NVIDIA Unreal Engine DLSS Super Resolution / DLAA Plugin

DLSS3 and the NVIDIA Unreal Engine DLSS Super Resolution Plugin

The NVIDIA *DLSS Super Resolution / DLAA* plugin is part of a wider suite of related NVIDIA performance and image quality improving technologies and corresponding NVIDIA Unreal Engine plugins:

- NVIDIA *Deep Learning Supersampling 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 *Deep Learning Supersampling 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 *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.

The NVIDIA Unreal Engine DLSS-SR plugin (documented here) provides:

- DLSS Super Resolution (DLSS-SR)
- Deep Learning Anti-Aliasing (DLAA)

The NVIDIA Unreal Engine Streamline plugin (available separately) provides:

- DLSS Frame Generation (also called DLSS-G or DLSS-FG)
- NVIDIA Reflex

The NVIDIA Unreal Engine NIS plugin (available separately) provides:

- NVIDIA Image Scaling

Quickstart

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

1. Enable the *DLSS-SR* plugin in the Editor, then restart the editor
2. DLSS/DLAA in the Editor: enable the following settings in the Project Plugin settings
    1. Enable DLSS to be turned on in Editor viewports (it is off by default)
    2. In the Viewport Options (triple-bar ≡ menu in the top left corner), set the Screen Percentage option to control the upscale percentage. Not all screen percentages are supported. Values in the range 50-67 are recommended for DLSS and 100 for DLAA.
3. DLSS/DLAA in Blueprint:
    1. The `EnableDLSS` DLSS blueprint library function provides a convenient method for setting the console variables and is recommended to be used when integrating support into a project's user interface and settings.
    2. The `GetDlssModeInformation` DLSS blueprint library function can be used to query optimal screen percentage to use for given quality mode. The screen percentage can then be used with `r.ScreenPercentage` in an `ExecuteConsoleCommand` node.
4. DLSS/DLAA in PIE: `Edit -> Editor Preferences -> Performance` turn off "Override game screen percentage settings with editor settings in PIE" (it is on by default)
5. DLSS in Game: make sure that the following console variables are set to enable DLSS:
    1. r.NGX.Enable 1 (can be overriden on the command line with -ngxenable)
    2. r.NGX.DLSS.Enable 1
    3. r.ScreenPercentage 66.7
6. DLAA in Game: make sure that the following console variables are set to enable DLAA
    1. r.NGX.Enable 1 (can be overriden on the command line with -ngxenable)
    2. r.NGX.DLSS.Enable 1
    3. r.ScreenPercentage 100
7. Check the log for `LogDLSS: NVIDIA NGX DLSS supported 1`
8. (Optionally) Enable the DLSS on screen indicator in the bottom left of the screen via `DLSS\Source\ThirdParty\NGX\Utils\ngx_driver_onscreenindicator.reg` to verify that DLSS is active

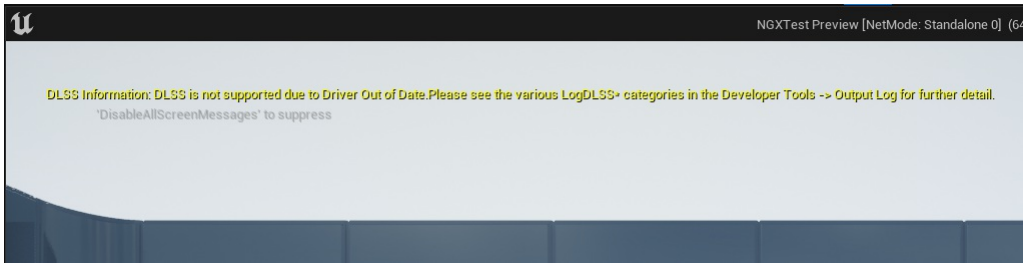
Troubleshooting

System requirements

- Windows 10, 64 bit
    - at least version v1709, Windows 10 Fall 2017 Creators Update 64-bit.
- NVIDIA Geforce Driver
    - Required: driver issued after March 3, 2022 (for instance 512.15)
- NVIDIA RTX GPU (GeForce, Titan or Quadro) with DLSS Super Resolution support
- UE project using either
    - Vulkan
    - DX11
    - DX12

Diagnosing DLSS Issues in the Editor

The DLSS plugin shows various common reasons why DLSS might not be working at the top of the screen (in non-Shipping build configurations). This message can also be turned off in the DLSS plugin settings, as discussed in the "DLSS plugin settings" section in this document.

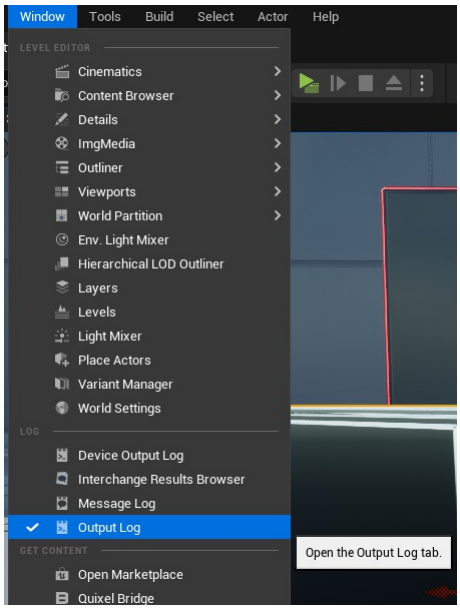


Additionally, the DLSS plugin modules write various information into the following UE log categories:

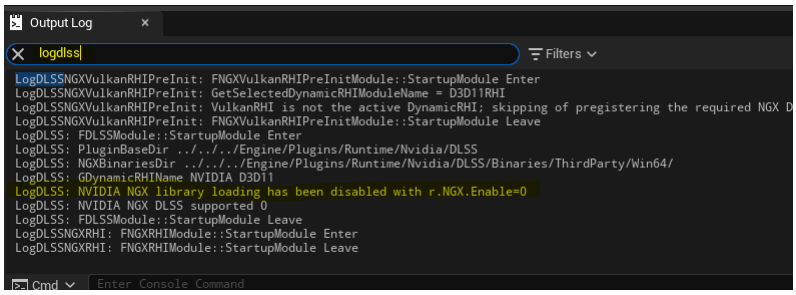
- LogDLSS
- LogDLSSEditor
- LogDLSSBlueprint
- LogDLSSNGXRHI
- LogDLSSNGXD3D11RHI
- LogDLSSNGXD3D12RHI
- LogDLSSNGXVulkanRHIPreInit
- LogDLSSNGXVulkanRHI

