FSR 3 Upscaling for Unity Documentation

Current version: FSR 3.1.0

About FSR 3

This Unity asset is created to interface with the open-source FSR 3 found here: https://github.com/GPUOpen-LibrariesAndSDKs/FidelityFX-SDK/tree/v1.1.0

FSR 3 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, FSR 3 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 FSR 3!
Note: FSR 3 works on all brands of graphics cards!

Current supported Unity Render Pipelines:

Built-in (BIRP), Universal Render Pipeline (URP) and High-Definition Render Pipeline (HDRP).

Current supported platforms*:

- Windows (DX11, DX12, Vulkan)
- Linux (Vulkan)
- MacOS (Metal)
- iOS (Metal)
- Android (Vulkan)
- Playstation 4
- Playstation 5
- Xbox One
- Xbox Series X|S
- Nintendo Switch
- PCVR (BIRP, URP)
- Standalone VR (BIRP, URP)

** For older and/or mobile platforms, the GPU rendering cost of FSR 3 can be higher than the gains, in our tests often FSR 3 did improve performance, but each project is different and can yield different results.*

Working on:

- PCVR (HDRP)

The Naked Dev Tools

Upscaling Tools

Each and every upscaler out there has different strong and weak spots, some require specific hardware, some are specifically made for slow or older devices.

Here is the list of all our other upscalers for Unity:

FSR 3 - Upscaling for Unity

FSR 3 is supported on almost every platform and most hardware! Making it the number #1 goto upscaler. However compared with DLSS or XeSS the visual fidelity of FSR 3 is considered to be slightly lower.

XeSS - Upscaling for Unity

XeSS is limited to Windows x64, DX11 and DX12, but works just as well on Intel, AMD and Nvidia GPUs! XeSS's visual fidelity is higher than FSR 3 and in some ways even better than DLSS.

DLSS - Upscaling for Unity

DLSS is limited to Windows x64, DX11 and DX12, and will only work on Nvidia RTX GPUs (20x0 series and up). DLSS's visual fidelity is higher than FSR 3 and in some ways better than XeSS.

SGSR - Upscaling for Unity

SGSR is supported on almost every platform and most hardware! However SGSR's visual fidelity is a lot lower than the other upscalers. But it's also way faster than all other upscalers, making it perfect to use on platforms with a high DPI like mobile devices!

Fallback Setup

For the absolute best visual upscaling results we recommend using the following fallback format:

1. DLSS
2. XeSS
3. FSR 3
4. SGSR
5. FSR 1

Various Tools

CACAO - Ambient Occlusion for Unity

CACAO is an ambient occlusion technique, created for AAA games, to produce the best possible ambient occlusion visuals while also offering the best performance possible.

About FSR 3 **1**
Current supported Unity Render Pipelines: 1
Current supported platforms*: 1
Working on: 1
The Naked Dev Tools **2**
Upscaling Tools 2
FSR 3 - Upscaling for Unity 2
XeSS - Upscaling for Unity 2
DLSS - Upscaling for Unity 2
SGSR - Upscaling for Unity 2
Fallback Setup 2
Various Tools 2
CACAO - Ambient Occlusion for Unity 2
Quick Start **5**
Quick Start: BIRP 6
Quick Start: URP 7
Quick Start: HDRP 8
Demo Scenes **10**
Important Information **11**
Multiple (stacking) Camera's 11
BIRP 11
URP 11
HDRP 11
Temporal Anti-Aliasing (TAA) 11
Alpha-Blended / Transparency 11
Camera Transform 12
Mipmaps - BIRP Only 12
Motion Vectors 12
Examples 12
Post Processing Effects (BIRP) 13
FSR 3 GPU Rendering time 14
FSR 1 BIRP fallback 14
Inspector **15**
Quality 15
FallBack - BIRP only 15
Sharpening 15
Sharpness 15
Generate Reactive Mask 15
Reactive Scale 15
Reactive Threshold 15
Reactive Binary Value 16
Auto Texture Update - BIRP only 16
Mip Map Update Frequency 16

Mipmap Bias Override - BIRP only 16
Public API **17**
Generic 17
public bool OnIsSupported() 17
public void OnResetCamera() 17
BIRP Only **17**
public void OnMipmapSingleTexture(Texture texture) 17
public void OnMipMapAllTextures() 17
public void OnResetAllMipMaps() 17
BIRP Custom Post Processing Layer 17
Editing Unity Post Processing package **19**
Download 19
Editing Render Pipeline source files **20**
HDRP source files **22**
HDCamera.cs 22
HDRenderPipeline.PostProcess 24
VR **25**
Current Limitations: 25
FAQ **26**
Known Issues & Limitations **28**
General 28
BIRP 28
URP 28
HDRP 28
Uninstall **29**
Support **29**
Special Thanks **29**

Quick Start

This chapter is written to add FSR 3 as fast as possible to your project. However, it is very much recommended to read the Important Information chapter. FSR 3 upscales very well when used properly, but it will take some tweaking to get the best quality possible for your project specifics.

Goto: Quick Start Built-in Render Pipeline
Goto: Quick Start Universal Render Pipeline
Goto: Quick Start High-Definition Render Pipeline

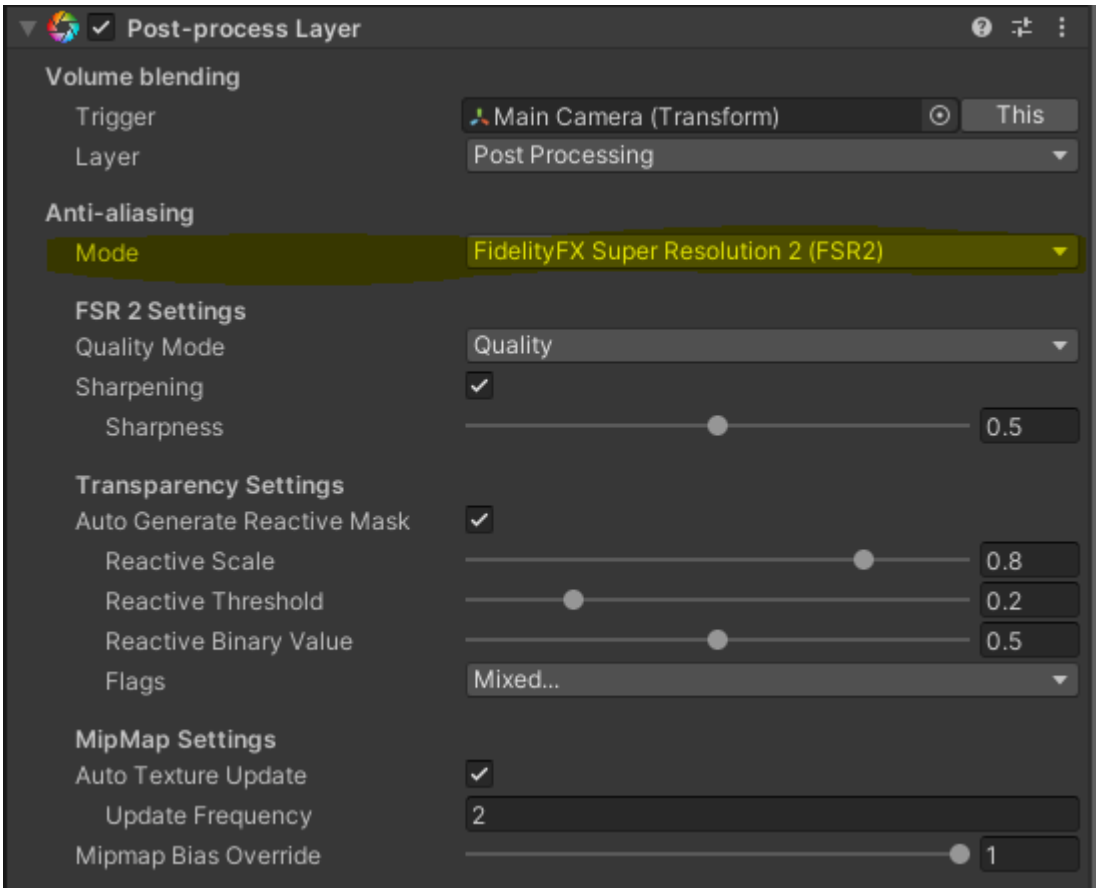
Quick Start: BIRP

Step 1: Import the FSR 3 Package in your project.

Now there are two options:
NOTE: Don't use both options at the same time!

Option 1: Add FSR3_BIRP.cs script to your main camera.

