Intel® XeSS Frame Generation (XeSS-FG) Developer Guide 1.1

Intel® XeSS Frame Generation (XeSS-FG) enables AI-based frame interpolation technology supported by Intel® Arc™ Graphics. Using deep learning to generate detailed, high-resolution frames, XeSS-FG improves gaming performance and fluidity without degrading image quality.

Table of Contents

Introduction

Intel® XeSS Frame Generation (XeSS-FG) is implemented as a sequence of compute shader passes, executed before a presentation event to generate an additional interpolated frame. XeSS-FG provides a proxy swapchain-based interface to simplify integration, abstracting the complexity of doing two presents for every game-rendered frame with smooth presentation pacing. The application initializes the library by passing a Direct3D* 12 (D3D12) device, which is being used for the main rendering. Intel® XeSS-FG can be enabled with a combination of technologies to deliver the best experience on both integrated and discrete graphics platforms. For optimal performance, we recommend the following frame rate targets:

- **40 FPS minimum (XeSS-SR or native input specification):** This provides adequate reconstruction for fluid gameplay, making it ideal for entry-level graphics.
- **60 FPS recommended (XeSS-SR or native input specification):** This ensures the best latency experience with the most fluid gameplay, targeting high-end graphics.

XeSS-FG supports all V-Sync and refresh rate modes (fixed refresh rate and variable refresh rate - VRR). For best results motion blur should be either disabled or modified to accommodate the additional frame rate resulting from XeSS-FG.

Requirements

- DirectX 12
- Windows 10/11 x64 - 10.0.19043/22000 or later
- Intel® Arc™ Graphics with Intel® Xe Matrix eXtensions (XMX)
- Intel® Graphics Driver Version 32.0.101.6252 or later
- Intel® XeLL 1.1.0 or later

Deployment

To use XeSS-FG in a project:

- Add "inc" folder to the include path
- Include xefg_swapchain.h, xefg_swapchain_d3d12.h and xefg_swapchain_debug.h
- Link with lib/libxess_fg.lib

The following file must be placed next to the executable or in the DLL search path:

- libxess_fg.dll
- Microsoft Visual C++ Redistributable version 14.40.33810 or later. The following libraries are required:
  - msvcp140.dll
  - vcruntime140.dll
  - vcruntime140_1.dll

Naming conventions and branding guidance

Please see “XeSS 2 Naming Structure and Examples.pdf” for approved naming conventions, branding guidance and settings menu examples.

Quick Start Guide

XeSS-FG integration into an application is a simple and straightforward process. At first the application must ensure it has all required inputs. If your application has support for XeSS-SR or TAA, then almost all required resources are already present. The only resource that is required for XeSS-FG but optional in XeSS-SR is the depth buffer. Here's the minimum set of required inputs:

- Motion vectors
  - Must be from current frame to previous.
  - `R16G16_FLOAT` or similar format.
  - Can be in NDC or in pixel space.
  - Can be jittered.
  - Preferred to be low-resolution (same as input to Super Sampling or TAAU) but can also be high-resolution (target resolution).
- Depth buffer
  - Can be inverse.
  - Buffer size must be the same as motion vectors buffer.
- Frame constants
  - View/Projection matrices required.
  - Optional jitter values if motion vectors are jittered.
  - Optional motion vector scale can be provided to match motion vector requirements.

Important! XeLL integration is a prerequisite for XeSS-FG integration. The application must provide a XeLL context to XeSS-FG before initialization – see XeLL Link for more details.

The following steps are required to properly integrate XeSS-FG into application:

- Create XeSS-FG context.
  - Can be done at application start.
  - `XefgSwapChainD3D12CreateContext`.
- Register logging callback
  - `xefgSwapChainSetLoggingCallback`
- Link to XeLL context
  - `xefgSwapChainSetLatencyReduction`
- Initialize context.
  - Should be done after swap chain creation.
  - `xefgSwapChainGetProperties`
  - `xefgSwapChainD3D12InitFromSwapChain` or `xefgSwapChainD3D12InitFromSwapChainDesc`
- Obtain pointer to XeSS-FG proxy swap chain.

    - This swap chain must be used instead of application swap chain.
    - `xefgSwapChainD3D12GetSwapChainPtr`
- Tag resources.
    - Each required resource must be tagged each frame.
    - `xefgSwapChainD3D12TagFrameResource`
        - Required `XEFG_SWAPCHAIN_RES_MOTION_VECTOR`
        - Required `XEFG_SWAPCHAIN_RES_DEPTH`
    - `xefgSwapChainTagFrameConstants`
    - `xefgSwapChainD3D12SetDescriptorHeap` call is required prior to presentation if and only if `XEFG_SWAPCHAIN_INIT_FLAG_EXTERNAL_DESCRIPTOR_HEAP` was used during XeSS-FG initialization.
- Perform frame presentation.
    - `xefgSwapChainSetPresentId` must be called before presentation.
- Application can pause XeSS-FG.
    - `xefgSwapChainSetEnabled`
- Application should destroy the XeSS-FG context before termination.
    - `xefgSwapChainDestroy`
- Turn off or reduce motion blur when XeSS-FG is ON

Programming Guide

XeSS-FG Integration steps

The XeSS-FG API provides an implementation of the `IDXGISwapChain` interface and contains functionality for resource life-time management and interaction with Intel® Graphics Driver for display pacing. The API manages an internal command list and submits GPU work on the application-provided command queue.

