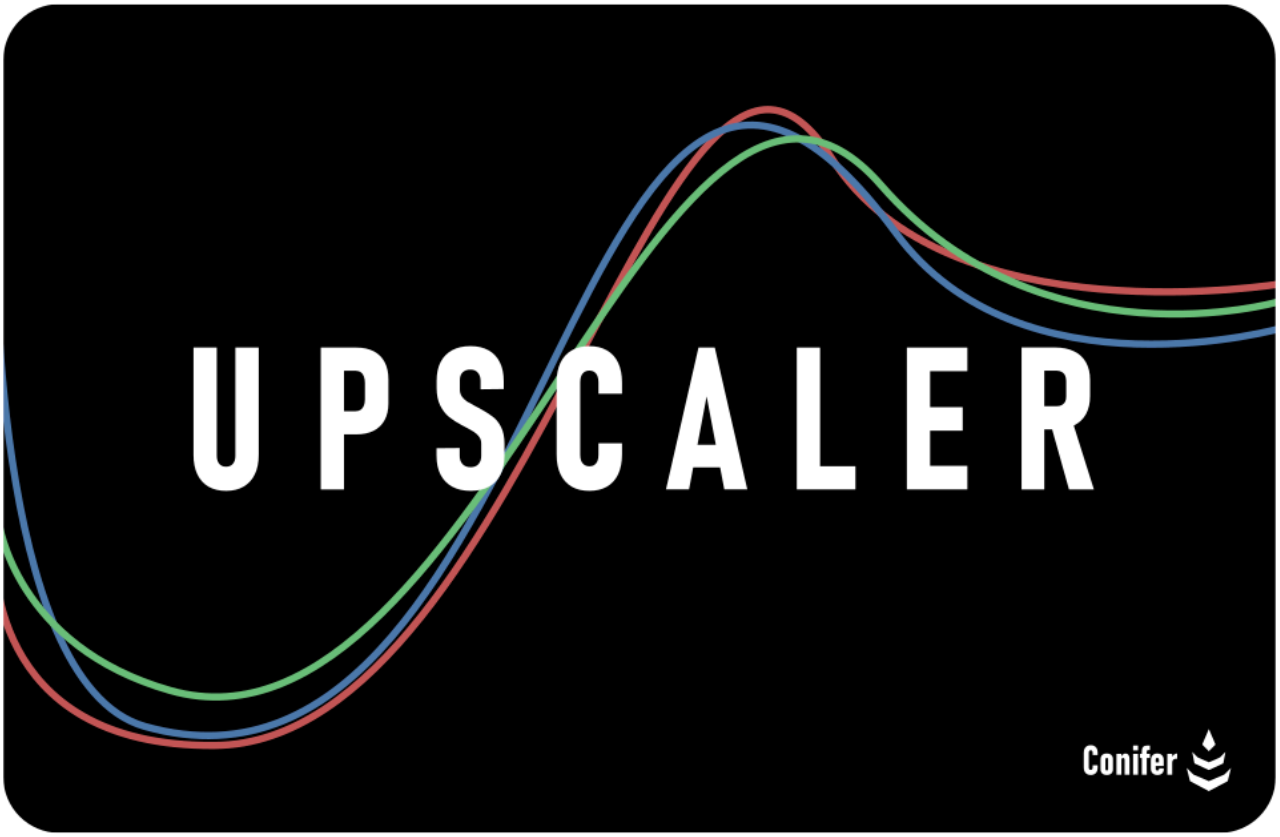
Conifer



Upscaler User Manual

v2.0.0

Table of Contents

1 Changes

- **v2.0.0**

NEW
- Updated to NVIDIA Streamline v2.7.2
- Added frame generation (BETA FEATURE, Vulkan only)

IMPROVED
- Compatibility with many setups and configurations including some 3rd party plugins

FIXED
- Many bugs that could result in anything from crashes to technique failures

- **v1.2.0**

NEW
- Updated to FidelityFX v11.3
- SGSR v1 and v2
- APIs for querying more information about a technique

FIXED
- Subtle, dark vertical lines appearing in FSR output

- **v1.1.3**

NEW
- Updated to FidelityFX v1.1.2

FIXED
- FSR receiving incorrect depth information

- **v1.1.2**

NEW
- Updated to FidelityFX v1.1.1

- **v1.1.1**

NEW
- Updated to NVIDIA Streamline 2.4.15

IMPROVED
- Greatly simplified update and installation processes

- **v1.1.0**

NEW
- Updated to FSR 3.1
- Updated to NVIDIA Streamline
- Updated to X^{e}SS 1.3.1
- Dropped Linux support