Option 2: Import our custom Post-Processing package here and place a Post-process Layer on your main camera then enable FSR 3 in the Anti-Aliasing settings on the Post-Process Layer.



Hit play!

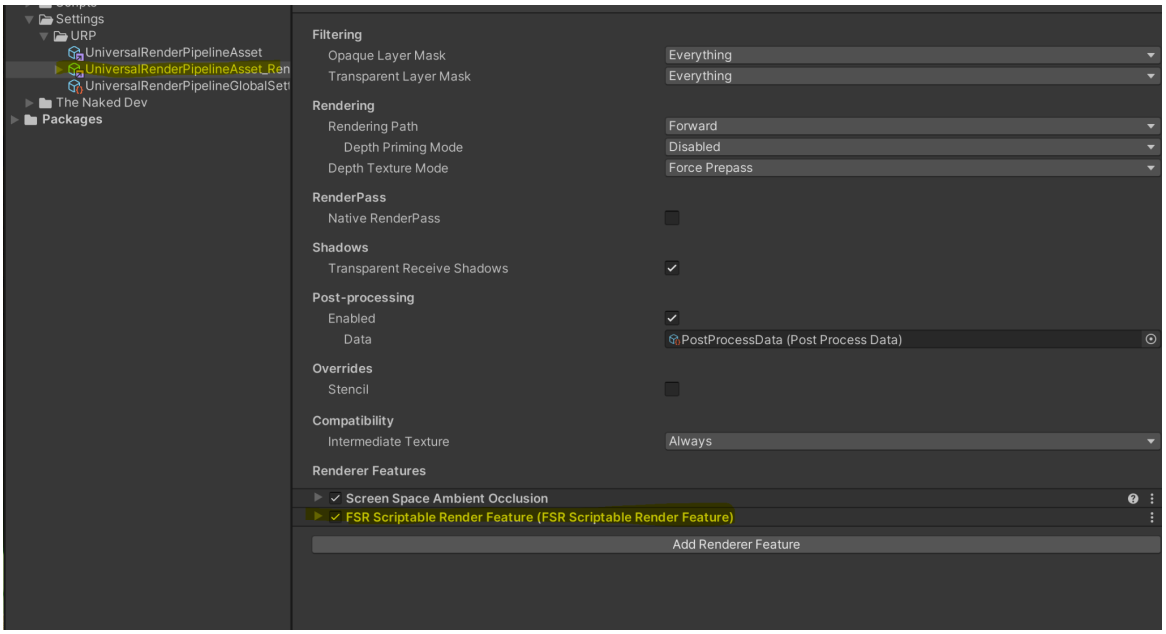
Note: Read more about using **Unity Post Processing**. Read chapter Post-Processing or check out the Demo's for more information. **Option 2** is the most recommended for 3rd party asset support and visual fidelity.

Quick Start: URP

Step 1: Import the FSR 3 Package in your project.

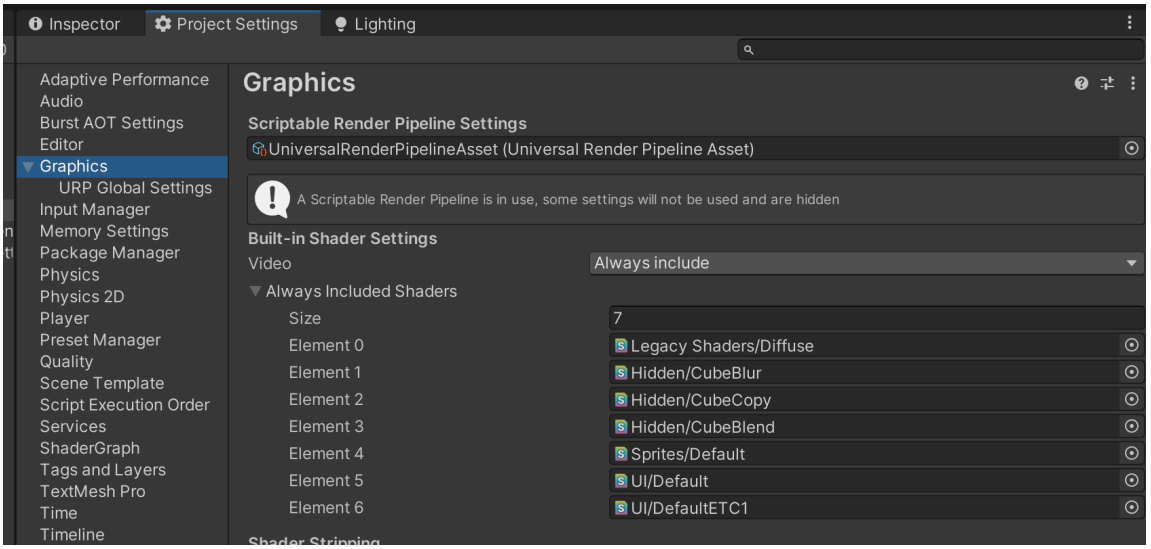
Step 2: Add FSR3_URP.cs script to your main camera.

Step 3: Add the “FSR 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!

NOTE: If you can’t add the FSR3_URP component to your Main Camera, make sure you have a Scriptable Render File in the Scriptable Render Pipeline Settings, FSR 3 uses this to check which render pipeline is active

