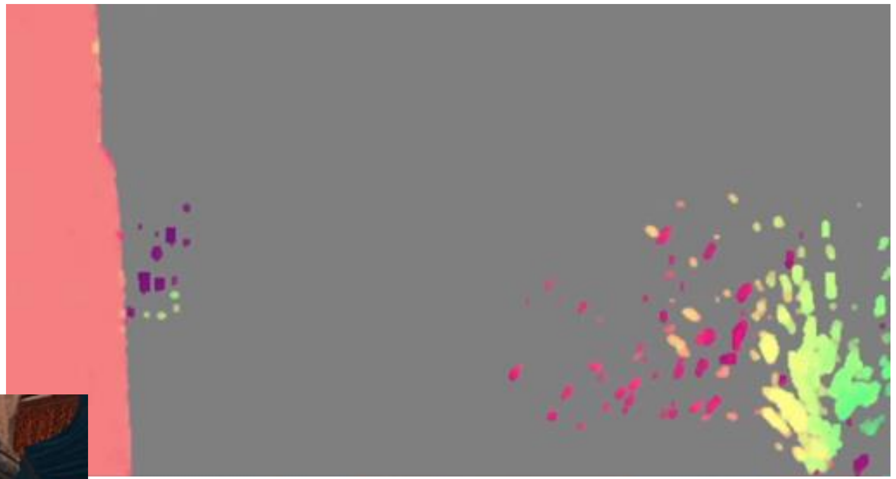
- Native HLSL Optical Flow reengineered from original OpenCL FMF codebase.
- Experimented with various options to configure operation.
  - Block and search size.
  - Trade-offs between what OF can detect versus false positive movements.
  - Performance impact.
- FSR 3 uses a single mode that gives good results across wide input range.



MOTION VECTOR FIELD

- Heuristics for how Game Motion Vectors and Optical Flow Vectors are combined.
  - Depends on how much confidence we can infer from both data streams.


Game Motion


Optical Flow

GENERATING INTERPOLATED FRAMES

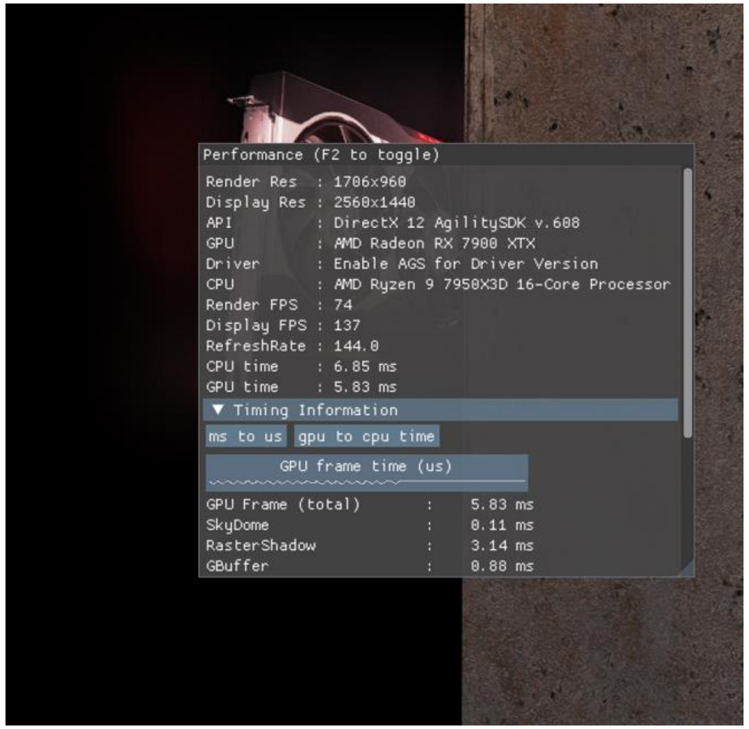
- Inputs.
  - Motion Vector fields.
  - Calculates disocclusions.
  - Reprojections.
  - Inpainting of gaps in confidence.

- Inpainting uses a mip pyramid of the frame to fill in any gaps that are considered low confidence data.
  - Low confidence data is when no information can be sampled from either N or N+1 input frame.

- Many complex issues that we solved; some others that can leave artefacts.
  - Challenges remain around translucent objects, fast dynamic shadows, vignette post processes.

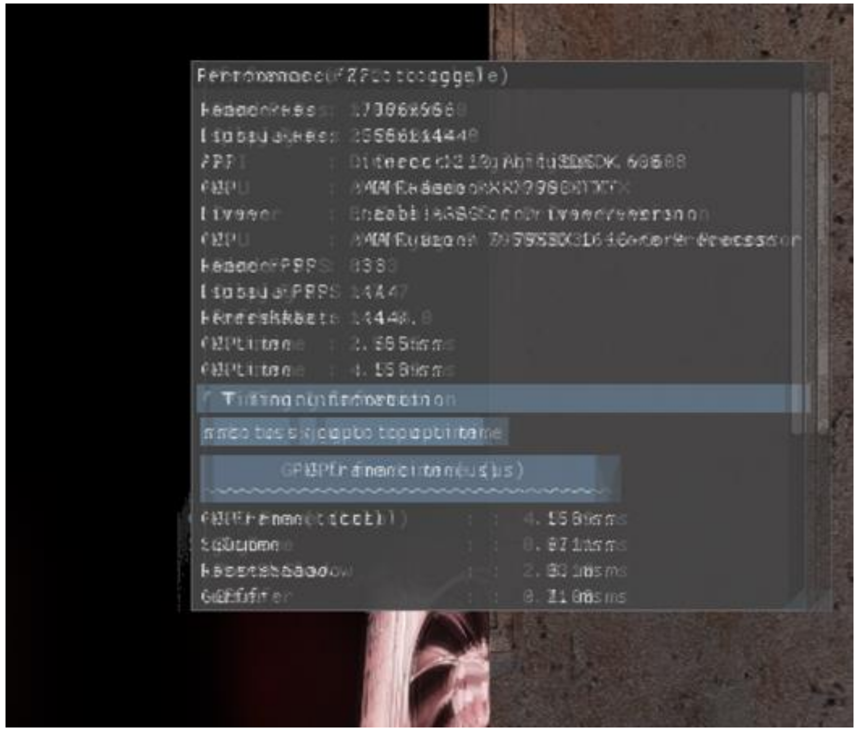
USER INTERFACE

- UI can be corrupted very easily when generating new frames.
  - Motion vectors from game are from the scene, not UI overlay.
  - Optical flow may be too coarse to assist with fine UI.



```
Performance (F2 to toggle)
Render Res  : 1706x960
Display Res : 2560x1440
API         : DirectX 12 AgilitySDK v.608
GPU         : AMD Radeon RX 7900 XTX
Driver      : Enable AGS for Driver Version
CPU         : AMD Ryzen 9 7950X3D 16-Core Processor
Render FPS  : 74
Display FPS : 137
RefreshRate : 144.0
CPU time    : 6.85 ms
GPU time    : 5.83 ms
▼ Timing Information
ms to us  gpu to cpu time
GPU frame time (us)
GPU Frame (total)         :     5.83 ms
SkyDome                   :     0.11 ms
RasterShadow              :     3.14 ms
GBuffer                   :     0.88 ms
```

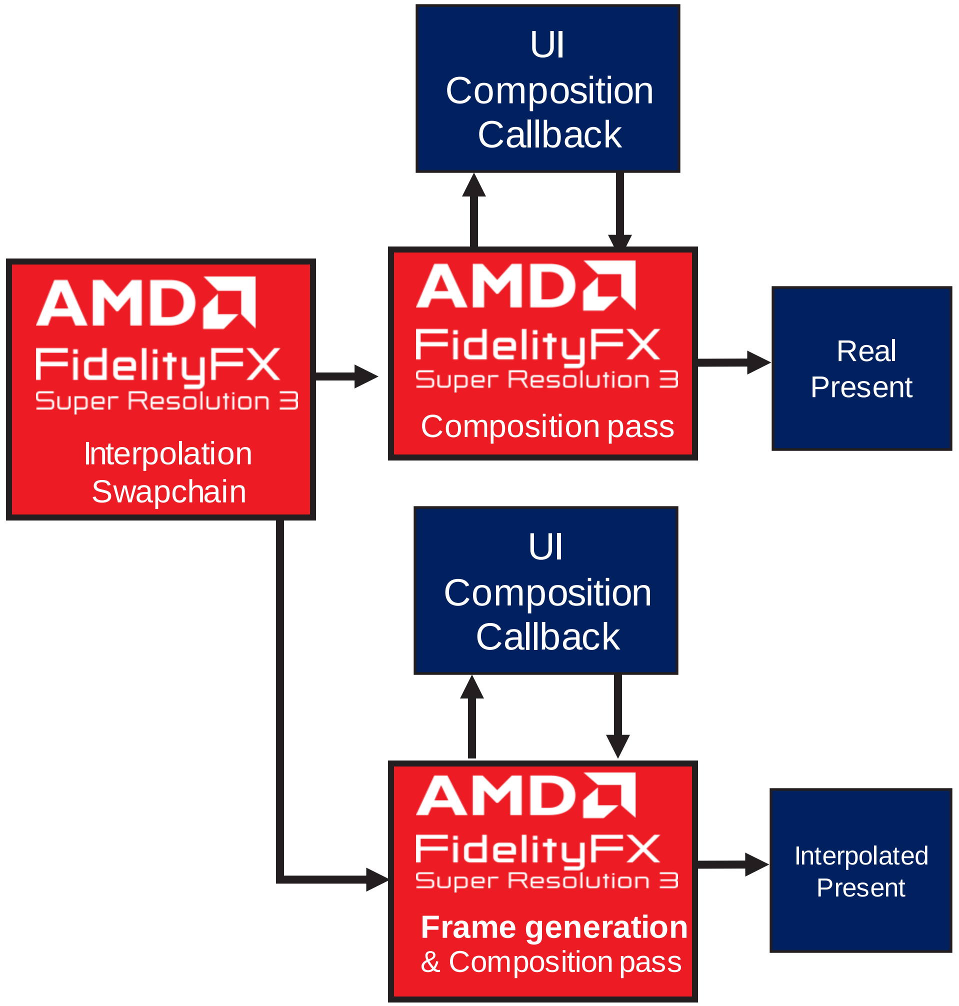
Real Frame in Motion



Generated frame in Motion
UI not separate

USER INTERFACE MODES

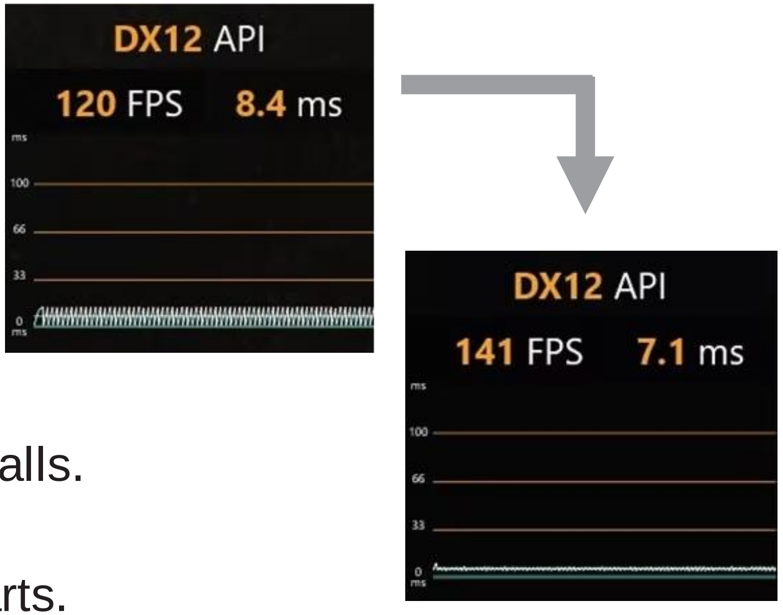
- We recommend rendering UI again on generated frames using our callback features.
  - Can yield full display-rate UI.
- Render-rate UI can be re-composited onto display-rate frames, however, may appear to update slower.
- We can also support a mode where FSR 3 is provided with resources with and without UI, and it can try to generate frames and keep artefacts low – but artefacts are possible.
  - Less artefacts than fully generating frames with UI included.
- Please refer to documentation provided with the Unreal Engine plugins for the options available there for artefact-free UI.


UI Composition Callback
AMD FidelityFX Super Resolution 3 Interpolation Swapchain
AMD FidelityFX Super Resolution 3 Composition pass
Real Present
UI Composition Callback
AMD FidelityFX Super Resolution 3 Frame generation & Composition pass
Interpolated Present

FRAME PACING

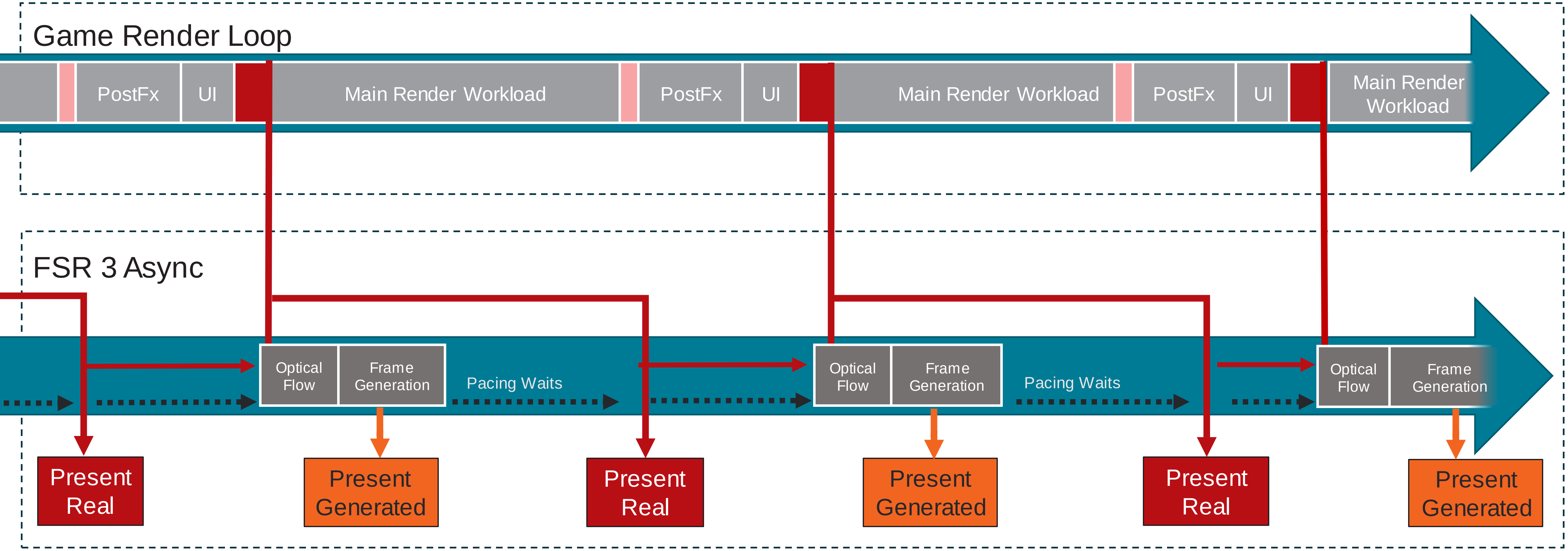
- Challenging – no driver involvement due to cross platform requirement.

- Things that hook the swapchain can change data in-flight, as well as add gpu render cost.
  - Cross platform solutions that hook the dx12 runtime.
  - Graphical overlays.
  - Performance measurement tools.
  - Screen recording tools.
  - Not running in fullscreen.



- Frame Interpolation Swapchain.
  - Decouples the game loop from presentation in order to pace present calls.
  - Some games may already do this.
  - Probably the hardest part to integrate as there are plenty of moving parts.

