



DLSS-RR Integration Guide

API Reference Manual

Document History

SWE-DLSS-001-PGRF

Version	Date	Authors	Description of Change
01	August 2023	abhelande, cwilkinson	Initial release
02	October 2023	abhelande, cwilkinson	Added Transparency Overlay API (3.4.10)
03	March 2024	abhelande, cwilkinson	<ul style="list-style-type: none">> Added Render Presets (3.12, 3.12.1)> Added support for Ultra-Performance and DLAA to existing Performance Modes (3.2)
04	August 2024	abhelande, cwilkinson	<ul style="list-style-type: none">> Use right technology name ' Ray Reconstruction'> Recommend default Render Preset in most cases.
05	January 2025	abhelande	<ul style="list-style-type: none">> Update VRAM + Speed for Transformer Model> Mark Presets A, B, C as deprecated (D is default)
05	February 2025	abhelande	<ul style="list-style-type: none">> Removed 'NVIDIA Confidential' from all Footers> Chapter 2 - remove Titan, Quadro> 3.1 Add note that DoF needs to absolutely be after RR otherwise use guide buffer> 3.3 Correction for DRS Not supported> 3.4.2 Expanded to 'this is the average specular reflectivity given a view direction.'> 3.4.7 Expand on Depth types that are supported> 3.4.8.1 parameter is plnMotionVectorsReflection. Remove "Note: This is expected to change to plnMotionVectorReflections. So we ask integrations to set both."> 3.4.11 Added Screen Space Sub Surface Scattering Guide> 3.4.12 Added Depth of Field Guide> 3.5 Added Section "Sampling and Noise"> 3.6 Added notes to DLSS RR recommendation for Sub pixel Jitter pattern> 5.1 remove mention ofnvsdk_ngx_dlssd_d3d.h
06	June 2025	abhelande	<ul style="list-style-type: none">> 3.13 Changed deprecation note to reflect that Presets A-C will be removed in next SDK> 5.5 Added Section for Querying Optimal Settings> 3.4.9 Added note: All matrices are Row Major Order and use left multiplication> 3.4.11 Added Color Before Transparency Guide> 8.1 Corrections to Debug Overlay section
07	August 2025	abhelande	<ul style="list-style-type: none">> 3.7 Corrected error that Presets are not supported> 3.13 Remove Presets A to C

Table of Contents

Abstract	6
Chapter 1. Introduction.....	7
Chapter 2. Getting Started.....	8
2.1 Querying for DLSS Ray Reconstruction Support on End User System and Getting the Latest Updates	8
2.2 DLSS-RR Execution Times and GPU RAM Usage	9
Chapter 3. DLSS-RR Integration.....	10
3.1 Pipeline Placement	10
3.2 DLSS-RR Execution Modes.....	11
3.3 Dynamic Resolution Support	11
3.4 Supported Resource Formats	11
3.4.1 Diffuse Albedo.....	11
3.4.1.1 NGX Parameter Name	11
3.4.1.2 SL Buffer Type	12
3.4.2 Specular Albedo.....	12
3.4.2.1 NGX Parameter Name	12
3.4.2.2 SL Buffer Type	12
3.4.3 Normals.....	12
3.4.3.1 NGX Parameter Name	12
3.4.3.2 SL Buffer Type	12
3.4.3.3 SL Parameter Notes	12
3.4.4 Roughness.....	12
3.4.4.1 NGX Parameter Name	13
3.4.4.2 NGX Parameter Notes	13
3.4.4.3 SL Buffer Type	13
3.4.4.4 SL Parameter Notes	13
3.4.5 Color Input	13
3.4.5.1 NGX Parameter Name	13
3.4.5.2 SL Buffer Type	13
3.4.6 Motion Vectors.....	13
3.4.6.1 NGX Parameter Name	13
3.4.6.2 SL Buffer Type	14
3.4.7 Depth Buffer	14
3.4.7.1 NGX Parameter Name	14
3.4.7.2 SL Buffer Type	14
3.4.8 Specular Motion Vector Reflections.....	14

3.4.8.1	NGX Parameter Name	14
3.4.8.2	SL Buffer Type	14
3.4.9	Specular Hit Distance	14
3.4.9.1	NGX Parameter Name	15
3.4.9.2	SL Buffer Type	15
3.4.10	Transparency Overlay	15
3.4.10.1	NGX Parameter Name	15
3.4.10.2	SL Buffer Type	15
3.4.11	Color Before Transparency Guide	15
3.4.11.1	NGX Parameter Name	16
3.4.11.2	SL Buffer Type	16
3.4.12	Screen Space Sub Surface Scattering Guide	16
3.4.12.1	NGX Parameter Name	16
3.4.12.2	SL Buffer Type	16
3.4.13	Depth of Field Guide	16
3.4.13.1	NGX Parameter Name	17
3.4.13.2	SL Buffer Type	17
3.4.14	Output	17
3.4.14.1	NGX Parameter Name	17
3.4.14.2	SL Buffer Type	17
3.5	Sampling and Noise	17
3.6	Sub-Pixel Jitter	18
3.7	Exposure, Auto-Exposure, Sharpness, and Presets	18
3.8	Scene Transitions	19
3.9	VRAM Usage	19
3.10	Current Frame Bias and VR Support	19
3.11	Current DLSS-RR Settings	19
3.12	DLSS-RR Logging	19
3.13	DLSS-RR Presets	19
3.13.1	Preset Selection Behavior with OTA Updates	21
3.14	Integrating DLSS-RR Using Streamline SDK	21
Chapter 4.	Distributing DLSS-RR in a Game	22
4.1	DLSS-RR Release Process	22
4.2	Distributable Libraries	22
Chapter 5.	DLSS-RR Code Integration	23
5.1	Adding DLSS-RR to a Project	23
5.1.1	Linking the NGX SDK	24
5.2	NGX SDK Initialization	24
5.3	Feature Creation	24
5.4	Feature Evaluation	26

5.5	Querying Optimal Settings.....	27
5.6	Other Feature APIs	27
Chapter 6.	Resource Management.....	28
Chapter 7.	Multi-GPU Support	29
Chapter 8.	Troubleshooting.....	30
8.1	DLSS-RR Debug Overlay	30
8.2	Error Codes	30
Appendix	31	
	Specular Albedo Calculation	31
Third Party Software		32
	Linux Driver Compatibility	32

List of Tables

Table 1. Allocated Memory for GPU Generation.....	9
Table 2. GeForce GPU in Performance Mode	9

Abstract

The DLSS-RR Integration Guide provides details on how to integrate and distribute DLSS-RR in a game or 3D application that Renders Content via Ray Tracing or Path Tracing pipelines. The guide also provides embedded sample code and links to a full sample implementation on GitHub.

