

Upscaler Offline Manual (PDF)

v1.0.0

Table of Contents

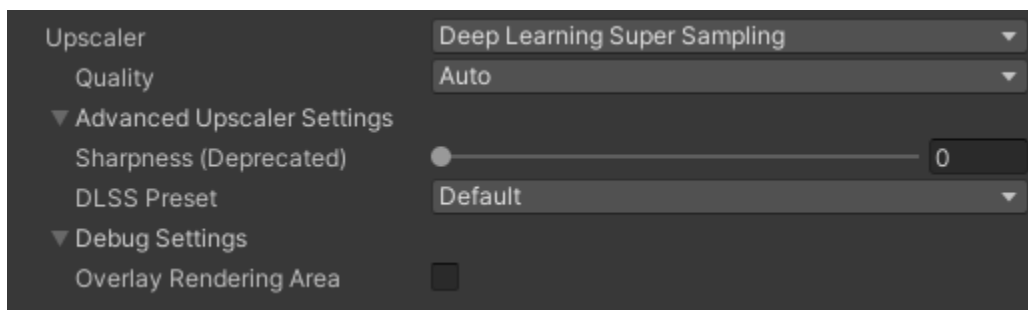
1 Getting Started.....	3
2 Using the Unity Editor GUI.....	3
2.1 Upscaler Settings.....	3
2.2 Basic Upscaler Settings.....	4
2.3 Debug Settings.....	5
3 URP Integration.....	5
4 BRP Integration.....	6
5 Jitter.....	6
6 Dynamic Resolution.....	6
7 Changing Settings.....	6
8 Handling Errors.....	7
9 Things to be Aware of.....	7
10 GUI Integration.....	8
11 Contact.....	9

1 Getting Started

After importing Upscaler be sure to restart Unity. This must be done so that Upscaler’s native plugin can load into memory. If you neglect to restart Unity at this point, expect Unity to crash as soon as you try to use Upscaler. Go to the Assets→Conifer→Upscaler→Scripts directory and drag and drop the **Upscaler.cs** script onto the camera that you want to upscale.

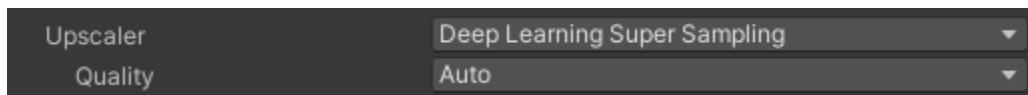
Consult [Using the Unity Editor GUI](#) for more information on how to control the upscaling from the Unity Editor. Consult [Changing Settings](#) to learn how to control the upscaling from within your scripts. Finally, check out [GUI Integration](#) in order to add upscaling controls to your game’s GUI.

2 Using the Unity Editor GUI



When correctly attached to the camera, Upscaler’s settings should appear as they do above in the camera’s inspector. This section provides detailed descriptions on the uses of each of the elements in this GUI.

2.1 Upscaler Settings



The “Upscaler” dropdown menu allows you to select one of the available upscalers. *Currently*, there are just two: “None” and “DLSS”. Selecting “None” will enable the dummy upscaler. This upscaler forces the game to render at the output resolution. It entirely removes Upscaler from Unity’s render pipeline. It hides all of the other settings; ultimately, it behaves as an off switch for Upscaler. Selecting “DLSS” 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. This setting will respect your selection in the “Quality” field.

The “Quality” dropdown menu allows you to select a quality mode at which to run DLSS. This setting is ignored if the “Upscaler” option is set to “None”. DLSS has five quality modes, four of which scale by a constant amount, and “Auto”, which chooses another quality mode as given by the output resolution. Consult the charts below for more information.

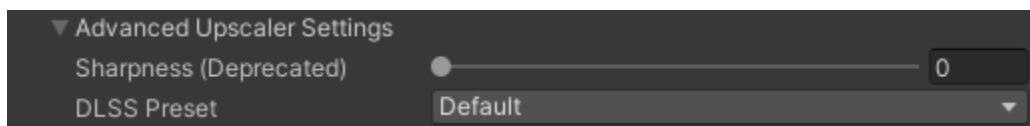
DLSS Quality Mode Effects on Render Resolution

Quality Mode	% less rendered pixels per-axis	% of total pixels rendered
DLAA	0%	100%
Quality	33.3%	44.4%
Balanced	42%	31%
Performance	50%	25%
Ultra Performance	66.67%	11.1%

DLSS “Auto” Quality Mode Choice

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

2.2 Basic Upscaler Settings



The “Sharpness” slider controls the amount of sharpening that DLSS will perform. Please note that this setting has been deprecated by NVIDIA. Even when DLSS is active and this slider is set to a non-zero value, it may be true that no sharpening effect is visible. NVIDIA has provided no replacement and has instead recommended rolling a custom sharpness solution. Conifer also recommends rolling a custom sharpness solution, as even when DLSS’ sharpening does work, it often produces very poor results.