IMPROVED
- Improved Inspector GUI
- Improved update process
- Improved performance

- Improved quality
- Reduced VRAM consumption

FIXED

- Bugs resulting in crashes
- Bugs related to installation

- **v1.0.0**

NEW

- FSR 2.2.1, XeSS 1.3, DLSS 3.7.10
- Windows and Linux
- Vulkan, DirectX 12, and DirectX 11

2 Getting Started

After importing *Upscaler* go to the `Assets` → `Conifer` → `Upscaler` → `Runtime` directory and drag and drop the `Upscaler.cs` script onto the camera that you want to upscale. Next, Unity must be restarted. If you neglect to restart Unity at this point, not all features of *Upscaler* will be available. This is due to our native plugin needing to hook into Unity's rendering engine's initialization phase.

Consult Upscaler Settings for information on how to control the upscaling from the Unity Editor. See Changing Settings to learn how to control the upscaling from within your scripts. Finally, check out GUI Integration in order to add upscaling controls to your game's GUI.

3 Using the Unity Editor GUI

This section provides detailed descriptions on the uses of each of the elements in *Upscaler*'s Inspector GUI.

3.1 Upscaler Settings

The "Upscaler" dropdown menu allows you to select one of the available upscalers (referred to as `technique`s in the code to avoid confusion). The "None" technique forces the game to render at the output resolution. It entirely removes *Upscaler* from Unity's render pipeline. It disregards all of the other settings; ultimately, it behaves as an off switch for *Upscaler*. Selecting "Deep Learning Super Sampling" will enable the DLSS on this camera. DLSS passes motion vectors, the depth buffer, and the color buffer through a neural network to quickly perform high quality upscaling. DLSS is only available on NVIDIA GPUs. Selecting "Fidelity FX Super Resolution" will enable FSR on this camera. FSR is an

open source project backed by AMD. It takes motion vectors, the depth buffer, and the color buffer, and processes them with cross-platform, cross-vendor, handcrafted upscaling algorithms. Selecting “Xe Super Sampling” will enable X^{e}SS. X^{e}SS is an AI powered upscaling algorithm that is developed by Intel. While it is able to work on any vendor’s GPUs it is specially designed to leverage the acceleration hardware found on Intel’s GPUs. Use an Intel GPU for best results. Selecting “Snapdragon Game Super Resolution Spatial” will enable SGSR v1, an algorithm derived from FSR v1 and remastered by Qualcomm for optimal performance on mobile devices. This option is not recommended for desktop GPUs. Finally, selecting “Snapdragon Game Super Resolution Temporal” will enable SGSR v2, an algorithm derived from temporal versions of FSR. This has also been optimized by Qualcomm for optimal performance on mobile devices, but it also performs very well on desktop GPUs while providing similar quality when compared with desktop focused options.

The “Quality” dropdown menu allows you to select a quality mode at which to run the technique. This setting is ignored if the “Upscaler” option is set to “None”.

Consult the charts below for more information. Note that actual numbers may slightly vary depending on the resolution due to the fact that images are sized with a whole number of pixels.

DLSS has six quality modes, five of which scale by a constant amount, and “Auto”.

DLSS Quality Mode Effects on Render Resolution

Quality Mode	% of pixels rendered per-axis	% of pixels rendered
AntiAliasing	100%	100%
Quality	66.6%	44.4%
Balanced	58%	33.6%
Performance	50%	25%
Ultra Performance[1]	33.3%	11.1%

[1] DLSS’ ‘Ultra Performance’ quality mode does **not** support dynamic resolution scaling.

FSR 3.1 has five quality modes, four of which scale by a constant amount and “Auto”.

FSR 3.1 Quality Mode Effects on Render Resolution

Quality Mode	% of pixels rendered per-axis	% of total pixels rendered
AntiAliasing	100%	100%
Quality	66.6%	44.4%
Balanced	58.8%	34.6%
Performance	50%	25%
Ultra Performance	33.3%	11.1%

XeSS has eight quality modes, seven of which scale by a constant amount and “Auto”.