This programming guide is an addendum to the *DLSS Programming Guide*, and it is highly recommended that you read through that first.

For information on using DLSS in Unreal Engine please see <https://www.unrealengine.com/marketplace/en-US/product/nvidia-dlss>



Note: Previous iterations of this document referred to DLSS-D for DLSS-Denoising. This has been renamed to DLSS Ray Reconstruction, or DLSS-RR.

This integration guide refers several times to the *DLSS Programming Guide*. For reference, the *DLSS Programming Guide* can be found at the following location:

https://github.com/NVIDIA/DLSS/blob/main/doc/DLSS_Programming_Guide_Release.pdf

Chapter 1. Introduction

DLSS-Ray Reconstruction (DLSS-RR) is a unified denoise + upscale model that can denoise any raytracing or path tracing signal without time-consuming fine tuning, and uses state-of-the-art temporal upscaling, all accelerated with RTX TensorCores.

Chapter 2. Getting Started

The following is needed to load and run DLSS for both Windows and Linux:

- > NVIDIA RTX GPU (GeForce)
- > The latest NVIDIA Graphics Driver is recommended. At a minimum for running DLSS Ray Reconstruction (DLSS 3.5+), you must have NVIDIA driver issued after August 23, 2023 (for instance 535.x).
- > Please see [Linux Driver Compatibility](#) information.

The following is needed to load and run DLSS Ray Reconstruction on Windows:

- > Windows PC with Windows 10 v1709 (Fall 2017 Creators Update 64-bit) or newer.
- > The development environment for integrating the DLSS Ray Reconstruction SDK into a game is:
 - Microsoft Visual Studio 2015 or newer.
 - Microsoft Visual Studio 2012 and 2013 are supported but may be deprecated in the future.

The following is needed to load and run DLSS Ray Reconstruction on Linux:

- > DLSS Ray Reconstruction SDK: glibc 2.11 or newer
- > DLSS Ray Reconstruction SDK Sample Code: gcc and g++ 8.4.0 or newer

2.1 Querying for DLSS Ray Reconstruction Support on End User System and Getting the Latest Updates

Please refer to sections 2.3 (Querying for DLSS Support on End User System) and 2.4 (Getting the Latest Updates) of the *DLSS Programming Guide*. Querying for DLSS-RR feature support, Vulkan Extensions and Getting Latest DLSS Ray Reconstruction updates is the same as those DLSS sections with the following differences:

Set **FeatureID** to *NVSDK NGX_Feature_RayReconstruction*.

FeatureInfo is Information Common to all NGX Features. Currently this is a list of paths that NGX can scan to find Feature specific DLLs. In the case of DLSS Ray Reconstruction, it is `nvngx_d1ssd.dll`.

2.2 DLSS-RR Execution Times and GPU RAM Usage

The exact execution time of DLSS-RR varies from engine to engine and is dependent on the integration. Factors such as how the engine memory manager works and whether additional buffer copies are required can affect the final performance.

As a rough guide, NVIDIA ran the DLSS-RR library using a command line utility (i.e., without a 3D renderer) across the full range of NVIDIA GeForce RTX GPUs and provides these results as a rough estimate of expected execution times. Developers can use these figures to estimate the potential savings DLSS provides.

Table 1. Allocated Memory for GPU Generation for Presets Default, D, E

	1920x1080	2560x1440	3840x2160
RTX 50 Series	121.09 MB	212.03 MB	467.97 MB
RTX 40 Series	121.09 MB	212.03 MB	467.97 MB
RTX 30 Series	163.21 MB	284.57 MB	623.64 MB
RTX 20 Series	163.21 MB	284.57 MB	623.64 MB

The following shows average expected DLSS-RR times where the DLSS-RR algorithm is executed in “Performance Mode” where the input is half the number of pixels as the output.

Table 2. Speed on GeForce GPU in Performance Mode for Presets Default, D, E

	1920x1080	2560x1440	3840x2160
RTX 2080 Ti	4.80 ms	8.20 ms	18.33 ms
RTX 3060	5.86 ms	10.39 ms	23.28 ms
RTX 3070	3.42 ms	6.06 ms	13.31 ms
RTX 3080 Ti	2.33 ms	3.97 ms	8.83 ms
RTX 4070 Ti	1.21 ms	2.09 ms	4.53 ms
RTX 4080	0.97 ms	1.66 ms	3.62 ms
RTX 4090	0.71 ms	1.11 ms	2.35 ms
RTX 5070	1.50 ms	2.58 ms	5.49 ms
RTX 5080	0.91 ms	1.51 ms	3.17 ms
RTX 5090	0.60 ms	0.91 ms	1.83 ms