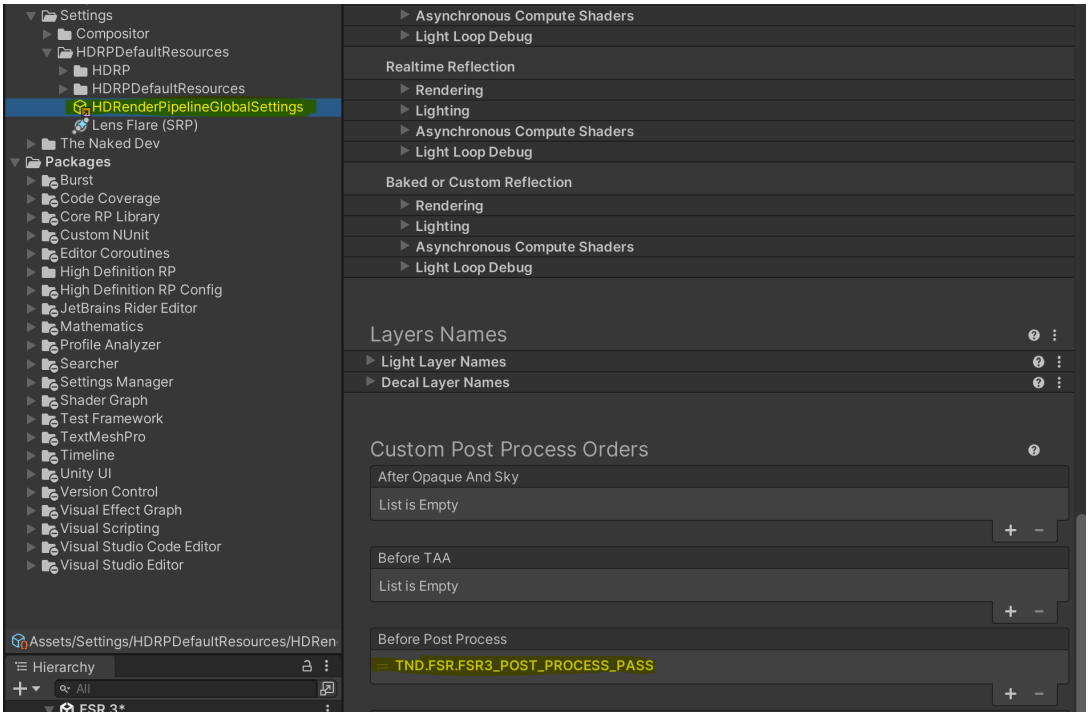


Quick Start: HDRP

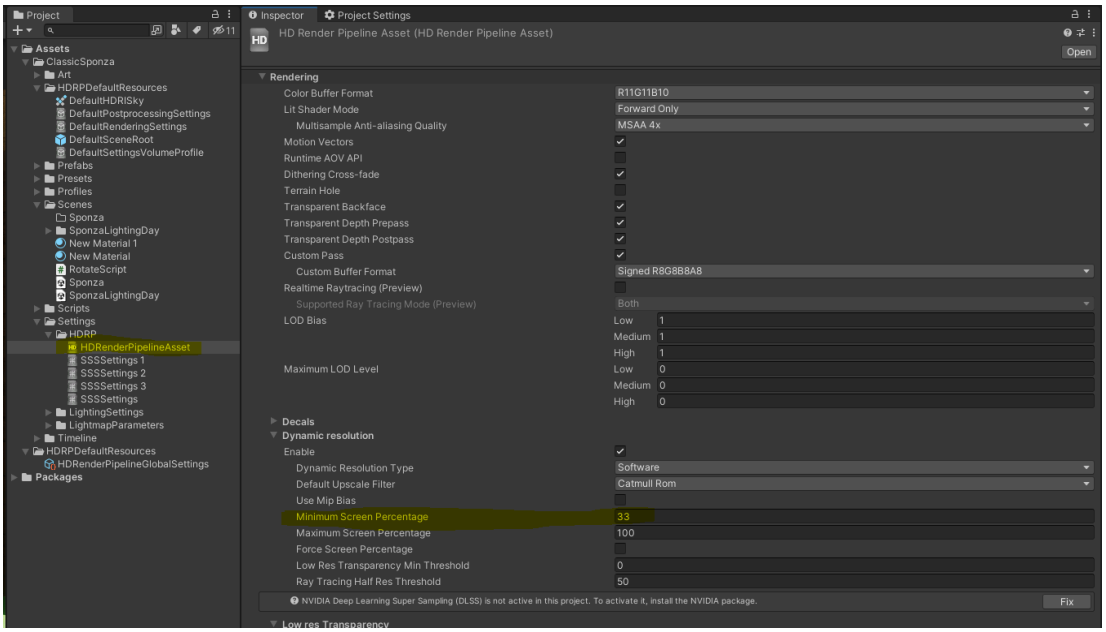
Step 1: Import the FSR 3 Package in your project.

Step 2: Edit the HDRP source files, see chapter HDRP source files

Step 3: Add the TND_FSR_FSR3RenderPass to the HDRenderPipelineGlobalSettings in “Before Post Process”.



Step 4: Enable Dynamic Resolution in HDRP settings and set the “Minimum Screen Percentage to 33. Keep the Default Upscale Filter to Catmull Rom. NOTE: if you use multiple RenderPipeline Asset files for different Quality settings, make sure you edit them all!



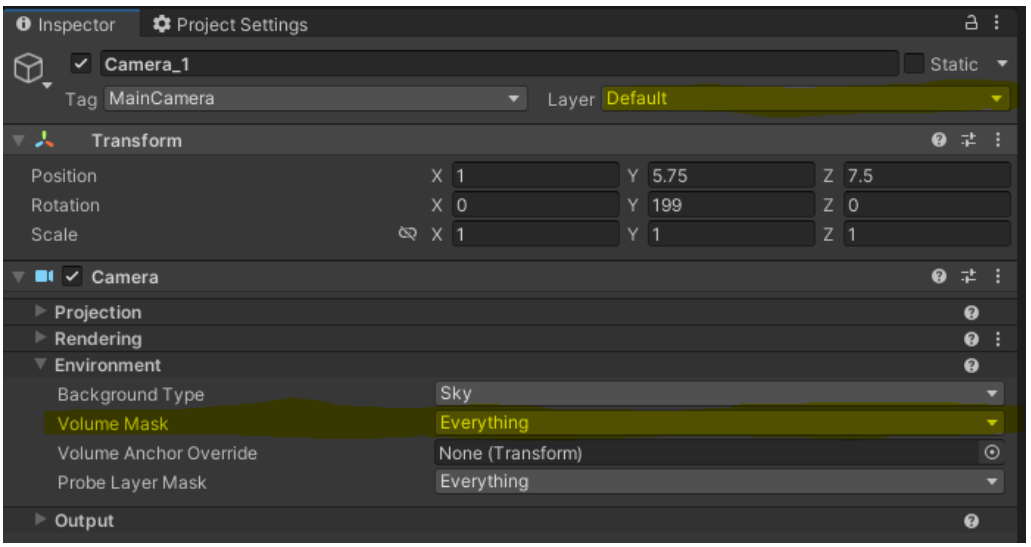
Step 5: Add FSR3_HDRP.cs script to your main camera.

Step 6: Press the “I have edited the source files” button.
Note: If you get compilation errors after you press the button, you have not properly edited the source files.

Hit play!

The next steps are optional, because FSR will do these steps for you if you skip them.

Optional Step 6: Make sure the “Volume Mask” of your camera includes the layer of your camera. If you skip this step, FSR will automatically do this for you.



Optional Step 7: Enable “Allow Dynamic Resolution” on the Camera. If you skip this step, FSR will automatically do this for you.

Demo Scenes

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

Download Demo Projects here

Note, the HDRP demo has been created in Unity 2021.3.33 LTS, and HDRP 12, the demo project might break on older or newer Unity versions because of the changes in the HDRP source.

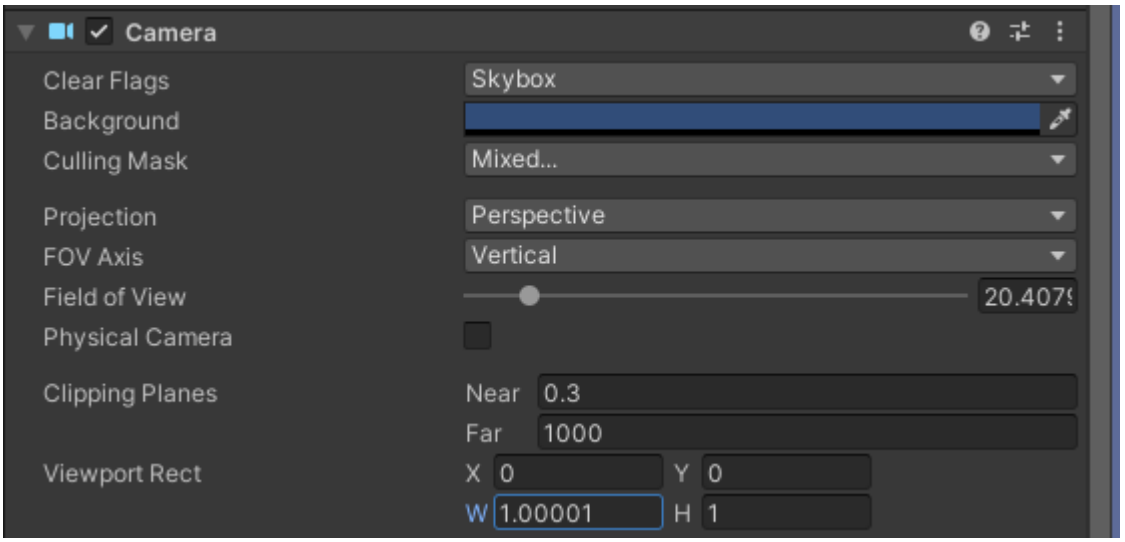
Important Information

Multiple (stacking) Camera's

BIRP

Multiple camera's is only supported when using the custom Post-Processing package.

Note: Unity "links" multiple cameras automatically, which is bad because the other cameras will also downscale, making everything look blurry. To prevent this, make sure the "Viewport Rect" values of all cameras are slightly different.



URP

Using multiple Base camera's is not supported, stacking camera's is supported.

To use Camera Stacks, simply drag the FSR 3 component on the Base camera. The overlay camera's will automatically be set up correctly.
Do not add the FSR component to the overlay cameras.
For more information about stacking camera's please read Unity's documentation.

HDRP

As Unity recommends to not use more than one camera, camera stacking and/or using multiple cameras is therefore not yet supported.

Temporal Anti-Aliasing (TAA)

FSR 3 has a very good built-in TAA solution. This is required for FSR 3 to do its work. There is no need to add any other TAA to the camera, since this will only add more blur and ghosting. Adding different types of Anti-Aliasing will also decrease the upscaling quality.

Alpha-Blended / Transparency

In Unity, alpha-blended and transparent objects are often not added to Unity's motion vector pass. For this FSR 3 has a feature called a Reactive Mask, by default enabled in the inspector. Enabling this setting will drastically improve Alpha-Blended / Transparent objects, without the need to add them to the motion-vector buffer.

Camera Transform

Since FSR 3 uses multiple frames to upscale, this can cause rendering artefacts when you have very fast camera transform changes. To improve this, call OnResetCamera on the same frame you change your camera transform, to let FSR 3 know it needs to drop previous rendered frames.