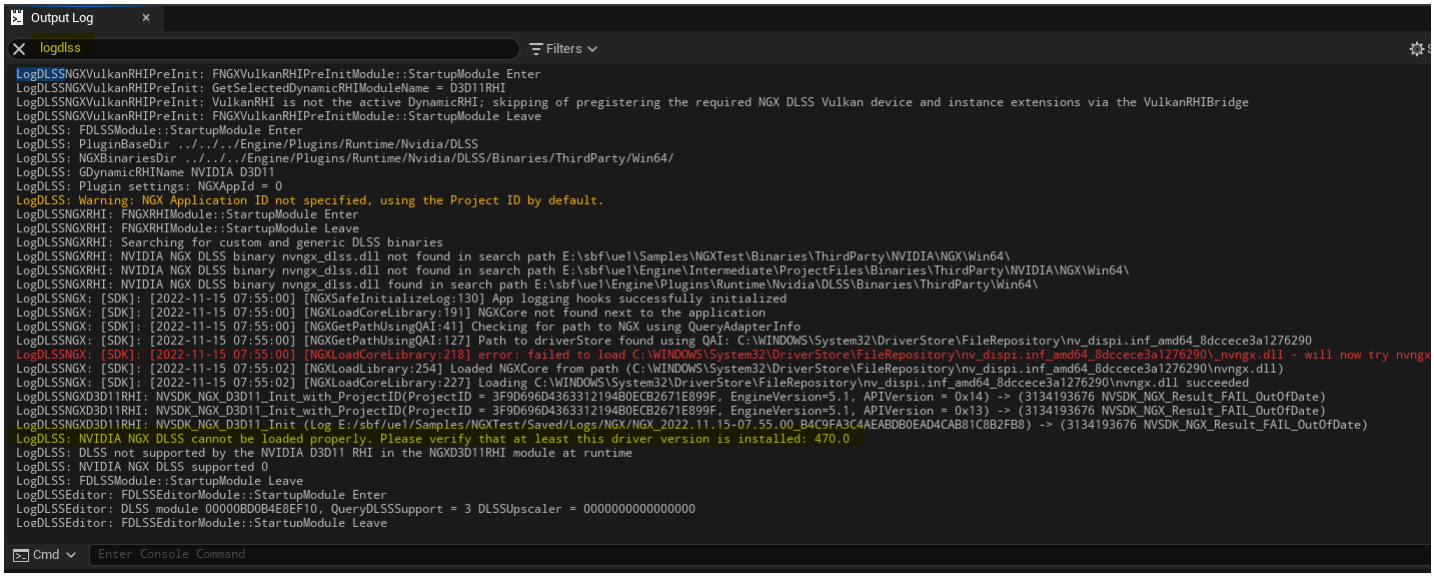
- LogDLSSNGX

Those can be accessed in the Editor under `Window -> Output Log`



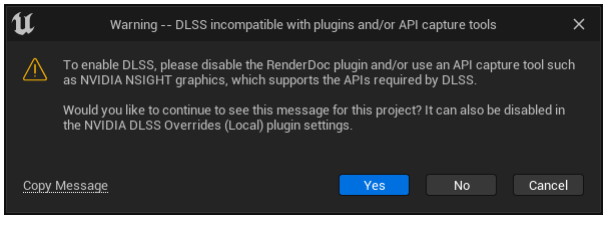
The Message log then can be filtered to show only the DLSS related messages to get more information on why DLSS might not be functioning as expected, as shown in these examples.





Incompatibilities with API Capture Tools such as RenderDoc

The Editor will show a warning at startup if DLSS incompatible API capture tools (such as RenderDoc) or plugins are used. To enable DLSS, please use an API capture tool such as NVIDIA NSIGHT Graphics, which support the NGX APIs required by DLSS.



Incompatibilities with Depth of Field

As DLSS workload occurs in the same spot as TAAU in the pipeline, visual differences in DoF are expected. To minimize the differences, it is recommended to use DLSS in 'Quality' or 'Ultra Quality' modes. It is possible to tweak the DoF settings in the camera actor to compensate for the differences. Please keep in mind this is content dependent. Thus for some scenes the differences can be minimal and avoided while in other cases it might be more challenging.

Verify Engine side plugin hooks

The following cvars should be set to those values by default:

- r.TemporalAA.Upscaler 1
- r.Reflections.Denoiser 2

Enabling NGX DLSS Logging on End User machines

The DLSS plugin also pipes the NGX DLSS logs into the UE logging system into the `LogDLSSNGX` log category. It is enabled by default and can be tweaked with the `r.NGX.LogLevel` console variable, or set with the -NGXLogLevel=X command line option.

This requires an NVIDIA GeForce **driver version 461.36** or later.

Enabling NGX DLSS Logging during Development

If r.NGX.EnableOtherLoggingSinks is set then additional NGX logging of the NVIDIA NGX software stack to files can be used as well, as discussed in the "NGX logging" chapter of the DLSS Programming Guide for details. The -NGXLogFileEnable and -NGXLogFileDisable command line options can also override the default setting. The DLSS SDK provides registry keys which can be set with the following .reg files which can be found in the plugin folder under `\DLSS\Source\ThirdParty\NGX\Utils\`:

- `ngx_log_on.reg`
- `ngx_log_off.reg`
- `ngx_log_verbose.reg`

The DLSS plugin will write those into subfolder under `$(ProjectDir)\Saved\Logs\` with a `NGX_$(TimeStamp)_$(GUID)` pattern

- `nvngx.log`
- `nvngx_dlss_2_1_34.log`
- `nvsdk_ngx.log`

DLSS On-Screen Indicator

The DLSS SDK provides registry keys which can be set with the following .reg files which can be found in the plugin folder under `\DLSS\Source\ThirdParty\NGX\Utils\`:

- `ngx_driver_onscreenindicator.reg`
- `ngx_driver_off_screenindicator.reg`

With the first registry key set, DLSS will display an indicator on-screen when it is enabled, enabling easier troubleshooting. The second registry key can be used to disable this indicator again.

Please see the DLSS Programming Guide for further details.

Command Line Options And Console Variables and Commands

Enabling DLSS (Engine Side)

The DLSS plugin uses various engine side hooks, which can be configured by the following cvars. Their default values

- r.TemporalAA.Upscaler (1, default)
  - Enable a custom TAAU upscaling plugin, such as the DLSS plugin
- r.Reflections.Denoiser (2, default)
  - Enable a custom denoising plugin. The DLSS plugin makes use of this to improve image quality for raytraced reflections by adding additionial TAA passes

Enabling Motion vectors for DLSS

DLSS requires correct motion vectors to function properly. The following console variable can be used to render motion vectors for all objects, and not just the ones with dynamic geometry. This can be useful if it's infeasible to e.g. change all meshes to stationary or dynamic.

- r.Velocity.ForceOutput (0, default)
  - Force the base pass to compute motion vector, regardless of FPrimitiveUniformShaderParameters.
  - 0: Disabled
  - 1: Enabled

Enabling DLSS/DLAA (Plugin Side)

- r.NGX.Enable (1, default) can also be overridden on the command line with **-ngxenable** and **-ngxdisable**
  - Whether the NGX library should be loaded. This allow to have the DLSS plugin enabled but avoiding potential incompatibilities by skipping the driver side NGX parts of DLSS.
- r.NGX.DLSS.Enable (1, default)
  - Enable/Disable DLSS/DLAA.

There are two alternatives for presenting DLSS in a UI, either listing all DLSS modes explictly or offering DLSS Off/Auto mode. For explicit DLSS modes, list all supported DLSS modes in the UI. Find the optimal screen percentage for the selected DLSS mode with `GetDLSSModeInformation`, set `r.ScreenPercentage` using that value, and call `EnableDLSS`. For DLSS auto mode, find the optimal screen percentage for the DLSS auto mode with `GetDLSSModeInformation`, if the screen percentage is not 0 then set `r.ScreenPercentage` using that value and call `EnableDLSS`.

The plugin includes two blueprint macros in the DLSSMacros asset for enabling DLSS and DLAA, one for explicit DLSS modes and one for DLSS auto mode. You can use a macro directly or copy it into your project to use as a starting point for your own work. You may need to enable "Show Engine Content" and "Show Plugin Content" in the content browser settings for the macros to be visible.