Chapter 3. DLSS-RR Integration

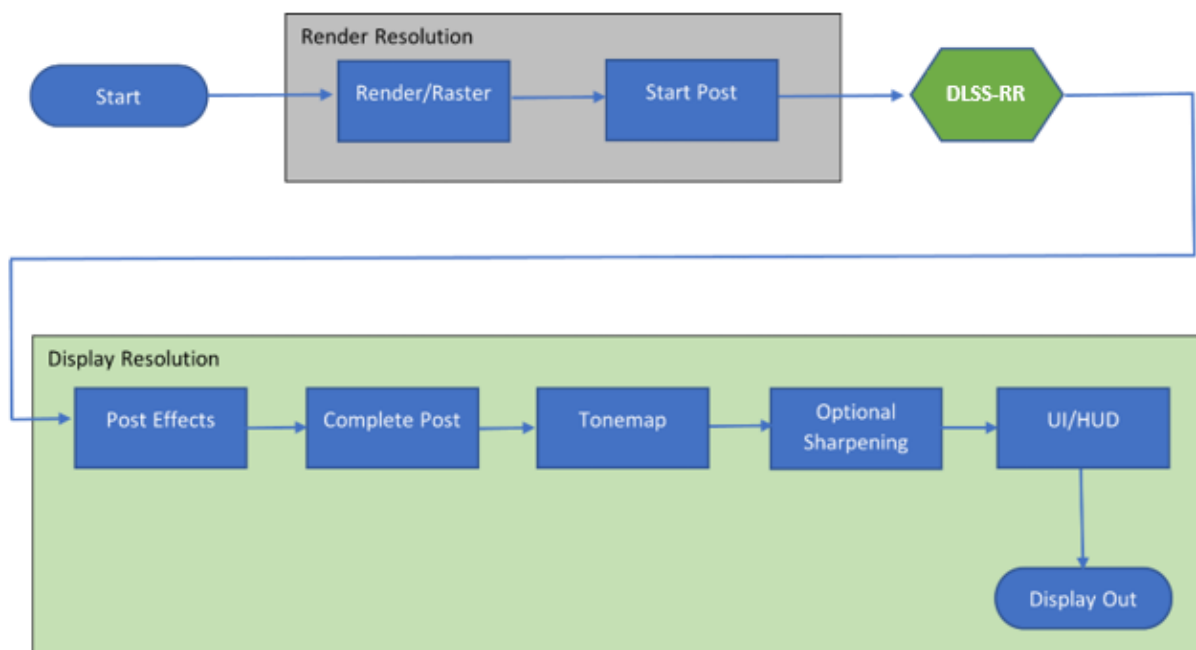
3.1 Pipeline Placement

Similar to DLSS-Super Resolution, the DLSS-Ray Reconstruction evaluation call must occur during the post-processing phase of the rendering pipeline, before tone mapping. For the best image quality, place DLSS-RR as close to the start of post-processing as possible (before any effects, or as few effects as possible, are applied to the frame).



Note: In particular Depth of Field absolutely needs to be after DLSS-RR. Alternatively use the latest preset - E and provide the guide buffer outlined in section

Once DLSS-RR is placed at the start of post-processing, all post effects must be able to handle the increased resolution and not have assumptions that processing can only occur at render resolution.



3.2 DLSS-RR Execution Modes

DLSS-Ray Reconstruction is an extension to DLSS, where-in the execution mode it operates in is the same as that set for DLSS-SR.

Integration of DLSS-RR requires a DLSS-SR style integration with an added toggle option to switch to using DLSS-RR. When DLSS-RR is enabled, it effectively overrides DLSS-SR execution (in other words, DLSS-SR is no longer being executed). Please consult the RTX UI Developer Guidelines for more information.

The input resolutions that the game should render and send to DLSS-RR for processing are determined by querying the “DLSS Optimal Settings” for each of the “PerfQualityValue” options. DLSS-RR currently supports the following DLSS “PerfQualityValues”:

1. Performance Mode
2. Balanced Mode
3. Quality Mode
4. Ultra-Performance Mode
5. DLAA (Deep Learning Anti-Aliasing) Mode

For more details on DLSS Optimal Settings, see section 5.2.8 of the *DLSS Programming Guide*.

3.3 Dynamic Resolution Support

Dynamic Resolution Support is not currently supported by DLSS Ray Reconstruction.

3.4 Supported Resource Formats

DLSS-RR uses formatted reads, therefore it should handle most input buffer formats and expects the following buffers as listed in the Evaluation parameter listing in [Feature Evaluation](#):

3.4.1 Diffuse Albedo

This is the diffuse component of Reflectance material. Any standard 3-channel format is provided at input resolution. Note that at this time sRGB formats are not supported.

3.4.1.1 NGX Parameter Name

pInDiffuseAlbedo

3.4.1.2 SL Buffer Type

`kBufferTypeAlbedo`

3.4.2 Specular Albedo

The specular component of Reflectance material which is the average specular reflectivity given a view direction. Any standard 3-channel format is provided at input resolution. Note that at this time sRGB formats are not supported.

A common integration error is to leave sky pixels uncleared. A good default albedo for sky is (0.5, 0.5, 0.5). See chapter 8 for tips how to verify the correctness of the guide buffers. Refer to the [Appendix Section](#) for instructions on how to generate this buffer.

3.4.2.1 NGX Parameter Name

`pInSpecularAlbedo`

3.4.2.2 SL Buffer Type

`kBufferTypeSpecularAlbedo`

3.4.3 Normals

Shading Normals (Normalized). Can be View Space or World Space. RGB16_FLOAT or RGB32_FLOAT provided at input resolution.

3.4.3.1 NGX Parameter Name

`pInNormals`

3.4.3.2 SL Buffer Type

`kBufferTypeNormalRoughness, kBufferTypeNormals`

3.4.3.3 SL Parameter Notes

If `kBufferTypeNormalRoughness` is used, you must set the `normalRoughnessMode` of `DLSSOptions` to `sl::DLSSDNormalRoughnessMode::ePacked`.

Otherwise, if Roughness is packed into the Alpha channel of the Normals use `sl::DLSSDNormalRoughnessMode::eUnPacked`.

3.4.4 Roughness

Linear Roughness of surface material is provided at input resolution. As a standalone texture, you are encouraged to use a single channel format. Otherwise, it should be

written into the R channel of that texture. Alternatively, you can pack Roughness into the Alpha channel of the Normals.