Important! XeLL integration is a prerequisite for XeSS-FG integration. The application must provide a XeLL context to XeSS-FG before initialization – see XeLL Link for more details. XeSS-FG uses XeLL for frame pacing support which is critical for reducing latency when frame generation is used.

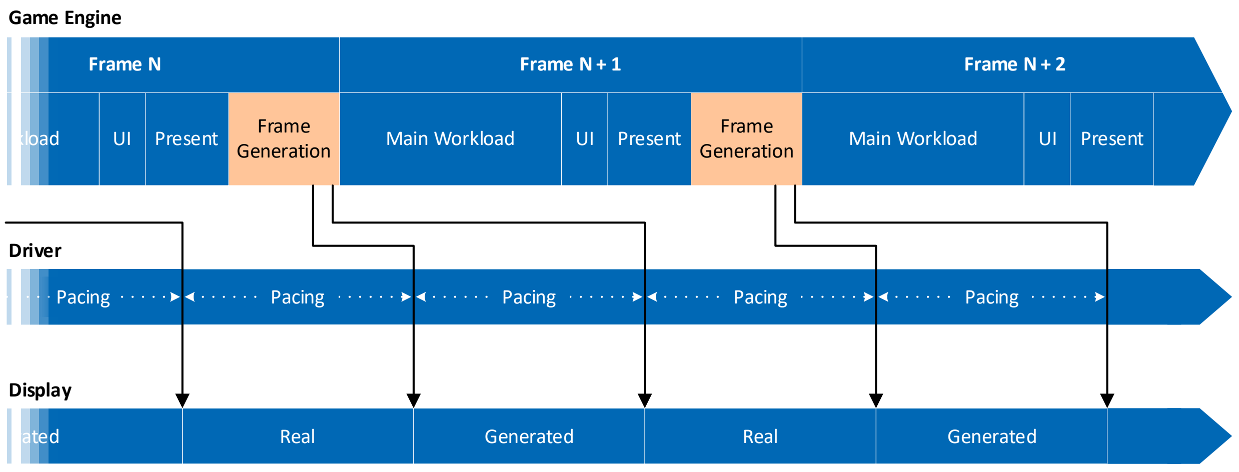


Figure 1. Library interaction with application's pipeline

Context Creation

To use XeSS-FG the application must create a XeSS-FG context via `xefgSwapChainD3D12CreateContext` call.

When creating a context, `xefgSwapChainD3D12CreateContext` will check for the necessary feature capabilities of the provided device and will fail if not sufficient. The application can query additional information for resource handling and interpolation options by calling `xefgSwapChainGetProperties`.

Logging Callback

The application can obtain additional diagnostic information by registering a logging callback using `xefgSwapChainSetLoggingCallback`:

```
static void LogCallback(const char* message,
      xefg_swapchain_logging_level_t level, void* userData);
xefg_swapchain_logging_level_t log_level =
      XEFG_SWAPCHAIN_LOGGING_LEVEL_WARNING;
xefgSwapChainSetLoggingCallback(xefgSwapChain, log_level, LogCallback,
userData);
```

There are four levels of logging:

- `XEFG_SWAPCHAIN_LOGGING_LEVEL_DEBUG`,
- `XEFG_SWAPCHAIN_LOGGING_LEVEL_INFO`,
- `XEFG_SWAPCHAIN_LOGGING_LEVEL_WARNING`,
- `XEFG_SWAPCHAIN_LOGGING_LEVEL_ERROR`.

We recommend using error level for production and warning or debug level for development.

XeLL Link

XeSS-FG requires a working XeLL integration. Please follow the XeLL developer guide.

Important! XeSS-FG will fail to initialize if there is no XeLL context provided, and frame generation will be disabled if XeLL is not initialized and enabled.

The link between both APIs can be established by calling xefgSwapChainSetLatencyReduction. The application must use the same frame counter with both APIs.

```
xefgSwapChainSetLatencyReduction(xefgSwapChain, xellContext);
```

Please make sure, that XeLL is enabled for the entire XeSS-FG context lifetime. It is allowed to disable XeLL when frame generation is disabled, but it must be re-enabled before FG enablement.