FRAME PACING

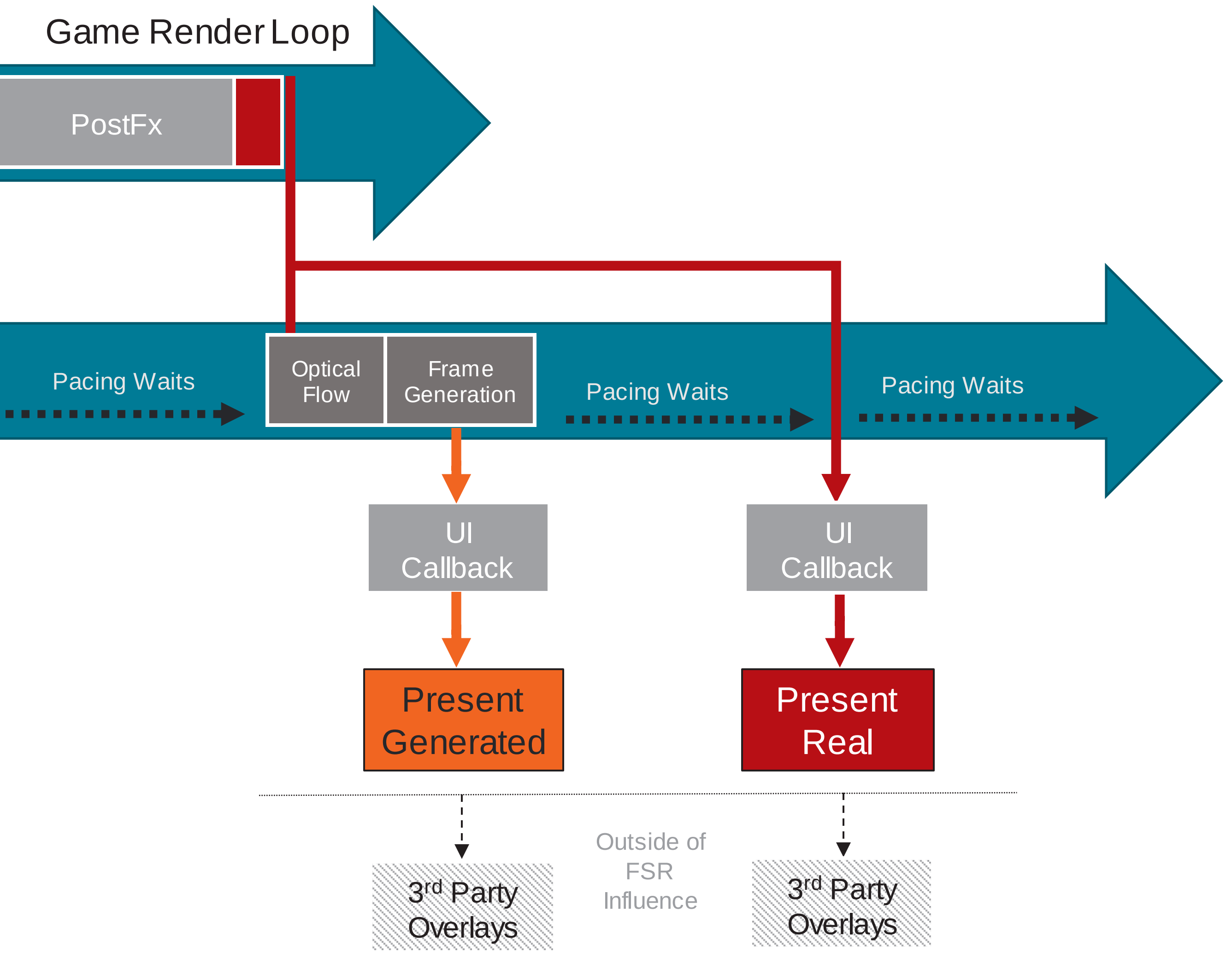


🟧 Generated Frame Datapath/Presents

🟥 Real Frame Datapath/Presents

🟥 Upscale

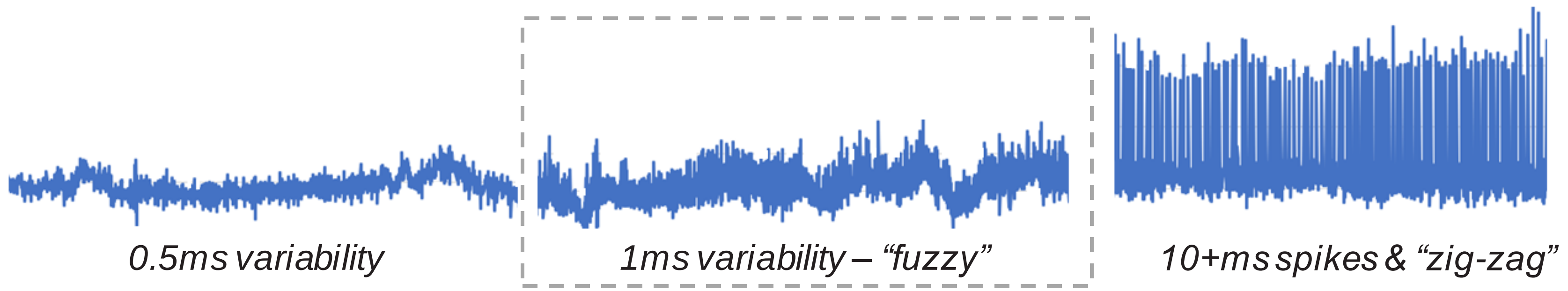
WHAT ARE THE PACING WAITS?



- Waits to ensure present calls occur within a range governed by a rolling average of previous frame times.
- UI Callback rendering cost requires consideration.
- We cannot measure the cost of overlays, as they occur outside of our control.
  - Which is why we recommend playing and benchmarking without them.

PACING IMPROVEMENTS FOR VARIABLE REFRESH RATE

- We discovered GPU HW, HW Scheduling, OS and Monitor can significantly change VRR experience
  - Used tooling to detect tears in displayed frames, indicating poor pacing.
- Fundamentally a question of perception, need many data points to make determination.



- We tested many implementations, across various hardware and frame-rate scenarios
  - Wild zig-zag frame time present-present graphs,
  - Normal frame time graphs with small variability,
  - Some more frame to frame variability – so could look "fuzzy".
- Also tested other frametime metrics that don't involve simple present timing.

- Result from testing was the "fuzzy" graphs provided best screen experience.
  - Hardware Scheduling can also help when enabled in the operating system.

PERFORMANCE AND INTEGRATION CONSIDERATIONS

Cost

Cost of generating a frame is low. Shader Model 6.2 minimum HW requirement.

Asynchronous compute

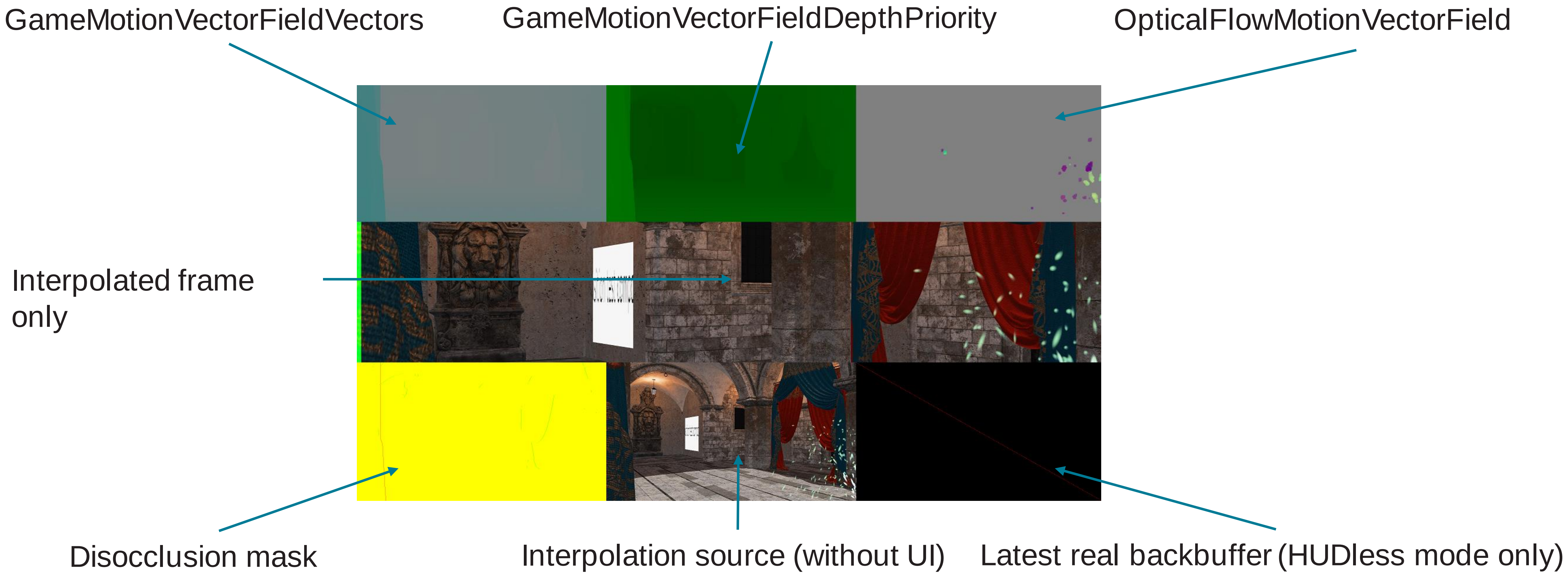
Can hide workloads and improve performance but requires careful integration. Non-async can have memory savings.

Latency

Latency is usually within the range of Upscaled-only to Native display resolution.

INTEGRATION DEBUG

- AMD 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)

HIGH PERFORMANCE RENDERING IN SNOWDROP USING FSR3

HAMPUS SIVERSSON

DISCLAIMER

- Not about FSR technology itself.
- Learnings from the integration side.

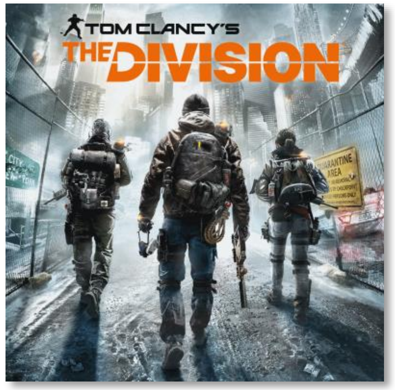
- Use or mention of third-party marks, logos, products, services, or solutions herein is for informational purposes only and no endorsement by AMD is intended or implied. GD-83.
- The information contained in this segment represents the view of Ubisoft as of the date presented. AMD and/or the third-party presenters have no obligation to update any forward-looking content in the above presentations. AMD is not responsible for the content of any third-party presentations and does not necessarily endorse the comments made therein. GD-84.

AGENDA

- Intro
- Integrating FSR
- External resources
- Snowdrop Pipeline
- Frame Generation

SNOWDROP

- The Division
- Avatar Frontiers of Pandora
- And more…



MOTIVATION

- Improve anti-aliasing solution
- Improve performance
- Unified solution across the board

FIDELITYFX SUPER RESOLUTION

- We landed on FSR. *(FSR2 at the time)*

AMD
FidelityFX
Super Resolution 3

FIDELITYFX SUPER RESOLUTION

- We landed on FSR. *(FSR2 at the time)*
- **Default solution**

AMD
FidelityFX
Super Resolution 3

FIDELITYFX SUPER RESOLUTION

- We landed on FSR. *(FSR2 at the time)*
- **Default solution**
  - Anti-aliasing
  - Upscaling

AMD FidelityFX Super Resolution 3

FIDELITYFX SUPER RESOLUTION

- We landed on FSR. *(FSR2 at the time)*
- **Default solution**
  - Anti-aliasing
  - Upscaling

- Used on all major platforms.

AMD FidelityFX Super Resolution 3

INTEGRATING FSR

INTEGRATING FSR

- Native backends
  - DX12
  - Vulkan
- Production ready and shippable.
- We wanted more control.

INTEGRATING FSR

- Custom Renderer
- Why not use what we already have?

INTEGRATING FSR

- Use FfxInterface to connect FSR to your own renderer.
- Gives lowest overhead possible.

```
FfxInterface interface;
interface.fpGetDeviceCapabilities     = Gfx_FSR_GetDeviceCapabilities;
interface.fpCreateBackendContext      = Gfx_FSR_CreateScratchContext;
interface.fpDestroyBackendContext     = Gfx_FSR_DestroyScratchContext;
interface.fpCreateResource            = Gfx_FSR_CreateResource;
interface.fpDestroyResource           = Gfx_FSR_DestroyResource;
interface.fpRegisterResource          = Gfx_FSR_RegisterResource;
interface.fpUnregisterResources       = Gfx_FSR_UnregisterResources;
interface.fpGetResource               = Gfx_FSR_GetInternalResource;
interface.fpGetResourceDescription    = Gfx_FSR_GetResourceDescription;
interface.fpCreatePipeline            = Gfx_FSR_CreatePipeline;
interface.fpDestroyPipeline           = Gfx_FSR_DestroyPipeline;
interface.fpScheduleGpuJob            = Gfx_FSR_ScheduleRenderJob;
interface.fpExecuteGpuJobs            = Gfx_FSR_ExecuteRenderJobs;
interface.scratchBuffer               = &scratchBuffer;
interface.scratchBufferSize           = sizeof scratchBuffer;
interface.device                      = device;
interface.fpSwapChainConfigureFrameGeneration = Gfx_FSR_SwapChainConfigureFrameGeneration;
```

INTEGRATING FSR

- Hooks are defined when setting up the FfxContext.

```
FfxFsr3ContextDescription createParams = {};
createParams.displaySize.width = aOutputResolution.x;
createParams.displaySize.height = aOutputResolution.y;
createParams.upscaleOutputSize.width = aDisplaySize.x;
createParams.upscaleOutputSize.height = aDisplaySize.y;
createParams.maxRenderSize.width = aMaxRenderSize.x;
createParams.maxRenderSize.height = aMaxRenderSize.y;
createParams.flags = FFX_FSR3_ENABLE_HIGH_DYNAMIC_RANGE | FFX_FSR3_ENABLE_DEPTH_INVERTED;
createParams.backBufferFormat = Gfx_FSR_GetSurfaceFormat(MR_Format::RGBA_UNORM8);

FfxDevice device = reinterpret_cast<FfxDevice>(MR_RenderCore::ourInstance);
Gfx_FSR_GetInterface(createParams.backendInterfaceSharedResources, device, myScratchBufferShared);
Gfx_FSR_GetInterface(createParams.backendInterfaceUpscaling, device, myScratchBufferUpscaling);
Gfx_FSR_GetInterface(createParams.backendInterfaceFrameInterpolation, device, myScratchBufferFrameGen);
FfxErrorCode errorCode = ffxFsr3ContextCreate(&myFSRContext, &createParams);
FFX_ASSERT(errorCode == FFX_OK);
```

INTEGRATING FSR

- Hooks are defined when setting up the FfxContext.

```
FfxFsr3ContextDescription createParams = {};
createParams.displaySize.width = aOutputResolution.x;
createParams.displaySize.height = aOutputResolution.y;
createParams.upscaleOutputSize.width = aDisplaySize.x;
createParams.upscaleOutputSize.height = aDisplaySize.y;
createParams.maxRenderSize.width = aMaxRenderSize.x;
createParams.maxRenderSize.height = aMaxRenderSize.y;
createParams.flags = FFX_FSR3_ENABLE_HIGH_DYNAMIC_RANGE | FFX_FSR3_ENABLE_DEPTH_INVERTED;
createParams.backBufferFormat = Gfx_FSR_GetSurfaceFormat(MR_Format::RGBA_UNORM_10_10_10_2);

FfxDevice device = reinterpret_cast<FfxDevice>(MR_RenderCore::ourInstance);
Gfx_FSR_GetInterface(createParams.backendInterfaceSharedResources, device, myScratchBufferShared);
Gfx_FSR_GetInterface(createParams.backendInterfaceUpscaling, device, myScratchBufferUpscaling);
Gfx_FSR_GetInterface(createParams.backendInterfaceFrameInterpolation, device, myScratchBufferFrameGen);
FfxErrorCode errorCode = ffxFsr3ContextCreate(&myFSRContext, &createParams);
FFX_ASSERT(errorCode == FFX_OK);
```

INTEGRATING FSR

- Lets look at an example:
  - `Gfx_FSR_ExecuteRenderJobs`

```
FfxInterface interface;
interface.fpGetDeviceCapabilities     = Gfx_FSR_GetDeviceCapabilities;
interface.fpCreateBackendContext      = Gfx_FSR_CreateScratchContext;
interface.fpDestroyBackendContext     = Gfx_FSR_DestroyScratchContext;
interface.fpCreateResource            = Gfx_FSR_CreateResource;
interface.fpDestroyResource           = Gfx_FSR_DestroyResource;
interface.fpRegisterResource          = Gfx_FSR_RegisterResource;
interface.fpUnregisterResources       = Gfx_FSR_UnregisterResources;
interface.fpGetResource               = Gfx_FSR_GetInternalResource;
interface.fpGetResourceDescription    = Gfx_FSR_GetResourceDescription;
interface.fpCreatePipeline            = Gfx_FSR_CreatePipeline;
interface.fpDestroyPipeline           = Gfx_FSR_DestroyPipeline;
interface.fpScheduleGpuJob            = Gfx_FSR_ScheduleRenderJob;
interface.fpExecuteGpuJobs            = Gfx_FSR_ExecuteRenderJobs;
interface.scratchBuffer               = &scratchBuffer;
interface.scratchBufferSize           = sizeof scratchBuffer;
interface.device                      = device;
interface.fpSwapChainConfigureFrameGeneration = Gfx_FSR_SwapChainConfigureFrameGeneration;
```

INTEGRATING FSR

- Lets look at an example:
  - `Gfx_FSR_ExecuteRenderJobs`

```
FfxErrorCode Gfx_FSR_ExecuteRenderJobs(FfxInterface* backendInterface, FfxCommandList commandList)
{
    Gfx_FSR_ScratchBuffer& scratch = Gfx_FSR_GetScratch(backendInterface);
    MR_RenderContext* ctx = Gfx_FSR_GetContext(commandList);

    for (FfxGpuJobDescription& job : scratch.myPendingJobs)
    {
        switch (job.jobType)
        {
        case FFX_GPU_JOB_CLEAR_FLOAT:
        {
            FfxClearFloatJobDescription& clear = job.clearJobDescriptor;
            Gfx_FSR_Resource& resource = scratch.myResources[clear.target.internalIndex];
            const MC_Vector4f clearValue(clear.color[0], clear.color[1], clear.color[2], clear.color[3]);
            ctx->ClearPixels(resource.uav, clearValue);
            break;
        }
        case FFX_GPU_JOB_COPY:
        {
            FfxCopyJobDescription& copy = job.copyJobDescriptor;
            Gfx_FSR_Resource& srcResource = scratch.myResources[copy.src.internalIndex];
            Gfx_FSR_Resource& dstResource = scratch.myResources[copy.dst.internalIndex];

            MR_PixelStorage* src = srcResource.myPersistentPixels.Get();
            MR_PixelStorage* dst = dstResource.myPersistentPixels.Get();

            ctx->CopyPixels(dst, src);
            break;
        }
        case FFX_GPU_JOB_COMPUTE:
            Gfx_FSR_ExecuteComputeJob(scratch, ctx, job.computeJobDescriptor);
            break;
        }
    }
    scratch.myPendingJobs.RemoveAll();
    return FFX_OK;
}
```