Blueprint functions:

- `EnableDLSS`, `IsDLSSEnabled`
- `IsDLSSSupported`, `QueryDLSSSupport`, `GetDLSSMinimumDriverVersion`, `GetDefaultDLSSMode`
- `IsDLSSModeSupported`, `GetSupportedDLSSModes`,`GetDLSSModeInformation`, `GetDLSSScreenPercentageRange`

DLSS Runtime Image Quality Tweaks

- r.NGX.DLSS.DilateMotionVectors (1, default)
  - 0: pass low resolution motion vectors into DLSS
  - 1: pass dilated high resolution motion vectors into DLSS. This can help with improving image quality of thin details.
- r.NGX.DLSS.Reflections.TemporalAA (1, default)
  - Apply a temporal AA pass on the denoised reflections
- r.NGX.DLSS.WaterReflections.TemporalAA (1, default)
  - Apply a temporal AA pass on the denoised water reflections
- r.NGX.DLSS.EnableAutoExposure
  - 0: Use the engine-computed exposure value for input images to DLSS
  - 1: Enable DLSS internal auto-exposure instead of the application provided one - enabling this can alleviate effects such as ghosting in darker scenes (default)
- r.NGX.DLSS.PreferNISSharpen (2, default)
  - Prefer sharpening with an extra NIS plugin sharpening pass instead of DLSS sharpening if the NIS plugin is also enabled for the project.
  - Requires the NIS plugin to be enabled, DLSS sharpening will be used otherwise.
    - 0: Softening/sharpening with the DLSS pass.
    - 1: Sharpen with the NIS plugin. Softening is not supported. Requires the NIS plugin to be enabled.

        - 2: Sharpen with the NIS plugin. Softening (i.e. negative sharpness) with the DLSS plugin. Requires the NIS plugin to be enabled.
    - **Note** This cvar is only evaluated when using the deprecated `SetDLSSSharpness` Blueprint function, from either C++ or a Blueprint event graph!
    - **Note** DLSS sharpening is deprecated, future plugin versions will remove DLSS sharpening. Use the NIS plugin for sharpening instead

DLSS Binaries

- r.NGX.BinarySearchOrder (0, default)
    - 0: automatic
        - use custom binaries from project and launch folder (*ProjectDir*)/*Binaries*/*ThirdParty*/*NVIDIA*/*NGX*/(Platform) if present
        - fallback to generic binaries from plugin folder
    - 1: force generic binaries from plugin folder, fail if not found
    - 2: force custom binaries from project or launch folder, fail if not found
    - 3: force generic development binaries from plugin folder, fail if not found. This is only supported in non-shipping build configurations

DLSS memory usage

- `stat DLSS`
    - shows how much GPU memory DLSS uses and how many DLSS features, i.e. instances of DLSS are allocated.
    - In steady state there should be 1 DLSS feature allocated per view. This value can increase temporarily, typically after changing the DLSS quality mode or resizing the window. This can be configured with the `r.NGX.FramesUntilFeatureDestruction` console variable

DLSS presets

The DLSS plugin offers render presets that allow tweaking different aspects of DLSS-SR upscaling. Usually there will be no need to change from the default presets. The DLSS presets are named A – G and are described in more detail in the DLSS Programming Guide.

A particular preset can be forced separately for DLAA and for each DLSS quality mode in the DLSS plugin settings (Edit -> Project Settings -> NVIDIA DLSS -> General Settings -> Advanced). Additionally, the DLSS preset can be globally overridden by setting the cvar `r.NGX.DLSS.Preset` to a value from 0 to 7 (0=project setting, 1=A, 2=B, 3=C, 4=D, 5=E, 6=F, 7=G).

DLSS presets are subject to change with new versions of DLSS and via over the air (OTA) updates. To opt out of OTA updates, unselect the "Allow OTA Update" option in the DLSS plugin settings (Edit -> Project Settings -> NVIDIA DLSS -> General Settings). OTA updates can also be overridden on the command line with **-ngxdisableota** and **-ngxenableota**. The interaction of OTA updates with DLSS presets is described in more detail in the DLSS Programming Guide.

NGX Project ID

The DLSS plugin by default uses the project identifier to initialize NGX and DLSS. On rare occasion, NVIDIA might provide a special NVIDIA NGX application ID. The following console variable determines which one is used.

r.NGX.ProjectIdentifier (0, default)

- 0: automatic:
    - use NVIDIA NGX Application ID if non-zero, otherwise use UE Project ID
- 1: force UE Project ID
- 2: force NVIDIA NGX Application ID (set via the Project Settings -> NVIDIA DLSS plugin)

Please refer to the "Distributing DLSS" section for further details.

Multi GPU Support (Experimental)

The DLSS plugin supports multiple GPUs in certain circumstances, as shown in the following table. There AFR stands for Alternate-Frame-Rendering, i.e. SLI or CrossFire, and SFR stands for Split-Frame-Rendering.

RHI	AFR	SFR
D3D12RHI	no	no
D3D11RHI	yes	no
VulkanRHI	no	no

Notes:

- D3D12RHI
    - AFR is not supported
        - Primarily due to higher level renderer code not maintaining TAA (and thus DLSS) history across non-consecutive frames on the same GPU
    - SFR is not yet supported in 5.1 engines
- D3D11RHI
    - AFR is supported via driver based, automatic SLI support
- VulkanRHI
    - The VulkanRHI (as of UE 4.27) does not implement explicit MGPU, and thus neither AFR nor SFR are available

The following console variables can be used to adjust how DLSS interacts with the GPU nodes

- r.NGX.DLSS.FeatureCreationNode (-1, default)
    - Determines which GPU the DLSS feature is getting created on
    - -1: Create on the GPU the command list is getting executed on
    - 0: Create on GPU node 0
    - 1: Create on GPU node 1
- r.NGX.DLSS.FeatureVisibilityMask (-1, default)
    - Determines which GPU the DLSS feature is visible to
    - -1: Visible to the GPU the command list is getting executed on
    - 1: visible to GPU node 0
    - 2: visible to GPU node 1
    - 3: visible to GPU node 0 and GPU node 1

Miscellaneous

- r.NGX.DLSS.AutomationTesting (0, default)
    - Whether the NGX library should be loaded when GIsAutomationTesting is true.(default is false)
    - Must be set to true before startup. This can be enabled for cases where running automation testing with DLSS is desired
- r.NGX.Automation.Enable (0, default)
    - Enable automation for NGX DLSS image quality and performance evaluation.
- r.NGX.Automation.ViewIndex (0, default)
    - Select which view to use with NGX DLSS image quality and performance automation.
- r.NGX.Automation.NonGameViews (0,default)
    - Enable non-game views for NGX DLSS image quality and performance automation.
- r.NGX.FramesUntilFeatureDestruction (3, default)
    - Number of frames until an unused NGX feature gets destroyed
- r.NGX.DLSS.MinimumWindowsBuildVersion (16299, default for v1709)
    - Sets the minimum Windows 10 build version required to enable DLSS
- r.NGX.LogLevel (1, default)
    - Determines the minimal amount of logging the NGX implementation. Please refer to the DLSS plugin documentation on other ways to change the logging level.