Mipmaps - BIRP Only

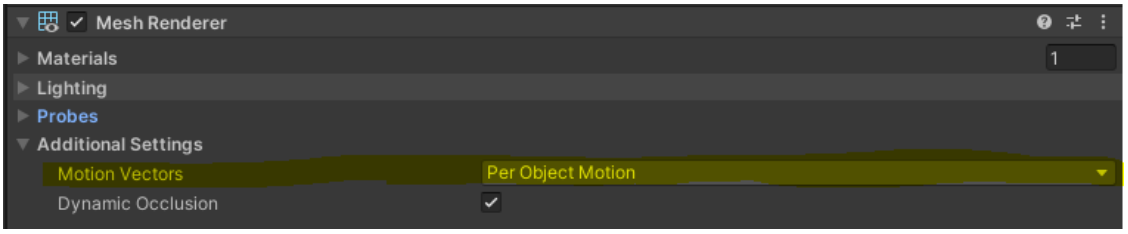
When FSR 3 downscales the source rendering, it is recommended to add a negative mipmap bias to all textures in your scene for improved quality. Textures will look very washed out otherwise. In the Inspector chapter, we explain how we created a way to update all mipmap biases on the textures automatically.

Motion Vectors

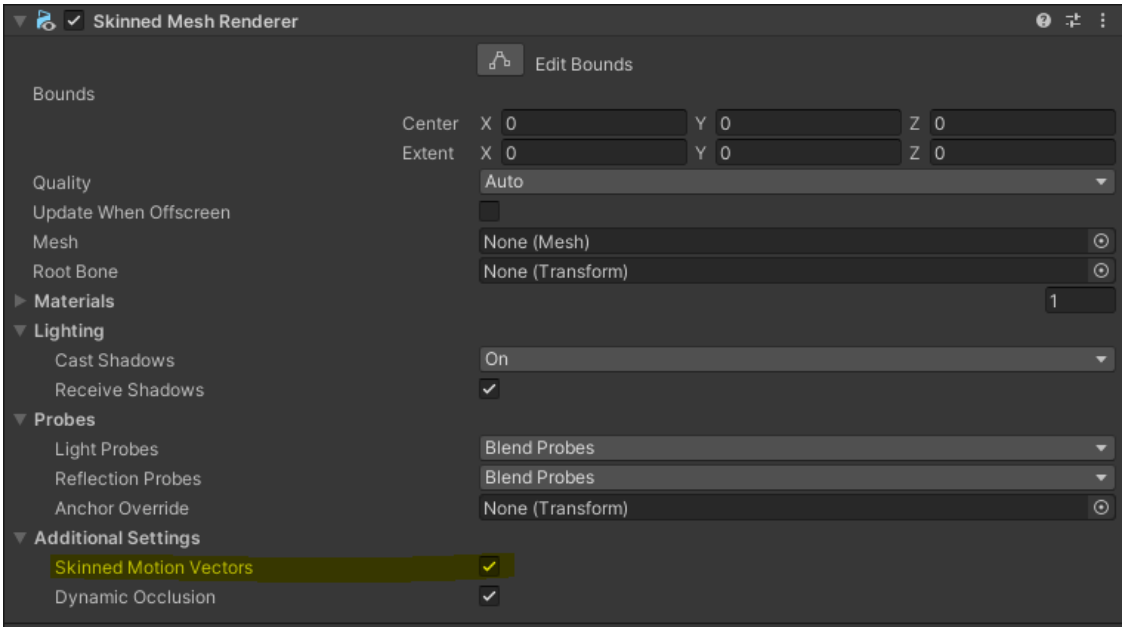
FSR 3 uses the motion-vector buffer to calculate how much an object moves, to improve upscaling. If you want opaque objects to be upscaled in FSR 3, always enable Motion Vectors.

Examples

Mesh Renderer

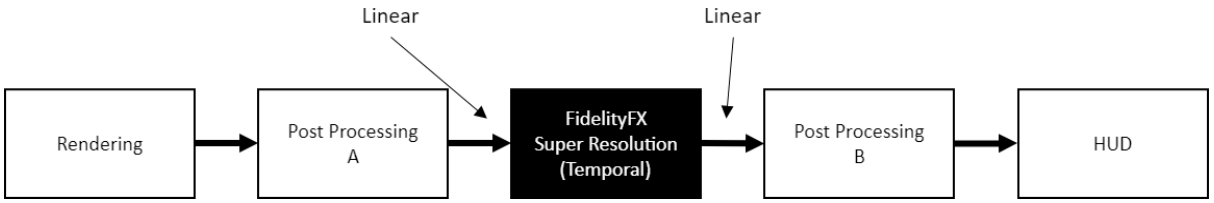


Skinned Mesh Renderer



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 FSR 3, others afterwards. Check out our demo's for examples.

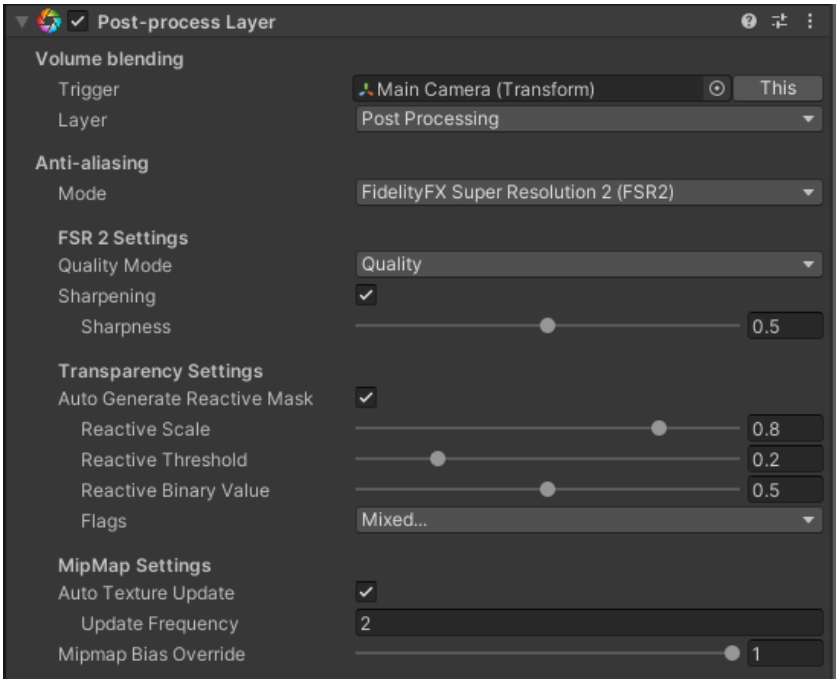


Post Processing order suggestions by AMD:
https://github.com/GPUOpen-Effects/FidelityFX-FSR2#placement-in-the-frame

In BIRP this gives us a bit of an issue.

The camera renders the scene at a lower resolution, which gets upscaled by FSR, we then order the camera to resize back to its original resolution and then we blit the upscaled image back. This is the most performant way, however it will break many Post Processing effects because to make this work, we need to change the CameraEvent from BeforeImageEffects to AfterForwardAlpha.

For this, we have edited a version of Unity's Post Processing 3.4.1, which you can download here, that fully inserts FSR 3 to the Post Processing layer allowing it to fully support FSR 3 the way AMD suggest.



FSR 3 GPU Rendering time

Target resolution	Quality	RX 7900 XTX	RX 6950 XT	RX 6900 XT	RX 6800 XT	RX 6800	RX 6700 XT	RX 6650 XT	RX 5700 XT	RX Vega 56	RX 590
3840×2160	Quality (1.5x)	0.7ms	1.1ms	1.2ms	1.2ms	1.4ms	2.0ms	2.8ms	2.4ms	4.9ms	5.4ms
	Balanced (1.7x)	0.6ms	1.0ms	1.0ms	1.1ms	1.4ms	1.8ms	2.6ms	2.2ms	4.1ms	4.9ms
	Performance (2x)	0.6ms	0.9ms	1.0ms	1.0ms	1.3ms	1.7ms	2.3ms	2.0ms	3.6ms	4.4ms
	Ultra perf. (3x)	0.5ms	0.8ms	0.8ms	0.9ms	1.1ms	1.5ms	1.8ms	1.7ms	2.9ms	3.7ms
2560×1440	Quality (1.5x)	0.3ms	0.5ms	0.5ms	0.5ms	0.7ms	0.9ms	1.2ms	1.1ms	1.9ms	2.3ms
	Balanced (1.7x)	0.3ms	0.5ms	0.5ms	0.5ms	0.6ms	0.8ms	1.1ms	1.0ms	1.7ms	2.1ms
	Performance (2x)	0.3ms	0.4ms	0.4ms	0.4ms	0.6ms	0.8ms	0.9ms	0.9ms	1.5ms	1.9ms
	Ultra perf. (3x)	0.2ms	0.4ms	0.4ms	0.4ms	0.5ms	0.7ms	0.8ms	0.8ms	1.2ms	1.7ms
1920×1080	Quality (1.5x)	0.2ms	0.3ms	0.3ms	0.3ms	0.4ms	0.5ms	0.6ms	0.6ms	1.0ms	1.3ms
	Balanced (1.7x)	0.2ms	0.3ms	0.3ms	0.3ms	0.4ms	0.5ms	0.6ms	0.6ms	0.9ms	1.2ms
	Performance (2x)	0.2ms	0.2ms	0.2ms	0.3ms	0.3ms	0.5ms	0.5ms	0.5ms	0.8ms	1.1ms
	Ultra perf. (3x)	0.1ms	0.2ms	0.2ms	0.2ms	0.3ms	0.4ms	0.4ms	0.4ms	0.7ms	0.9ms

This image shows the GPU rendering times that FSR 3 costs each frame to work. Other platforms, such as mobiles and/or older consoles the GPU rendering time is higher.