INTEGRATING FSR

- Lets look at an example:
  - `Gfx_FSR_ExecuteRenderJobs`

```
FfxErrorCode Gfx_FSR_ExecuteRenderJobs(FfxInterface* backendInterface, FfxCommandList commandList)
{
    Gfx_FSR_ScratchBuffer& scratch = Gfx_FSR_GetScratch(backendInterface);
    MR_RenderContext* ctx = Gfx_FSR_GetContext(commandList);

    for (FfxGpuJobDescription& job : scratch.myPendingJobs)
    {
        switch (job.jobType)
        {
        case FFX_GPU_JOB_CLEAR_FLOAT:
        {
            FfxClearFloatJobDescription& clear = job.clearJobDescriptor;
            Gfx_FSR_Resource& resource = scratch.myResources[clear.target.internalIndex];
            const MC_Vector4f clearValue(clear.color[0], clear.color[1], clear.color[2], clear.color[3]);
            ctx->ClearPixels(resource.uav, clearValue);
            break;
        }
        case FFX_GPU_JOB_COPY:
        {
            FfxCopyJobDescription& copy = job.copyJobDescriptor;
            Gfx_FSR_Resource& srcResource = scratch.myResources[copy.src.internalIndex];
            Gfx_FSR_Resource& dstResource = scratch.myResources[copy.dst.internalIndex];

            MR_PixelStorage* src = srcResource.myPersistentPixels.Get();
            MR_PixelStorage* dst = dstResource.myPersistentPixels.Get();

            ctx->CopyPixels(dst, src);
            break;
        }
        case FFX_GPU_JOB_COMPUTE:
            Gfx_FSR_ExecuteComputeJob(scratch, ctx, job.computeJobDescriptor);
            break;
        }
    }
    scratch.myPendingJobs.RemoveAll();
    return FFX_OK;
}
```

INTEGRATING FSR

- Lets look at an example:
  - `Gfx_FSR_ExecuteRenderJobs`

```
FfxErrorCode Gfx_FSR_ExecuteRenderJobs(FfxInterface* backendInterface, FfxCommandList commandList)
{
    Gfx_FSR_ScratchBuffer& scratch = Gfx_FSR_GetScratch(backendInterface);
    MR_RenderContext* ctx = Gfx_FSR_GetContext(commandList);

    for (FfxGpuJobDescription& job : scratch.myPendingJobs)
    {
        switch (job.jobType)
        {
        case FFX_GPU_JOB_CLEAR_FLOAT:
        {
            FfxClearFloatJobDescription& clear = job.clearJobDescriptor;
            Gfx_FSR_Resource& resource = scratch.myResources[clear.target.internalIndex];
            const MC_Vector4f clearValue(clear.color[0], clear.color[1], clear.color[2], clear.color[3]);
            ctx->ClearPixels(resource.uav, clearValue);
            break;
        }
        case FFX_GPU_JOB_COPY:
        {
            FfxCopyJobDescription& copy = job.copyJobDescriptor;
            Gfx_FSR_Resource& srcResource = scratch.myResources[copy.src.internalIndex];
            Gfx_FSR_Resource& dstResource = scratch.myResources[copy.dst.internalIndex];

            MR_PixelStorage* src = srcResource.myPersistentPixels.Get();
            MR_PixelStorage* dst = dstResource.myPersistentPixels.Get();

            ctx->CopyPixels(dst, src);
            break;
        }
        case FFX_GPU_JOB_COMPUTE:
            Gfx_FSR_ExecuteComputeJob(scratch, ctx, job.computeJobDescriptor);
            break;
        }
    }
    scratch.myPendingJobs.RemoveAll();
    return FFX_OK;
}
```

INTEGRATING FSR

- Let's go one step deeper:
  - Gfx_FSR_ExecuteComputeJobs

```
FfxErrorCode Gfx_FSR_ExecuteRenderJobs(FfxInterface* backendInterface, FfxCommandList commandList)
{
    Gfx_FSR_ScratchBuffer& scratch = Gfx_FSR_GetScratch(backendInterface);
    MR_RenderContext* ctx = Gfx_FSR_GetContext(commandList);

    for (FfxGpuJobDescription& job : scratch.myPendingJobs)
    {
        switch (job.jobType)
        {
        case FFX_GPU_JOB_CLEAR_FLOAT:
        {
            FfxClearFloatJobDescription& clear = job.clearJobDescriptor;
            Gfx_FSR_Resource& resource = scratch.myResources[clear.target.internalIndex];
            const MC_Vector4f clearValue(clear.color[0], clear.color[1], clear.color[2], clear.color[3]);
            ctx->ClearPixels(resource.uav, clearValue);
            break;
        }
        case FFX_GPU_JOB_COPY:
        {
            FfxCopyJobDescription& copy = job.copyJobDescriptor;
            Gfx_FSR_Resource& srcResource = scratch.myResources[copy.src.internalIndex];
            Gfx_FSR_Resource& dstResource = scratch.myResources[copy.dst.internalIndex];

            MR_PixelStorage* src = srcResource.myPersistentPixels.Get();
            MR_PixelStorage* dst = dstResource.myPersistentPixels.Get();

            ctx->CopyPixels(dst, src);
            break;
        }
        case FFX_GPU_JOB_COMPUTE:
            Gfx_FSR_ExecuteComputeJob(scratch, ctx, job.computeJobDescriptor);
            break;
        }
    }
    scratch.myPendingJobs.RemoveAll();
    return FFX_OK;
}
```

INTEGRATING FSR

- `Gfx_FSR_ExecuteComputeJobs`

```
FfxErrorCode Gfx_FSR_ExecuteComputeJob(Gfx_FSR_ScratchBuffer& aScratch, FfxComputeJobDescription& aJob)
{
  MR_ShaderState* shader = static_cast<MR_ShaderState*>(aJob.pipeline.pipeline);
  Gfx_FSR_BindingsBuffer bindings;
  MR_RenderContext* aCtx = MR_RenderContext::GetCurrent();
  for (uint i=0; i<aJob.pipeline.srvTextureCount; ++i)
  {
    FfxResourceBinding& binding = aJob.pipeline.srvTextureBindings[i];
    Gfx_FSR_Resource& resource = aScratch.myResources[aJob.srvTextures[i].internalIndex];
    MR_Texture* srv = resource.mySRV;
    bindings.srvBindings[binding.slotIndex] = srv->GetDescriptorIndex();
    aCtx->MarkResourceForBindless(*srv->GetStorage(), MR_TRANSITION_TO_SHADER_READ);
  }
  for (uint i=0; i<aJob.pipeline.uavTextureCount; ++i)
  {
    FfxResourceBinding& binding = aJob.pipeline.uavTextureBindings[i];
    Gfx_FSR_Resource& resource = aScratch.myResources[aJob.uavTextures[i].internalIndex];
    uint highestMipIndex = resource.myUAVs.CountU() - 1;
    MR_Texture* uav = resource.myUAVs[MC_Min(aJob.uavTextureMips[i], highestMipIndex)];
    bindings.uavBindings[binding.slotIndex] = uav->GetDescriptorIndex();
    aCtx->MarkResourceForBindless(*uav->GetStorage(), MR_TRANSITION_TO_SHADER_WRITE);
  }
  for (uint i=0; i<aJob.pipeline.constCount; ++i)
  {
    uint numInts = aJob.cbs[i].num32BitEntries;
    uint size = numInts * sizeof(uint32);
    aCtx->SetConstantBuffer(size, aJob.cbs[i].data, i);
  }

  aCtx->SetConstantBuffer(sizeof(bindings), &bindings, 3);
  aCtx->SetShaderState(shader);
  aCtx->Dispatch(aJob.dimensions[0], aJob.dimensions[1], aJob.dimensions[2]);
  return FFX_OK;
}
```

INTEGRATING FSR

- `Gfx_FSR_ExecuteComputeJobs`
  - Gather resource info
  - Mark for bindless

```
FfxErrorCode Gfx_FSR_ExecuteComputeJob(Gfx_FSR_ScratchBuffer& aScratch, FfxComputeJobDescription& aJob)
{
  MR_ShaderState* shader = static_cast<MR_ShaderState*>(aJob.pipeline.pipeline);
  Gfx_FSR_BindingsBuffer bindings;
  MR_RenderContext* aCtx = MR_RenderContext::GetCurrent();
  r (uint i=0; i<aJob.pipeline.srvTextureCount; ++i)

    FfxResourceBinding& binding = aJob.pipeline.srvTextureBindings[i];
    Gfx_FSR_Resource& resource = aScratch.myResources[aJob.srvTextures[i].internalIndex];
    MR_Texture* srv = resource.mySRV;
    bindings.srvBindings[binding.slotIndex] = srv->GetDescriptorIndex();
    aCtx->MarkResourceForBindless(*srv->GetStorage(), MR_TRANSITION_TO_SHADER_READ);

  r (uint i=0; i<aJob.pipeline.uavTextureCount; ++i)

    FfxResourceBinding& binding = aJob.pipeline.uavTextureBindings[i];
    Gfx_FSR_Resource& resource = aScratch.myResources[aJob.uavTextures[i].internalIndex];
    uint highestMipIndex = resource.myUAVs.CountU() - 1;
    MR_Texture* uav = resource.myUAVs[MC_Min(aJob.uavTextureMips[i], highestMipIndex)];
    bindings.uavBindings[binding.slotIndex] = uav->GetDescriptorIndex();
    aCtx->MarkResourceForBindless(*uav->GetStorage(), MR_TRANSITION_TO_SHADER_WRITE);

  [illegible]
  {
    uint numInts = aJob.cbs[i].num32BitEntries;
    uint size = numInts * sizeof(uint32);
    aCtx->SetConstantBuffer(size, aJob.cbs[i].data, i);
  }

  aCtx->SetConstantBuffer(sizeof(bindings), &bindings, 3);
  aCtx->SetShaderState(shader);
  aCtx->Dispatch(aJob.dimensions[0], aJob.dimensions[1], aJob.dimensions[2]);
  return FFX_OK;
}
```

INTEGRATING FSR

- `Gfx_FSR_ExecuteComputeJobs`
  - `Gather resource info`
  - `Mark for bindless`
  - `Pass SDK constants`

```
FfxErrorCode Gfx_FSR_ExecuteComputeJob(Gfx_FSR_ScratchBuffer& aScratch, FfxComputeJobDescription& aJob)
{
  MR_ShaderState* shader = static_cast<MR_ShaderState*>(aJob.pipeline.pipeline);
  Gfx_FSR_BindingsBuffer bindings;
  MR_RenderContext* aCtx = MR_RenderContext::GetCurrent();
  for (uint i=0; i<aJob.pipeline.srvTextureCount; ++i)
  {
    FfxResourceBinding& binding = aJob.pipeline.srvTextureBindings[i];
    Gfx_FSR_Resource& resource = aScratch.myResources[aJob.srvTextures[i].internalIndex];
    MR_Texture* srv = resource.mySRV;
    bindings.srvBindings[binding.slotIndex] = srv->GetDescriptorIndex();
    aCtx->MarkResourceForBindless(*srv->GetStorage(), MR_TRANSITION_TO_SHADER_READ);
  }
  for (uint i=0; i<aJob.pipeline.uavTextureCount; ++i)
  {
    FfxResourceBinding& binding = aJob.pipeline.uavTextureBindings[i];
    Gfx_FSR_Resource& resource = aScratch.myResources[aJob.uavTextures[i].internalIndex];
    uint highestMipIndex = resource.myUAVs.CountU() - 1;
    MR_Texture* uav = resource.myUAVs[MC_Min(aJob.uavTextureMips[i], highestMipIndex)];
    bindings.uavBindings[binding.slotIndex] = uav->GetDescriptorIndex();
    aCtx->MarkResourceForBindless(*uav->GetStorage(), MR_TRANSITION_TO_SHADER_WRITE);
  }
  for (uint i=0; i<aJob.pipeline.constCount; ++i)
  {
    uint numInts = aJob.cbs[i].num32BitEntries;
    uint size = numInts * sizeof(uint32);
    aCtx->SetConstantBuffer(size, aJob.cbs[i].data, i);
  }

  aCtx->SetConstantBuffer(sizeof(bindings), &bindings, 3);
  aCtx->SetShaderState(shader);
  aCtx->Dispatch(aJob.dimensions[0], aJob.dimensions[1], aJob.dimensions[2]);
  return FFX_OK;
}
```

INTEGRATING FSR

- Gfx_FSR_ExecuteComputeJobs
  - Gather resource info
  - Mark for bindless
  - Pass SDK constants
  - Pass bindless descriptors

```
FfxErrorCode Gfx_FSR_ExecuteComputeJob(Gfx_FSR_ScratchBuffer& aScratch, FfxComputeJobDescription& aJob)
{
  MR_ShaderState* shader = static_cast<MR_ShaderState*>(aJob.pipeline.pipeline);
  Gfx_FSR_BindingsBuffer bindings;
  MR_RenderContext* aCtx = MR_RenderContext::GetCurrent();
  for (uint i=0; i<aJob.pipeline.srvTextureCount; ++i)
  {
    FfxResourceBinding& binding = aJob.pipeline.srvTextureBindings[i];
    Gfx_FSR_Resource& resource = aScratch.myResources[aJob.srvTextures[i].internalIndex];
    MR_Texture* srv = resource.mySRV;
    bindings.srvBindings[binding.slotIndex] = srv->GetDescriptorIndex();
    aCtx->MarkResourceForBindless(*srv->GetStorage(), MR_TRANSITION_TO_SHADER_READ);
  }
  for (uint i=0; i<aJob.pipeline.uavTextureCount; ++i)
  {
    FfxResourceBinding& binding = aJob.pipeline.uavTextureBindings[i];
    Gfx_FSR_Resource& resource = aScratch.myResources[aJob.uavTextures[i].internalIndex];
    uint highestMipIndex = resource.myUAVs.CountU() - 1;
    MR_Texture* uav = resource.myUAVs[MC_Min(aJob.uavTextureMips[i], highestMipIndex)];
    bindings.uavBindings[binding.slotIndex] = uav->GetDescriptorIndex();
    aCtx->MarkResourceForBindless(*uav->GetStorage(), MR_TRANSITION_TO_SHADER_WRITE);
  }
  for (uint i=0; i<aJob.pipeline.constCount; ++i)
  {
    uint numInts = aJob.cbs[i].num32BitEntries;
    uint size = numInts * sizeof(uint32);
    aCtx->SetConstantBuffer(size, aJob.cbs[i].data, i);
  }

  a[illegible]tx->SetConstantBuffer(sizeof(bindings), &bindings, 3);
  a[illegible]
  aCtx->Dispatch(aJob.dimensions[0], aJob.dimensions[1], aJob.dimensions[2]);
  return FFX_OK;
}
```

INTEGRATING FSR

- `Gfx_FSR_ExecuteComputeJobs`
  - Gather resource info
  - Mark for bindless
  - Pass SDK constants
  - Pass bindless descriptors
  - Set shader
  - Dispatch

```
FfxErrorCode Gfx_FSR_ExecuteComputeJob(Gfx_FSR_ScratchBuffer& aScratch, FfxComputeJobDescription& aJob)
{
  MR_ShaderState* shader = static_cast<MR_ShaderState*>(aJob.pipeline.pipeline);
  Gfx_FSR_BindingsBuffer bindings;
  MR_RenderContext* aCtx = MR_RenderContext::GetCurrent();
  for (uint i=0; i<aJob.pipeline.srvTextureCount; ++i)
  {
    FfxResourceBinding& binding = aJob.pipeline.srvTextureBindings[i];
    Gfx_FSR_Resource& resource = aScratch.myResources[aJob.srvTextures[i].internalIndex];
    MR_Texture* srv = resource.mySRV;
    bindings.srvBindings[binding.slotIndex] = srv->GetDescriptorIndex();
    aCtx->MarkResourceForBindless(*srv->GetStorage(), MR_TRANSITION_TO_SHADER_READ);
  }
  for (uint i=0; i<aJob.pipeline.uavTextureCount; ++i)
  {
    FfxResourceBinding& binding = aJob.pipeline.uavTextureBindings[i];
    Gfx_FSR_Resource& resource = aScratch.myResources[aJob.uavTextures[i].internalIndex];
    uint highestMipIndex = resource.myUAVs.CountU() - 1;
    MR_Texture* uav = resource.myUAVs[MC_Min(aJob.uavTextureMips[i], highestMipIndex)];
    bindings.uavBindings[binding.slotIndex] = uav->GetDescriptorIndex();
    aCtx->MarkResourceForBindless(*uav->GetStorage(), MR_TRANSITION_TO_SHADER_WRITE);
  }
  for (uint i=0; i<aJob.pipeline.constCount; ++i)
  {
    uint numInts = aJob.cbs[i].num32BitEntries;
    uint size = numInts * sizeof(uint32);
    aCtx->SetConstantBuffer(size, aJob.cbs[i].data, i);
  }

  aCtx->SetConstantBuffer(sizeof(bindings), &bindings, 5);
  aCtx->SetShaderState(shader);
  aCtx->Dispatch(aJob.dimensions[0], aJob.dimensions[1], aJob.dimensions[2]);
  return FFX_OK;
}
```

INTEGRATING FSR

- Gfx_FSR_ExecuteComputeJobs
  - Gather resource info
  - Mark for bindless
  - Pass SDK constants
  - Pass bindless descriptors
  - Set shader
  - Dispatch

```
FfxErrorCode Gfx_FSR_ExecuteComputeJob(Gfx_FSR_ScratchBuffer& aScratch, FfxComputeJobDescription& aJob)
{
  MR_ShaderState* shader = static_cast<MR_ShaderState*>(aJob.pipeline.pipeline);
  Gfx_FSR_BindingsBuffer bindings;
  MR_RenderContext* aCtx = MR_RenderContext::GetCurrent();
  for (uint i=0; i<aJob.pipeline.srvTextureCount; ++i)
  {
    FfxResourceBinding& binding = aJob.pipeline.srvTextureBindings[i];
    Gfx_FSR_Resource& resource = aScratch.myResources[aJob.srvTextures[i].internalIndex];
    MR_Texture* srv = resource.mySRV;
    bindings.srvBindings[binding.slotIndex] = srv->GetDescriptorIndex();
    aCtx->MarkResourceForBindless(*srv->GetStorage(), MR_TRANSITION_TO_SHADER_READ);
  }
  for (uint i=0; i<aJob.pipeline.uavTextureCount; ++i)
  {
    FfxResourceBinding& binding = aJob.pipeline.uavTextureBindings[i];
    Gfx_FSR_Resource& resource = aScratch.myResources[aJob.uavTextures[i].internalIndex];
    uint highestMipIndex = resource.myUAVs.CountU() - 1;
    MR_Texture* uav = resource.myUAVs[MC_Min(aJob.uavTextureMips[i], highestMipIndex)];
    bindings.uavBindings[binding.slotIndex] = uav->GetDescriptorIndex();
    aCtx->MarkResourceForBindless(*uav->GetStorage(), MR_TRANSITION_TO_SHADER_WRITE);
  }
  for (uint i=0; i<aJob.pipeline.constCount; ++i)
  {
    uint numInts = aJob.cbs[i].num32BitEntries;
    uint size = numInts * sizeof(uint32);
    aCtx->SetConstantBuffer(size, aJob.cbs[i].data, i);
  }

  aCtx->SetConstantBuffer(sizeof(bindings), &bindings, 3);
  aCtx->SetShaderState(shader);
  aCtx->Dispatch(aJob.dimensions[0], aJob.dimensions[1], aJob.dimensions[2]);
  return FFX_OK;
}
```

