NVIDIA Image Scaling Unreal Engine Plugin

Table of Contents

DLSS 4 and the NVIDIA Unreal Engine NVIDIA Image Scaling plugin

The NVIDIA *Image Scaling* Unreal Engine plugin is part of a wider suite of related NVIDIA performance and image quality improving technologies and corresponding NVIDIA Unreal Engine plugins:

- NVIDIA DLSS *Frame Generation (DLSS-FG)* boosts frame rates by using AI to render additional frames. *DLSS-FG* requires a GeForce RTX 40 series graphics card.
- NVIDIA DLSS *Multi Frame Generation (DLSS-MFG)* uses AI to boost frame rates by generating up to 3 additional high-quality frames, all while optimizing responsiveness with NVIDIA Reflex. DLSS Multi Frame Generation is available for GeForce RTX 50 Series GPUs.
- NVIDIA DLSS *Super Resolution (DLSS-SR)* boosts frame rates by rendering fewer pixels and using AI to output high resolution frames. *DLSS-SR* requires an NVIDIA RTX graphics card.
- NVIDIA DLSS *Ray Reconstruction (DLSS-RR)* uses AI to enhance image quality and generate additional pixels for intensive ray-traced scenes. Ray Reconstruction replaces hand-tuned denoisers with an NVIDIA supercomputer-trained AI network that generates higher-quality pixels in between sampled rays. This feature is available for all RTX GPUs and requires Super Resolution to be enabled.
- NVIDIA *Deep Learning Anti Aliasing (DLAA)* is used to improve image quality. *DLAA* requires an NVIDIA RTX graphics card.
- NVIDIA *Image Scaling (NIS)* provides best-in class upscaling and sharpening for non-RTX GPUs, both NVIDIA or 3rd party. Please refer to the NVIDIA *Image Scaling* Unreal Engine plugin for further details.
- NVIDIA *RTX Dynamic Vibrance* enhances visual clarity by dynamically tuning color saturation and contrast.
- NVIDIA *Reflex Low Latency* technology, synchronizes the CPU frame submission with GPU processing for just in time rendering.

NVIDIA *DLSS 4* combines DLSS Multi Frame Generation, DLSS Frame Generation, DLSS Super

Resolution, DLSS Ray Reconstruction, NVIDIA DLAA, NVIDIA Reflex Low Latency and delivers boosted frame rates, great responsiveness, and better than native IQ.

What's documented in this file

The NVIDIA Unreal Engine NIS plugin provides:

- NVIDIA Image Scaling

What's available and documented elsewhere

The NVIDIA Unreal Engine Streamline plugins provide:

- DLSS Frame Generation (also called DLSS-G or DLSS-FG)
- DLSS Multi Frame Generation (also called DLSS-MFG)
- NVIDIA Reflex Low latency
- NVIDIA RTX Dynamic Vibrance (also called DeepDVC)

The NVIDIA Unreal Engine DLSS-SR plugin provides:

- DLSS Super Resolution (DLSS-SR)
- Deep Learning Anti-Aliasing (DLAA)
- DLSS Ray Reconstruction (DLSS-RR)
- Movie Render Queue support for DLSS-SR,DLSS-RR, DLAA

Quickstart

Please refer to the relevant section in this document for additional details.

1. Enable the *NIS* plugin in the Editor, then restart the editor
2. NIS in Blueprint: The `SetNISMode` and `SetNISSharpness` function of the NIS blueprint library provides convenient functions for setting the following console variables and are recommended to be used when integrating support into a project's user interface and settings.
3. NIS upscaling in game: The following console variables can be set to enable NIS:
    1. r.NIS.Enable 1
    2. r.NIS.Upscaling 1
    3. r.ScreenPercentage 50
    4. r.TemporalAA.Upsampling 0
    5. r.TemporalAA.Upscaler 0
    6. *Optional* r.NIS.Sharpness 0.5
4. NIS sharpening in game: The following console variables can be set to enable a NIS sharpening pass regardless of whether temporal or spatial upscaling is used.
    1. r.NIS.Enable 1
    2. r.NIS.Sharpness 0.5