FSR 1 BIRP fallback

Sometimes FSR 3 is too performance heavy for older or mobile devices. Hence we added official support for FSR 1 for BIRP (As URP and HDRP already support FSR 1 natively).

Inspector

Quality

Off: Disables FSR 3.
Native AA: 1.0x scaling (Native, AA only!)
Ultra Quality: 1.2x scaling
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 “FSRScaleFactor” variable.

FallBack - BIRP only

The current Anti-Aliasing will be changed to the fallback option when FSR 3 is not supported.

Sharpening

Off - On

All Temporal Anti-Aliasing solutions add some blur, to prevent this, FSR 3 features a sharpening filter. With these settings you can enable/disable this filter.

Sharpness

0.0 - 1.0

With this slider you can change the amount of sharpening.

Generate Reactive Mask

Off - On

Unless you do not use any Transparent or Alpha Blended objects in your project, we recommend keeping this enabled. Otherwise, disabling it will free up some additional memory.

Reactive Scale

0.0 - 1.0

Larger values result in more reactive pixels. Recommended default value is 0.9f

Reactive Threshold

0 - 1.0

Smaller values will increase stability at hard edges of translucent objects. Recommended default value is 0.1f.

Reactive Binary Value
0 - 1.0

Larger values result in more reactive pixels. Recommended default value is 0.5f

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 FSR 3, 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 FSR_URP.cs, FSR_HDRP or FSR_BIRP.cs camera components, 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 FSR 3 is supported on the current hardware. Its recommended to use this function, before enabling FSR 3.

public void OnResetCamera()
Resets the camera for the next frame, clearing all the buffers saved from previous frames in order to prevent artefacts.
Should be called in or before PreRender oh the frame where the camera makes a jumpcut.
It is automatically disabled the frame after.

BIRP Only

public void OnMipmapSingleTexture(Texture texture)
Updates a single texture to the set MipMap Bias.
Should be called when an object is instantiated, or when the ScaleFactor is changed.
Use this function if you are not making use of the Auto Texture Update feature.

public void OnMipMapAllTextures()
Updates all textures currently loaded to the set MipMap Bias.
Should be called when a lot of new textures are loaded, or when the ScaleFactor is Changed.
Use this function if you are not making use of the Auto Texture Update feature.

public void OnResetAllMipMaps()
Resets all currently loaded textures to the default MipMap Bias.

BIRP Custom Post Processing Layer

To change any of the settings of the Post-process Layer you'll need a reference to it, afterwards you can address the upscaler settings like this:

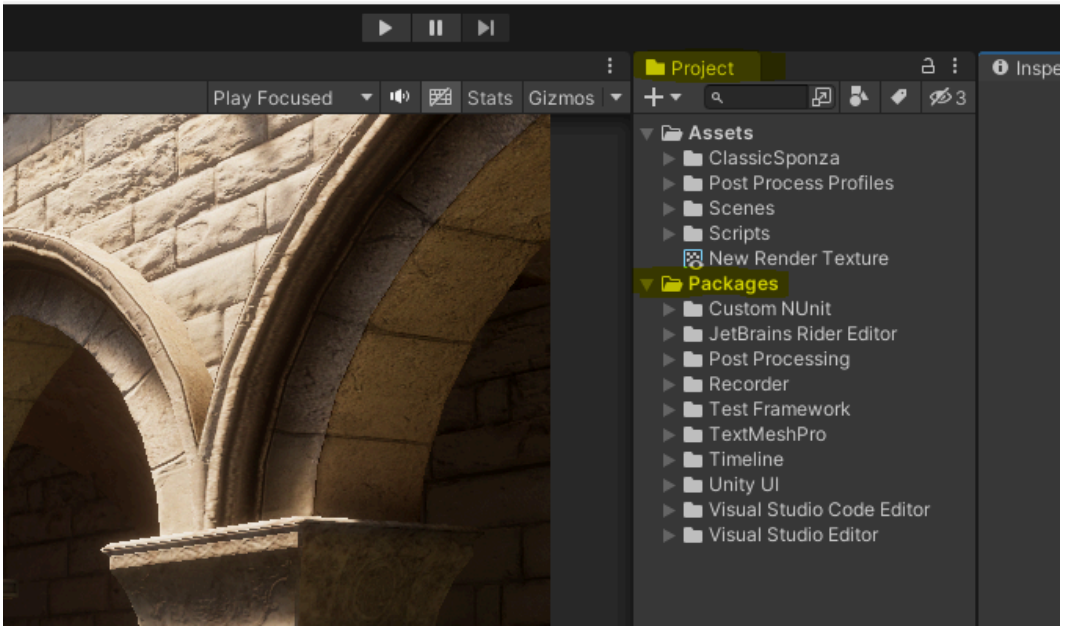
```
using TND.FSR;

public class ChangeQuality : MonoBehaviour
{
    void Start()
    {
        _postProcessingLayer.fsr3.qualityMode = FSR3_Quality.Quality;
    }
}
```

Editing Unity Post Processing package

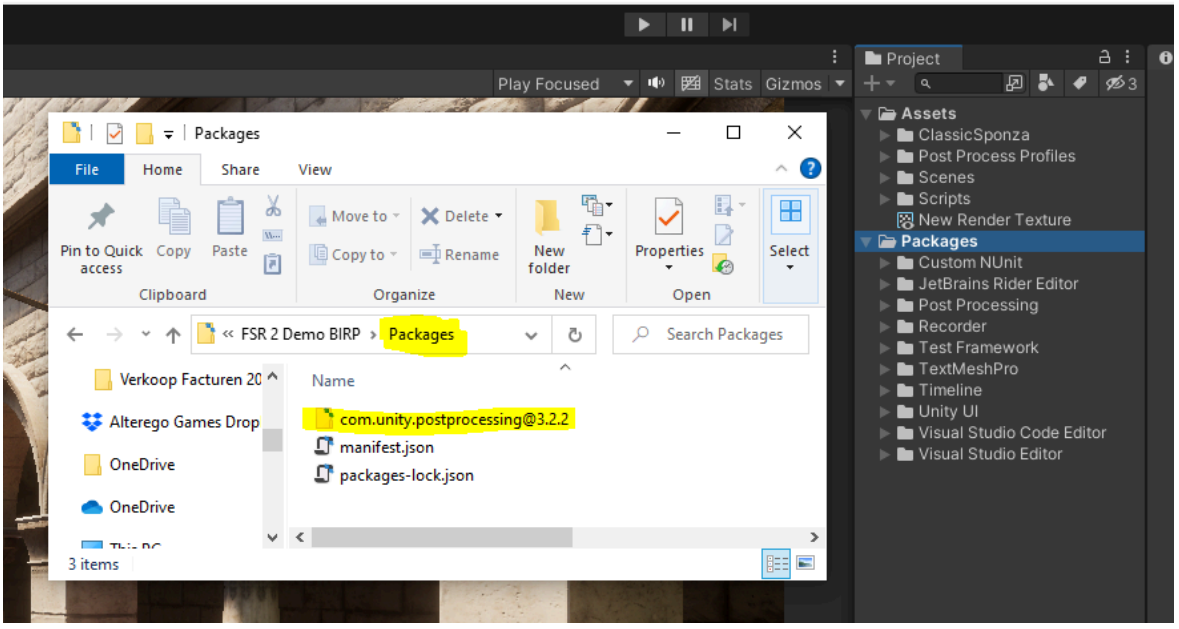
Download

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"