INTEGRATING FSR

- Resources and Pipelines are created on context creation.

```
FfxInterface interface;
interface.fpGetDeviceCapabilities    = Gfx_FSR_GetDeviceCapabilities;
interface.fpCreateBackendContext     = Gfx_FSR_CreateScratchContext;
interface.fpDestroyBackendContext    = Gfx_FSR_DestroyScratchContext;
interface.fpCreateResource           = Gfx_FSR_CreateResource;
interface.fpDestroyResource          = Gfx_FSR_DestroyResource;
interface.fpRegisterResource         = Gfx_FSR_RegisterResource;
interface.fpUnregisterResources      = Gfx_FSR_UnregisterResources;
interface.fpGetResource              = Gfx_FSR_GetInternalResource;
interface.fpGetResourceDescription   = Gfx_FSR_GetResourceDescription;
interface.fpCreatePipeline           = Gfx_FSR_CreatePipeline;
interface.fpDestroyPipeline          = Gfx_FSR_DestroyPipeline;
interface.fpScheduleGpuJob           = Gfx_FSR_ScheduleRenderJob;
interface.fpExecuteGpuJobs           = Gfx_FSR_ExecuteRenderJobs;
interface.scratchBuffer              = &scratchBuffer;
interface.scratchBufferSize          = sizeof scratchBuffer;
interface.device                     = device;
interface.fpSwapChainConfigureFrameGeneration = Gfx_FSR_SwapChainConfigureFrameGeneration;
```

INTEGRATING FSR

- Resources and Pipelines are created on context creation.
  - External resources are registered in runtime.

```
FfxInterface interface;
interface.fpGetDeviceCapabilities     = Gfx_FSR_GetDeviceCapabilities;
interface.fpCreateBackendContext      = Gfx_FSR_CreateScratchContext;
interface.fpDestroyBackendContext     = Gfx_FSR_DestroyScratchContext;
interface.fpCreateResource            = Gfx_FSR_CreateResource;
interface.fpDestroyResource           = Gfx_FSR_DestroyResource;
interface.fpRegisterResource          = Gfx_FSR_RegisterResource;
interface.fpUnregisterResources       = Gfx_FSR_UnregisterResources;
interface.fpGetResource               = Gfx_FSR_GetInternalResource;
interface.fpGetResourceDescription    = Gfx_FSR_GetResourceDescription;
interface.fpCreatePipeline            = Gfx_FSR_CreatePipeline;
interface.fpDestroyPipeline           = Gfx_FSR_DestroyPipeline;
interface.fpScheduleGpuJob            = Gfx_FSR_ScheduleRenderJob;
interface.fpExecuteGpuJobs            = Gfx_FSR_ExecuteRenderJobs;
interface.scratchBuffer               = &scratchBuffer;
interface.scratchBufferSize           = sizeof scratchBuffer;
interface.device                      = device;
interface.fpSwapChainConfigureFrameGeneration = Gfx_FSR_SwapChainConfigureFrameGeneration;
```

INTEGRATING FSR

- Resources and Pipelines are created on context creation.
  - External resources are registered in runtime.

```
FfxInterface interface;
interface.fpGetDeviceCapabilities    = Gfx_FSR_GetDeviceCapabilities;
interface.fpCreateBackendContext     = Gfx_FSR_CreateScratchContext;
interface.fpDestroyBackendContext    = Gfx_FSR_DestroyScratchContext;
interface.fpCreateResource           = Gfx_FSR_CreateResource;
interface.fpDestroyResource          = Gfx_FSR_DestroyResource;
interface.fpRegisterResource         = Gfx_FSR_RegisterResource;
interface.fpUnregisterResources      = Gfx_FSR_UnregisterResources;
interface.fpGetResource              = Gfx_FSR_GetInternalResource;
interface.fpGetResourceDescription   = Gfx_FSR_GetResourceDescription;
interface.fpCreatePipeline           = Gfx_FSR_CreatePipeline;
interface.fpDestroyPipeline          = Gfx_FSR_DestroyPipeline;
interface.fpScheduleGpuJob           = Gfx_FSR_ScheduleRenderJob;
interface.fpExecuteGpuJobs           = Gfx_FSR_ExecuteRenderJobs;
interface.scratchBuffer              = &scratchBuffer;
interface.scratchBufferSize          = sizeof scratchBuffer;
interface.device                     = device;
interface.fpSwapChainConfigureFrameGeneration = Gfx_FSR_SwapChainConfigureFrameGeneration;
```

RESOURCE CREATION

- Custom resource types.
- Initialize based on usage.

RESOURCE CREATION

- Custom resource types.
- Initialize based on usage.

- FFX_RESOURCE_FLAGS_ALIASABLE
- Allows transient resources.
- Allocation can be done just-in-time.
- Minimize memory overhead.

RESOURCE CREATION

- Allocated just before ffxFsr3ContextDispatchX.
  - For each context.
- Discarded right after.

```
myContext->myScratchBuffers[GFX_FSR_SCRATCH_TYPE_UPSCALING].CreateTempResources();

if (ffxFsr3ContextDispatchUpscale(&myContext->myFSRContext, &dispatchDesc) != FFX_OK)
{
    MC_ERROR("ffxFsr3ContextDispatchUpscale failed: 0x%08x", errorCode);
    return false;
}

myContext->myScratchBuffers[GFX_FSR_SCRATCH_TYPE_UPSCALING].FreeTempResources();
```

SHADERS

- Custom intermediate language for shader files.
- FSR ships with native HLSL source.

SHADERS

- Custom intermediate language for shader files.
- FSR ships with native HLSL source.
- Convert native HLSL into mshader.

```
SamplerState bilinear : register(s0);
Texture2D Input : register(t0);
float4 MainPS() : SV_Target0
{
    float2 uv = float2(0.5, 0.5);
    float4 color = Input.Sample(bilinear, Input, uv);
    return color;
}
```

SHADERS

- Custom intermediate language for shader files.
- FSR ships with native HLSL source.
- Convert native HLSL into mshader.

```
SamplerState bilinear : register(s0);
Texture2D Input : register(t0);
float4 MainPS() : SV_Target0
{
    float2 uv = float2(0.5, 0.5);
    float4 color = Input.Sample(bilinear, Input, uv);
    return color;
}
```

→

```
pixel
{
    MR_Sampler2D input : MR_Texture0;
    main(out float4 color : MR_Color)
    {
        float2 uv = float2(0.5, 0.5);
        color = MR_Sample(input, uv);
    }
}
```

SHADERS

- Custom intermediate language for shader files.
- FSR ships with native HLSL source.
- Convert native HLSL into mshader
- Add extras *(Bindless, custom load functions etc.)*
- Respect permutations

```
SamplerState bilinear : register(s0);
Texture2D Input : register(t0);
float4 MainPS() : SV_Target0
{
    float2 uv = float2(0.5, 0.5);
    float4 color = Input.Sample(bilinear, Input, uv);
    return color;
}
```

$\rightarrow$

```
pixel
{
    MR_Sampler2D input : MR_Texture0
    main(out float4 color : MR_Color)
    {
        float2 uv = float2(0.5, 0.5);
        color = MR_Sample(input, uv);
    }
}
```

SHADERS

- Converted to bindless resources.
  - Avoided issue with limited bindful registers.

```
FfxFloat32 LoadInputDepth(FfxUInt32x2 iPxPos)
{
#ifdef USE_MRENDER_BINDLESS
    uint descriptorIndex = Gfx_GetSRVDescriptorIndex(FSR3UPSCALER_BIND_SRV_INPUT_DEPTH);
    return MR_SampleTexelLod0Bindless(Gfx_Texture2Ds, descriptorIndex, iPxPos).x;
#else
    return r_input_depth[iPxPos];
#endif
}
```

SHADERS

- Converted to bindless resources.
  - Avoided issue with limited bindful registers.

```
FfxFloat32 LoadInputDepth(FfxUInt32x2 iPxPos)
{
#ifdef USE_MRENDER_BINDLESS
    uint descriptorIndex = Gfx_GetSRVDescriptorIndex(FSR3UPSCALER_BIND_SRV_INPUT_DEPTH);
    return MR_SampleTexelLod0Bindless(Gfx_Texture2Ds, descriptorIndex, iPxPos).x;
#else
    return r_input_depth[iPxPos];
#endif
}
```

In the case of ffx_fsr3upscaler_depth_clip_pass.hlsl

$\downarrow$

```
FfxFloat32 LoadInputDepth(FfxUInt32x2 iPxPos)
{
    uint descriptorIndex = Gfx_GetSRVDescriptorIndex(8);
    return MR_SampleTexelLod0Bindless(Gfx_Texture2Ds, descriptorIndex, iPxPos).x;
}
```

HLSL CONVERSION

- For each permutation:
  1. Gather resource usage from native shader file.
  2. Preprocess native shader file.
  3. Parse preprocessed file with internal parser.
  4. Add extras, strip unnecessary parts.
  5. Write .mshader file.
  6. (Optional) Load written .mshader for validation.

EXTERNAL RESOURCE REQUIREMENTS

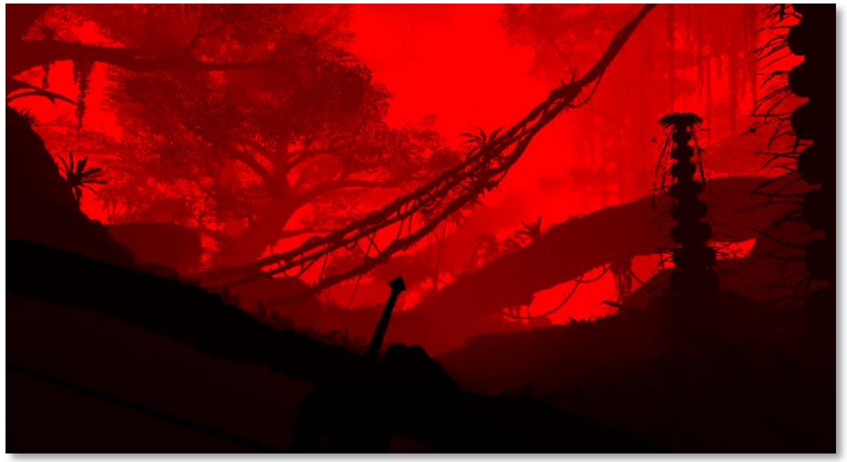
EXTERNAL RESOURCE REQUIREMENTS

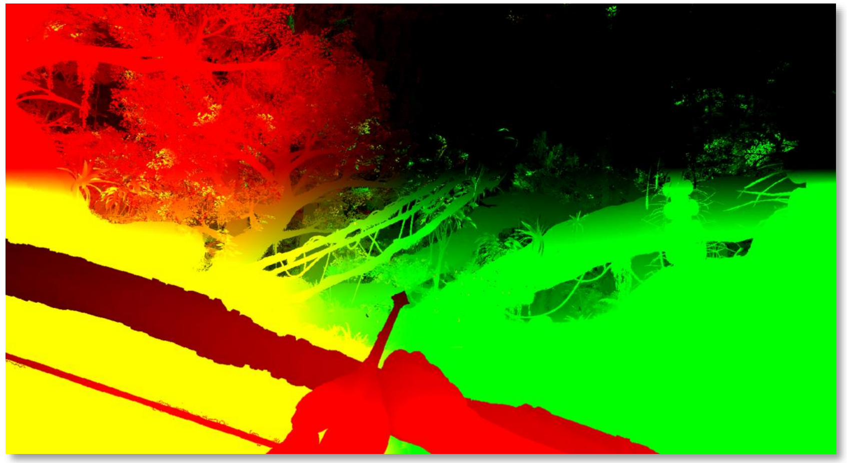
- ”External” as in not internal to FSR.
- Needs to be provided to FSR.

EXTERNAL RESOURCE REQUIREMENTS

Shared Context

- Depth
- Motion vectors





EXTERNAL RESOURCE REQUIREMENTS

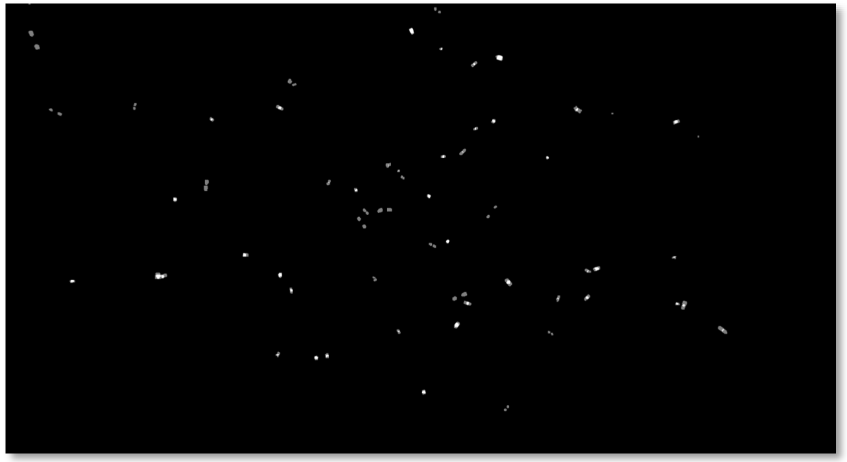
Upscaling Context

- Reactiveness (optional)
- Transparency and composition (optional)

EXTERNAL RESOURCE REQUIREMENTS

Upscaling Context

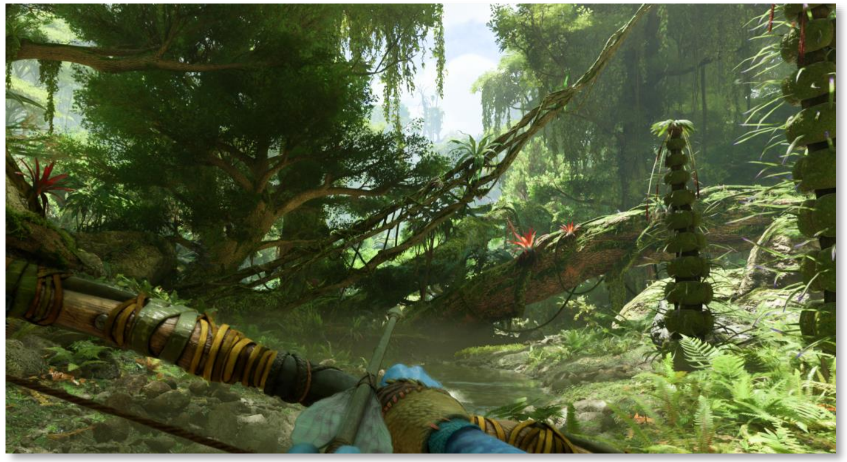
- Reactiveness (optional)
- ~~Transparency and composition (optional)~~