Initialization

There are two ways to initialize XeSS-FG:

- `xefgSwapChainD3D12InitFromSwapChain`
  - Queries parameters from provided swapchain.
  - Application must provide swapchain pointer with ref counter equal to 1.
  - **Application-provided swapchain will be released!** Function will fail if it can't release swapchain.
  - After successful call, provided swapchain pointer will be invalid.
  - Creates new swapchain for window.
  - Returns error in exclusive fullscreen mode.

- `xefgSwapChainD3D12InitFromSwapChainDesc`
    - New swapchain for window will be created.
    - Application shouldn't have any swapchain associated with window.
    - Won't fail in exclusive fullscreen mode but frame generation can't be enabled.

Once XeSS-FG is initialized the application must obtain a pointer to the proxy swap chain using `xefgSwapChainD3D12GetSwapChainPtr`.

Important! All interaction with the swap chain from that point on must be done by using the proxy swap chain only!

XeSS-FG will store a reference to the command queue provided during initialization and execute the interpolation work using this queue. The application must ensure this command queue still exists at the time it uses the XeSS-FG swap chain.

```
ComPtr<IDXGISwapChain4> swapChain = CreateApplicationSwapChain();
xefg_swapchain_d3d12_init_params_t params = {};
params.pApplicationSwapChain = swapChain.Get();
params.maxInterpolatedFrames = 1;
xefgSwapChainD3D12InitFromSwapChain(
        xefgSwapChain, commandQueue, &params);
xefgSwapChainD3D12GetSwapChainPtr(
        xefgSwapChain, IID_PPV_ARGS(&swapChain));
```

Please note that XeSS-FG is disabled by default and can be enabled by calling `xefgSwapChainSetEnabled`.

XeSS-FG allocates GPU resources of one of the two categories:

- Persistent allocations, such as network weights, and other constant data.
- Temporary allocations, such as network activations.

The application can optionally provide external resource heaps. Required sizes of these resource heaps can be obtained by calling xefgSwapChainGetProperties. The allocated heaps must use the default heap type (D3D12_HEAP_TYPE_DEFAULT). And to ensure optimal performance this heap should have HIGH memory residency priority. Use resource heap pointers and offsets in the corresponding initialization parameters.

The application can optionally provide an external descriptor heap, where XeSS-FG creates all its internal resource descriptors. You have to get the amount of required descriptors by calling xefgSwapChainGetProperties. You have to specify the XEFG_SWAPCHAIN_INIT_FLAG_EXTERNAL_DESCRIPTOR_HEAP flag on initialization and pass the descriptor heap and offset into xefgSwapChainD3D12SetDescriptorHeap. The descriptor heap must be of type D3D12_SRV_UAV_CBV_DESCRIPTOR_HEAP and shader visible.

The application can use initialization parameters to modify library behavior by providing initialization flags and/or configuration for UI handling modes.

```
xefg_swapchain_d3d12_init_params_t params = {};
params.initFlags = XEFG_SWAPCHAIN_INIT_FLAG_USE_NDC_VELOCITY;
params.uiMode = XEFG_SWAPCHAIN_UI_MODE_BACKBUFFER_HUDLESS;
```

See more initialization parameters and their requirements below.

Initialization Flags

The application can use different initialization flags to configure XeSS-FG:

- `XEFG_SWAPCHAIN_INIT_FLAG_NONE` – default
- `XEFG_SWAPCHAIN_INIT_FLAG_INVERTED_DEPTH` – depth texture uses inverted values
- `XEFG_SWAPCHAIN_INIT_FLAG_EXTERNAL_DESCRIPTOR_HEAP` – uses application-provided external descriptors heap
- `XEFG_SWAPCHAIN_INIT_FLAG_HIGH_RES_MV` – high-resolution motion vectors texture (swap chain resolution)
- `XEFG_SWAPCHAIN_INIT_FLAG_USE_NDC_VELOCITY` – motion vectors use normalized device coordinates (NDC)
- `XEFG_SWAPCHAIN_INIT_FLAG_JITTERED_MV` – motion vectors are not un-jittered
- `XEFG_SWAPCHAIN_INIT_FLAG_UITEXTURE_NOT_PREMUL_ALPHA` – RGB values in the UI texture are not pre-multiplied by the corresponding alpha value from the alpha channel

UI Composition Modes

XeSS-FG offers several options for handling user interface (UI) elements for interpolated frames. The application may provide additional resources:

- **HUD-less Color -** A full scene color buffer with all post-processing applied, same texture format and color space as the back buffer, but without any UI or HUD elements.

- **UI-Only Texture -** A texture that contains only the UI/HUD elements with appropriate alpha values to denote transparency, same texture format and color space as the back buffer. The contents of the UI-only texture must satisfy the following formula:

  ```
  Final_Color.RGB = UI.RGB + (1 - UI.Alpha) * Hudless.RGB
  ```

Important! For better UI composition quality and to minimize UI distortions, the application must provide both additional resources: HUD-less Color and UI-only texture.

