DLSS Upscaling for Unity Documentation

About DLSS

DLSS is an upscaling technique, creating high quality and resolution frames based on lower resolution input. By using this, projects could have drastically lower GPU requirements than without.

Only if your project is limited by GPU performance, DLSS will gain you a higher framerate. If a project is limited by CPU performance, all it will do is make the GPU workload lower. While this may seem like a big limitation, it also means a laptop will use way less battery power when using DLSS!

Current supported Unity Render Pipelines:

Built-in Render Pipeline (BIRP) and Universal Render Pipeline (URP).
High-Definition Render Pipeline (HDRP) already has a native DLSS version.

Current supported platforms*:

- Windows x64 (DX11, DX12, Vulkan)

**DLSS only works on NVIDIA RTX cards.*

Quick Start

This chapter is written to add DLSS as fast as possible to your project. However, it is very much recommended to read the Important Information chapter on current DLSS limitations.

Quick Start: BIRP

Step 1: Import the DLSS Package in your project.

Step 2: Import our custom Post-Processing package here and place a Post-process Layer on your main camera then enable DLSS in the Anti-Aliasing settings on the Post-Process Layer.

Hit play!

Quick Start: URP

Step 1: Import the DLSS Package in your project.

Step 2: Add DLSS_URP.cs script to your main camera.

Step 3: Add the “DLSS Scriptable Render Feature” to your Universal Renderer Data files. (Add it to all the URP data files you will use in your project, for example if you use different ones per quality setting).

Hit play!

Important Information

Multiple Cameras

DLSS Upscaling for Unity doesn't currently support multiple cameras using DLSS, it may even make Unity crash if you try. Generally it is best practice to limit the use of multiple cameras at the same time due to degrading performance.

Temporal Anti-Aliasing (TAA)

DLSS has a very good built-in AA solution. This internal AA is not optional. There is no need to add any other AA to the camera, since this will only add more blur and ghosting. Adding different types of Anti-Aliasing will also decrease the upscaling quality.

Mipmaps - BIRP Only

When DLSS downscales the source rendering, it is recommended to add a negative mipmap bias to all textures in your scene for improved quality. Textures will look very washed out otherwise. In the Inspector chapter, we explain how we created a way to update all mipmap biases on the textures automatically.

Post Processing Effects (BIRP)

For every upscaling technique, is it very important to know how to use Post Processing effects. Some effects will need to be added before DLSS, others afterwards. Check out our demo's for examples.

To make this happen in the most efficient way we have edited a version of Unity's Post Processing 3.2.3, which you can download here, that fully inserts DLSS to the Post Processing layer allowing it to support DLSS as is intended.

Demo Scenes

With the Quality.cs script you can toggle between quality modes by pressing the spacebar.

Download Demo Projects here

Inspector

Quality
Quality: 1.5x scaling
Balanced: 1.7x scaling
Performance: 2.0x scaling
Ultra Performance: 3.0x scaling

Example: If the native resolution is 1920x1080, selecting the **Performance** option will change the rendering resolution to 960x540, which will then be upscaled back to 1920x1080. Changing the quality setting will update the “m_scaleFactor” variable.

FallBack - BIRP only
The current Anti-Aliasing will be changed to the fallback option when DLSS is not supported.

Anti-Ghosting
0 - 1

All Temporal Anti-Aliasing solutions add some ghosting, this can be minimised by using the Anti-Ghosting slider.

Auto Texture Update - BIRP only
Off - On

As previously explained, it is recommended to update the MipMap Bias of all used textures. In Unity, the only way to do this is by script, texture by texture. This is less than ideal, so we added a feature to automatically update all textures currently loaded in memory. In real-world projects, we saw no noticeably extra CPU cost.
Note: It seems URP and HDRP already automatically do mipmap biassing, so here we disabled this feature by default.

Mip Map Update Frequency
0.0 - Infinite

This settings determines how often the Auto Texture Update checks for new loaded textures to update the Mipmap Bias for.