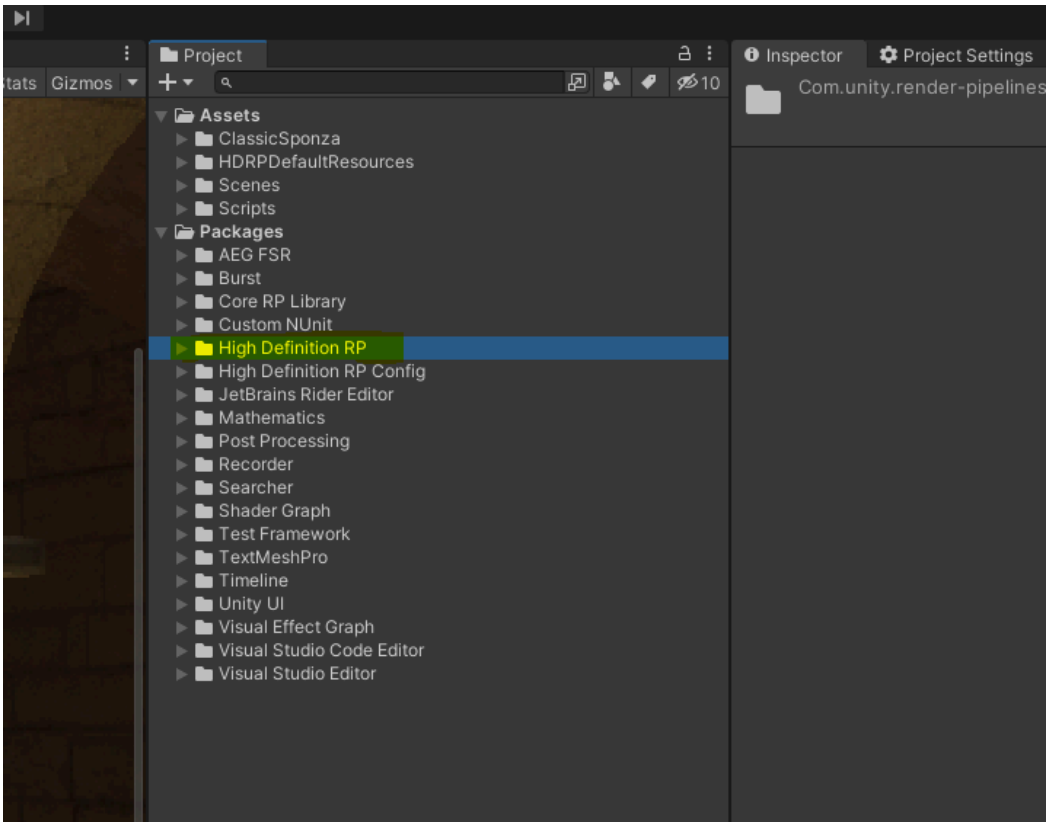
Step 2: Unzip the downloaded package into the Packages folder



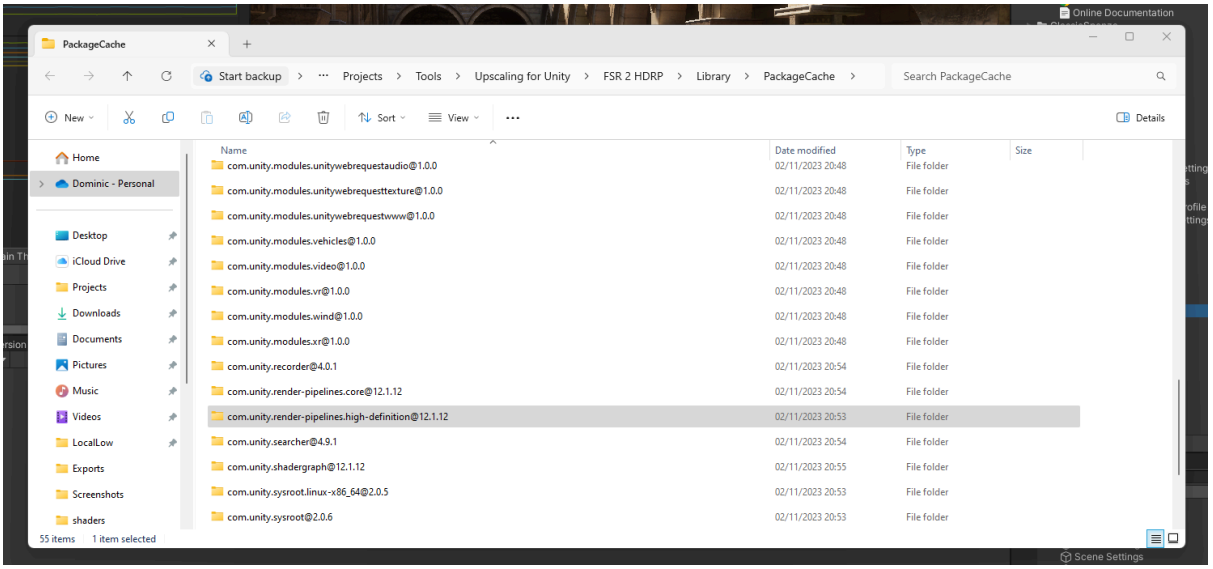
All done, now you should be able to use the single camera setup in BIRP.

Editing Render Pipeline source files

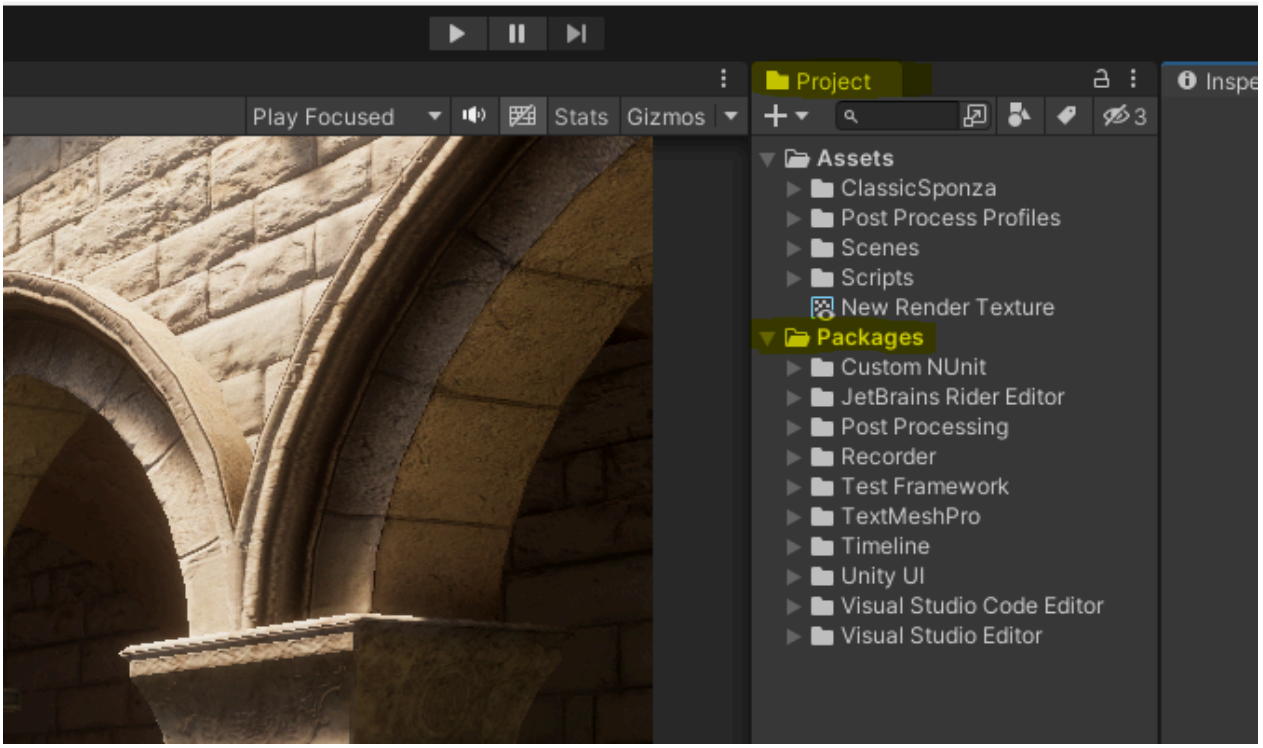
Step 1: Locate the Packages folder in your project, unfold it and press Right-Mouse Button on the “High Definition RP” and select “Show in Explorer”