XeSS Quality Mode Effects on Render Resolution

Quality Mode	% of pixels rendered per-axis	% of total pixels rendered
AntiAliasing[2]	100%	100%
Ultra Quality Plus	76.9%	59.2%
Ultra Quality	66.7%	44.5%
Quality	58.8%	34.4%
Balanced	50%	25%
Performance	43.5%	18.9%
Ultra Performance	33.3%	11.1%

SGSR Spatial has seven quality modes, seven of which scale by a constant amount and “Auto”.

SGSR Spatial Quality Mode Effects on Render Resolution

Quality Mode	% of pixels rendered per-axis	% of total pixels rendered
Ultra Quality Plus	76.9%	59.2%
Ultra Quality	66.7%	44.5%
Quality	58.8%	34.4%
Balanced	50%	25%
Performance	43.5%	18.9%
Ultra Performance	33.3%	11.1%

[2] XeSS’ ‘AntiAliasing’ quality mode does **not** support dynamic resolution scaling.

SGSR Temporal has seven quality modes, seven of which scale by a constant amount and “Auto”.

SGSR Temporal Quality Mode Effects on Render Resolution

Quality Mode	% of pixels rendered per-axis	% of total pixels rendered
AntiAliasing	100%	100%
Ultra Quality Plus	76.9%	59.2%
Ultra Quality	66.7%	44.5%
Quality	58.8%	34.4%
Balanced	50%	25%
Performance	43.5%	18.9%
Ultra Performance	33.3%	11.1%

The comparisons shown in the charts above are always between the total number of pixels in the given resolution and the total number of pixels in the output resolution.

The ‘Auto’ quality mode applies to each technique in the same way. This particular method of determining the chosen quality mode is suggested in the DLSS Developer Guide.

“Auto” Quality Mode

Output Resolution	Chosen Quality Mode
Less than or equal to 2560x1440	Quality
Less than or equal to 3840x2160	Performance
Greater than 3840x2160	Ultra Performance

3.2 Advanced Settings

Upscaler’s advanced settings change based on the selected *Upscaler*. Note that XeSS does not have an advanced settings foldout menu.

3.2.1 Advanced DLSS Settings

The “DLSS Preset” dropdown menu gives you several options to help DLSS work better with your game. Most games will not need to deviate from the “Default” preset, but some games may benefit from them.

- Use “Stable” if the contents of your camera’s view changes slowly. This option favors older information for better antialiasing at the expense of a higher probability of ghosting. It may show a sharper image when used with the “AntiAliasing” quality mode.

- Use “Fast Paced” if the contents of your camera’s view changes quickly. This option favors new information for better clarity at the expense of antialiasing quality.
- Use “Anti Ghosting” if some of the contents of your camera’s view lacks motion vectors. This can be useful when working with third party plugins that do not correctly handle motion vectors.

3.2.2 Advanced FSR Settings

A non-zero “Sharpness” value adds FSR’s internal sharpening pass after upscaling. The value of the slider controls the amount of sharpening that FSR will perform. Conifer recommends increasing the sharpness as the upscaling ratio becomes more extreme.

The “Use Reactive Mask” checkbox enables automatic reactive mask generation in FSR. Though default FSR configurations will work acceptably in many scenarios, this opens up a number of fine tuning options. The defaults for these options are set to values that Conifer found works well for our test scene, though AMD also recommends that each title refine the numbers to provide the best results.

- The “Reactivity Max” setting can be used to set the maximum reactivity value that the automatic generation process can produce. More reactivity favors newer information at the expense of aliased and blocky results.
- The “Reactivity Scale” setting can be used to scale the results of the automatic generation process. Larger values will result in a more reactive pixel
- The “Reactivity Threshold” setting can be used to control which parts of the image are made reactive. Increase this to make more of the image reactive.

3.2.3 Advanced SGSR1 Settings

Using SGSR Spatial’s “Sharpness” option incurs no extra frametime cost. Its effect is evident, but does not result in a ‘crunchy’ image.

SGSR Spatial’s “Use Edge Direction” option incurs a noticeable frametime cost, but results in a much improved upscale. Its visual impact reduces fireflies, and blocky patterns in SGSR Spatial’s default output. Conifer recommends keeping this on if it is within your performance budget.

3.2.4 Advanced SGSR2 Settings

The “SGSR Method” setting allows choosing between three different methods for computing the SGSR algorithm. These methods offer a range of tradeoffs between performance and quality.