Mipmap Bias Override - BIRP only
0.0 - 1.0

When using DLSS, and changing the MipMap bias, it is possible that there will be additional texture flickering. If you notice too much texture flickering, try lowering this setting until you have the desired balance between quality and texture stability. If you have no texture flickering, keep this to 1 for best quality.

Public API

When using the DLSS_URP.cs camera component, you can call the following API functions on those components. When using the custom PostProcessing package for BIRP, you can change the values of the Post-processing Layer just like you'd normally would when changing settings on it.

Generic

public bool OnIsSupported()
Use this function to check if DLSS is supported on the current hardware. It's recommended to use this function, before enabling DLSS .

public bool OnSetQuality(DLSSQuality value)
Use this function to change the DLSS quality settings.

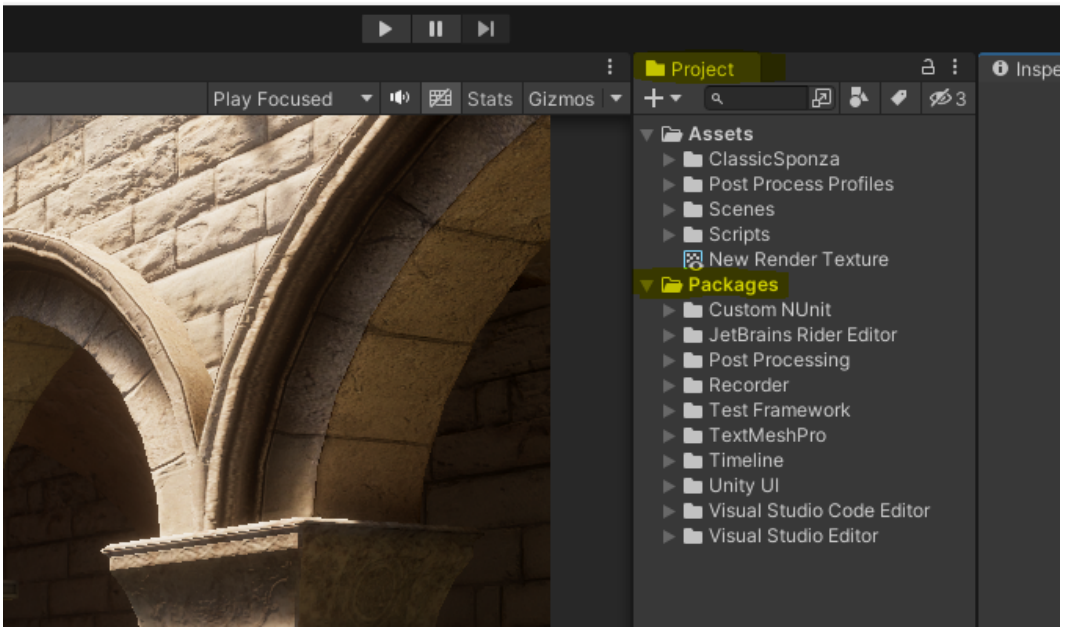
Editing Unity Post Processing package

Download

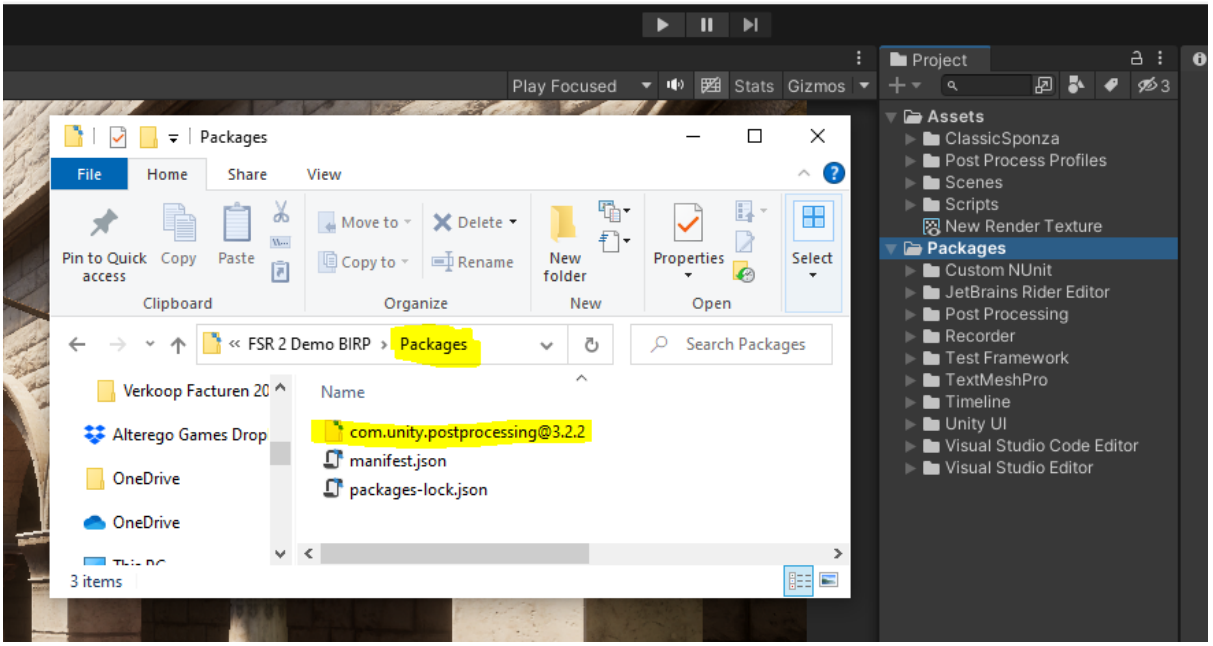
First of all, download our edited Unity Post Processing package:
Download Link or add https://github.com/DominicdeGraaf/Unity-Post-Processing-Stack.git to the Unity Package Manager

Use this edited package only when you're planning to use BIRP and want to be able to use Unity's Post Processing Stack V2.

Step 1: Locate the Packages folder in your project, press Right-Mouse Button on it and select "Show in Explorer"



Step 2: Unzip the downloaded package into the Packages folder



All done, now you should be able to use the single camera setup in BIRP.

FAQ

Q: Does DLSS offer free performance?