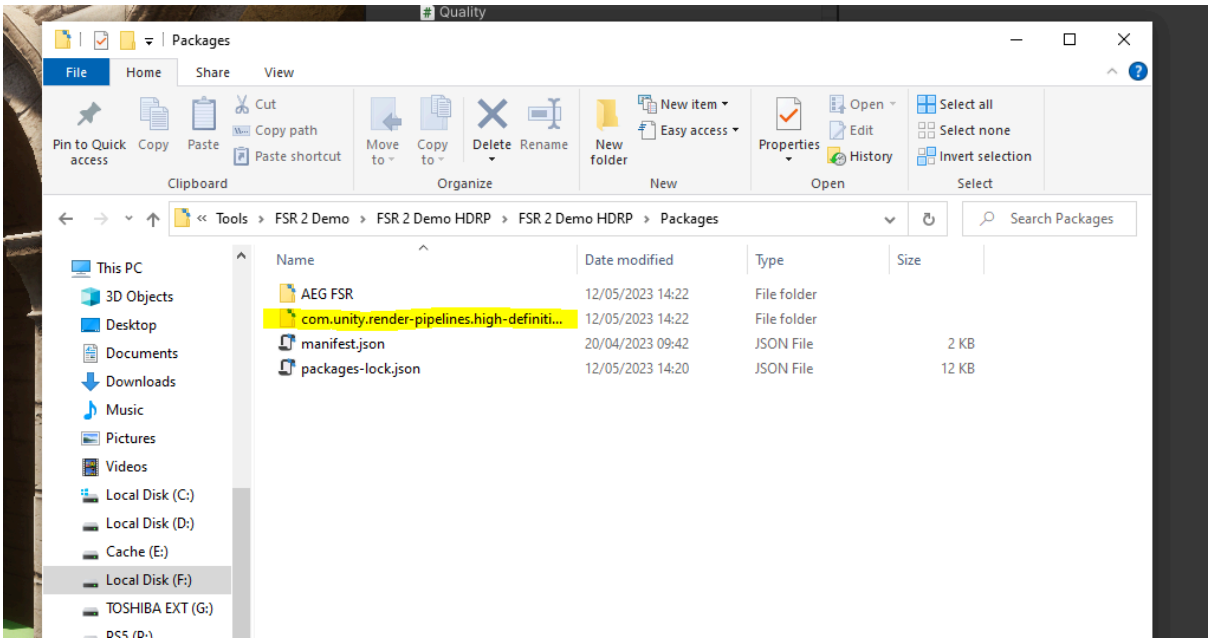
Step 2: Copy the com.unity.render-pipelines.high-definition@xx.x.xx



Step 3: Locate the Packages folder in your project, press Right-Mouse Button on it and select “Show in Explorer”



Step 4: And paste the previously copied folder into this Packages folder!



Now the source files of the High-Definition Render Pipeline are editable!

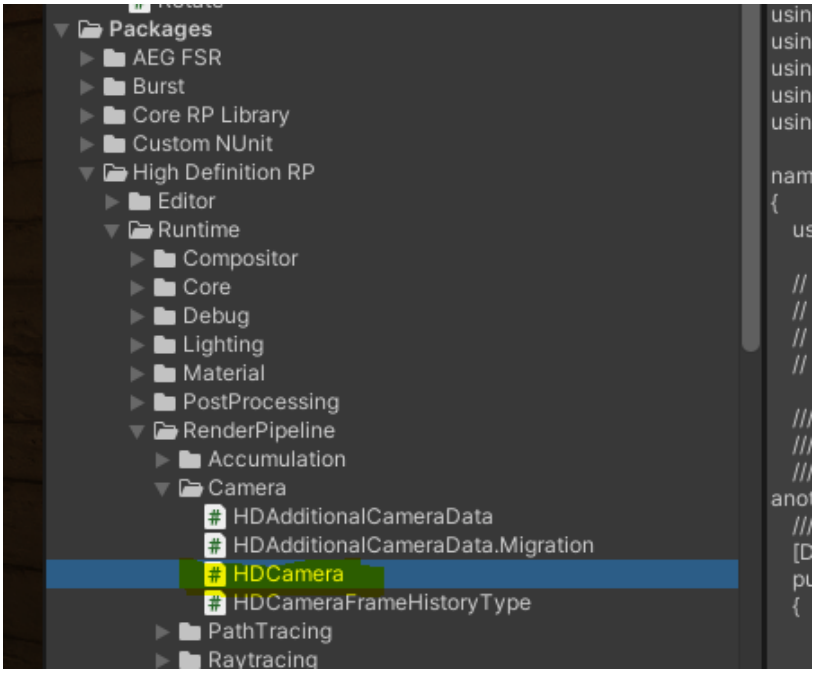
HDRP source files

To make FSR 3 work in HDRP we need to edit two files. If you are unsure on how to edit Render Pipeline Source files, read about it here.

NOTE: If you're already using one of our other upscalers for HDRP and already changed the source files, you don't have to do it again!

NOTE 2: Please take note that different HDRP versions require different edits

HDCamera.cs



UNITY 2022.3 LTS or older (with HDRP 13.X.X)

In the script, locate:

```
internal bool UpsampleHappensBeforePost()
{
    return IsDLSSEnabled() || IsTAAUEnabled();
}
```

And replace the whole function with:

```
public bool tndUpscalerEnabled = false;

internal bool IsTNDUpscalerEnabled()
{
    return tndUpscalerEnabled;
}

internal bool UpsampleHappensBeforePost()
{
    return IsTNDUpscalerEnabled() || IsDLSSEnabled() || IsTAAUEnabled();
}
```

UNITY 2023.1 or newer (and certain 2022.3 with HDRP 14.0.x)

In the script, locate:

```
internal DynamicResolutionHandler.UpsamplerScheduleType
UpsampleSyncPoint()
{
   if (IsDLSSEnabled())
   {
      return
HDRenderPipeline.currentAsset.currentPlatformRenderPipelineSettings.dyna
micResolutionSettings.DLSSInjectionPoint;
   } else if (IsTAAUEnabled())
   {
       return DynamicResolutionHandler.UpsamplerScheduleType.BeforePost;
   } else
   {
       return DynamicResolutionHandler.UpsamplerScheduleType.AfterPost;
   }
}
```

And replace the whole function with:

```
public bool tndUpscalerEnabled = false;

internal bool IsTNDUpscalerEnabled() {
    return tndUpscalerEnabled;
}

internal DynamicResolutionHandler.UpsamplerScheduleType
UpsampleSyncPoint() {
   if(IsDLSSEnabled()) {
         return
HDRenderPipeline.currentAsset.currentPlatformRenderPipelineSettings.dyna
micResolutionSettings.DLSSInjectionPoint;
   } else if(IsTAAUEnabled()) {
         return
DynamicResolutionHandler.UpsamplerScheduleType.BeforePost;
   } else if(IsTNDUpscalerEnabled()) {
         return
DynamicResolutionHandler.UpsamplerScheduleType.BeforePost;
   } else {
         return
DynamicResolutionHandler.UpsamplerScheduleType.AfterPost;
  }
}
```

UNITY 6 or newer (with HDRP 17.0.x)

In the script, locate the function:

```
internal DynamicResolutionHandler.UpsamplerScheduleType
UpsampleSyncPoint()
```