    - 0: off
    - 1: on
    - 2: verbose
- r.NGX.EnableOtherLoggingSinks (0, default)
    - Determines whether the NGX implementation will turn on additional log sinks LogDLSSNGXRHI
    - 0: off
    - 1: on
- r.NGX.RenameNGXLogSeverities (1, default)
    - Renames 'error' and 'warning' in messages returned by the NGX log callback to 'e_rror' and 'w_arning' before passing them to the UE log system
    - 0: off
    - 1: on, for select messages during initalization
    - 2: on, for all messages
- r.NGX.DLSS.ReleaseMemoryOnDelete (1, default)
    - Enable/disable releasing DLSS related memory on the NGX side when DLSS features get released

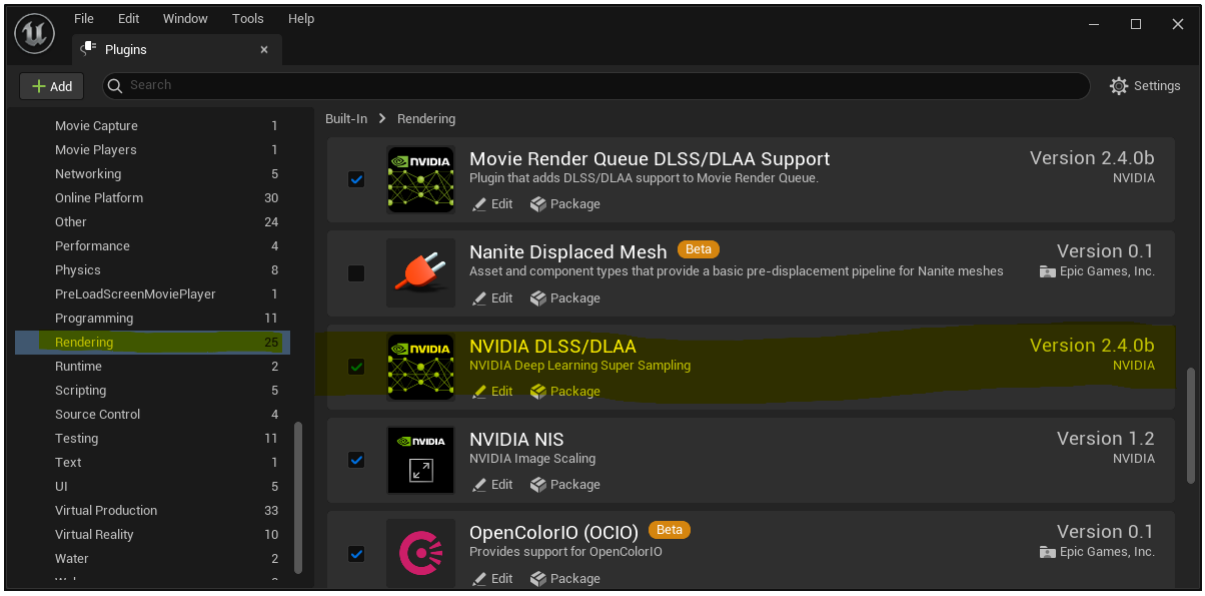
DLSS and the NIS NVIDIA Image Scaling plugin

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 suggested UI implementations.

It is recommend to use the NIS plugin for sharpening in place of the deprecated DLSS sharpening feature.

DLSS/DLAA in the Editor

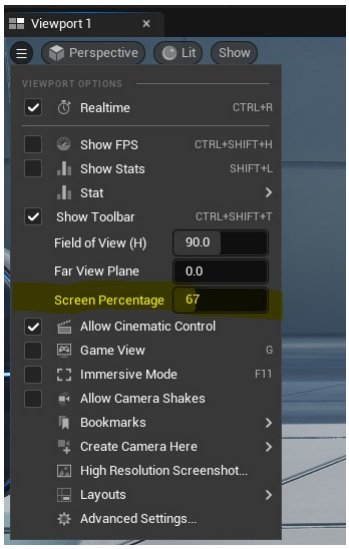
Enabling DLSS/DLAA for a project



Enabling DLSS/DLAA in Level Editor Viewports

First enable the "Enable DLSS/DLAA to be turned on in Editor viewports" checkbox in the project plugin settings `Project Settings -> Plugins -> NVIDIA DLSS` or the local override `Project Settings -> Plugins -> NVIDIA DLSS (Local)`. Note that DLSS/DLAA is off by default in editor viewport windows.

With DLSS/DLAA enabled, change the Screen Percentage in an editor viewport window to control the upscaling amount. With a screen percentage of 100, DLAA will be in effect. Sliding the screen percentage between 50% and 99% will enable DLSS upscaling.



DLSS Plugin Settings

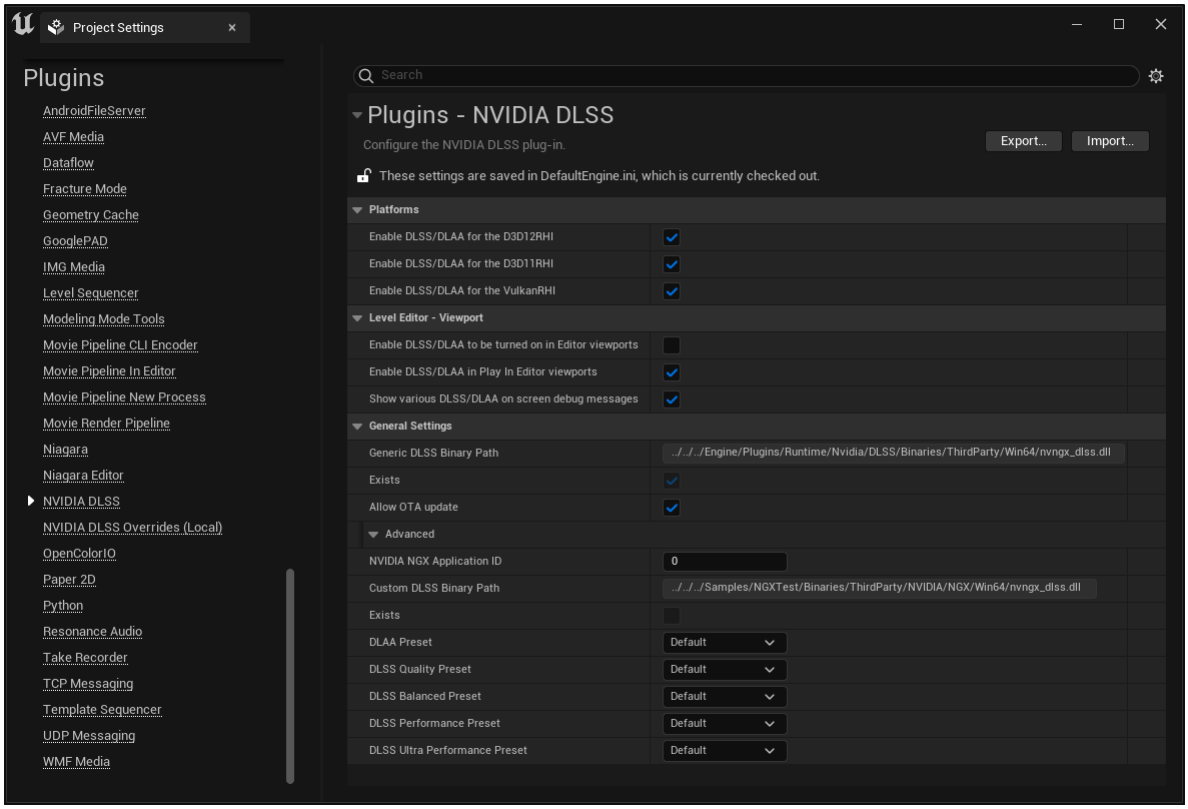
Some of the "Level Editor - Viewport" settings are split across two config files and settings pages to tailor how DLSS is interacting with the editor user experience.