3.4.4.1 NGX Parameter Name

`pInRoughness`

3.4.4.2 NGX Parameter Notes

When packing Roughness into the Alpha channel of the Normals, you must set the `InRoughnessMode` parameter of the `NVSDK NGX_DLSSD_Create_Params` at Feature Creation to `NVSDK NGX_DLSS_Roughness_Mode_Packed`.

3.4.4.3 SL Buffer Type

`kBufferTypeRoughness`, `kBufferTypeNormalRoughness`

3.4.4.4 SL Parameter Notes

If using `kBufferTypeNormalRoughness`, you must set the `normalRoughnessMode` of `DLSSOptions` to `sl::DLSSDNormalRoughnessMode::ePacked`.

3.4.5 Color Input

This is the Noisy Ray Traced Input Color. Any standard 3-channel format is provided at input resolution.

3.4.5.1 NGX Parameter Name

`pInColor`

3.4.5.2 SL Buffer Type

`kBufferTypeScalingInputColor`

3.4.6 Motion Vectors

This refers to the dense motion vector field. This includes, for example, camera motion or the motion of dynamic objects. `RG16_FLOAT` or `RG32_FLOAT` is provided at input resolution. (For more information, see section 3.6 of the *DLSS Programming Guide*).

3.4.6.1 NGX Parameter Name

`pInMotionVectors`

3.4.6.2 SL Buffer Type

`kBufferTypeMotionVectors`

3.4.7 Depth Buffer

This is the same depth data used to generate motion vector data (View Space Depth or HW Depth). Any format with one channel in it (for instance R32_FLOAT or D32_FLOAT), as well as any depth-stencil format (for instance D24S8). Either view-space depth or HW depth can be provided.

Provided at input resolution.

For more information see section 3.8 of the DLSS Programming Guide).

3.4.7.1 NGX Parameter Name

`pInDepth`. (Set `InUseHWDepth` creation parameter for HW Depth use.

3.4.7.2 SL Buffer Type

`kBufferTypeLinearDepth` or `kBufferTypeDepth`

3.4.8 Specular Motion Vector Reflections

DLSS-RR uses Specular Motion Vectors to improve image quality of reflections during motion. The application can either provide these directly or, alternatively, provide Specular Hit Distance with 1 and 2 matrices.

This refers to the dense motion vector field for Reflections (Virtually Reflected Geometries). For example, this could include camera motion or the motion of dynamic objects. RG16_FLOAT or RG32_FLOAT is provided at input resolution.

3.4.8.1 NGX Parameter Name

`pInMotionVectorsReflection`

3.4.8.2 SL Buffer Type

`kBufferTypeSpecularMotionVectors`

3.4.9 Specular Hit Distance

This is the World Space distance between the Specular Ray Origin and Hit Point. Specular Ray Origin must be on the Primary Surface. Floating Point Scalar Value (FP16, or FP32).

Additionally, the application needs to provide its World To View Matrix and View To Clip Space Matrix.



Note: This is only needed if Specular Motion Vectors are not provided.
All matrices are Row Major Order and use left multiplication

3.4.9.1 NGX Parameter Name

`pInSpecularHitDistance` with `pInWorldToViewMatrix`, `pInViewToClipMatrix`

3.4.9.2 SL Buffer Type

`kBufferTypeSpecularHitDistance`

3.4.10 Transparency Overlay

Optional only.

A buffer that has particles or other transparent effects rendered into it instead of passing it as part of the input color. Content in this buffer will not be denoised and upscaled only.

Single standard 4-channel input – where RGB must be premultiplied with Alpha, Alpha channel is the blending factor.

Or 2 separate standard 3-channel inputs - One representing color (RcGcBc), other representing alpha (RaGaBa)

Please also see Color Before Transparency Guide (section 3.4.11)

3.4.10.1 NGX Parameter Name

`pInTransparencyLayer`, `pInTransparencyLayerOpacity` (6 channel only)

3.4.10.2 SL Buffer Type

`kBufferTypeTransparencyLayer`, `kBufferTypeTransparencyLayerOpacity` (6 channel only)

3.4.11 Color Before Transparency Guide

Optional only.

Snapshot of noisy color buffer before any transparencies are rendered on top. The main purpose of this guide is to help with ghosting or disappearing particles. If this is not a problem in your application, this buffer is probably not needed.

It is important to use the same image format as for the noisy color input since small precision differences between the buffers may cause DLSS-RR to interpret the pixel as having transparencies, which can lead to artifacts under difficult lighting conditions.

In certain cases, rendering some transparent objects into the guide may improve image quality. If this guide is needed in the first place, the recommendation is to start with just opaque pixels, and if there are image quality problems, try adding the problematic transparent objects to the guide.

3.4.11.1 NGX Parameter Name

`pInColorBeforeTransparency`

3.4.11.2 SL Buffer Type

`kBufferTypeColorBeforeTransparency`

3.4.12 Screen Space Sub Surface Scattering Guide

Optional only.

In general the buffer should contain $\text{luminance}(\text{colorAfterSSS} - \text{colorBeforeSSS})$.

If SSS blur is computed based on diffuse light, diffuse albedo should be applied before computing luminance, in

which case the formula used should be:

$\text{luminance}((\text{diffuseAfterSSS} - \text{diffuseBeforeSSS}) * \text{diffuseAlbedo})$

where $\text{luminance}(s) = (s.r + (2 * s.g) + s.b) / 4$

Pixels without SSS material must have 0 in the guide.

1-channel FP16 format provided at input resolution.

3.4.12.1 NGX Parameter Name

`pInScreenSpaceSubsurfaceScatteringGuide`

3.4.12.2 SL Buffer Type

`kBufferTypeScreenSpaceSubsurfaceScatteringGuide`

3.4.13 Depth of Field Guide

Optional only – and only compatible with Preset E.