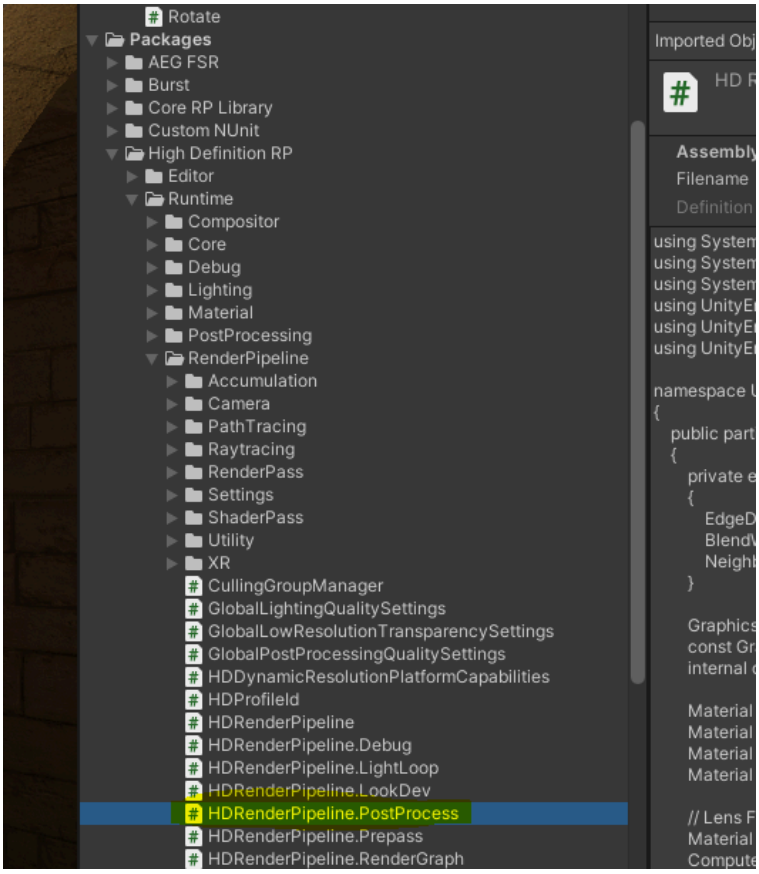
Add “ || IsTNDUpscalerEnabled() “ to the IsFSR2Enabled if statement.

```
 if(IsFSR2Enabled() || IsTNDUpscalerEnabled()) {
         Return
HDRenderPipeline.currentAsset.currentPlatformRenderPipelineSettings.dyna
micResolutionSettings.FSR2InjectionPoint;
   }
```

And add the following lines just above the function

```
public bool tndUpscalerEnabled = false;

internal bool IsTNDUpscalerEnabled() {
    return tndUpscalerEnabled;
}

internal DynamicResolutionHandler.UpsamplerScheduleType
UpsampleSyncPoint() {
    …
    …
    …
    …
    …
    …
    …
    …
```

HDRenderPipeline.PostProcess



All versions pre Unity 6 (with HDRP 15.X.X) or older

In the script locate:

```
source = CustomPostProcessPass(renderGraph, hdCamera, source,
depthBuffer, normalBuffer, motionVectors,
m_GlobalSettings.beforePostProcessCustomPostProcesses,
HDProfileId.CustomPostProcessBeforePP);
```

And replace the line with:

```
source = CustomPostProcessPass(renderGraph, hdCamera, source,
depthBuffer, normalBuffer, motionVectors,
m_GlobalSettings.beforePostProcessCustomPostProcesses,
HDProfileId.CustomPostProcessBeforePP);
if (hdCamera.IsTNDUpscalerEnabled())
{
    SetCurrentResolutionGroup(renderGraph, hdCamera,
ResolutionGroup.AfterDynamicResUpscale);
}
```

UNITY 6 or newer (with HDRP 17.0.x)

In the script locate:

```
source = BeforeCustomPostProcessPass(renderGraph, hdCamera, source,
depthBuffer, normalBuffer, motionVectors,
m_CustomPostProcessOrdersSettings.beforePostProcessCustomPostProcesses,
HDProfileId.CustomPostProcessBeforePP);
```

And replace the line with:

```
source = BeforeCustomPostProcessPass(renderGraph, hdCamera, source,
depthBuffer, normalBuffer, motionVectors,
m_CustomPostProcessOrdersSettings.beforePostProcessCustomPostProcesses,
HDProfileId.CustomPostProcessBeforePP);
if (hdCamera.IsTNDUpscalerEnabled())
{
    SetCurrentResolutionGroup(renderGraph, hdCamera,
ResolutionGroup.AfterDynamicResUpscale);
}
```

VR

The latest update of FSR - Upscaling for mobile has added support for PCVR for BIRP and URP. If you do run into any issues, please get in contact!

Current Limitations:

For now onlyMulti-pass is supported. Single-pass support is still under investigation.

FAQ

Q: Does FSR 3 offer free performance?
A: Almost, in many cases it does. However FSR 3 does 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: My performance with FSR 3 is worse than without!
A: While in most cases FSR 3 will have great performance results, some platforms or projects the tradeoff between the downscaling performance gain is not enough to counter the rendering cost of FSR 3. Unfortunately on mobile devices this is often the case.

Q: Will FSR 3 work for every kind of project?
A: No, in projects that are CPU bound, FSR 3 will probably only hinder performance. However because FSR 3 includes an industry leading AAA quality Temporal Anti-Aliasing, your projects fidelity might improve over Unity's standard anti-aliasing.

Q: FSR 3 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: My Vegetation is blurry when using FSR 3.
A: This is an issue that the vegetation shaders do not add their vertex displacement (movement) to the motion vectors. Unfortunately this is an issue that should be fixed by the creators of the vegetation. Assets like the "Nature Renderer" or "Nature Shaders" have this functionality.

Q: FSR 3 makes my game look jittery.
A: Try changing the Reactive Mask values, this should eliminate the jitters. A small jitter is expected behaviour on lower quality settings, just like normal TAA. Additionally, if you are on URP, make sure that all your third party Post-Processing effects are set to "**BeforeRenderingPostProcessing**, as they are often defaulted to "**AfterRenderingTransparents**" and this can make the reactive mask very jittery!

Q: I'm seeing really weird artefacts.
A: FSR 3 needs the Depth and Motion Vector buffers to work. It enables these buffers by default, however there are plugins known that disable these buffers. Make sure your camera is rendering the Depth & Motion Vectors!

Q: FSR 3 flips out when adding the FSR script to a another camera
A: FSR 3 Upscaling currently only supports 1 upscaled camera!

Q: FSR 3 is amazing!
A: Indeed FSR 3 is a great upscaling technique, thank AMD for making it publicly available and do leave us a review!

Known Issues & Limitations

General

- Multiple cameras may work, but are not officially supported

BIRP

-

URP

- Unity 2022.1 is not supported. **Older and newer versions are!**
- URP 12 and newer is supported. Older versions will not work due to missing Motion Vector support.
- Ideally most Post Processing is rendered after FSR 3 (some before), 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.
- In some situations when using the Particle System, you'll have to disable "Generate Mip-Maps", because of a Unity issue they will use a too low Mip-Map and not scale up correctly.

HDRP

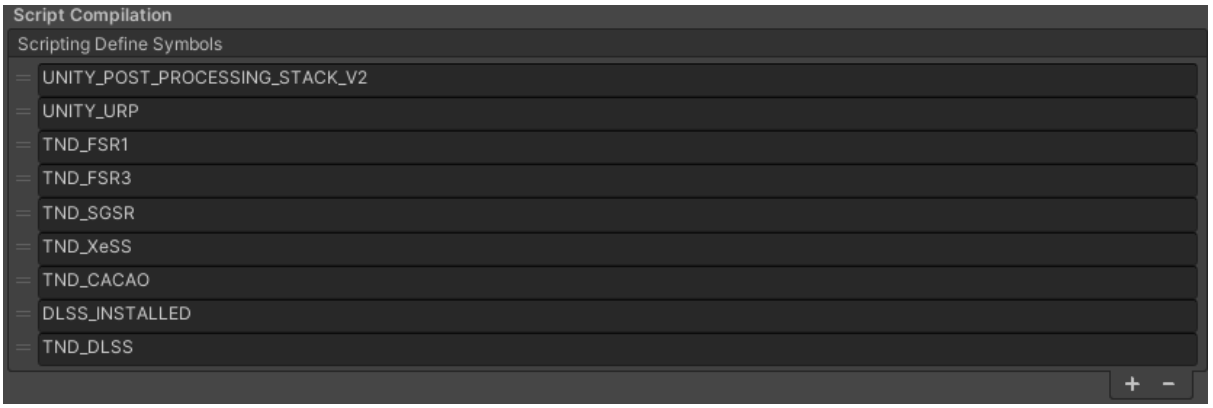
- Requires changes to the source of the HDRP package.
- Sometimes to make FSR 3 work correctly, the "Dynamic Resolution Type" in the HDRenderPipelineAsest file should be set to "Software" instead of "Hardware"

Uninstall

Step 1: Delete the “FSR” folder in “The Naked Dev” folder

Optional Step 2 (BIRP ONLY): Delete the Custom Post-Processing Package folder from the “Packages” folder

Optional Step 3: If you are still getting compile errors after deleting the asset, go to the Scripting Define Symbols in the Player settings and manually delete the define symbols starting with “**TND_**” and “**UNITY_URP/UNITY_BIRP/UNITY_HDRP**”.



If you are still getting compile errors, make sure you are not referencing the upscaling asset in a component of your own.

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

Special Thanks

Special thanks to Nico de Poel for allowing us to use his FSR 3 backend to make sure our asset supports so many platforms!