Troubleshooting

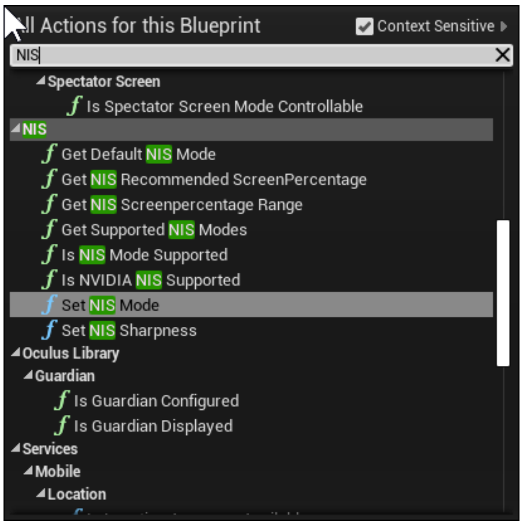
System requirements

- A GPU supporting SM5
- UE 4.27.1 or higher
- A project targeting the SM5 renderer and using either:
    - Vulkan
    - DX11
    - DX12

NIS Blueprint library

The UNISLibrary blueprint library provides functionality to query whether NIS and which modes are supported. It also provides convenient functions to enable the underlying NIS console variables. The tooltips of each function provide additional information.

Using the UNISLibrary via blueprint or C++ (by including the NISBlueprint module in a game project) is recommended over setting the console variables directly. This will make sure that any future updates will be picked up by simply updating the NIS plugin, without having to update the game logic.



DLSS and the NIS NVIDIA Image Scaling plugin (UE 4.27.1+)

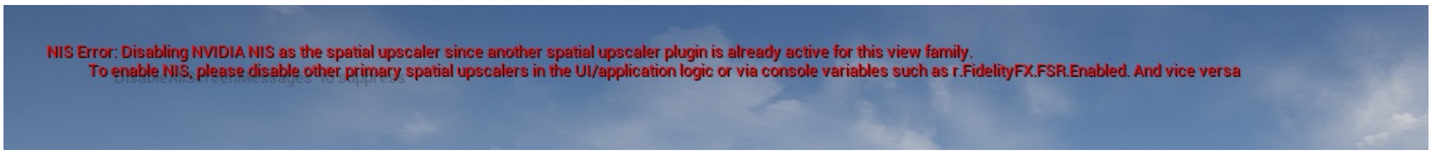
The *DLSS* plugin and NVIDIA Image Scaling (*NIS*) plugins can be enabled together for the same project. Please see the `RTX UI Developer Guidelines` document for recommended UI implementations.

When both the *DLSS* and *NIS* plugins are enabled for a project, NIS sharpening will be used instead of DLSS sharpening. See `r.NGX.DLSS.PreferNISSharpen` in the *DLSS* plugin for details.

Interactions between NIS and other spatial upscaler plugins

The engine supports having multiple spatial upscaling plugins enabled for a single project. However only one can be **active** at the same time. The engine will assert in `FViewFamily::SetPrimarySpatialUpscalerInterface` if a plugin tries to set spatial upscaler interfaces when another plugin has already set it before.

To avoid this assert and a subsequent engine crash, the *NIS* plugin tries to detect cases where another spatial upscaler plugin might be active already and disables itself, showing a message on screen like this:


```
NIS Error: Disabling NVIDIA NIS as the spatial upscaler since another spatial upscaler plugin is already active for this view family.
    To enable NIS, please disable other primary spatial upscalers in the UI/application logic or via console variables such as r.FidelityFX.FSR.Enabled. And vice versa
```

The fix is to change UI and gameplay logic to have only one spatial upscaler plugin **active** at run time.

Panini Projections

The NIS plugin does *not* support panini projections at this time.

Command Line Options And Console Variables and Commands

Enabling NIS (Engine Side)

The *NIS* plugin uses various engine side hooks, which can be configured by the following cvars.

Note By default the engine prefers temporal upscaling (either built-in, or via temporal upscaling plugin such as the *DLSS* plugin). As such some console variables need to be changed to turn temporal upscaling off.