A: Almost, in many cases it does. However DLSSdoes use a small bit of GPU performance, so when using Quality mode the gains can be little. However if your project is CPU bound, you will not likely see much performance gains, just a lower GPU usage.

Q: Will DLSS work for every kind of project?
A: No, in projects that are CPU bound, DLSS will probably only hinder performance. However because DLSS includes an industry leading AAA quality Anti-Aliasing, your project's fidelity might improve over Unity's standard anti-aliasing.

Q: DLSS is not working or I am having an issue, help!
A: If you encounter any issues, you can contact us by emailing to info@thenakeddev.com, joining our discord in the "Unity Tools" channel or on the Unity Forum.

Q: Can I use older/newer DLSS versions?
A: Yes you can! The nvngx_dlss.dll file located in the editor installation can be replaced with any DLSS version downloadable from here:
https://www.techpowerup.com/download/nvidia-dlss-dll/

Q: Where is the Frame Generation?
A: I'm looking into the possibilities of adding DLSS 3 Frame Generation feature into Unity.

Known Issues & Limitations

General

- Currently multiple cameras are not supported

BIRP

- To make Unity's Post Processing Stack work correctly we need source changes which can be downloaded here.

URP

- Ideally most Post Processing is rendered before DLSS, due to the nature of each different URP version, this is not possible as of yet. In future updates this might be added for better quality Post Processing.

Support

If you encounter any issues, you can contact us by emailing to info@thenakeddev.com, joining our discord in the "Unity Tools" channel or on the Unity Forum.