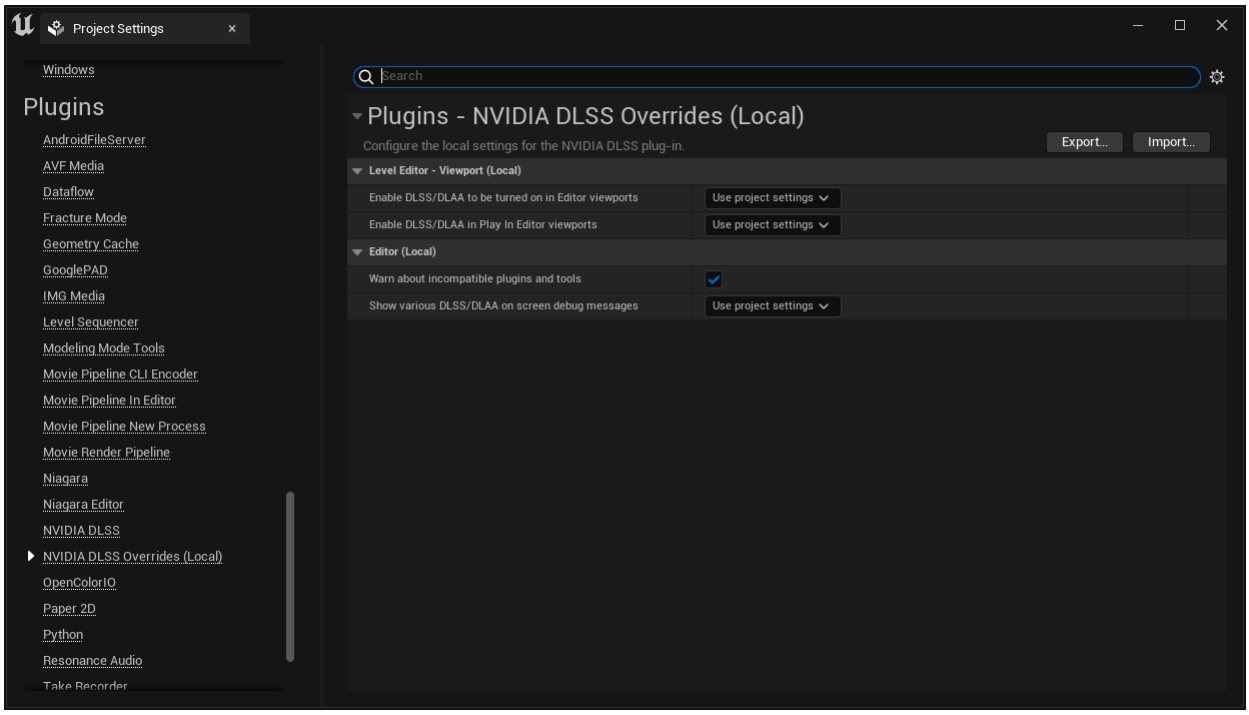
For example, a cross-platform game project might find it more practical by default to only have DLSS enabled in "Play In Editor Viewports" or in "game mode" in order to maintain a consistent content authoring experience across the range of supported platforms. However projects (e.g. an architecture visualization project with notable raytracing workloads), might find it more useful to have DLSS enabled during the content authoring. Either way each user can override those settings locally:

- `Project Settings -> Plugins -> NVIDIA DLSS`
    - stored in DefaultEngine.ini
    - typically resides in source control.

- settings here are shared between users

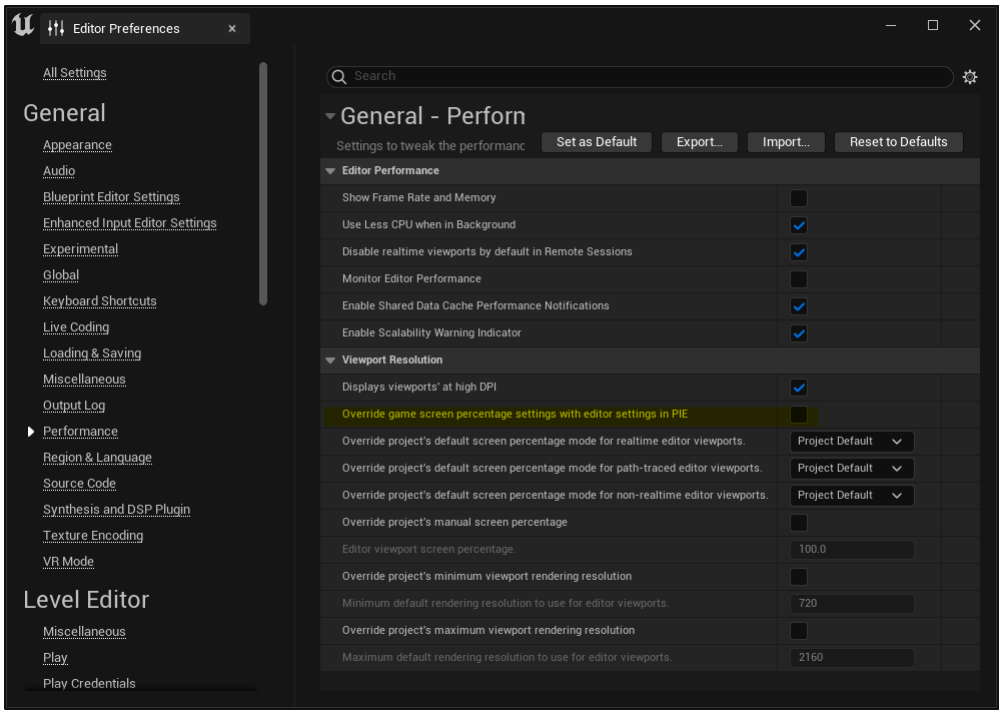
- `Project Settings -> Plugins -> NVIDIA DLSS (Local)`
  - stored UserEngine.ini
  - not recommended to be checked into source control.
  - allow a user to override project wide settings if desired. Defaults to "use project settings"





Editor Settings

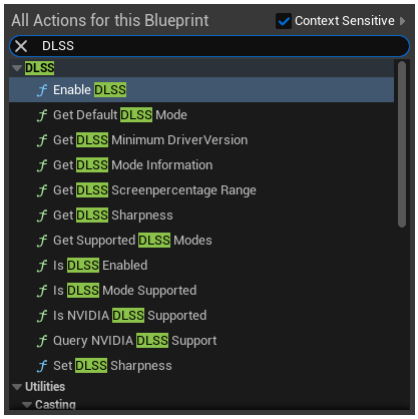
By default, the editor overrides the screen percentage when running in PIE. This can interfere with application logic that changes the screen percentage to set a particular DLSS quality level. To allow the application to set screen percentage, go to `Edit -> Editor Preferences -> Performance` and turn off "Override game screen percentage settings with editor settings in PIE". Restart the editor after changing this setting.



DLSS Blueprints

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

Using the UDLSSLibrary via blueprint or C++ (by including the DLSSBlueprint 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 DLSS plugin, without having to update the game logic.