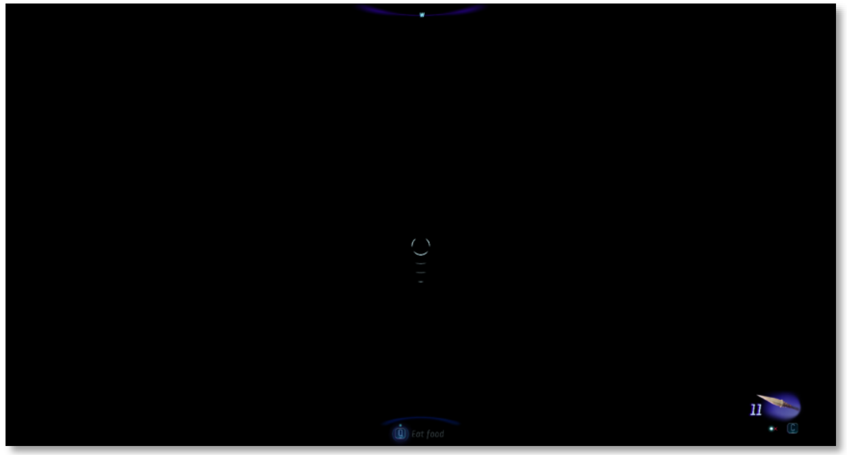


EXTERNAL RESOURCE REQUIREMENTS

Frame Generation Context

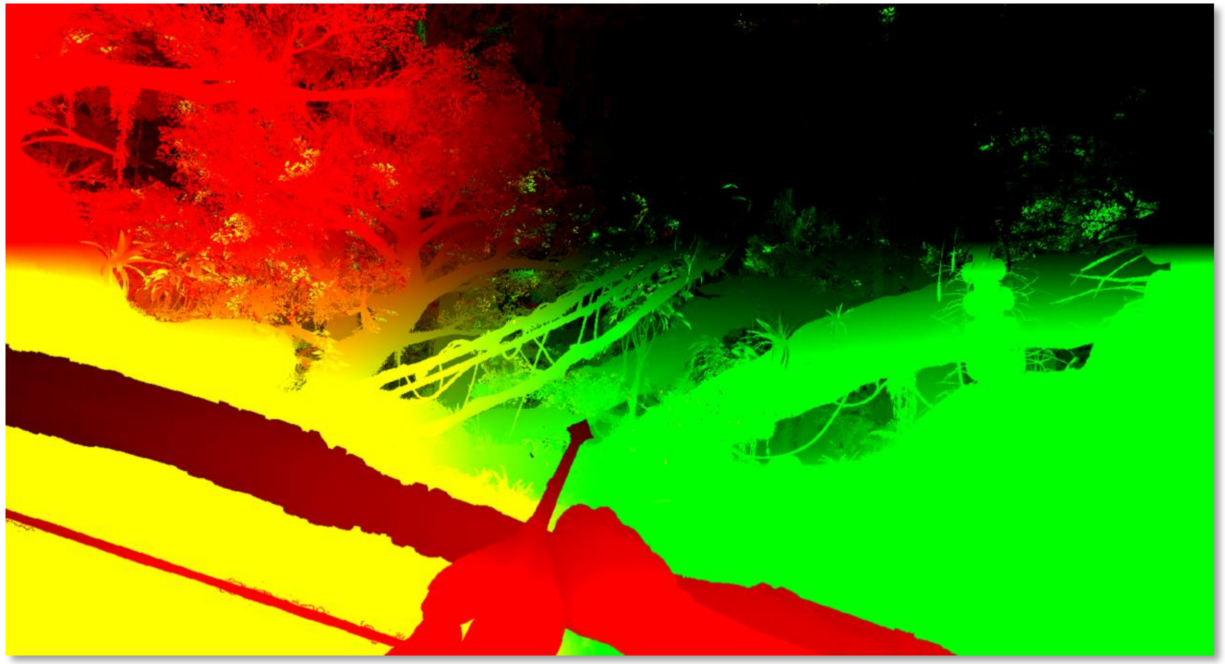
- HUD-less game scene
- Separated UI





MOTION VECTORS

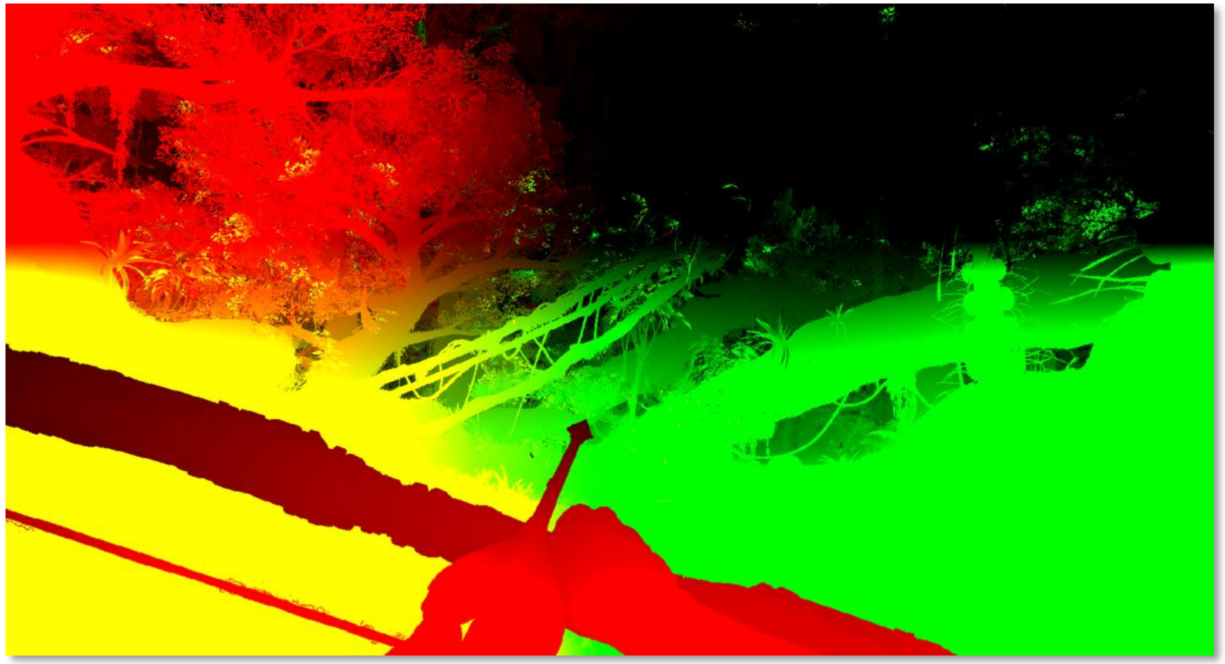
- Precision critical
- Upscaling algorithm is sensitive to errors.
- Frame interpolation makes it worse.



MOTION VECTORS

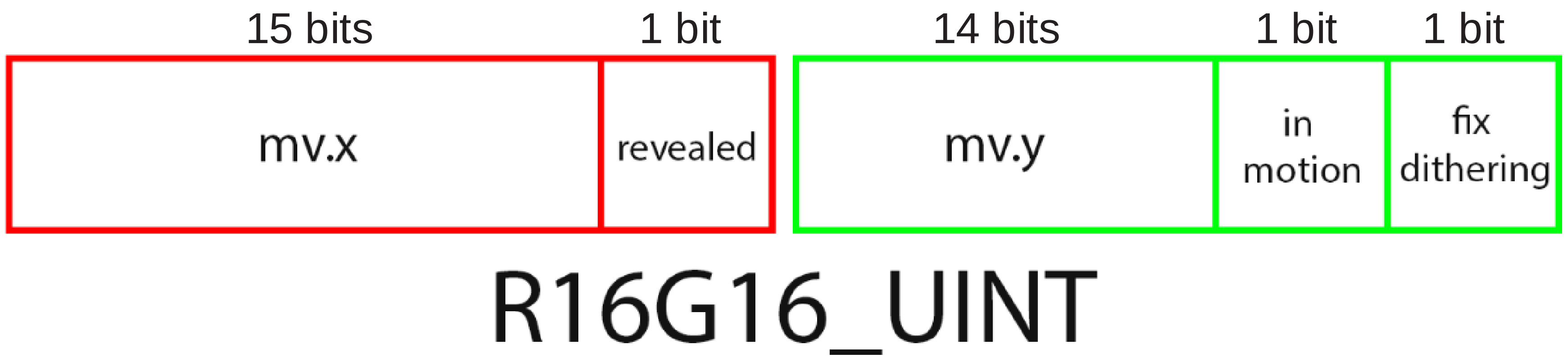
- Precision critical
- Upscaling algorithm is sensitive to errors.
- Frame interpolation makes it worse.

- Vertex modifiers are problematic.
- Mismatching last frame data.



MOTION VECTORS

- Snowdrop motion vectors encodes metadata.
- Would require decoding pass.
- Complex encoding, but for simplicity:



MOTION VECTORS

- Sample the global motion vector resource directly.
- Exclude the metadata.

```
MR_Texture2D<uint2> MotionVectors : MR_MotionVectors;
FfxFloat32x2 LoadInputMotionVector(FfxUInt32x2 iPxDilatedMotionVectorPos)
{
    uint2 encodedMV = MR_SampleTexelLod0(MotionVectors, iPxDilatedMotionVectorPos);

    float2 offset;
    bool fixDithering;
    bool revealed;
    bool inMotion;
    Gfx_DecodeMotionVector(encodedMV, offset, fixDithering, revealed, inMotion);

    float2 fSrcMotionVector = offset;
    FfxFloat32x2 fUvMotionVector = fSrcMotionVector * MotionVectorScale();

#if FFX_FSR3UPSCALER_OPTION_JITTERED_MOTION_VECTORS
    fUvMotionVector -= MotionVectorJitterCancellation();
#endif

    return fUvMotionVector;
}
```

MOTION VECTORS

- Sample the global motion vector resource directly.
- Exclude the metadata.

```
MR_Texture2D<uint2> MotionVectors : MR_MotionVectors;
FfxFloat32x2 LoadInputMotionVector(FfxUInt32x2 iPxDilatedMotionVectorPos)
{
    uint2 encodedMV = MR_SampleTexelLod0(MotionVectors, iPxDilatedMotionVectorPos);

    float2 offset;
    bool fixDithering;
    bool revealed;
    bool inMotion;
    Gfx_DecodeMotionVector(encodedMV, offset, fixDithering, revealed, inMotion);

    float2 fSrcMotionVector = offset;
    FfxFloat32x2 fUvMotionVector = fSrcMotionVector * MotionVectorScale();

#if FFX_FSR3UPSCALER_OPTION_JITTERED_MOTION_VECTORS
    fUvMotionVector -= MotionVectorJitterCancellation();
#endif

    return fUvMotionVector;
}
```

MOTION VECTORS

- Sample the global motion vector resource directly.
- Exclude the metadata.

```
MR_Texture2D<uint2> MotionVectors : MR_MotionVectors;
FfxFloat32x2 LoadInputMotionVector(FfxUInt32x2 iPxDilatedMotionVectorPos)
{
    uint2 encodedMV = MR_SampleTexelLod0(MotionVectors, iPxDilatedMotionVectorPos);

    float2 offset;
    bool fixDithering;
    bool revealed;
    bool inMotion;
    Gfx_DecodeMotionVector(encodedMV, offset, fixDithering, revealed, inMotion);
    float2 fSrcMotionVector = offset;
    FfxFloat32x2 fUvMotionVector = fSrcMotionVector * MotionVectorScale();

#if FFX_FSR3UPSCALER_OPTION_JITTERED_MOTION_VECTORS
    fUvMotionVector -= MotionVectorJitterCancellation();
#endif

    return fUvMotionVector;
}
```

MOTION VECTORS

- Sample the global motion vector resource directly.
- Exclude the metadata.

```
MR_Texture2D<uint2> MotionVectors : MR_MotionVectors;
FfxFloat32x2 LoadInputMotionVector(FfxUInt32x2 iPxDilatedMotionVectorPos)
{
    uint2 encodedMV = MR_SampleTexelLod0(MotionVectors, iPxDilatedMotionVectorPos);

    float2 offset;
    bool fixDithering;
    bool revealed;
    bool inMotion;
    Gfx_DecodeMotionVector(encodedMV, offset, fixDithering, revealed, inMotion);

    float2 fSrcMotionVector = offset;
    FfxFloat32x2 fUvMotionVector = fSrcMotionVector * MotionVectorScale();

#if FFX_FSR3UPSCALER_OPTION_JITTERED_MOTION_VECTORS
    fUvMotionVector -= MotionVectorJitterCancellation();
#endif

    return fUvMotionVector;
}
```

MOTION VECTORS

- Sample the global motion vector resource directly.
- Exclude the metadata.

```
MR_Texture2D<uint2> MotionVectors : MR_MotionVectors;
FfxFloat32x2 LoadInputMotionVector(FfxUInt32x2 iPxDilatedMotionVectorPos)
{
    uint2 encodedMV = MR_SampleTexelLod0(MotionVectors, iPxDilatedMotionVectorPos);

    float2 offset;
    bool fixDithering;
    bool revealed;
    bool inMotion;
    Gfx_DecodeMotionVector(encodedMV, offset, fixDithering, revealed, inMotion);

    float2 fSrcMotionVector = offset;
    FfxFloat32x2 fUvMotionVector = fSrcMotionVector * MotionVectorScale();

#if FFX_FSR3UPSCALER_OPTION_JITTERED_MOTION_VECTORS
    fUvMotionVector -= MotionVectorJitterCancellation();
#endif

    return fUvMotionVector;
}
```

MOTION VECTORS

- Sample the global motion vector resource directly.
- Exclude the metadata.

```
MR_Texture2D<uint2> MotionVectors : MR_MotionVectors;
FfxFloat32x2 LoadInputMotionVector(FfxUInt32x2 iPxDilatedMotionVectorPos)
{
    uint2 encodedMV = MR_SampleTexelLod0(MotionVectors, iPxDilatedMotionVectorPos);

    float2 offset;
    bool fixDithering;
    bool revealed;
    bool inMotion;
    Gfx_DecodeMotionVector(encodedMV, offset, fixDithering, revealed, inMotion);

    float2 fSrcMotionVector = offset;
    FfxFloat32x2 fUvMotionVector = fSrcMotionVector * MotionVectorScale();

#if FFX_FSR3UPSCALER_OPTION_JITTERED_MOTION_VECTORS
    fUvMotionVector -= MotionVectorJitterCancellation();
#endif

    return fUvMotionVector;
}
```

REACTIVENESS

- Controls the history lerp
  - 1.0 means no history.
  - 0.0 means full lerp.

REACTIVENESS

- Controls the history lerp
  - 1.0 means no history.
  - 0.0 means full lerp.

- In Snowdrop, reactiveness is binary
  - It’s legacy and problematic.
  - Global reactiveness value.
  - Mostly used for particles.

REACTIVENESS

- Controls the history lerp
  - 1.0 means no history.
  - 0.0 means full lerp.

- In Snowdrop, reactiveness is binary
  - It’s legacy and problematic.
  - Global reactiveness value.
  - Mostly used for particles.

- For the future
  - Move to UNORM texture for reactiveness.
  - Allows fine-grained control of the history.

REACTIVENESS

- Similar to motion vectors, we use custom sampling.
  - Read global stencil texture.
  - Return global reactiveness value if match.

```
FfxFloat32 LoadReactiveMask(FfxUInt32x2 iPxPos)
{
    FfxUInt32 stencil = MR_SampleStencilTexelLod0(StencilTexture, iPxPos);

    FfxFloat32 reactive = 0;
    if (stencil & Gfx_Stencil_QuickAA)
        reactive = Gfx_FSR3_Bindings.reactivenessValue;

    return reactive;
}
```

REACTIVENESS

- Similar to motion vectors, we use custom sampling
  - Read global stencil texture.
  - Return global reactiveness value if match.

```
FfxFloat32 LoadReactiveMask(FfxUInt32x2 iPxPos)
{
    FfxUInt32 stencil = MR_SampleStencilTexelLod0(StencilTexture, iPxPos);

    FfxFloat32 reactive = 0;
    if (stencil & Gfx_Stencil_QuickAA)
        reactive = Gfx_FSR3_Bindings.reactivenessValue;

    return reactive;
}
```

REACTIVENESS

- Similar to motion vectors, we use custom sampling
  - Read global stencil texture.
  - Return global reactiveness value if match.

```
FfxFloat32 LoadReactiveMask(FfxUInt32x2 iPxPos)
{
    FfxUInt32 stencil = MR_SampleStencilTexelLod0(StencilTexture, iPxPos);

    FfxFloat32 reactive = 0;
    if (stencil & Gfx_Stencil_QuickAA)
        reactive = Gfx_FSR3_Bindings.reactivenessValue;

    return reactive;
}
```

REACTIVENESS

- Similar to motion vectors, we use custom sampling
  - Read global stencil texture.
  - Return global reactiveness value if match.

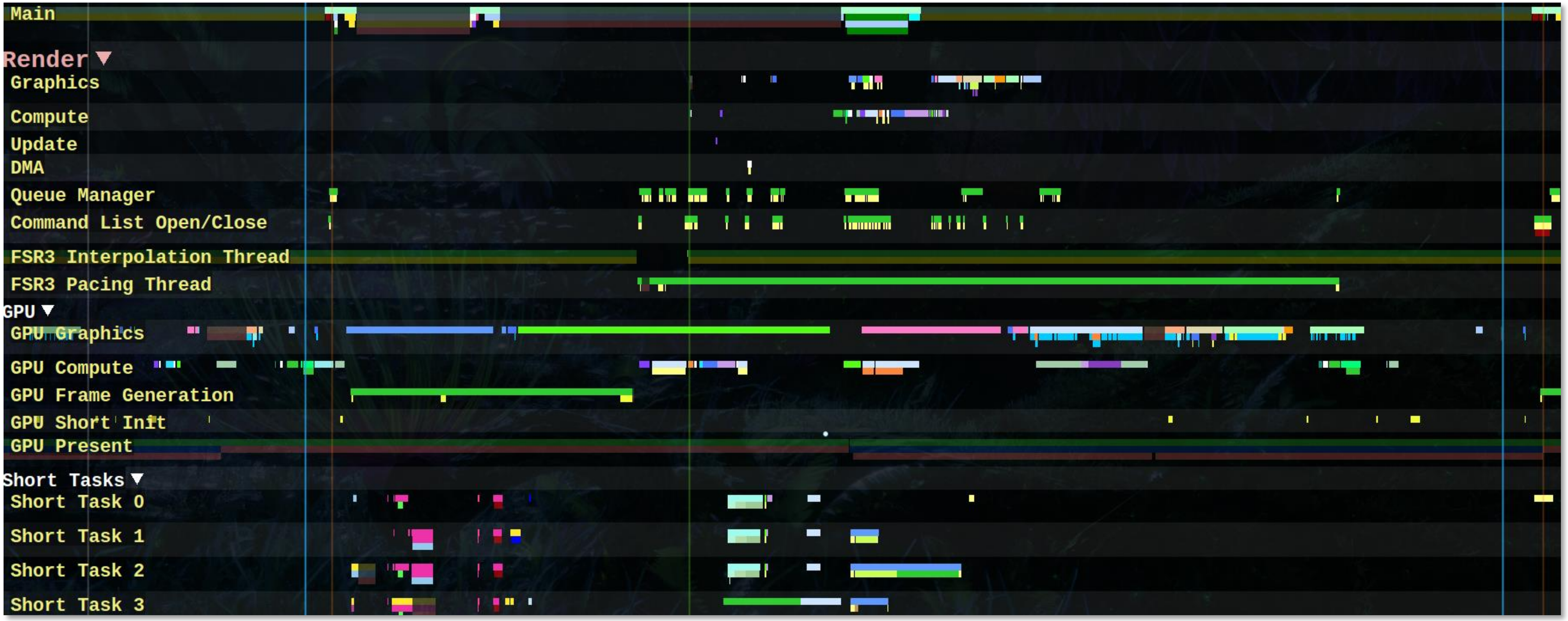
```
FfxFloat32 LoadReactiveMask(FfxUInt32x2 iPxPos)
{
    FfxUInt32 stencil = MR_SampleStencilTexelLod0(StencilTexture, iPxPos);

    FfxFloat32 reactive = 0;
    if (stencil & Gfx_Stencil_QuickAA)
        reactive = Gfx_FSR3_Bindings.reactivenessValue;

    return reactive;
}
```