- r.TemporalAA.Upsampling 0
  - Disable built-in temporal upsampling (i.e. TAAU/TSR)
- r.TemporalAA.Upscaler 0
  - Disable a custom TAAU upscaling plugin, such as the DLSS plugin
- r.ScreenPercentage 50 ... 100
  - Adjust the screen percentage

NIS supports a set of quality modes with recommended screen percentages, in addition to a custom screen percentage. The recommended screen percentages can be retrieved via the `GetNISRecommendedScreenPercentage` Blueprint function or as excerpted below from the accompanying [RTX Developer Guidelines](RTX UI Developer Guidelines.pdf) document:

Mode	r.ScreenPercentage
Ultra Quality	77
Quality	66.667
Balanced	59
Performance	50
Custom	50 100

Note: For convenience, the `SetNISMode` blueprint function sets r.ScreenPercentage based on the NIS mode as well.

Blueprint functions:

- `SetNISMode`
- `IsNISSupported`
- `IsNISModeSupported`, `GetNISModes`,`GetNISRecommendedScreenPercentage`, `GetNISScreenPercentageRange`

Enabling NIS (Plugin Side)

- r.NIS.Enable (1, default)
  - Enable/disable NIS upscaling and/or sharpening
- r.NIS.Upscaling (1,default),
  - Enable NIS Upscaling. Also requires r.TemporalAA.Upscaler 0

Blueprint functions:

- `SetNISMode`
- `IsNISSupported`
- `IsNISModeSupported`, `GetNISModes`,`GetNISRecommendedScreenPercentage`, `GetNISScreenPercentageRange`

NIS Runtime Image Quality Tweaks

- r.NIS.Sharpness (0.0, default)
  - 0.0 to 1.0: Sharpening to apply to either primary NIS pass or the secondary NIS pass. If 0.0 the secondary NIS sharpening pass will not be executed
- r.NIS.HDRMode (-1, default)
  - -1: Automatic. Determines the NIS HDR mode based on ETonemapperOutputDevice
  - 0: None
  - 1: Linear
  - 2: PQ

Blueprint functions:

- `SetNISSharpness`

Miscellaneous

- r.NIS.HalfPrecision,(-1, default)
    - Enable/disable half precision in the NIS shaders and selects which permutation is used
    - -1: automatic. Pick the appropriate FP16 permutation based on shader model and RHI
    - 0: Float32, disable half precision
    - 1: Min16Float, half precision, intended for UE4 DX11 SM5
    - 2: Min16FloatDXC, half precision, intended for UE4 DX12 SM5
    - 3: Float16DXC, half precision, intended for UE5 DX12 SM6

Note: Half precision support has only been tested with DX11 and DX12. Most reliable results are configuring the project to run with Shader Model 6 and DX12 on Unreal Engine 5.

Enabling Automatic Mip-Map LOD bias selection for NIS

The application is recommended to set the mip-map bias (also called the texture LOD bias) to a value lower than 0. This will improve the overall image quality as textures are sampled at the display resolution rather than the lower render resolution in use with NIS. For details, please refer to section *5.1 Mip-Map Bias* in the NIS Programming Guide

The engine doesn't do this when running with spatial upscaling plugins. In this case, the application is recommended to adjust this manually. The `SetNISMode` Blueprint library function performs this resolution dependent mip-map LOD bias adjustment and can be configured with the following console variables:

- r.NIS.Upscaling.AutomaticMipMapLODBias (1, default)
    - Enable automatic setting of r.MipMapLODBias based on the effective NIS screen percentage
    - NOTE: This is only applied when using the UNISLibrary::SetNISMode blueprint function
- r.NIS.Upscaling.AutomaticMipMapLODBias.Offset (0, default)
    - Allows offsetting the automatic resolution dependent mip map LOD bias by this amount
    - NOTE: This is only applied when using the UNISLibrary::SetNISMode blueprint function

Blueprint functions:

- `SetNISMode`

Note: The engine automatically computes a screenpercentage based mip map LOD bias when running with temporal upscaling, either built-in (i.e. TAAU/TSR) or via a plugin such as DLSS.

NIS API and UI Documentation

The NIS Programming Guide provides details about the NIS SDK shaders which are used by the plugin to implement NIS.

The RTX Developer Guidelines (Chinese) provides details about recommended game settings and UI for NIS and DLSS.