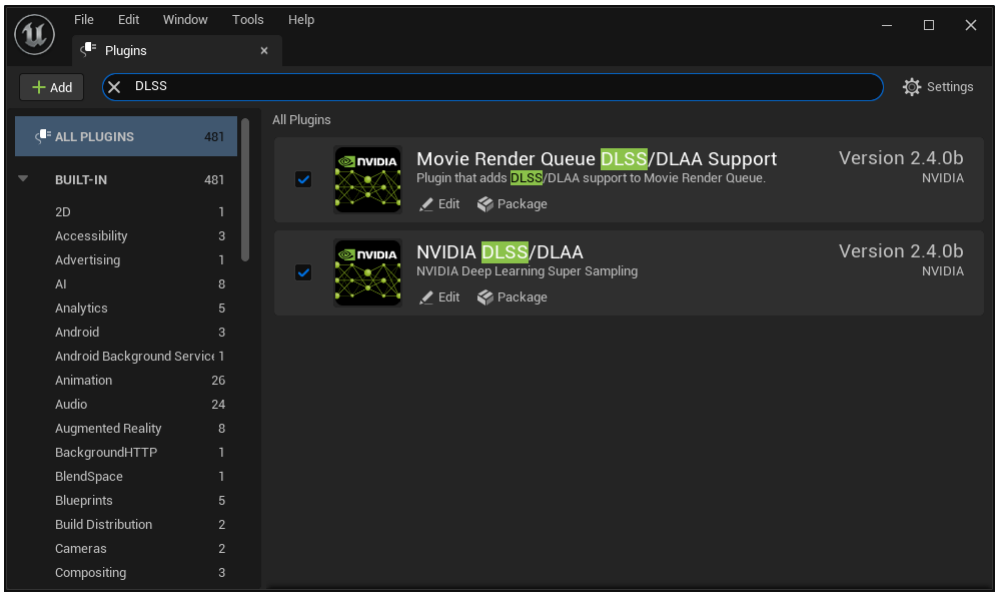


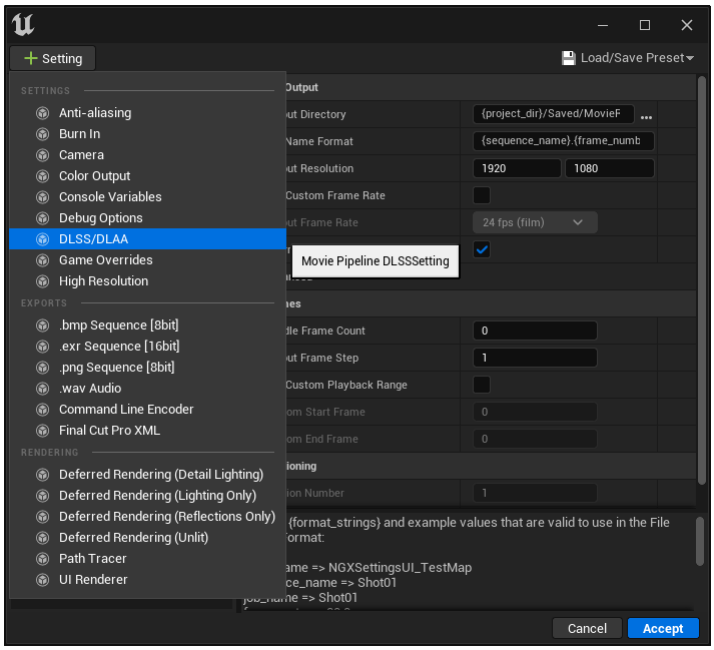
DLSS Movie Render Queue Support

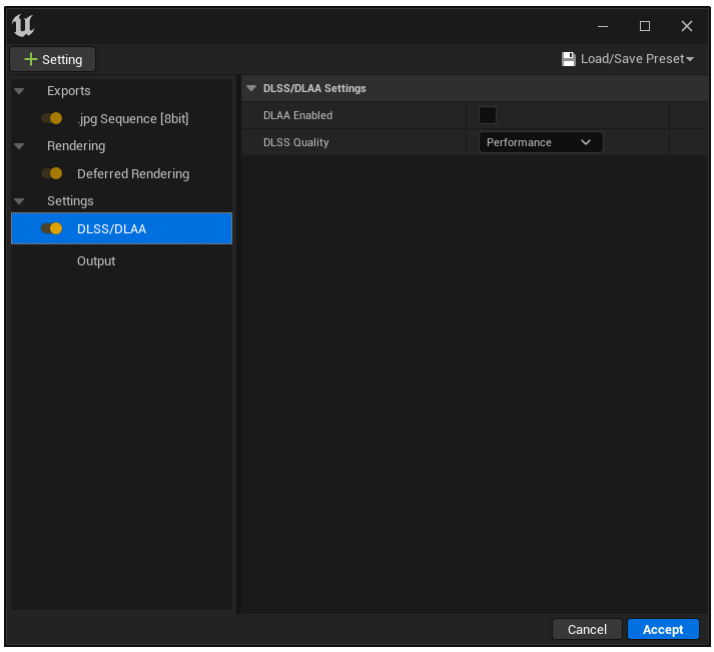
DLSS is supported when rendering movies with the Movie Render Queue plugin.

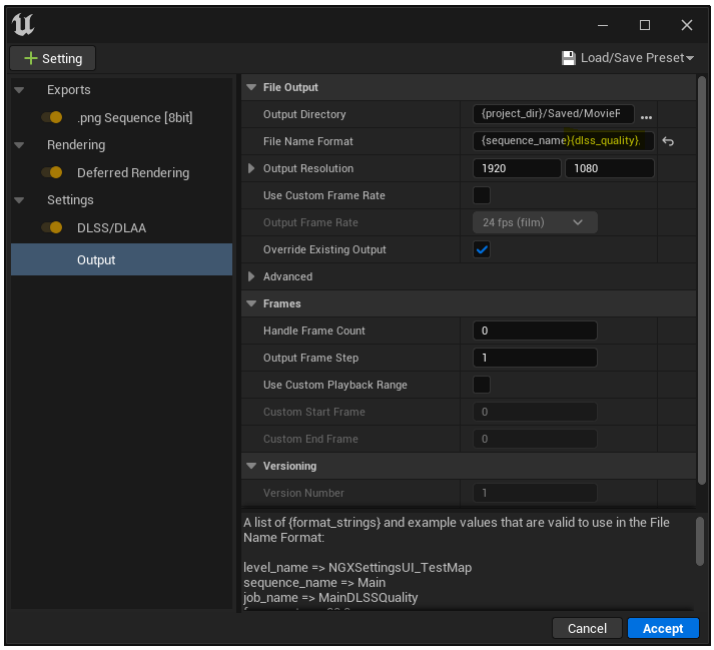
0. Enable the *Movie Render Queue* and *DLSS* plugins in the Editor
1. Enable the *Movie Render Queue DLSS Support* plugin in the Editor, then restart the editor
2. In the configuration, add the `Settings -> DLSS` page
3. In the DLSS settings page, change the desired DLSS quality mode
    1. Note: Unsupported DLSS modes will show a warning at the bottom of the window
4. Optional: The `Settings -> Output -> File Name Format` page supports a `{dlss_quality}` format tag

Note: Only the *Deferred Rendering* render pass is supported with DLSS, all other passes use the built-in TAA


File
Edit
Window
Tools
Help
Plugins
Add
DLSS
Settings
ALL PLUGINS 481
BUILT-IN 481
2D 1
Accessibility 3
Advertising 1
AI 8
Analytics 5
Android 3
Android Background Servic 1
Animation 26
Audio 24
Augmented Reality 8
BackgroundHTTP 1
BlendSpace 1
Blueprints 5
Build Distribution 2
Cameras 2
Compositing 3
All Plugins
Movie Render Queue DLSS/DLAA Support
Plugin that adds DLSS/DLAA support to Movie Render Queue.
Version 2.4.0b
NVIDIA
Edit
Package
NVIDIA DLSS/DLAA
NVIDIA Deep Learning Super Sampling
Version 2.4.0b
NVIDIA
Edit
Package


Setting
Load/Save Preset
SETTINGS
Anti-aliasing
Burn In
Camera
Color Output
Console Variables
Debug Options
DLSS/DLAA
Game Overrides
High Resolution
EXPORTS
.bmp Sequence [8bit]
.exr Sequence [16bit]
.png Sequence [8bit]
.wav Audio
Command Line Encoder
Final Cut Pro XML
RENDERING
Deferred Rendering (Detail Lighting)
Deferred Rendering (Lighting Only)
Deferred Rendering (Reflections Only)
Deferred Rendering (Unlit)
Path Tracer
UI Renderer
Movie Pipeline DLSSSetting
Cancel
Accept