By default, XeSS-FG uses the following UI composition mode:

- `XEFG_SWAPCHAIN_UI_MODE_AUTO` - Automatically determines the best UI handling mode based on the availability of additional resources (HUD-less color and UI-only texture).

The application can manually select UI Composition mode. With both HUD-less color and UI-only texture resources provided, use one of the following UI composition modes:

- `XEFG_SWAPCHAIN_UI_MODE_HUDLESS_UITEXTURE` - Interpolates on the HUD-less color buffer and blends UI from the UI texture with alpha using the formula above.
  Use this mode If the game can provide completely correct UI texture satisfying the formula

- `XEFG_SWAPCHAIN_UI_MODE_BACKBUFFER_HUDLESS_UITEXTURE` - Interpolates on the HUD-less color buffer and blends UI from the UI texture with additional UI refinement by UI extraction from the backbuffer.
  Use this mode if the game can provide the UI texture but doesn't completely satisfy the blending formula. For example, this is the preferred option for Unreal Engine*-based games.

In case either one or both additional resources cannot be provided by the application:

- `XEFG_SWAPCHAIN_UI_MODE_BACKBUFFER_HUDLESS` - Interpolates on the HUD-less color buffer and extracts the UI from the back buffer. The resulting visual quality of the UI composition may be worse than if UI-only texture is also provided, especially for semi-transparent UI elements.
  Use this mode if the game cannot provide the UI-only texture but can provide HUD-less color texture.

- `XEFG_SWAPCHAIN_UI_MODE_BACKBUFFER_UITEXTURE` - Interpolates on the back buffer and refines the UI using the UI texture with alpha.
  Use this mode if the game cannot provide the HUD-less color texture but can provide UI-Only texture.

- `XEFG_SWAPCHAIN_UI_MODE_NONE` - Interpolates on the back buffer without any special UI handling. This mode is ***not recommended*** as it may produce unacceptable visual quality artifacts in the interpolation of UI elements.

UI Composition Modes Comparison Table

The following table shows what resources must be provided for different UI handling modes:

ID	Mode Name	HUD-less Color	UI-Only Texture
0	AUTO	Optional	Optional
1	NONE	Not Used	Not Used
2	BACKBUFFER_UITEXTURE	Not Used	Required
3	HUDLESS_UITEXTURE	Required	Required
4	BACKBUFFER_HUDLESS	Required	Not Used
5	BACKBUFFER_HUDLESS_UITEXTURE	Required	Required

Performance Implications

Each UI composition mode has different performance characteristics. For instance, modes that require additional inputs, such as the UI-only texture, may incur a slight performance overhead due to the extra processing needed for blending. Here's an overview of the performance impact for each mode:

- `XEFG_SWAPCHAIN_UI_MODE_NONE` - This mode has no performance impact as it does not perform any additional processing for UI elements.
- `XEFG_SWAPCHAIN_UI_MODE_BACKBUFFER_HUDLESS_UITEXTURE` - While providing the most robust composition for the UI elements, this mode may introduce slightly higher overhead of frame generation cost on the GPU compared to other modes.
- **All other modes** - These modes involve minor performance overhead due to additional processing for UI handling, such as texture blending and UI extraction from the back buffer.

Error Handling

When integrating XeSS-FG into your application, it's important to handle potential errors that may arise during the selection and use of UI handling modes. Below are common error scenarios:

- **Invalid Mode** - If provided value is not a constant from `xefg_swapchain_ui_mode_t`. The library will return `XEFG_SWAPCHAIN_RESULT_ERROR_INVALID_ARGUMENT` error code on the initialization, indicating an invalid mode selection.

- **Missing Resources** - If the HUD-less color buffer or UI-only texture is missing when required by the selected mode, XeSS-FG will report

`XEFG_SWAPCHAIN_RESULT_ERROR_MISMATCH_INPUT_RESOURCES` using Present Status (see below for more information).

HDR Display Support

XeSS-FG provides support for High Dynamic Range (HDR) displays with the R10G10B10A2_UNORM pixel format. When using HDR, the back buffer, HUD-less texture and UI-only texture must have the same pixel format and must be in HDR10/BT.2100 color space. Since 2-bit alpha may not be enough to represent semi-transparent UI elements, in order to avoid flickering, `XEFG_SWAPCHAIN_UI_MODE_BACKBUFFER_HUDLESS_UITEXTURE` may be preferred. Please note that XeSS-FG does not support the FP16 HDR format and scRGB color space. Please refer to https://learn.microsoft.com/en-us/windows/win32/direct3darticles/high-dynamic-range#option-2-use-uint10rgb10-pixel-format-and-hdr10bt2100-color-space for more details.