Input buffer for specifying Depth of Field (DOF) guide.

In general the buffer should contain luminance($\text{colorAfterDOF} - \text{colorBeforeDOF}$), where $\text{luminance}(s) = (s.r + (2 * s.g) + s.b) / 4$

Pixels without DOF must have 0 in the guide.

Alternatively, depth of field can be applied after DLSS-RR, in which case this buffer can be omitted.

1-channel FP16 format provided at input resolution.

3.4.13.1 NGX Parameter Name

`pInDepthOfFieldGuide`

3.4.13.2 SL Buffer Type

`kBufferTypeDepthOfFieldGuide`

3.4.14 Output

Output refers to the destination for the denoised, full-resolution frame. Any standard 3 or 4-channel format is provided at output resolution.

3.4.14.1 NGX Parameter Name

`pInOutput`

3.4.14.2 SL Buffer Type

`KBufferTypeScalingOutputColor`

3.5 Sampling and Noise

DLSS RR assumes independent samples and requires that sampling used to generate Inputs must have minimal correlation both spatially and temporally

- Do use high quality hash functions and white noise
 - Correlations should be minimized between all lower-dimensional projections of the random samples in the path space.
 - Any hash function with low number of ‘bigcrush’ failures is recommended (<https://jcgf.org/published/0009/03/02/>)
 - xxhash32 is a good example
 - In general the sampling should allow the resulting image to converge to a noise free sampling (Ground truth).

- The following practices are to be avoided
 - Use of checkerboard rendering
 - Do Not use hash functions with correlated sampling
 - Use of screen space dithering in the sampling algorithms
 - Sharing of sampling patterns across the screen
 - This however does not apply to initial pixel jitter as DLSS RR is given this as input separately.
 - Blue noise sampling may be fine as long as the period is large enough and adheres to correlation requirement above.
- Reduction of noise in the Input signals is very beneficial
 - ReSTIR DI and ReSTIR GI are excellent and recommended techniques
 - Reduction of noise is especially beneficial in difficult lighting situations.
- When using ReSTIR GI or RTX DI
 - Randomize the temporal reuse step
 - RR assumes independent samples, which is violated by ReSTIR temporal and spatial reuse.
 - Permutation sampling helps avoid correlation artifacts. The basic idea is to randomize the reused sample location so that the resulting set of reservoirs has as much spatial and temporal variation as possible (ie. avoid using the same samples for neighboring pixels' reservoirs).

3.6 Sub-Pixel Jitter

Please refer to section 3.7 (Sub-Pixel Jitter) of the *DLSS Programming Guide*.

In addition, for DLSS-RR – There is no reason to limit the number of samples. Use of many more jitter positions (at least 32) is also highly recommended.

3.7 Exposure, Auto-Exposure, Sharpness

These are not supported by DLSSRay Reconstruction. In the *DLSS Programming Guide* for DLSS-RR Integrations, please ignore sections 3.9 (Exposure Parameter), 3.10 (Auto Exposure), 3.11 (Additional Sharpening).

3.8 Scene Transitions

Just like DLSS-SR, DLSS-RR provides the `InReset` parameter to the `DLSSD_Eval` parameter list.

Please refer to section 3.13 (Scene Transitions) of the *DLSS Programming Guide* for more details.

3.9 VRAM Usage

Please refer to section 3.14 (VRAM Usage) of the *DLSS Programming Guide*.

3.10 Current Frame Bias and VR Support

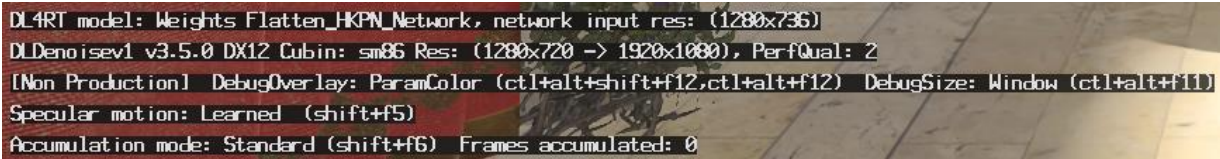
Please refer to sections 3.15 (Biasing the Current Frame) and 3.16 (Multi-View and Virtual Reality Support) of *DLSS Programming Guide*.

3.11 Current DLSS-RR Settings

These work in the same way as described in section 3.17 (Current DLSS Settings) of the *DLSS Programming Guide*.



Note: The overlay is different for DLSS-RR.

A screenshot of a game scene with a semi-transparent debug overlay at the bottom. The overlay contains the following text: "DL4RT model: Weights Flatten_HKPN_Network, network input res: (1280x736)", "DLDenoiseV1 v3.5.0 DX12 Cubin: sm86 Res: (1280x720 -> 1920x1080), PerfQual: 2", "[Non Production] DebugOverlay: ParamColor (ctrl+alt+shift+f12,ctrl+alt+f12) DebugSize: Window (ctrl+alt+f11)", "Specular motion: Learned (shift+f5)", and "Accumulation mode: Standard (shift+f6) Frames accumulated: 0".

DL4RT model: Weights Flatten_HKPN_Network, network input res: (1280x736)
DLDenoiseV1 v3.5.0 DX12 Cubin: sm86 Res: (1280x720 -> 1920x1080), PerfQual: 2
[Non Production] DebugOverlay: ParamColor (ctrl+alt+shift+f12,ctrl+alt+f12) DebugSize: Window (ctrl+alt+f11)
Specular motion: Learned (shift+f5)
Accumulation mode: Standard (shift+f6) Frames accumulated: 0

3.12 DLSS-RR Logging

DLSS-RR logging is built of the same NGX Logging utilities as described in section 3.18 (NGX Logging) of the *DLSS Programming Guide*.