Setting
Load/Save Preset
Exports
.jpg Sequence [8bit]
Rendering
Deferred Rendering
Settings
DLSS/DLAA
Output
DLSS/DLAA Settings
DLAA Enabled
DLSS Quality
Performance
Cancel
Accept



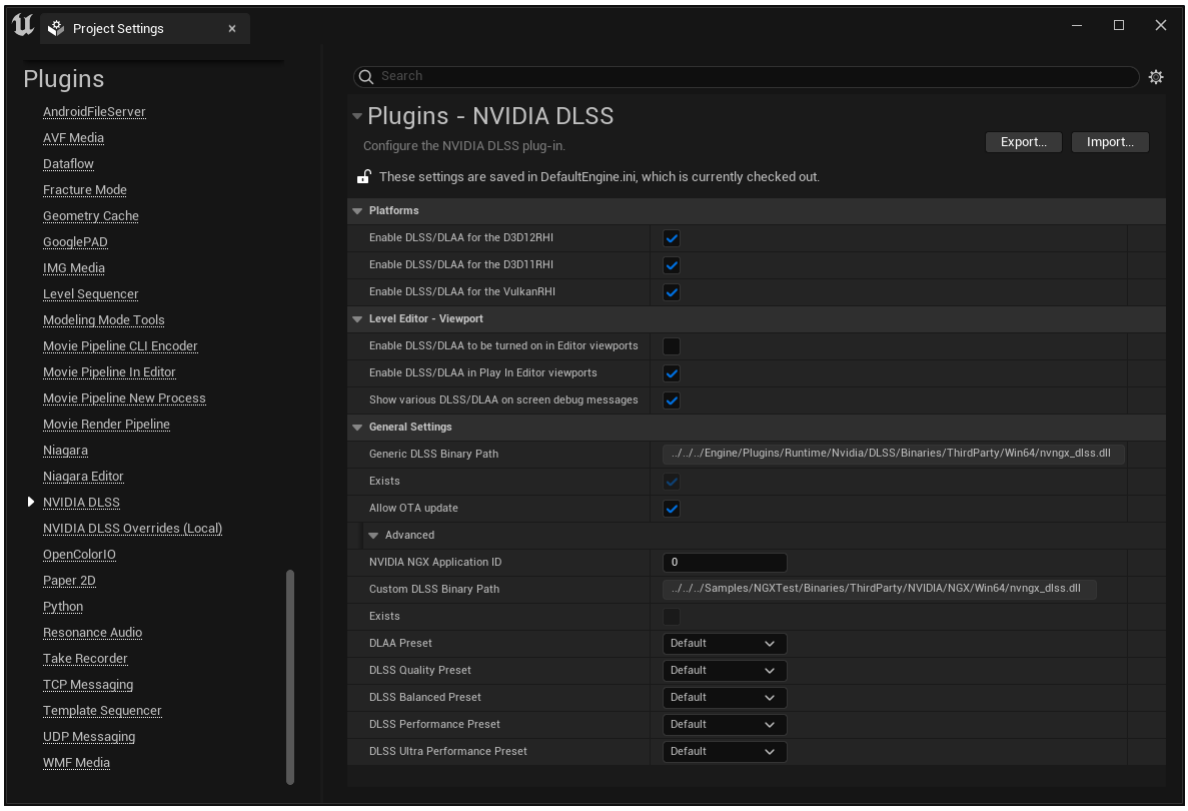
Distributing DLSS

The DLSS plugin ships with a ready-to-use production DLSS binary (without watermarks) and uses the project identifier to initialize NGX and DLSS. This is the common case for distribution to end users and does not require further actions from either your or NVIDIA's side. On rare occasion NVIDIA however might provide:

1. a custom project specific DLSS binary
2. an NVIDIA application ID

In that case those can be configured in the advanced plugin settings. Additionally please also ensure that the r.NGX.ProjectIdentifier console variable is set to either 0 (the default) or 2. The project plugin settings can be used to configure those (please see above).

1. The custom, project specific DLSS binary `nvngx_dlss.dll` should be put into the project under `$(ProjectDir)/Binaries/ThirdParty/NVIDIA/NGX/$(Platform)`
2. Setting the NVIDIA NGX application ID for the project.



Please refer to "Chapter 4 Distributing DLSS in a Game" in the the DLSS Programming Guide for details.

Upgrading from DLSS plugin versions for earlier engines

In general, it is strongly recommended to use the DLSS on-screen indicator when upgrading from engine versions before 5.1 to verify that you are still getting DLSS and DLAA enabled properly in your application.

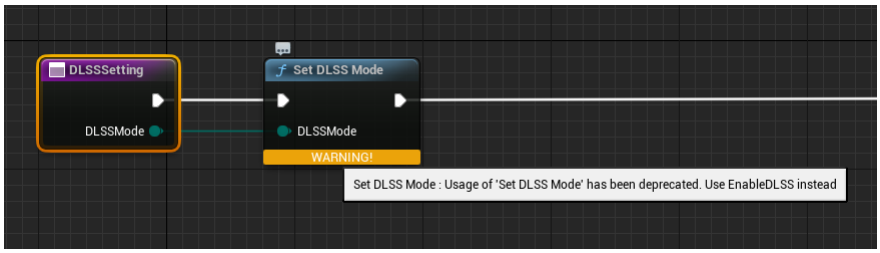
Screen percentage

Before Unreal Engine 5.1, the DLSS plugin controlled upscaling screen percentage directly, overriding anything set through the `r.ScreenPercentage` cvar or other methods. In Blueprints or in code, the application would request a particular DLSS quality mode from the plugin, and the plugin would internally set the quality mode and the appropriate screen percentage.

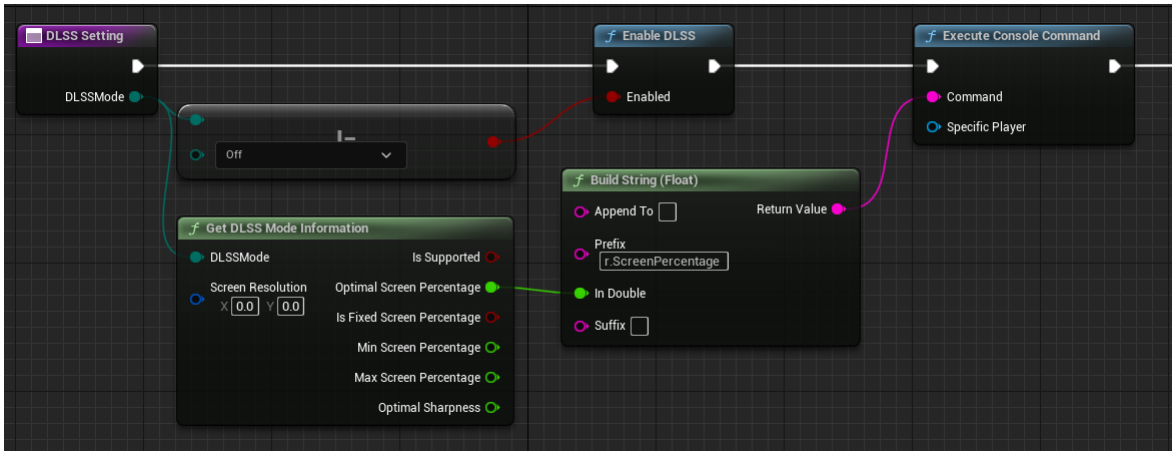
Starting with Unreal Engine 5.1, plugins do not override the upscaling screen percentage. So now to get a particular DLSS quality mode, the application must set the screen percentage corresponding to the optimal value for the desired DLSS quality mode. If DLSS is enabled, the plugin will internally set the best quality mode for the current screen percentage. GetDLSSModeInformation can be used from blueprints or from C++ to find the optimal screen percentage for a given DLSS quality mode. For DLAA, set a screen percentage of 100%.