Enabling Frame Generation

After creating the context and proxy swap chain, the application can enable frame generation by calling `xefgSwapChainSetEnabled`. It is highly recommended that the application enables XeSS-FG as early as possible.

```
xefgSwapChainSetEnabled(xefgSwapChain, true);
```

Frame generation cannot be enabled while application is in the fullscreen exclusive mode. If application enters fullscreen exclusive mode while XeSS-FG is enabled, frame generation will be disabled and the swapchain will work in a “passthrough mode”, i.e. just presenting game-rendered frames as without XeSS-FG.

After leaving fullscreen exclusive mode, `xefgSwapChainSetEnabled` must be called again to enable frame generation.

The application can toggle XeSS-FG on or off at any time, except when there are already tagged resources with `XEFG_SWAPCHAIN_RESOURCE_VALIDITY_ONLY_NOW` validity, and the command queue was not executed yet. Each call to disable will mark history for clean up next time frame generation is enabled. When enabling frame generation, the application must ensure that XeLL latency reduction is enabled.

It is recommended to disable XeSS-FG when the game is in the menu or any paused state or if no rendering commands get submitted.

When XeSS-FG is disabled, the proxy swapchain still has a small overhead. The application must destroy the XeSS-FG context and recreate the swap chain if they would like to achieve zero overhead.

Resource Tagging

The application must tag resources that are required for frame generation each frame by calling `xefgSwapChainD3D12TagFrameResource` with the corresponding frame counter value and filled `xefg_swapchain_d3d12_resource_data_t` structure.

It is important that all resources except for motion vectors and depth must be of the same size between frames. If the application needs to change resource size, it must reinitialize XeSS-FG.

Each call to tag a resource must specify one of following resource validity modes:

- `XEFG_SWAPCHAIN_RV_UNTIL_NEXT_PRESENT` – this flag says that the application guarantees the resource will be alive until the next present call. XeSS-FG will use this to optimize execution.
- `XEFG_SWAPCHAIN_RV_ONLY_NOW` – this flag says that resource is only valid now and XeSS-FG will record a copy operation to the provided command list.

Application must provide valid internal state for each tagged resource. It's recommended to have resources in following state based on resource validity:

- For `XEFG_SWAPCHAIN_RV_UNTIL_NEXT_PRESENT` use `D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE`.
- For `XEFG_SWAPCHAIN_RV_ONLY_NOW` use `D3D12_RESOURCE_STATE_COPY_SOURCE`.

Here is an example of resource tagging for input color frames:

```
xefg_swapchain_d3d12_resource_data_t hudlessColor = {};
hudlessColor.type = XEFG_SWAPCHAIN_RES_HUDLESS_COLOR;
hudlessColor.validity = XEFG_SWAPCHAIN_RV_UNTIL_NEXT_PRESENT;
hudlessColor.resourceSize = { width, height };
hudlessColor.pResource = renderTargets[m_frameIndex]
hudlessColor.incomingState =
  D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE;

xefgSwapChainD3D12TagFrameResource(xefgSwapChain, commandList,
  frameCounter, &hudlessColor));
```

Different types of resources are described in more detail below.

Inputs and Outputs

XeSS-FG will automatically get the required back buffer and color resources from the proxy swap chain.

Additionally, XeSS-FG requires a minimum set of inputs for every frame:

- Velocity texture (Motion Vectors)
- Depth texture
- Set of frame constants

Depending on the selected UI handling mode the application may need to provide additional resources:

- HUD-less color texture
- UI texture

The application provides these inputs to XeSS-FG by tagging them: resources are tagged using `xefgSwapChainD3D12TagFrameResource`, frame constants are tagged using `xefgSwapChainTagFrameConstants`.

If the application initialized XeSS-FG with `XEFG_SWAPCHAIN_INIT_FLAG_EXTERNAL_DESCRIPTOR_HEAP` the corresponding descriptor heap must be provided prior to Present being issued.

Motion Vectors

Motion vectors specify the screen-space motion in pixels from the current frame to the previous frame. If the application uses NDC for motion vectors it should pass an additional flag (XEFG_SWAPCHAIN_INIT_FLAG_USE_NDC_VELOCITY) during XeSS-FG context initialization. XeSS-FG accepts motion vectors in the format `R16G16_FLOAT` or similar, where the R channel encodes the motion in x, and the G channel motion in y. The motion vectors do not include motion induced by the camera jitter. Motion vectors can be low-res (default and recommended), or high-res. Low-res motion vectors are represented by a 2D texture at the resolution of the input to Super Sampling pass or TAAU, whereas high-res motion vectors are represented by a 2D texture at the target resolution.