3.13 DLSS-RR Presets

Presets tweak different aspects of DLSS RR to cater to different scaling ratios, game content, etc. DLSS RR presets are enabled on a per-scaling ratio basis. For example, to enable preset D for performance mode:

```
auto preset = NVSDK_NGX_RayReconstruction_Hint_Render_Preset::
NVSDK_NGX_Parameter_RayReconstruction_Hint_Render_Preset_D;

NVSDK_NGX_Parameter_SetUI(ngxParams,
NVSDK_NGX_RayReconstruction_Hint_Render_Preset_Performance, preset);
```



Note: As of NGX SDK 310.4.0, Presets A, B, and C have been removed.
Please use the default preset, or select Preset D or E.

The following are the Presets that can be set per Performance mode using the NVSDK_NGX_Parameter_SetUI API:

```
// These can be set via NVSDK_NGX_Parameter_SetUI
typedef enum NVSDK_NGX_RayReconstruction_Hint_Render_Preset
{
    NVSDK_NGX_RayReconstruction_Hint_Render_Preset_Default, // default
    behavior, weights may or may not be updated after OTA
    NVSDK_NGX_RayReconstruction_Hint_Render_Preset_D,
    NVSDK_NGX_RayReconstruction_Hint_Render_Preset_E,
} NVSDK_NGX_RayReconstruction_Hint_Render_Preset;

#define NVSDK_NGX_Parameter_RayReconstruction_Hint_Render_Preset_DLAA
"RayReconstruction.Hint.Render.Preset.DLAA"
#define NVSDK_NGX_Parameter_RayReconstruction_Hint_Render_Preset_Quality
"RayReconstruction.Hint.Render.Preset.Quality"
#define NVSDK_NGX_Parameter_RayReconstruction_Hint_Render_Preset_Balanced
"RayReconstruction.Hint.Render.Preset.Balanced"
#define NVSDK_NGX_Parameter_RayReconstruction_Hint_Render_Preset_Performance
"RayReconstruction.Hint.Render.Preset.Performance"
#define
NVSDK_NGX_Parameter_RayReconstruction_Hint_Render_Preset_UltraPerformance
"RayReconstruction.Hint.Render.Preset.UltraPerformance"
#define NVSDK_NGX_Parameter_RayReconstruction_Hint_Render_Preset_UltraQuality
"RayReconstruction.Hint.Render.Preset.UltraQuality"
```

Currently we recommend games stick to

NVSDK_NGX_RayReconstruction_Hint_Render_Preset_Default for all modes. The Develop DLL in the SDK will let you toggle between other presets for experimentation.

DLSS RR defaults and presets are subject to change with new versions of DLSS RR. Note that NVSDK_NGX_Parameter_RayReconstruction_Hint_Render_Preset_DLAA will work only when the input and output render sizes are set to the same value as described above.

3.13.1 Preset Selection Behavior with OTA Updates

Please refer to section 3.12.1 (Preset Selection Behavior with OTA Updates) of the *DLSS Programming Guide*.

3.14 Integrating DLSS-RR Using Streamline SDK

DLSS-RR can be integrated into any application through the Streamline SDK. You will need the following:

- > Streamline SDK
- > DLSS-RR Plugin for Streamline – `s1.dlss_d.dll` available in the Streamline SDK
- > DLSS-RR Binary – a compatible `nvngx_dlssd.dll` which can be used by `s1.dlss_d.dll`

The DLSS-RR Plugin for Streamline (`s1.dlss_d.dll`) serves as a wrapper for `nvngx_dlssd.dll`. Refer to the Streamline SDK package included with DLSS-RR.

Chapter 4. Distributing DLSS-RR in a Game

NVIDIA has encapsulated the DLSS-RR technology to ensure that developers can use the functionality with minimal build or packaging changes. Follow the steps in this section to include DLSS-RR in game or application builds.

4.1 DLSS-RR Release Process

NVIDIA hosts DLSS publicly for all development partners. To ensure the best quality experience for end-users, we encourage developers to thoroughly test DLSS on different DLSS modes, output resolutions, and varying game content.

4.2 Distributable Libraries

The release library for DLSS is located in the `./lib/_/rel/` folder of the GitHub repository. Ensure that after game installation, the file resides in the same folder as the game executable (or DLL if you are building a plugin).



Important: Both the development and release DLLs (located in `./lib/_/dev/`, `./lib/_/rel/`) included in the NDA SDK package **must not** be included or distributed in any public release. These libraries include on-screen notices and have an overlay designed for debugging only. *To obtain a distributable DLL, please contact NVIDIA.*

NGX AI-Powered Feature	NGX Feature DLL
DLSS-RR	<code>nvngx_dlssd.dll</code> / <code>libnvidia-ngx-dlssd.so</code>



Note: The DLSS-RR library can be loaded from an alternate directory path if required. The entry point, `NVSDK_NGX_D3D12_Init_ext`, accepts a parameter, `InFeatureInfo` which has a `NVSDK_NGX_PathListInfo` item that contains a list of paths to search through. For more information, see sections 5.1 and 5.2 of the *DLSS Programming Guide*.

For other notes, please refer to sections 4.2.1 (Removing the DLSS Library), 4.2.2 (Signing the DLSS Library), 4.2.3 (Note of inclusion of third-party code) in the *DLSS Programming Guide*.

Chapter 5. DLSS-RR Code Integration

5.1 Adding DLSS-RR to a Project

The NGX DLSS-RR SDK includes following header files:

- > `nvsdk_ngx.h`
- > `nvsdk_ngx_defs.h`
- > `nvsdk_ngx_dlssd.h`
- > `nvsdk_ngx_dlssd_vk.h`
- > `nvsdk_ngx_helpers.h`
- > `nvsdk_ngx_helpers_vk.h`
- > `nvsdk_ngx_params.h`
- > `nvsdk_ngx_vk.h`
- > `nvsdk_ngx_cuda.h`
- > `nvsdk_ngx_helpers_dlssd_cuda.h`

The header files are located in the `./include` folder. In the game project, include `nvsdk_ngx_helpers.h`.