SNOWDROP RENDER PIPELINE

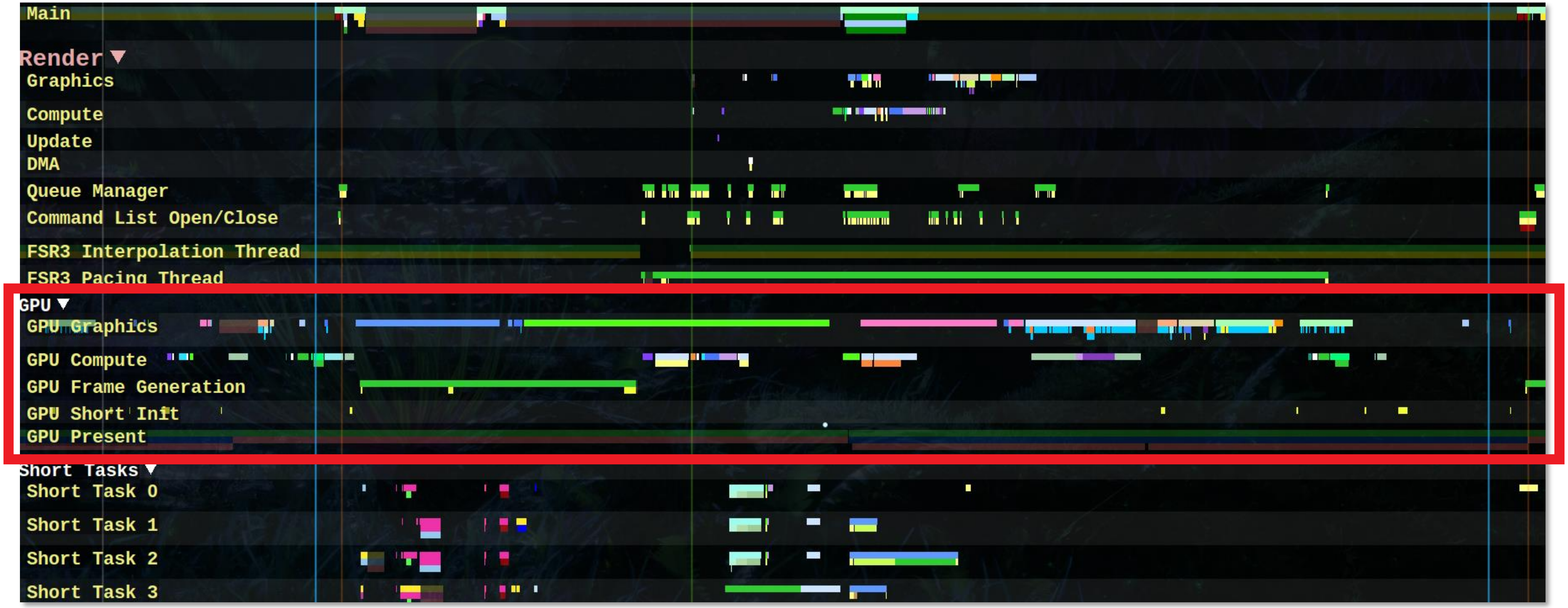
SNOWDROP RENDER PIPELINE

- Where does it all fit into our render pipeline?
  - Pre-post effects
  - Present

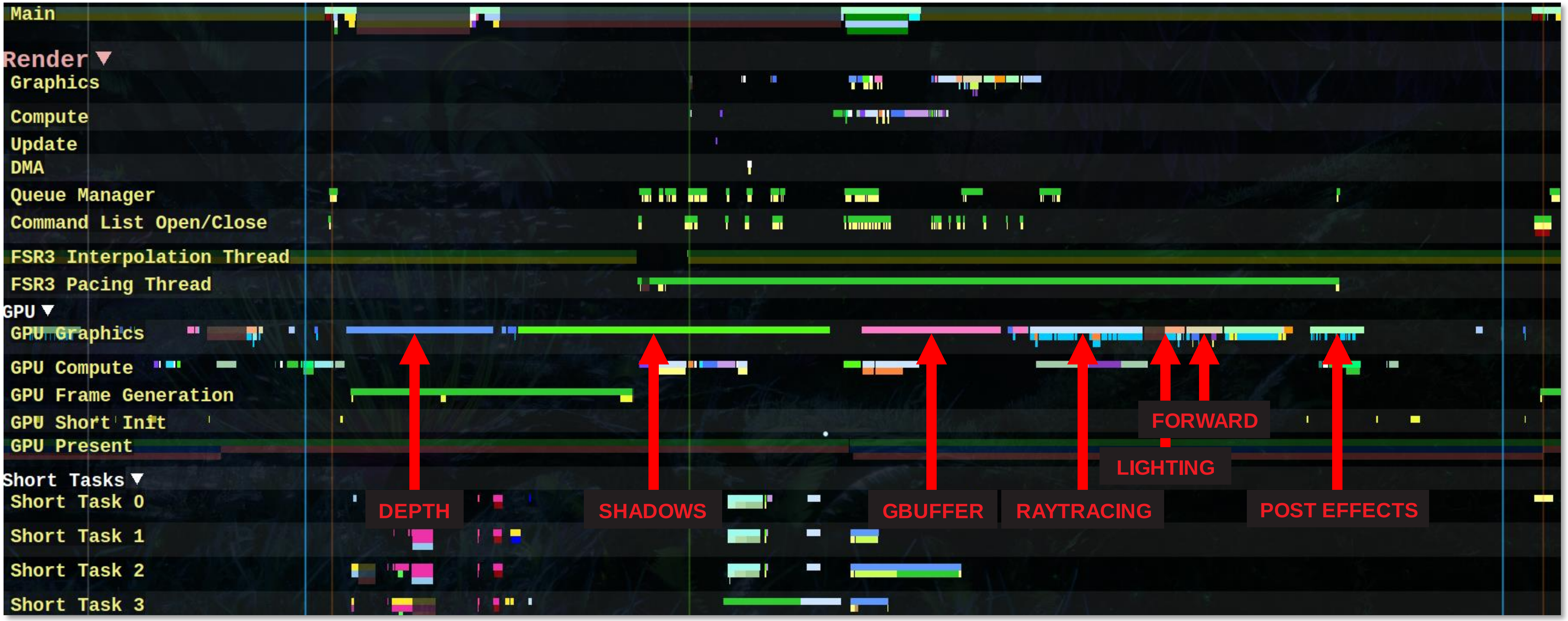
SNOWDROP RENDER PIPELINE



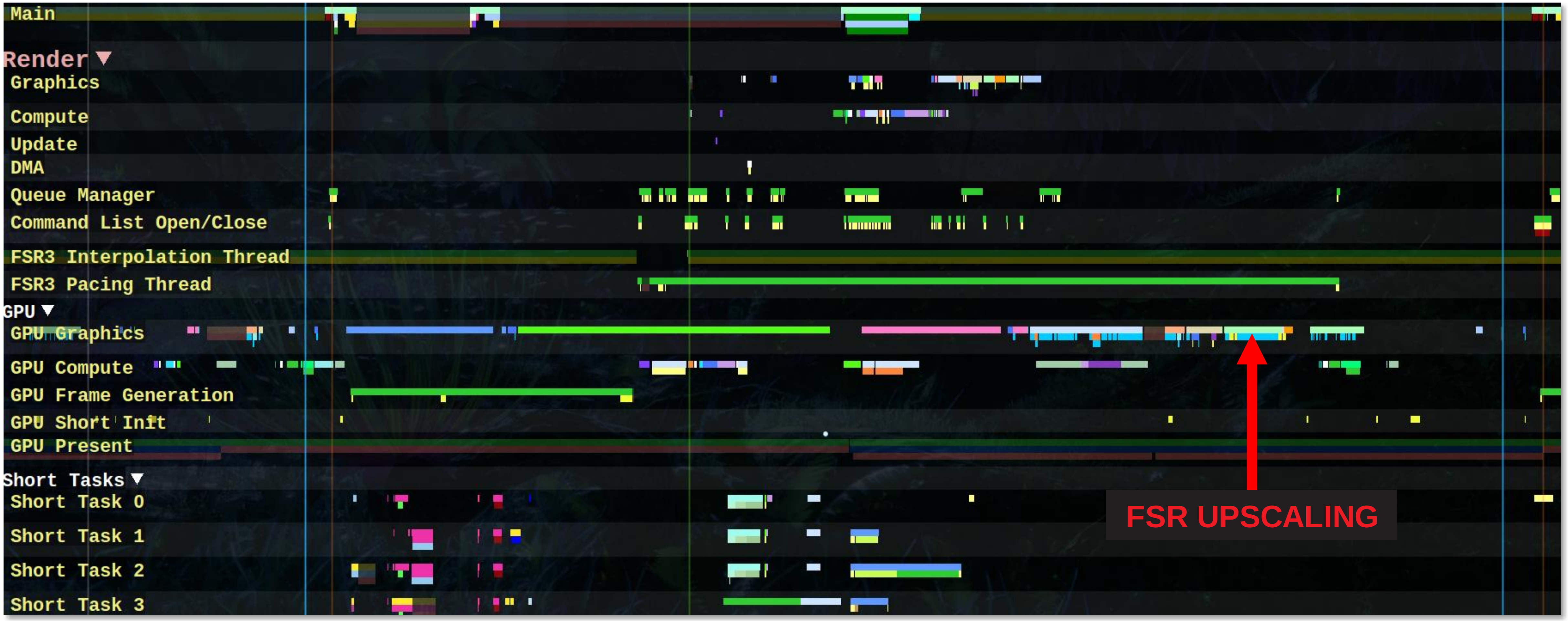
SNOWDROP RENDER PIPELINE



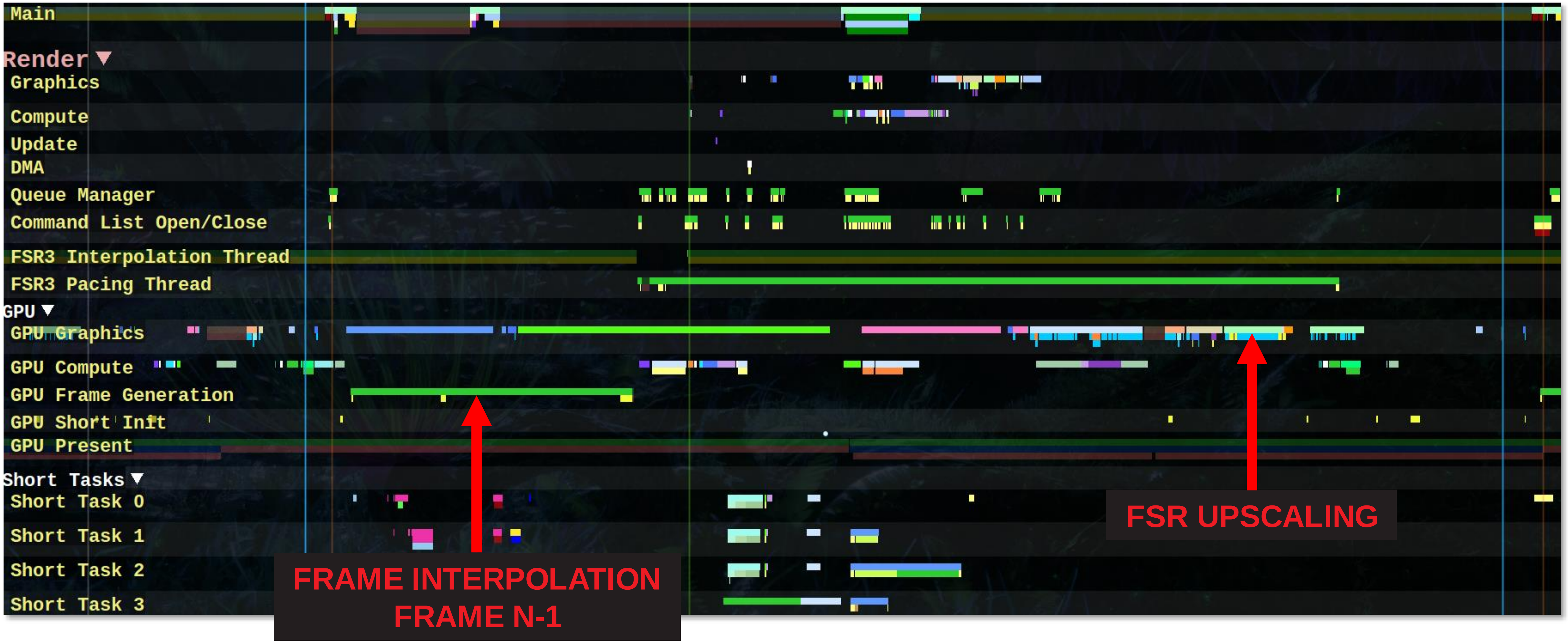
SNOWDROP RENDER PIPELINE



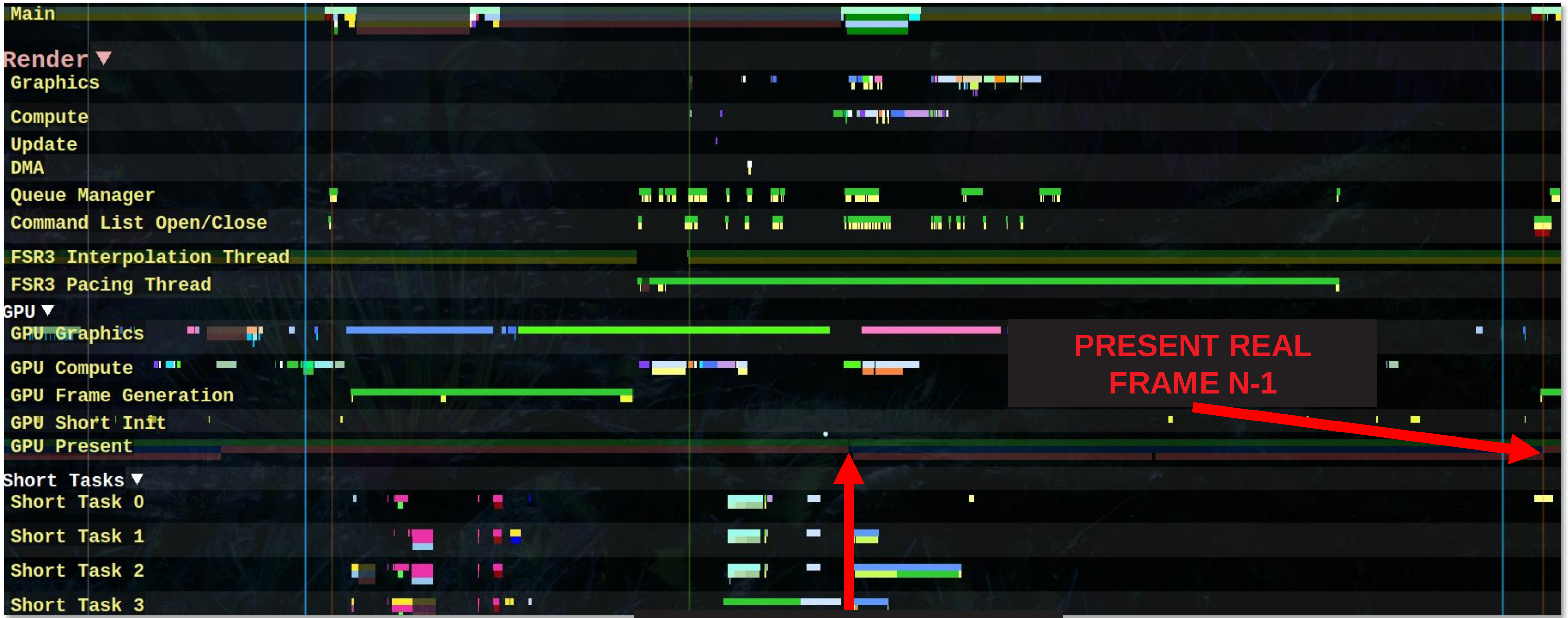
SNOWDROP RENDER PIPELINE



SNOWDROP RENDER PIPELINE

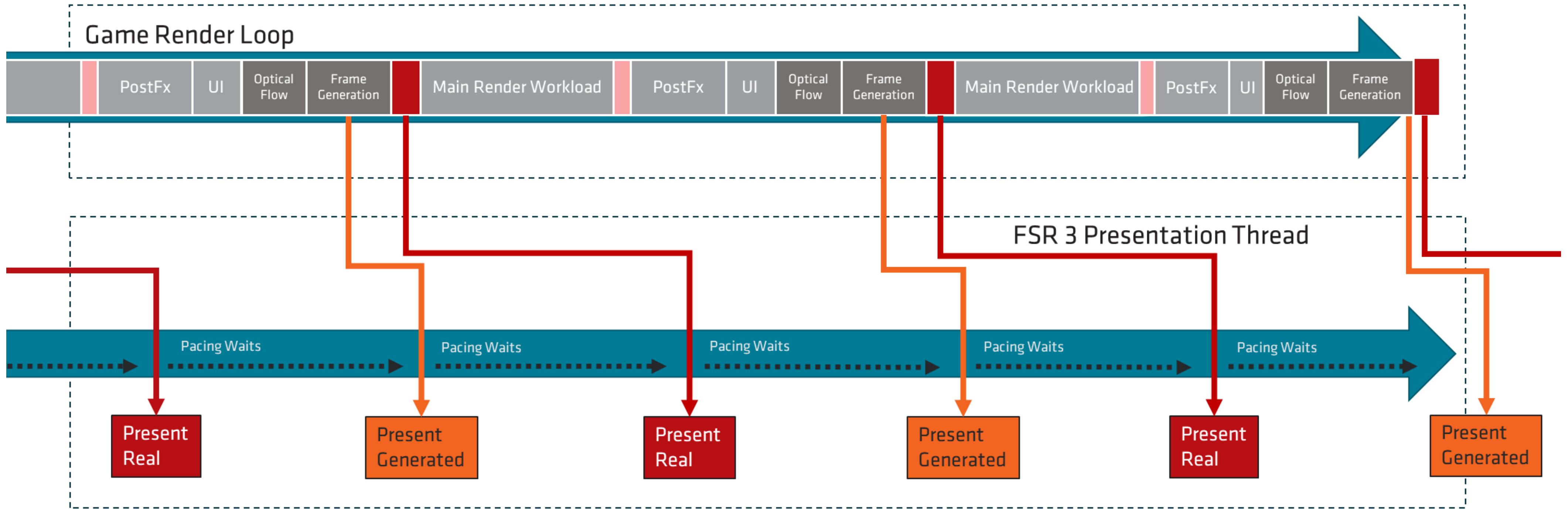


SNOWDROP RENDER PIPELINE



SNOWDROP RENDER PIPELINE

FSR 3 – PRESENT QUEUE UPSCALING AND FRAME GENERATION PIPELINE



🟧 Generated Frame Datapath/Presents

🟥 Real Frame Datapath/Presents

▨ Upscale

FRAME GENERATION

FRAME GENERATION

- Analytically generates extra frames.
- Improves display rate.
- Makes the game feel more fluid.