Important! Using high-res motion vectors can greatly reduce performance of frame generation especially in higher resolutions, and it is generally recommended to use low-res motion vectors

In the case of high-res motion vectors, the velocity component resulting from camera animations is computed at the target resolution in a deferred pass, using the camera transformation and depth values. However, the velocity component related to particles and object animations is typically computed at the input resolution and stored in the G-Buffer. This velocity component is upsampled and combined with the camera velocity to produce the texture for high-res motion vectors. XeSS-FG also expects the high-res motion vectors to be dilated. For example, the motion vectors represent the motion of the foremost surface in a small neighborhood of input pixels (such as 3 * 3). High-res motion vectors can be computed in a separate pass by the user.

Low-res motion vectors are not dilated, and directly represent the velocity sampled at each jittered pixel position. XeSS-FG internally up-samples motion vectors to the target grid and uses the depth texture to dilate them. Figure 2 shows the same motion specified with low-res and high-res motion vectors.

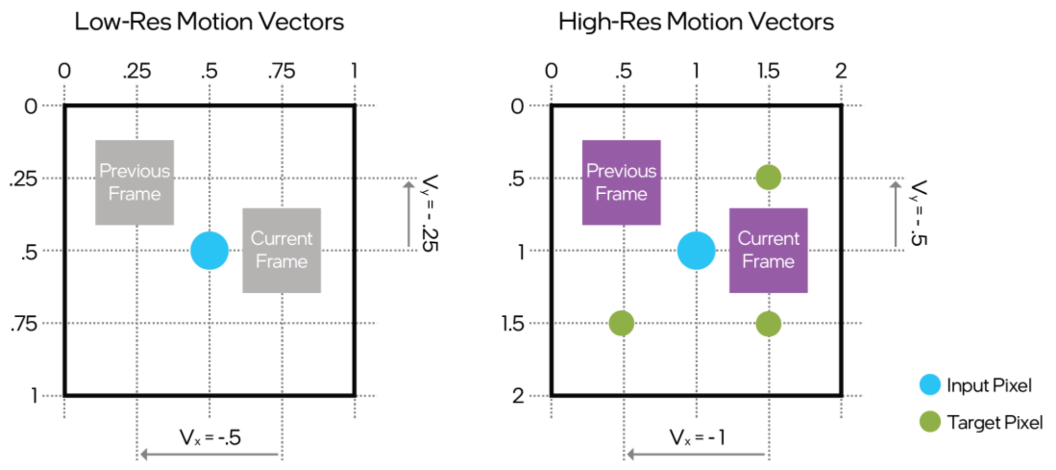


Figure 2. Convention for specifying the low-res and high-res motion vector to XeSS-FG.

Some game engines only render objects into the G-buffer, and quickly compute the camera velocity in the TAA shader. In such cases, an additional pass is required before XeSS-FG execution to merge object and camera velocities and generate a flattened velocity buffer. In such scenarios, high-res motion vectors might be a better choice, as the flattening pass can be executed at the target resolution.

Depth

Any depth format, such as `D32_FLOAT` or `D24_UNORM`, is supported. By default, XeSS-FG assumes that smaller depth values are closer to the camera. However, several game engines use inverted depth, and this can be enabled by setting `XEFG_SWAPCHAIN_INIT_FLAG_INVERTED_DEPTH`.

Buffer size must be same as Motion Vectors buffer size.

HUD-less color texture

HUD-less color texture is a full scene color texture with all post-processing applied, identical in color space and size to the back buffer, but without any UI or HUD elements.

It is required to use same texture format as main back buffer.

UI texture

UI texture – is a texture that contains only UI elements, including HUD, with appropriate alpha values to denote transparency. The contents of UI-only texture must satisfy the following formula:

```
    Final_Color.RGB = UI.RGB + (1 - UI.Alpha) * Hudless.RGB
```

This texture should have the same texture format, color space and size as back buffer.

Frame Constants

The application must provide a `xefg_swapchain_frame_constant_data_t` structure on each frame to properly update configuration and set up a scene data by calling `xefgSwapChainTagFrameConstants`. The structure includes the following data:

- `viewMatrix` - scene view matrix in a row-major order without applied jitter
- `projectionMatrix` - camera projection matrix in a row-major order without applied jitter
- `jitterOffsetX/Y` - Jitter offset in a range [-0.5, 0.5]
- `motionVectorScaleX/Y` - Motion vector scale factors
- `resetHistory` – if this flag is set XeSS-FG will ignore previous frame data and won't perform interpolation for current frame. Should be set when big difference in output happens
- `frameRenderTime` – frame render time on GPU in milliseconds.