- The “Compute 3 Pass” option is the slowest of the three methods, but tends to result in the best quality upscale. Conifer recommends this option for most desktop GPUs and high-end mobile GPUs.
- The “Compute 2 Pass” option is faster than the “Compute 3 Pass” option, but looks better than the “Fragment 2 Pass” option. This option may produce a more stable output in regions with high frequency textures when compared to “Compute 3 Pass”. Conifer recommends this option for low-end desktop GPUs and most mobile GPUs.
- The “Fragment 2 Pass” option is the fastest of the lot, but tends to have blurry or blocky results. Conifer recommends this option if Compute Shaders are not available on your target platform, and instead of falling back to SGSR Spatial if possible.

3.3 Debug Settings

Debug settings should be left off for release builds, but may be useful if encountering issues with some of the features provided by Upscaler. Below are some of the debug settings that do not affect a specific feature.

- ‘Force History Reset’ instructs *Upscaler* to clear the history every frame. This can be useful for a variety of things such as better understanding how clearing the history behaves or seeing how long it takes the technique to recover from a history reset.
- ‘Global Log Level’ allows filtering out excess log messages that may be irrelevant from *Upscaler*. Log messages from Unity’s initialization phases are only given after the ‘Global Log Level’ is set (probably the first time the *Upscaler* Inspector GUI is opened).

3.3.1 Upscaling Debug Settings

- ‘View Debug Images’ shows all the inputs that FSR is receiving, and what its internal phases are outputting. This is only available when using FSR.
- ‘Dynamic Resolution’ allows you to change the render resolution straight from the Unity Editor.
- ‘Overlay Rendering Area’ allows you to visualize the screen area that is actually being rendered. This can be useful for visualizing Dynamic Resolution changes while testing.

3.3.2 Frame Generation Debug Settings

All settings mentioned below are only available when using frame generation. Frame generation itself is only available when using the Vulkan graphics API.

- 'View Frame Generation Debug Images' shows all the inputs that FSR's frame generation algorithm is receiving as well as its current output. This option will force only generated frames to be rendered as it requires information from both the previous frame and the generated frame to display.
- 'Show Tear Lines' displays a green line on the left side of the screen during generated frames and a color shifting line on the left side of the screen during all frames. This is useful for visually synching the display with frame generation. When V-Sync is enabled there should be no tears in these lines.
- 'Show Reset Indicators' displays a blue bar at the top of the screen when the frame generation's internal reset detector does a soft reset. When this happens is not configurable. A red bar is displayed when an external hard reset request is received. An external hard reset request can be sent using *Upscaler's* API or by enabling 'Force History Reset' in the Editor GUI.
- 'Show Pacing Indicator' has no visible effect. AMD's documentation does not describe its function. It is included here only for completeness.
- 'Only Present Generated Frames' skips the present calls sent for non-generated frames and only schedules the presents for generated frames. This can be used to see the effects of any artifacting caused by frame generation more clearly.

4 URP Integration

While integrating with the URP, it is not only important to add the `Upscaler` script to the camera, but the `UpscalerRendererFeature` must also be added to the URP Renderer which that camera is using. Failing to do either will result in an error. A help box will appear in the *Upscaler*'s Inspector GUI on the camera when an `UpscalerRendererFeature` is not present in that camera's URP Renderer.

4.1 Upscaling

During the `SetupRenderPasses` the camera's target is reconfigured to one of *Upscaler*'s internal render resolution render targets built on top of the `RenderTexture`s used by the camera's original render targets.. From there rendering proceeds as normally until upscaling occurs in the `RenderPassEvent.BeforeRenderingPostProcessing`

event. After performing upscaling, *Upscaler* rescales the camera's targets, then copies the upscaled output into it. Rendering proceeds normally again from there.

4.2 Frame Generation

Before post processing is performed the motion vectors for the current frame are intercepted. After all other rendering happens the hudless and depth buffers are intercepted and passed to the frame generator. The frame generator will automatically generate and display new frames between every two consecutive frames.

4.3 A note on Unity's Screen Space Ambient Occlusion

If using the Screen Space Ambient Occlusion (SSAO) Renderer Feature, 'After Opaque' must be enabled to make it compatible with *Upscaler*. Please note that SSAO works at a high resolution, but that work is thrown away when it is blitted onto the render resolution image.

5 Dynamic Resolution

Change the resolution using the `Upscaler.RenderResolution` property. The resolution will be clamped to within a valid range as defined by `MaxRenderResolution` and `MinRenderResolution`. While *Upscaler* does support dynamic resolution, not all quality modes support it. Attempting to adjust dynamic resolution while using a quality mode that does not support dynamic resolution will fail as *Upscaler* will clamp the requested resolution to the only allowed value. See Upscaler Settings for more information about quality modes. *Upscaler*'s Editor GUI will notify you when such a quality mode is in use.