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 ghosting.
- 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.

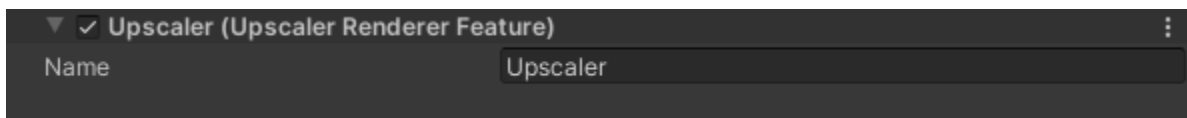
2.3 Debug Settings



The only setting available here allows you to visualize the screen area that is actually being rendered. This can be useful for visualizing Dynamic Resolution changes while testing.

3 URP Integration

While integrating with the URP, it is not only important to add the **Upscaler** script to the camera; 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 Editor GUI on the camera when an **UpscalerRendererFeature** is not present in that camera’s URP Renderer.



The solitary [ScriptableRenderPass](#) in the **UpscalerRendererFeature** is bound to the [RenderPassEvent.BeforeRenderingPostProcessing](#) event. After performing upscaling, it blits the depth buffer into a present resolution image, then binds that along with the upscaled results as the camera’s target. Some reflection was required here to ensure that other post processing passes that follow the upscale will use correctly sized images.

4 BRP Integration

The BRP is not supported as of version 1.0.0, but BRP integration is planned.

5 Jitter

Jitter values are queried from the C++ backend then added to the camera's projection matrix to offset the camera by that many pixels. When integrating Upscaler into your rendering system, be aware that [Camera.ResetProjectionMatrix](#) is called at this time. This happens so that the last jitter values are cleared from the projection matrix before we add new ones.

6 Dynamic Resolution

While Upscaler does support dynamic resolution, not all of DLSS' quality modes support it, namely Ultra Performance mode and DLAA mode. Attempting to enable dynamic resolution with one of these modes active will automatically disable dynamic resolution then log a message to the console explaining what happened.

Change the resolution using '[UniversalRenderPipelineAsset.renderScale](#)'. Modifying this value will automatically update the resolution that Upscaler upscales from. Note that while upscaling is active Upscaler will clamp this value between the allowed values for the current 'Quality' mode as queried from the C++ native library.

7 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>();
```

Use **Upscaler.QuerySettings** to get a copy of the settings currently in use by the upscaler object.

```
var settings = upscaler.QuerySettings();
```

Make your desired changes.

```
settings.upscaler = settings.upscaler == Settings.Upscaler.None ?
Settings.Upscaler.DLSS : Settings.Upscaler.None;
```

Use **Upscaler.ApplySettings** to push your changes to the active upscaler.

```
upscaler.ApplySettings(settings);
```

When calling this function, be aware that it can throw **Upscaler.Status** errors if something goes wrong. It is advisable to handle errors that this function returns in-place.

8 Handling Errors

Errors can occur when requesting a settings change. By default, when this happens the default error handler will reset the current upscaler 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](#). If you do not perform any settings changes then the active upscaler 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.

9 Things to be Aware of

The following is a list of advice both from Conifer and from NVIDIA as given in their [DLSS developer guide](#).

1. Disable all forms of anti-aliasing including but not limited to MSAA, TAA, FXAA, and SMAA.

2. Textures should use a default mip bias calculated using the formula below when being dynamically added to the scene. When using dynamic resolution, use the lowest possible resolution for the **renderingResolution**. When settings that require a new DLSS feature to be created are changed, the default mip bias handler will set all textures in the scene to have the correct mip bias. You are responsible for any dynamically added textures.

$$mipBias = \log_2\left(\frac{RenderingResolution.x}{OutputResolution.x}\right) - 1.0$$

3. Use DLSS 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 DLSS 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.

4. The Upscaler plugin is incompatible with RenderDoc. While this is unavoidable due to RenderDoc being incompatible with DLSS itself (as of RenderDoc v1.31 and likely long into the future), an alternative is available. Use NVIDIA's Nsight Graphics for graphics debugging. You can connect to Unity's Editor by launching it from Nsight Graphics. Upscaler will be able to load successfully, and will not interfere with any captures. Nsight Graphics is available for both Windows and Linux and was used to help create this plugin.
5. Read the [DLSS developer guide](#) for more information.

10 GUI Integration

Consult the [DLSS UI Developer Guidelines PDF](#) for more information about integration with in-game GUIs.

Do note that in contrast to what is suggested in the above guidelines PDF, the 'Auto' setting does not disable DLSS 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 DLSS off, they turn off DLSS. If you choose to follow the guidelines PDF more closely you will need to perform the required resolution check yourself.

11 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.