Example:

```
xefg_swapchain_frame_constant_data_t constData = {};
XMFLOAT4X4 float4x4;
float fx = m_constantBufferData.offset.x;
float fy = m_constantBufferData.offset.y;
XMStoreFloat4x4(&float4x4, DirectX::XMMatrixTranslation(fx, fy, 0));
memcpy(constData.viewMatrix, float4x4.m, sizeof(float) * 16);
XMStoreFloat4x4(&float4x4, DirectX::XMMatrixIdentity());
memcpy(constData.projectionMatrix, float4x4.m, sizeof(float) * 16);

constData.jitterOffsetX = constData.jitterOffsetY = 0.0f;
constData.motionVectorScaleX = constData.motionVectorScaleY = 1.0f;

xefgSwapChainTagFrameConstants(xefgSwapChain, frameCounter, &constData );
```

Descriptor Heap

An `ID3D12DescriptorHeap` must be provided if XeSS-FG was initialized with `XEFG_SWAPCHAIN_INIT_FLAG_EXTERNAL_DESCRIPTOR_HEAP` prior to present being called, through `xefgSwapChainD3D12SetDescriptorHeap`.

The provided descriptor heap must have at least `xefg_swapchain_properties_t::requiredDescriptorCount` entries to accommodate all descriptors during the interpolation execution.

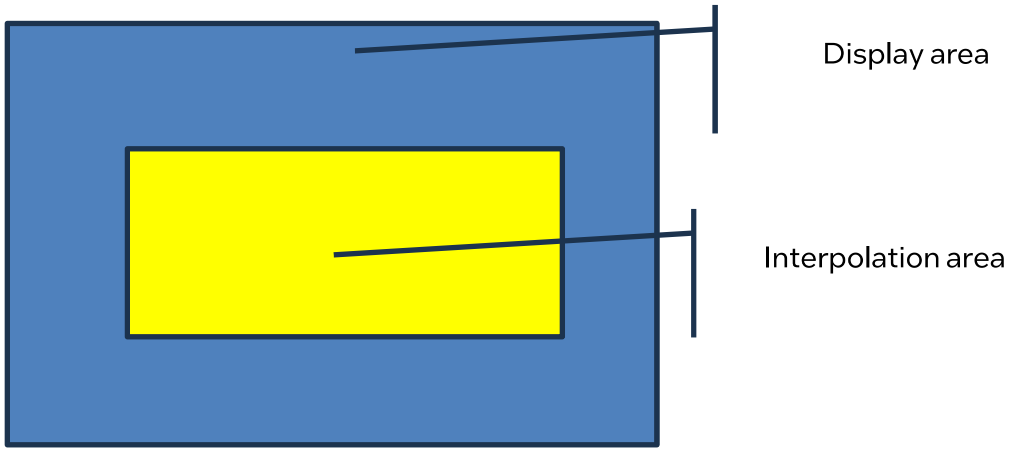
One must be mindful of providing different offsets or different heaps per frame as frame generation execution may still be ongoing while the next frame is being programmed by the application. Once the descriptor heap and offset are no longer in use they can be provided again for use in a new frame interpolation.

Letterboxing

Display area: The size of the back buffer. Defined by the swap chain initialization size.

Interpolation area: a region of interest within the color resource (HUDless, back buffer, UI) for the interpolation. Defined by the resource size and offset.

To achieve the letterboxing effect, XeSS-FG supports partial frame interpolation, i.e., interpolation within a defined interpolation area. The image below shows the relationship between interpolation and display areas inside the back buffer resource:



To enable such interpolation mode, the application must follow this procedure:

1. Initialize or re-initialize XeSS-FG with the required display size.
2. For each frame:
    a. Tag HUDless and/or UI resource with required interpolation size (*resourceBase* and *resourceSize*, the members of the

   xefg_swapchain_d3d12_resource_data_t structure) by invoking *xefgSwapChainD3D12TagFrameResource* function.

   b. Tag back buffer resource with required interpolation size (*resourceBase* and *resourceSize*, the members of the *xefg_swapchain_d3d12_resource_data_t* structure) by invoking *xefgSwapChainD3D12TagFrameResource* function. Note: XeSS-FG ignores other members of the xefg_swapchain_d3d12_resource_data_t structure.

The interpolation will work only when the following conditions are met:

1. The interpolation area (width and height) in HUDless, UI, and back buffer resources is the same for a given frame.
2. The interpolation area (width and height) is the same in the previous and next frames.

Otherwise, interpolation will be skipped, and the previously rendered frame will be presented.

During the interpolation execution, the display area in the back buffer is entirely fulfilled by the XeSS-FG:

1. Pixels within the interpolation area are interpolated.
2. Pixels outside the interpolation area are copied from the previously generated back buffer resource.

Asynchronous shaders pipeline build

To improve overall startup time, XeSS-FG supports asynchronous shader compilation:

1. Create the XeSS-FG context early during game initialization.
2. Right after that call *xefgSwapChainD3D12BuildPipelines* in non-blocking mode. XeSS-FG will immediately start compiling kernels in the background, while the game keeps loading.
3. Initialize XeSS-FG at a later time (e.g. when the user chooses it in the menu). By that time compilation has hopefully finished.
4. The shader's compilation status can be queried by calling *xefgSwapChainGetPipelineBuildStatus*.