Note: Vulkan headers follow the above naming convention with a “vk” suffix and must be included with Vulkan applications. CUDA headers follow the above naming convention with a “cuda” suffix and must be included with CUDA applications.

During development, copy the `nvngx_dlssd.dll` / `libnvidia-ngx-dlssd.so` file from the `./lib/_/dev/` directory on GitHub to the folder where the main game executable or DLL is located, so the NGX runtime can properly find and load the DLL. If required by the game, the game build, or the packaging system, the DLSS library can be packaged in a different location to the main executable. In this case, the entry point, `NVSDK NGX_D3D12_Init_ext`, accepts a parameter, `InFeatureInfo` which has a `NVSDK NGX_PathListInfo` item that contains a list of paths to search through.

For more information, see section 5.1 of the *DLSS Programming Guide*.

- > The develop DLL (in the **dev** folder) is intended for use while integrating and developing with DLSS Ray Reconstruction and includes debug features.
- > The release DLL should be used in the final version of the application (in the **rel** folder).

- > Please do not release / ship a develop DLL with your application.

5.1.1 Linking the NGX SDK

Please refer to sections 5.1.1 (Linking the NGX SDK on Windows) and 5.1.2 (Linking the NGX SDK on Linux) of the *DLSS Programming Guide*.

5.2 NGX SDK Initialization

Please refer to section 5.2 (Initializing the NGX SDK Object) of the *DLSS Programming Guide*.

5.3 Feature Creation

DLSS-RR Feature creation is an extension to DLSS Feature Creation (refer Section 5.3 of the *DLSS Programming Guide*) with the following additions:

Integrations can simply include `nvsdk_ngx_helpers_dlssd.h`, wherever they have included `nvsdk_ngx_defs.h`, `nvsdk_ngx_params.h`.

For Feature Creation, use the following guidelines:

- > `NVSDK NGX_DLSS_Denoise_Mode` enum has been provided in `nvsdk_ngx_defs_dlssd.h`.

```
// Only NVSDK NGX_DLSS_Denoise_Mode_DLUnified is supported
typedef enum NVSDK NGX_DLSS_Denoise_Mode
{
    NVSDK NGX_DLSS_Denoise_Mode_Off = 0,
    NVSDK NGX_DLSS_Denoise_Mode_DLUnified = 1, // DL based unified upscaler
} NVSDK NGX_DLSS_Denoise_Mode;
```

- > The following struct has been provided in `nvsdk_ngx_params_dlssd.h`:

```
typedef struct NVSDK NGX_DLSSD_Create_Params
{
    NVSDK NGX_DLSS_Denoise_Mode InDenoiseMode;
    NVSDK NGX_DLSS_Roughness_Mode InRoughnessMode;
    NVSDK NGX_DLSS_Depth_Type InUseHWDepth;

    unsigned int InWidth;
    unsigned int InHeight;
    unsigned int InTargetWidth;
    unsigned int InTargetHeight;
    /**/ OPTIONAL /**/
    NVSDK NGX_PerfQuality_Value InPerfQualityValue;
    int InFeatureCreateFlags;
    bool InEnableOutputSubrects;
} NVSDK NGX_DLSSD_Create_Params;
```


- > `InDenoiseMode` should be set to one of the `NVSDK NGX_DLSS_Denoise_Mode_DL_Unified`.

To create DLSS-RR features, use one of the following convenience functions provided in `nvsdk_ngx_helpers_dlssd.h`, depending on which graphics API is being used.

```
static inline NVSDK NGX_Result NGX_D3D12_CREATE_DLSSD_EXT(
    ID3D12GraphicsCommandList *pInCmdList,
    unsigned int InCreationNodeMask,
    unsigned int InVisibilityNodeMask,
    NVSDK NGX_Handle **ppOutHandle,
    NVSDK NGX_Parameter *pInParams,
    NVSDK NGX_DLSSD_Create_Params *pInDlssDCreateParams)
{
    // Set parameters...
    // ...
    return NVSDK NGX_D3D12_CreateFeature(pInCmdList,
NVSDK NGX_Feature_SuperSamplingDenoising, pInParams, ppOutHandle);
}

static inline NVSDK NGX_Result NGX_D3D11_CREATE_DLSSD_EXT(
    ID3D11DeviceContext *pInCtx,
    NVSDK NGX_Handle **ppOutHandle,
    NVSDK NGX_Parameter *pInParams,
    NVSDK NGX_DLSSD_Create_Params *pInDlssDCreateParams)
{
    // Set parameters...
    // ...
    return NVSDK NGX_D3D11_CreateFeature(pInCtx,
NVSDK NGX_Feature_SuperSamplingDenoising, pInParams, ppOutHandle);
}

static inline NVSDK NGX_Result NGX_VULKAN_CREATE_DLSSD_EXT1(
    VkDevice InDevice,
    VkCommandBuffer InCmdList,
    unsigned int InCreationNodeMask,
    unsigned int InVisibilityNodeMask,
    NVSDK NGX_Handle **ppOutHandle,
    NVSDK NGX_Parameter *pInParams,
    NVSDK NGX_DLSSD_Create_Params *pInDlssDCreateParams)
{
    // Set parameters...
    // ...
    if(InDevice)
        return NVSDK NGX_VULKAN_CreateFeature1(InDevice, InCmdList,
NVSDK NGX_Feature_SuperSamplingDenoising, pInParams, ppOutHandle);
    else
        return NVSDK NGX_VULKAN_CreateFeature(InCmdList,
NVSDK NGX_Feature_SuperSamplingDenoising, pInParams, ppOutHandle);
}
```

```
static inline NVSDK_NGX_Result NGX_CUDA_CREATE_DLSSD_EXT( NVSDK_NGX_Handle**
ppOutHandle, NVSDK_NGX_Parameter* pInParams,
NVSDK_NGX_CUDA_DLSSD_Create_Params* pInDlssDCreateParams)

static inline NVSDK_NGX_Result NGX_CUDA_CREATE_DLSSD_EXT1( NVSDK_NGX_CUDADevice
InDevice, NVSDK_NGX_Handle* ppOutHandle, NVSDK_NGX_Parameter* pInParams,
NVSDK_NGX_CUDA_DLSSD_Create_Params* pInDlssDCreateParams)
```