The old SetDLSSMode and EnableDLAA blueprint functions have been deprecated and should be replaced with EnableDLSS, and then the appropriate screen percentage can be set by executing the `r.ScreenPercentage` console command from blueprints or code.

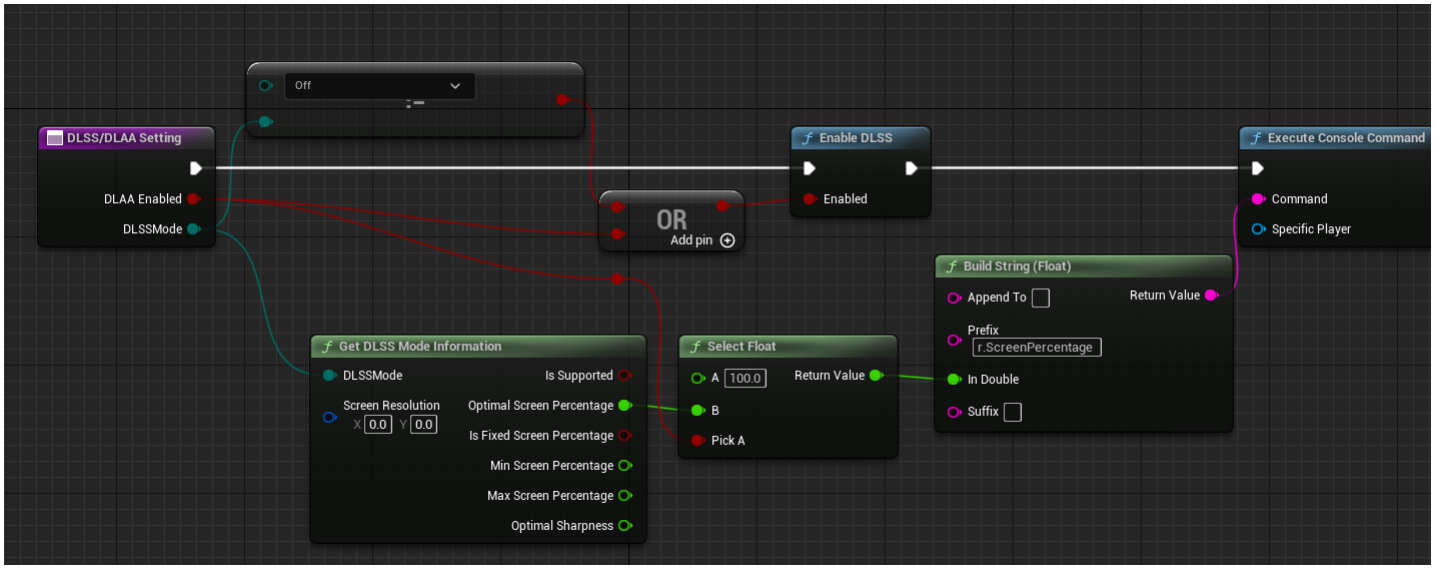
Before updating, you will see deprecation warnings for the old blueprint functions.



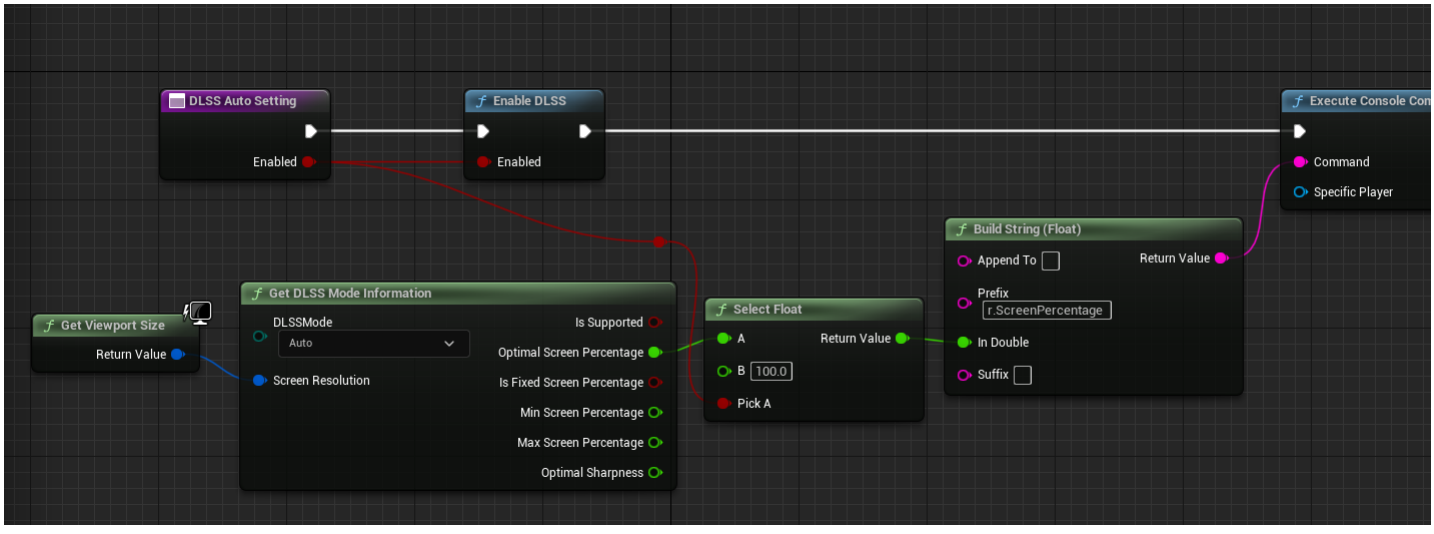
To offer the same DLSS quality mode options as before, check the optimal upscaling screen percentage for a quality mode and use the "r.ScreenPercentage" console command to set it.



DLAA is enabled by setting screen percentage to 100.



If you want your application to offer DLSS Off/Auto options instead of selecting individual DLSS quality modes, you can supply the screen resolution along with the fake "Auto" quality mode to GetDLSSModeInformation and a recommended screen percentage will be provided.



If your application changed `r.ScreenPercentage` before upgrading to Unreal Engine 5.1 or later, you may find that DLSS/DLAA behaves differently after upgrading. This is because before engine version 5.1, the DLSS plugin was

able to override the screen percentage value when DLSS or DLAA were enabled. When using the DLSS on-screen indicator, if you see that the upscale resolution doesn't change when changing DLSS modes, check for any blueprints or code that may be setting `r.ScreenPercentage` directly.

DLSS sharpening

The DLSS sharpening feature has been deprecated. It's recommended to use the NVIDIA Image Scaling (NIS) plugin for sharpening instead.

Performance benefits of DLSS-SR

Unreal Engine 5 scales the input resolution of the rendered scene automatically according to the current output resolution, ranging from 50 to 100%. Since all upscalers use the same automatic input resolution, performance should be similar across the available options. DLSS-SR has the advantage of delivering the best overall image quality thanks to the AI and deep learning techniques used to upscale and enhance the output. As a result, DLSS-SR has the ability to support resolutions below 50% with a 33% input resolution "Ultra Performance" mode to deliver significant performance gains while still preserving image quality.

DLSS API and UI Documentation

The DLSS Programming Guide provides details about the NVIDIA NGX APIs which are used by the plugin to implement DLSS.

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

The NVIDIA Developer Blog Tips: Getting the Most out of the DLSS Unreal Engine 4 Plugin provides best practices along with other tips and tricks to use NVIDIA DLSS in Unreal Engine games and applications.