FRAME GENERATION

- Custom swapchain implementation.
- Frame pacing is tricky.

FRAME GENERATION

- Custom swapchain implementation.
- Frame pacing is tricky.

- Opens up for platform agnostic frame gen. ☺

CUSTOM SWAPCHAIN

- Inspired by AMDs swapchain implementation.
- Few tweaks to make it fit.

CUSTOM SWAPCHAIN

- Inspired by AMDs swapchain implementation.
- Few tweaks to make it fit.
    - Fences!

CUSTOM SWAPCHAIN

- Inspired by AMDs swapchain implementation.
- Few tweaks to make it fit.
    - Fences!

- Tested various sleeps for pacing.

CUSTOM SWAPCHAIN

- Inspired by AMDs swapchain implementation.
- Few tweaks to make it fit.
    - Fences!

- Tested various sleeps for pacing.

- V-sync makes it even harder.
    - Cannot miss vertical blank.
    - Separate cpu thread to handle timing calculations.

CUSTOM SWAPCHAIN

- Allow passthrough mode.
  - Avoids cost of recreating swapchain.

CUSTOM SWAPCHAIN

- Allow passthrough mode.
  - Avoids cost of recreating swapchain.

- Mode that does ”regular” present.
- No interpolation scheduled.

RECREATING SWAPCHAIN

- Minimal overhead
- Requires CPU-GPU sync.
- Requires full reconstruction to switch command queue.

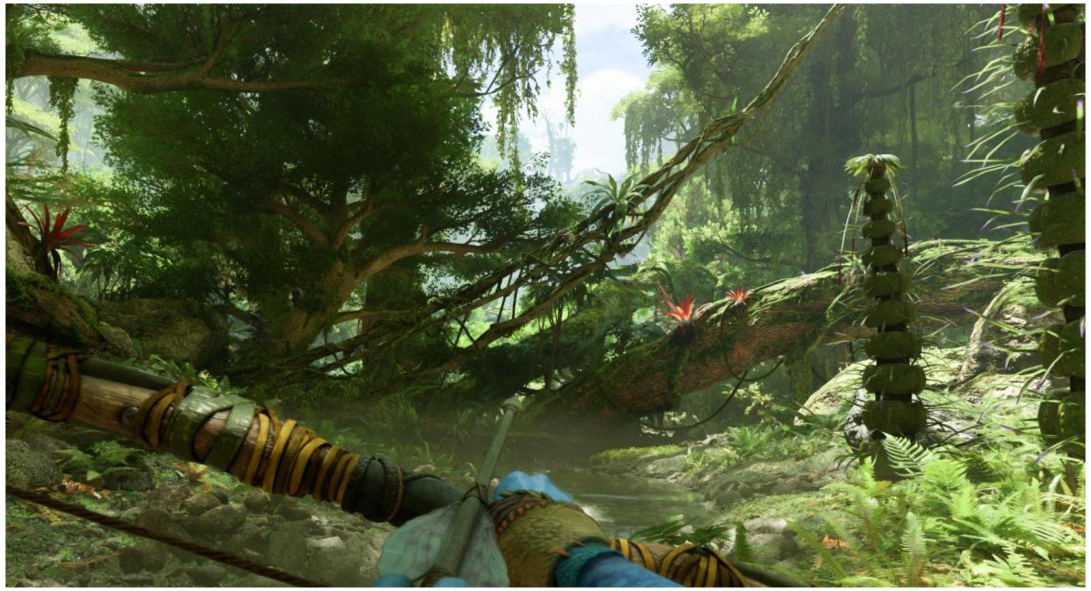
RECREATING SWAPCHAIN

- Minimal overhead
- Requires CPU-GPU sync.
- Requires full reconstruction to switch command queue.

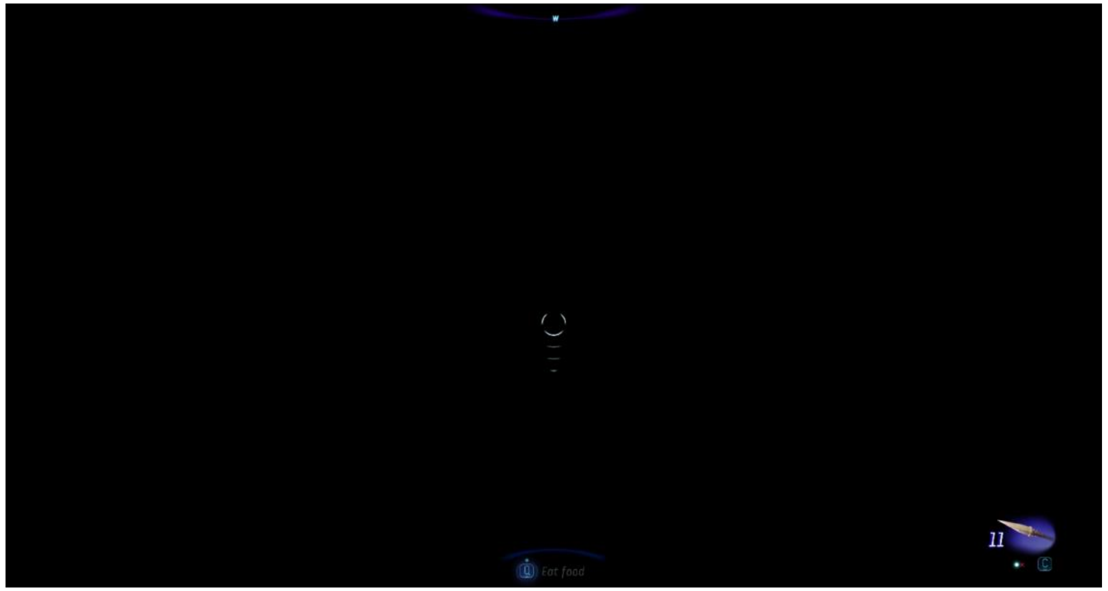
- Can be problematic for third party software.
  - Abort recreation and revert.

```
DXGI ERROR: IDXGIFactory::CreateSwapChain: Only one flip model swap chain can be associate with an HWND, IWindow, or composition
surface at a time. ClearState() and Flush() may need to be called on the D3D11 device context to trigger deferred destruction of old
swapchains. [ MISCELLANEOUS ERROR #297: ]
```

COMPOSITING UI



+

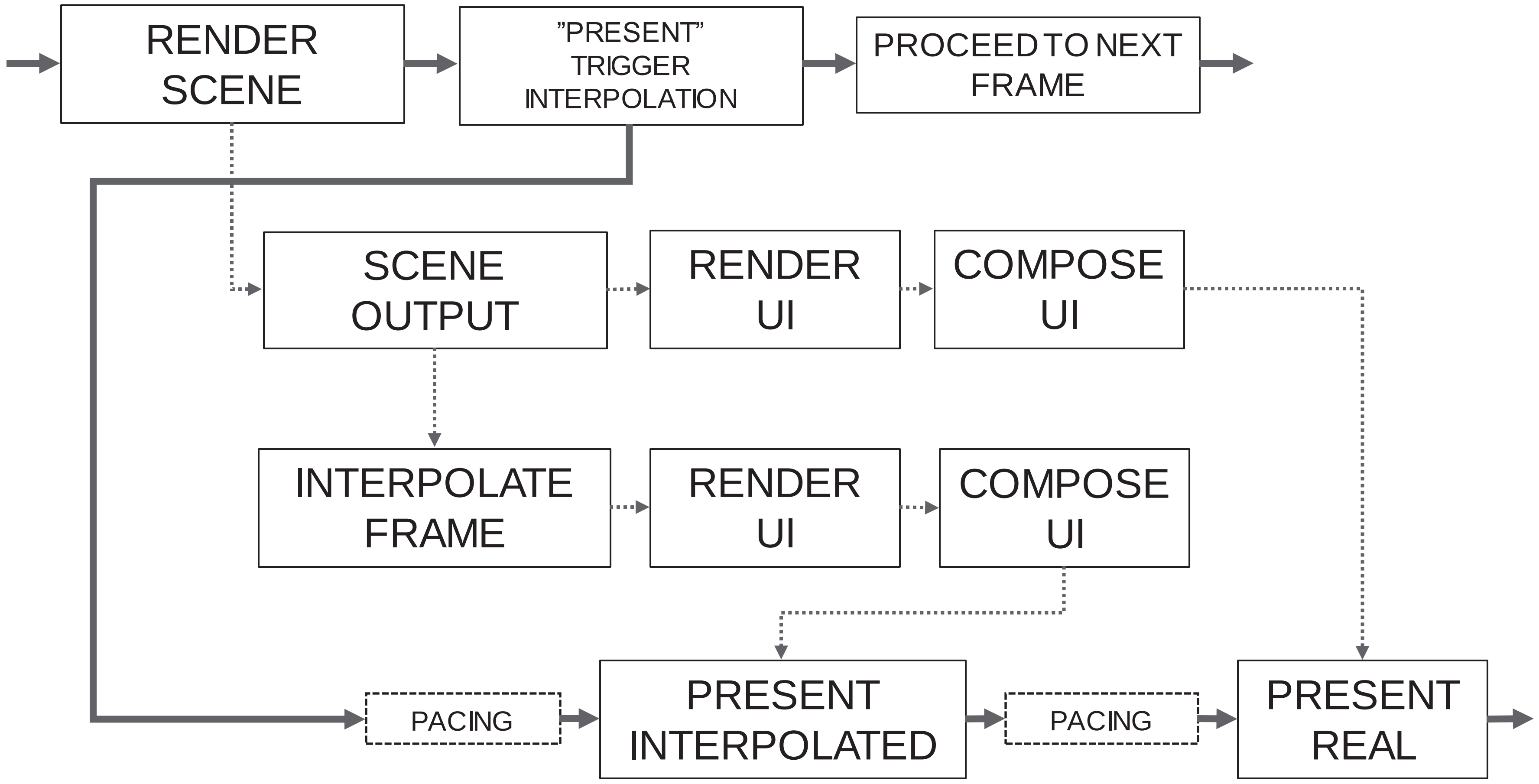


COMPOSITING UI

- Render at full-rate.
- Render at half-rate.

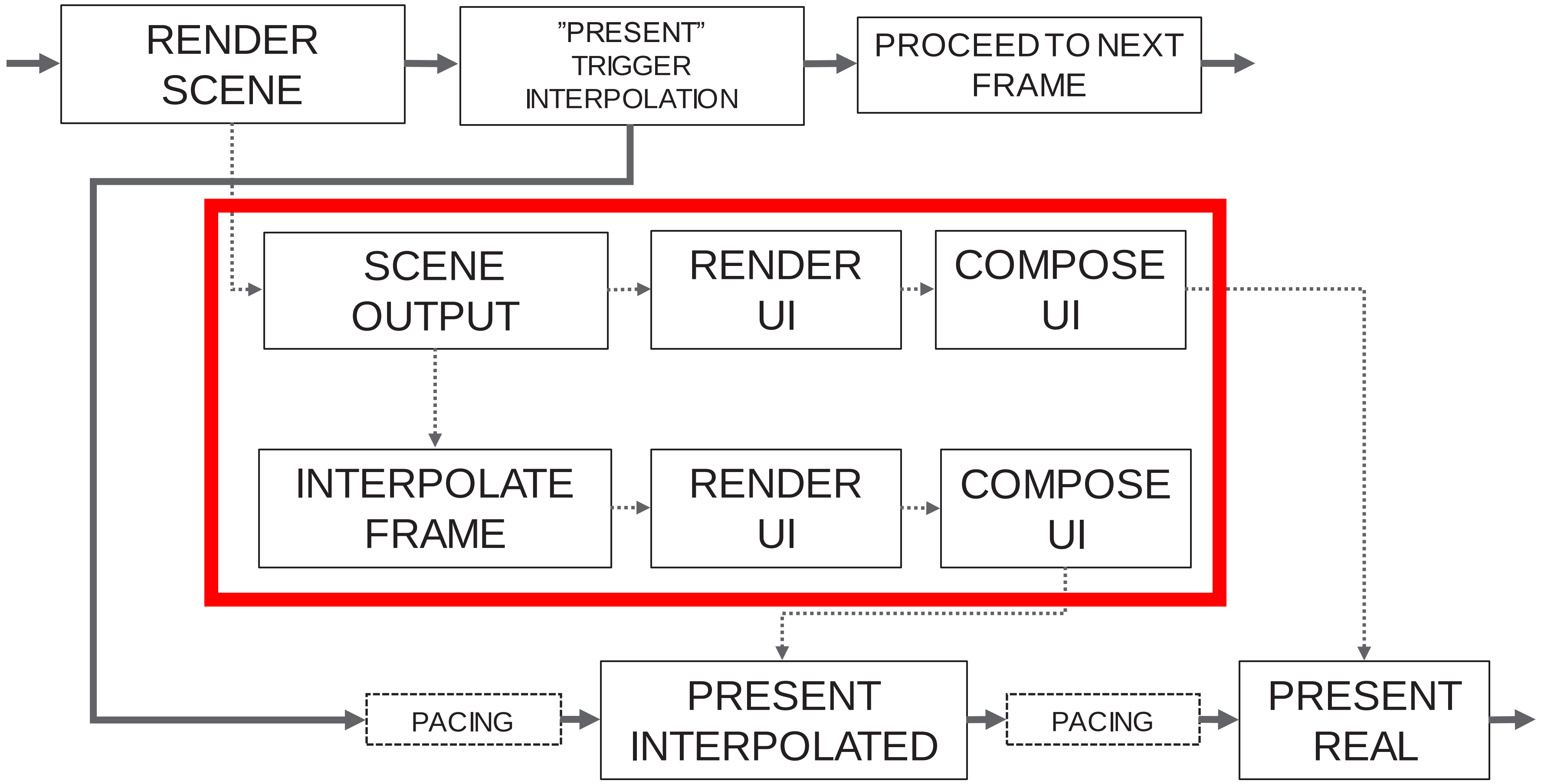
COMPOSITING UI

- Render at full-rate.



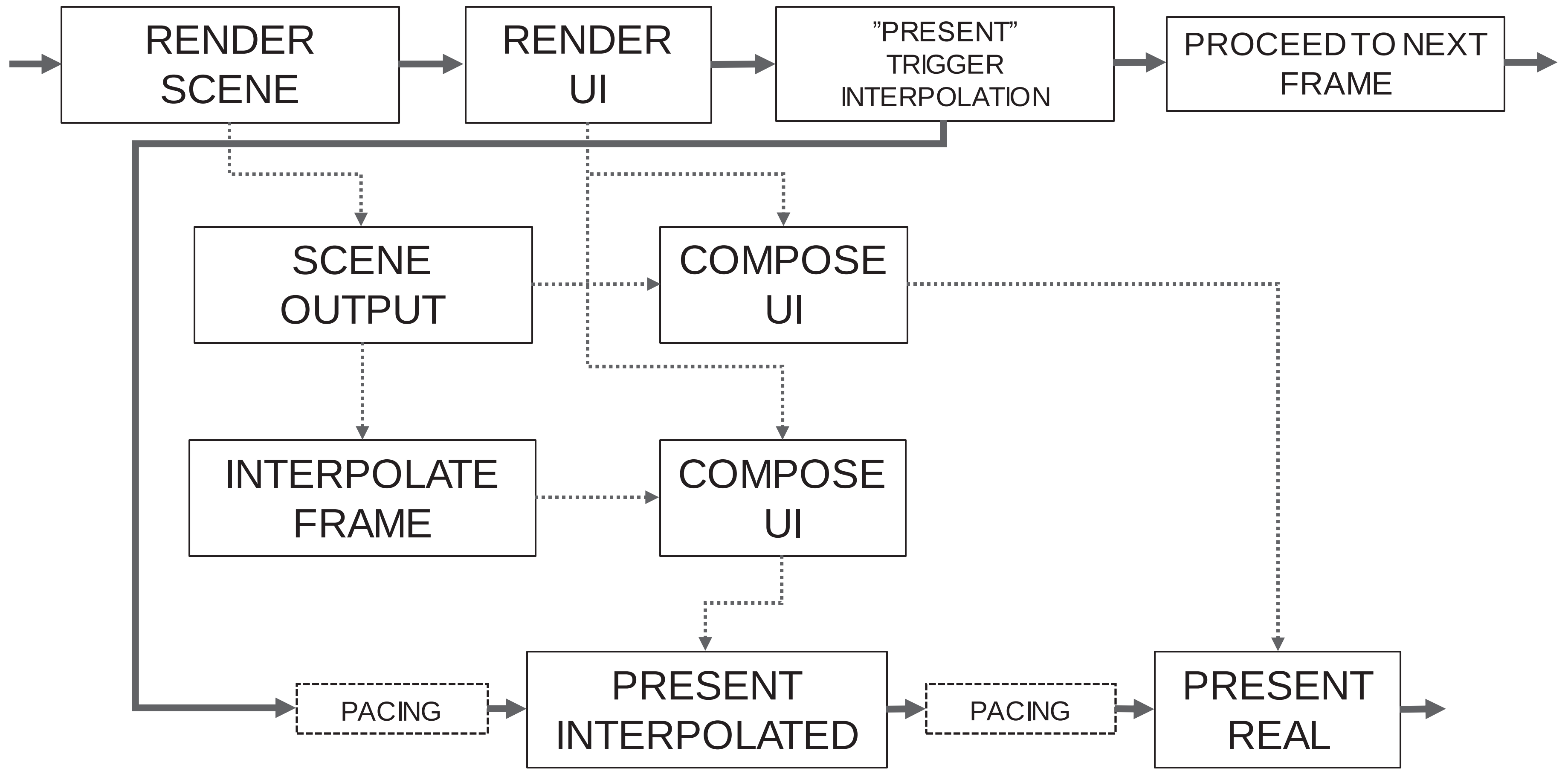
COMPOSITING UI

- Render at full-rate.