6 Changing Settings

Upscaler allows you to change the settings through C# scripts using its API. Below is an example script that shows how one can get the `Upscaler` object, then use the API it provides to change upscaling settings on-the-fly.

Obtain an instance of the `Upscaler` class from the main camera. It is highly recommended that you ensure that the camera actually exists. The example below works, but is not error protected at all. Use it with caution.

```
Upscaler.Upscaler upscaler = Camera.main!.GetComponent<Upscaler.Upscaler>();
```

You can immediately change settings. For example to change the active `technique` use:

```
_upscaler.technique = Upscaler.Technique.FidelityFXSuperResolution;
```

Your changes will come into effect during the next frame and any errors will be handled by the registered error handler then. See Handling Errors for more information about error handling.

7 Handling Errors

Errors can occur when requesting a settings change. By default, when this happens the default error handler will reset the current technique to “None”. This will remove the *Upscaler* plugin from the pipeline ensuring that *Upscaler* does not cause any rendering errors. If this is not the behavior that you want, you can choose to override the error handler. You may want to instead revert back to the last settings that worked, print the error to the console, or anything else.

First, to override the error handler, replace the empty lambda with the function that you wish to respond to errors in the code below.

```
Upscaler.Upscaler upscaler = Camera.main!.GetComponent<Upscaler.Upscaler>();
upscaler.ErrorCallback = (status, message) => { };
```

Next, if you wish to perform any settings changes within your error handler, refer to section Changing Settings. Note that error handlers should call the `ApplySettings` function after making any settings changes.. If there is still an error status after the registered error callback is called then the active technique will be set to “None” to prevent fatal errors.

The error handler is always called from the `Update` method of the `Upscaler` class in the frame after the error to which it is responding occurs or, in the case of a settings related error, the same frame.

8 Things to be Aware of

The following is a list of advice from Conifer, NVIDIA, AMD, and Intel as given in the DLSS developer guide, FSR documentation pages, and XeSS developer guide.

1. When a technique is active, disable all other forms of anti-aliasing including but not limited to MSAA, TAA, FXAA, and SMAA. *Upscaler* will notify you if any Unity provided anti-aliasing is enabled, but it is up to you to ensure that other third party packages are not injecting their own anti-aliasing solutions.
    1. This can be ignored when using SGSR Spatial. Antialiasing will improve visual quality still, but if frametime allow for an extra antialiasing pass, it may be worth simply using SGSR Temporal + Fragment 2 Pass instead..
2. Use *Upscaler* for rendering directly to the screen **OR** rendering to an image that will be post processed before being presented directly to the screen. Do **NOT** use *Upscaler* to upscale things such as:
    1. The view through portals.
    2. The view of a security camera onto a monitor.
    3. Static textures.
    4. Shadow maps.
    5. A depth pass.

    Or any number of possible non direct-to-screen views.
3. DLSS is incompatible with RenderDoc as of RenderDoc v1.35. *Upscaler* will still load correctly and other techniques will work, but DLSS will be unavailable. Use NVIDIA's Nsight Graphics for graphics debugging if you require debugging abilities with DLSS enabled. You can connect to Unity's Editor by launching it from Nsight Graphics. *Upscaler* will be able to load and use DLSS successfully, and will not interfere with any captures. Nsight Graphics was used to help create this plugin.
4. Read the relevant parts of the DLSS developer guide, FSR documentation pages, and XeSS developer guide for more information.

9 GUI Integration

Consult the DLSS UI Developer Guidelines PDF, the FidelityFX Naming Guidelines, and/or the XeSS developer guide for more information about integration with in-game GUIs.

Do note that in contrast to what is suggested in the above guidelines, the ‘Auto’ setting does not disable upscaling under any circumstances. Conifer chose to do this because we believe that the settings that the user selects should be honored. If the user wants upscaling off, they turn off upscaling. If you choose to follow the guidelines more closely you will need to perform the required resolution check yourself. Furthermore, in an effort to unify the settings between techniques, an ‘Auto’ setting has been added to FSR and XeSS that follows the guidelines presented by the DLSS developer guide.

10 Contact

This document will be kept up-to-date with the latest releases of *Upscaler*. The document version number will be the same as the *Upscaler* version number. If any errors in this documentation are found or you have any further questions, please report them to briankirk@conifercomputing.com.