Execution

XeSS-FG execution happens automatically on the present call to the proxy swap chain. Actual presentation happens asynchronously using a separate thread to pace frames properly.

All requested interpolated frames will be generated, presented in order, and paced correctly. XeSS-FG manages resources efficiently, performing copies if necessary.

The application must call `xefgSwapChainSetPresentId` before each swap chain's `Present`. This call is required to correctly identify resources tagged for this frame and failure to make this call will result in visual corruption or execution failure. It is important to call `xefgSwapChainSetPresentId` after tagging all resources.

```
    xefgSwapChainSetPresentId(xefgSwapChain, frameCounter);
    ++frameCounter;
    swapChain->Present(…);
```

Present Status

In order to improve application state tracking XeSS-FG provides `xefgSwapChainGetLastPresentStatus` call, which returns `xefg_swapchain_present_status_t` structure with last interpolation result, number of frames queued to the presentation and indicator if XeSS-FG was enabled.

```
    xefg_swapchain_present_status_t lastPresentStatus = {};
    xefgSwapChainGetLastPresentStatus(xefgSwapChain, &lastPresentStatus);
```

Important! It will only work when being called after presentation and before next call to `xefgSwapChainSetPresentId`, because data are associated with present ID.

It is recommended to call it as soon after `Present` call as possible.

Changing Window State

When the application is resizing the window or changing the fullscreen state, it is necessary to call `ResizeBuffers`. The application must ensure that all outstanding resources are released to be able to recreate them. It is important also for tagged resources, because at this point history will be reset and any on-the-fly work related to tagged resources must be completed.

XeSS-FG updates the swap chain queue associated with the swap chain and ignores the queues provided to `ResizeBuffers1` substituting them with an internal queue instead.

Shutdown

Before calling `xefgSwapChainDestroy` the application must ensure that:

- All operations related to XeSS-FG have been completed and there will be no tagging of resources/constants or enabling of XeSS-FG anymore.
- All references for XeSS-FG swap chain have been released.

- XeLL context is **not** destroyed: XeSS-FG context should be destroyed before XeLL context.

Calling `xefgSwapChainDestroy` will clear/release all objects related to frame generation, including the XeSS-FG swap chain API context handle. Function will return error if the XeSS-FG swap chain has any additional references. This means the swap chain wasn't destroyed, and the application must release all swap chain references first.

XeSS-FG uses COM-pointers to hold references for some resources received from the application. XeSS-FG will release such resources.

After calling `xefgSwapChainDestroy` the swap chain must be recreated if the application intends to continue rendering, even without XeSS-FG or other frame generation technologies.

Compatibility with 3rdparty frame generation software

XeSS-FG is not compatible with any 3rd-party frame generation software.

If the application wants to switch from XeSS-FG to any other frame generation technology, it must fully shutdown XeSS-FG. If the application wants to switch to XeSS-FG it should shut down any other running frame generation technology and perform full initialization starting from swap chain creation. XeSS-FG does not support sharing of the swap chain with any other libraries.

Debugging

Most common way to debug API is to attach the logger callback and check log messages, but if this is not enough and more sophisticated options are needed then there are two additional tools – XeSS Inspector tool and an API for debug options:

- `XEFG_SWAPCHAIN_DEBUG_FEATURE_SHOW_ONLY_INTERPOLATION` – only generated frames will be seen on the screen. If frame generation fails – current back buffer will be presented instead, this behavior can be changed by additional flag `XEFG_SWAPCHAIN_DEBUG_FEATURE_PRESENT_FAILED_INTERPOLATION`.
- `XEFG_SWAPCHAIN_DEBUG_FEATURE_TAG_INTERPOLATED_FRAMES` – showing purple boxes and vertical lines on interpolated frame to distinguish it from rendered one, as well as visually inspect tearing issues
- `XEFG_SWAPCHAIN_DEBUG_FEATURE_PRESENT_FAILED_INTERPOLATION` – enables black frame presentation if interpolation failed, otherwise current back buffer will be present.

Debug options can be enabled using `xefgSwapChainEnableDebugFeature`. This function can be called at any time between context creation and destruction.

Example:

```
xefgSwapChainEnableDebugFeature(xefgSwapChain,
XEFG_SWAPCHAIN_DEBUG_FEATURE_SHOW_ONLY_INTERPOLATION, true, nullptr);
```

Versioning

The library uses a `<major>.<minor>.<patch>` versioning format, and Numeric 90+ scheme, for development stage builds. The version is specified by a 64-bit sized structure (xefg_swapchain_version_t), in which:

- A major version increment indicates a new API, and potentially a break in functionality.
- A minor version increment indicates an API improvement such as optional inputs or flags.
- A patch version increment may include performance or quality tweaks or fixes for known issues.

The version is baked into the XeSS-FG SDK release and can be accessed using the function `xefgSwapChainGetVersion`.

Notices