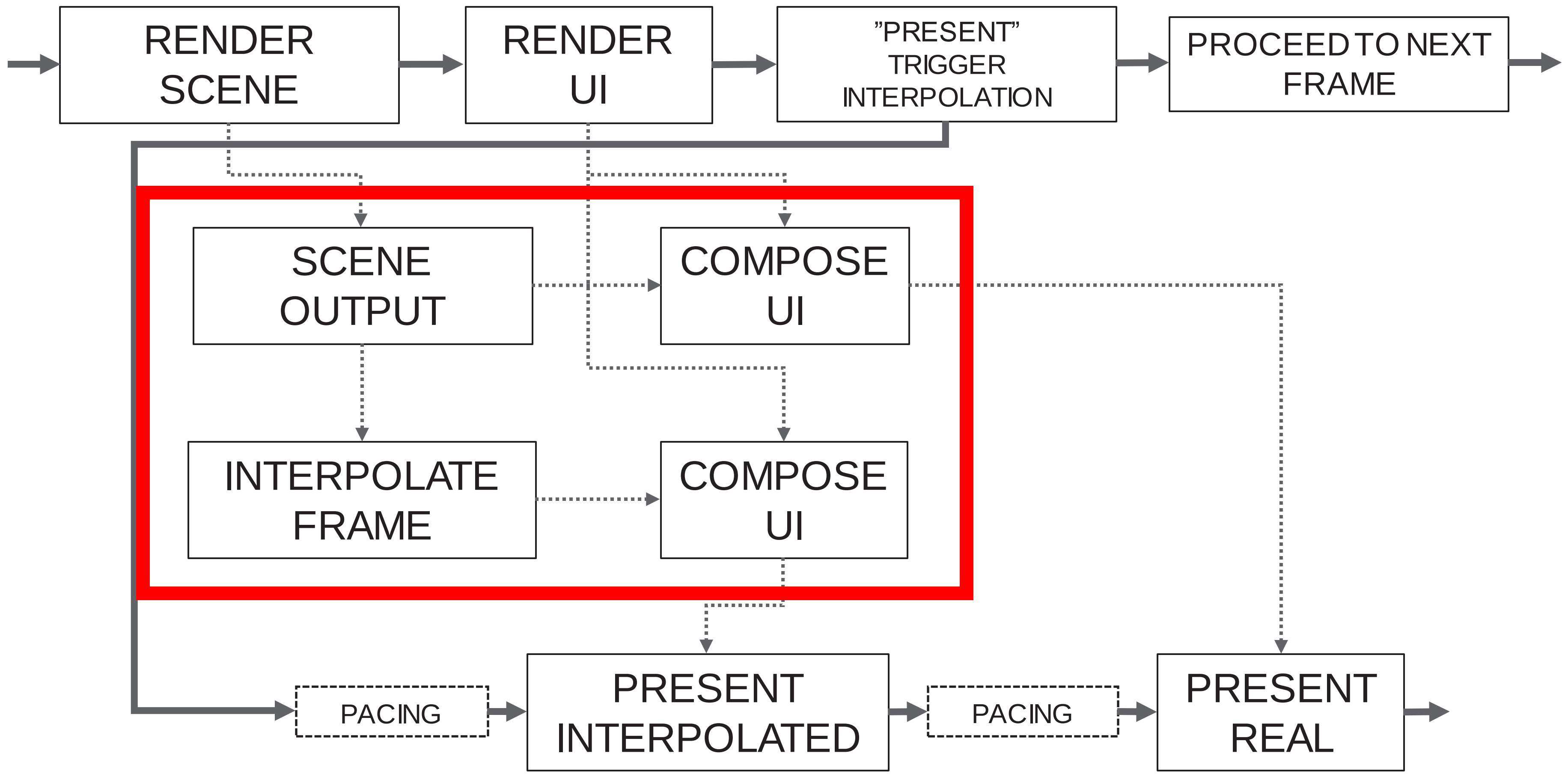
COMPOSITING UI

- Render at half-rate



COMPOSITING UI

- Render at half-rate



COMPOSITING UI

- Full-rate required change to pipeline.
- We chose half-rate compositing.

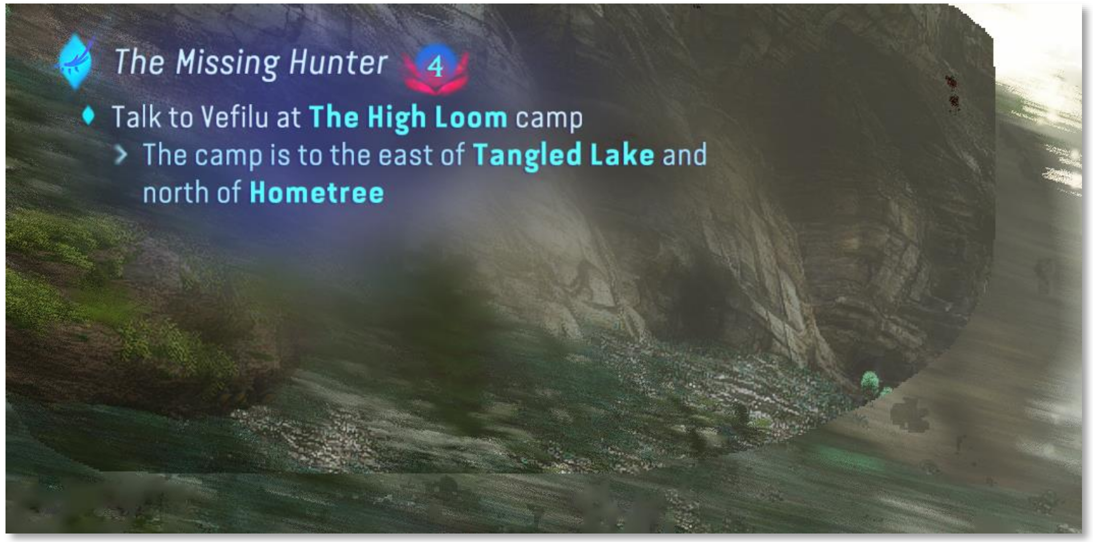
COMPOSITING UI

- Full-rate required change to pipeline.
- We chose half-rate compositing.
    - Requires extra copy of UI texture.

COMPOSITING UI

- Full-rate required change to pipeline.
- Render at half-rate.
  - Requires extra copy of UI texture.

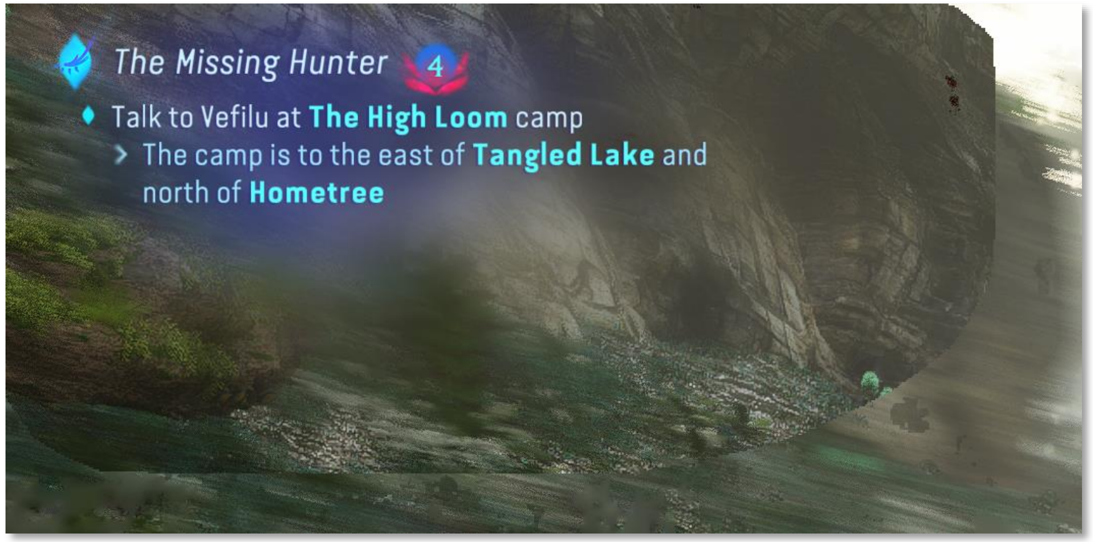
- Blurring the background is very problematic.


The Missing Hunter 4
Talk to Vefilu at The High Loom camp
The camp is to the east of Tangled Lake and north of Hometree

COMPOSITING UI

- Full-rate required change to pipeline.
- Render at half-rate.
  - Requires extra copy of UI texture.

- Blurring the background is very problematic.
  - Disable it. ☺


The Missing Hunter 4
Talk to Vefilu at The High Loom camp
The camp is to the east of Tangled Lake and north of Hometree

CONCLUSION

- Superior quality compared to our own TAA.
- High performance
- On all our major platforms.
  - *Frame generation is PC-only, at the moment...*

SPECIAL THANKS

UBISOFT

- Gauthier Viau
- Oleksandr Koshlo
- Daniel Wesslén
- Dennis Persson
- Gregor Ehrenstein
- Aaron Koppensteiner
- Charlie Mrad

AMD

- Tobias Fast
- Lou Kramer
- Gareth Thomas
- Jonas Gustavsson
- Luke Valentine
- FSR Development Team

And of course, to everyone else I've bothered during the development,
THANK YOU!

AMD FIDELITYFX™ SUPER RESOLUTION 3.1

COLIN RILEY - AMD

together we advance_

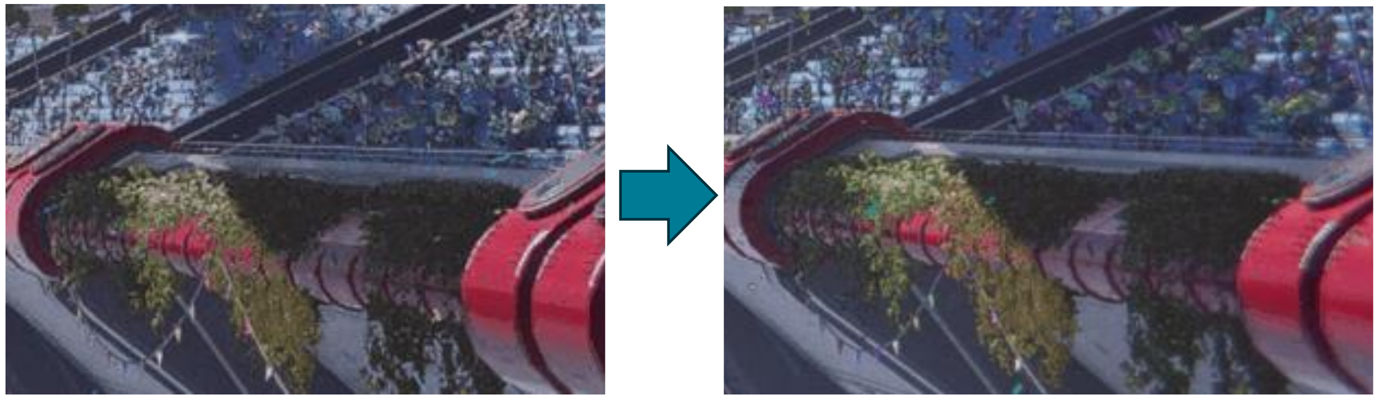
INTRODUCING AMD FSR 3.1

- Upscaling improvements.
- Various fixes.
- Vulkan® and Xbox® Game Development Kit support.
- Upscale can now be separated from Frame Generation.
- AMD FidelityFX™ API.
  - Locking down ABI.
  - Can unlock “upgradability” of FSR DLLs.

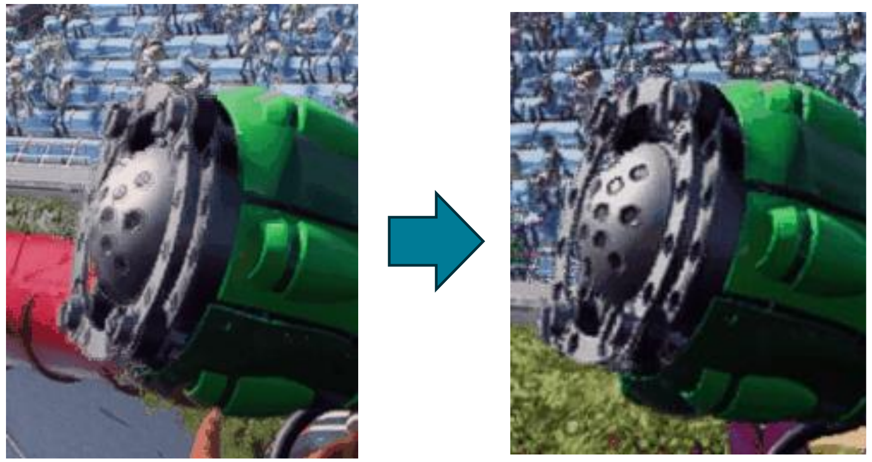
AMD FidelityFX Super Resolution 3

UPSCALING IMPROVEMENTS

- Internal changes to high/low frequency signals.
- Better preservation of detail.
- Reduction of temporal instability.


FSR 2.2 FSR 3.1

Temporal Stability (animated)


FSR 2.2 FSR 3.1

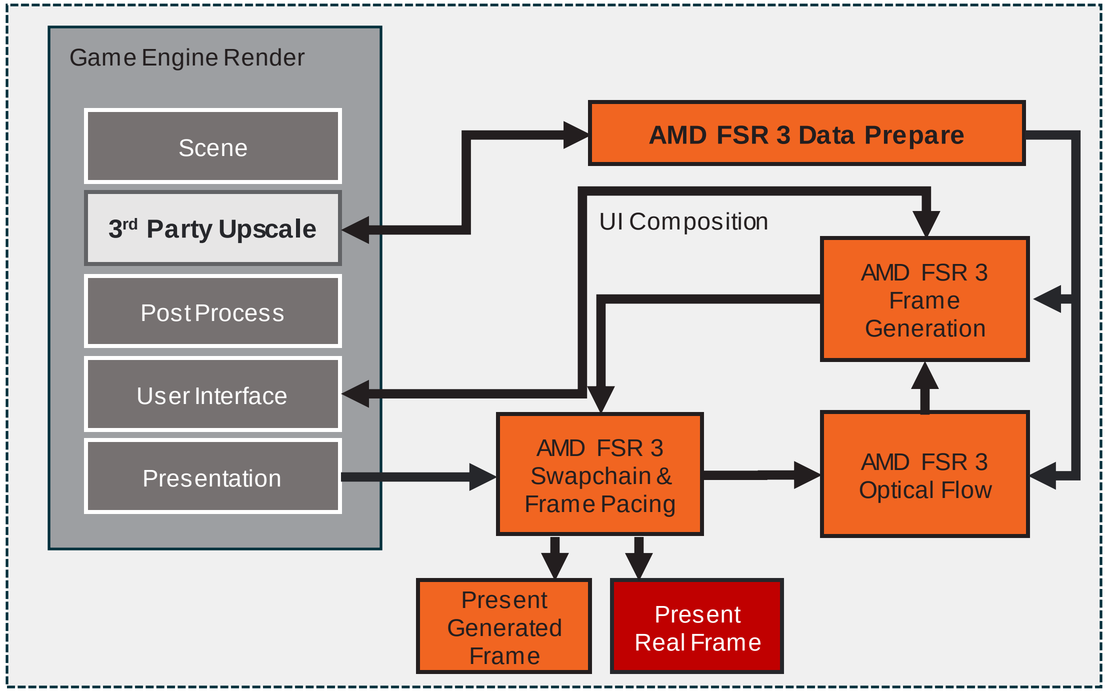
Ghosting Reduction (animated)

1080p Output Resolution,
FSR Performance mode.

SEPARATING UPSCALING FROM FRAME GENERATION

- AMD FSR 3.0 required FSR 3 upscaling to execute and generate information prior to FSR 3 Frame Generation.
  - This saved GPU resources, to allow for more FPS.
  - It did restrict upscale choices, however.

- Now FSR can run in a mode where Frame Generation runs with any 3rd party upscale component!
  - An additional step is required to precompute some data from the upscaler inputs.



New FSR Dataflow Option

Visit our website
https://gpuopen.com

AMD FSR 3.1 – AVAILABLE SOON!

- Follow GPUOpen to be notified when the new version is available for download!

Follow us on X
https://twitter.com/GPUOpen

Follow us on Mastodon
https://mastodon.gamedev.place/@gpuopen

Follow us on Zhihu
https://www.zhihu.com/org/gpuopen-7

MANY THANKS

- Stefan Petersson
- Krzysztof Kaminski
- Stephan Hodes
- Mark Satterthwaite
- Tom Hansson
- Oskar Homburg
- Lou Kramer
- Nadav Geva
- Nick Thibieroz

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 (FSR) versions 1, 2, and 3 are available on select games which require game developer integration and are supported on select AMD products. AMD does not provide technical or warranty support for AMD FidelityFX Super Resolution enablement on other vendors' graphics cards. See https://www.amd.com/en/technologies/fidelityfx-super-resolution for additional information. GD-187A.

© 2024 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.

THANK YOU!

Questions?

AMD GPUOpen

UBISOFT

UBISOFT snowdrop

AMD together we advance_

AMD EPYC

AMD RYZEN

AMD RADEON

AMD

together we advance_