5.4 Feature Evaluation

DLSS-RR Feature Evaluation is an extension to DLSS Feature Evaluation (refer Section 5.4 of the *DLSS Programming Guide*) with the following additions:

- > A new Evaluate Parameter struct for ray reconstruction-specific parameters is provided in `nvsdk_ngx_helpers_dlssd.h` and for each graphics API.
- > A new Evaluate function is also provided in `nvsdk_ngx_helpers_dlssd.h` for evaluating a ray reconstruction feature for each graphics API, which takes the DLSS-RR evaluation parameters above.

```
static inline NVSDK_NGX_Result NGX_VULKAN_EVALUATE_DLSSD_EXT(
    VkCommandBuffer InCmdList,
    NVSDK_NGX_Handle *pInHandle,
    NVSDK_NGX_Parameter *pInParams,
    NVSDK_NGX_VK_DLSSD_Eval_Params *pInDlssDEvalParams)
{
    // Set parameters...
    return NVSDK_NGX_VULKAN_EvaluateFeature_C(InCmdList, pInHandle, pInParams,
    NULL);
}

static inline NVSDK_NGX_Result NGX_D3D11_EVALUATE_DLSSD_EXT(
    ID3D11DeviceContext *pInCtx,
    NVSDK_NGX_Handle *pInHandle,
    NVSDK_NGX_Parameter *pInParams,
    NVSDK_NGX_D3D11_DLSSD_Eval_Params *pInDlssDEvalParams)
{
    // Set parameters...
    return NVSDK_NGX_D3D11_EvaluateFeature_C(pInCtx, pInHandle, pInParams,
    NULL);
}

static inline NVSDK_NGX_Result NGX_D3D12_EVALUATE_DLSSD_EXT(
    ID3D12GraphicsCommandList *pInCmdList,
    NVSDK_NGX_Handle *pInHandle,
    NVSDK_NGX_Parameter *pInParams,
    NVSDK_NGX_D3D12_DLSSD_Eval_Params *pInDlssDEvalParams)
{
    // Set parameters...
```

```

    return NVSDK NGX_D3D12_EvaluateFeature_C(pInCmdList, pInHandle, pInParams,
    NULL);
}

```

```

static inline NVSDK_NGX_Result NGX_CUDA_EVALUATE_DLSSD_EXT(
NVSDK_NGX_Handle* pInHandle, NVSDK_NGX_Parameter* pInParams,
NVSDK_NGX_CUDA_DLSSD_Eval_Params* pInDlssDEvalParams)

```

- > DLSS-RR parameter allocation is identical to DLSS.

```

// ParametersDLSSD can be used as an argument to all functions above
result = NVSDK_NGX_VULKAN_AllocateParameters(&ParametersDLSSD);

```

5.5 Querying Optimal Settings

Obtaining DLSS-RR Optimal settings is an extension to DLSS Optimal Resolutions Query (refer Section 5.2.8, 5.2.8.1 of the *DLSS Programming Guide*) with the following changes:

- > A new Optimal Settings Query is provided in `nvsdk_ngx_helpers_dlssd.h` which take similar input to those of DLSS. Note that the sharpeness output parameter must always be ignored.

```

static inline NVSDK_NGX_Result NGX_DLSSD_GET_OPTIMAL_SETTINGS (
    // Query function parameters...
)

```

5.6 Other Feature APIs

Please refer to the notes in Sections 5.4.1 to Sections 5.7 (except 5.2.8 which is covered above) of the *DLSS Programming Guide*.

Chapter 6. Resource Management

Please refer to section 6 (Resource Management) of the *DLSS Programming Guide*.

Chapter 7. Multi-GPU Support

Please refer to section 7 (Resource Management) of the *DLSS Programming Guide*.

Chapter 8. Troubleshooting

8.1 DLSS-RR Debug Overlay

The DLSS-RR SDK development library (dev DLL) includes a debugging overlay which allows you to visualize the inputs used by DLSS-RR. To enable and cycle through the various debug layers, use the CTRL+ALT+F12 hotkey. The debug layer appears as an overlay in the top right side of the screen and includes:

1. Current input color buffer ([Section 3.4.5](#))
2. Current frame's input Motion Vectors ([Section 3.4.6](#)).
3. Current frame's input Depth buffer (black is the near plane passing through blue, purple, pink with white being the far plane) ([Section 3.4.7](#)).
4. Jitter Offset – The history of submitted jitter offsets in an XY scatter plot [-1, 1] (green is the current offset, white and red represent old offsets that are within the correct boundary [-0.5, 0.5] and outside it, respectively). If old, interior offsets are colored yellow; this means that the recommended number of phases has not been met yet (see section 3.7 of the *DLSS Programming Guide* for more information).
5. Current frame's input Normals ([Section 3.4.3](#))
6. Specular Hit Distance ([Section 3.4.9](#))
7. Current frame's input Roughness ([Section 3.4.4](#)).
8. Current frame's input Diffuse Albedo ([Section 3.4.1](#))
9. Current frames's input Specular Albedo ([Section 3.4.2](#))
10. Current frame's input Specular Motion Vectors ([Section 3.4.8](#))

To toggle the debug visualization between an overlay window in the top right of the screen and fullscreen, use the CTRL+ALT+F11 hotkey.



Note: The DLSS production library does not include the Debug Overlay.

8.2 Error Codes

Please refer section 8.7 (Error Codes) of the *DLSS Programming Guide*.

Appendix

Specular Albedo Calculation

Specular Albedo is computed with this function.

```
float3 EnvBRDFApprox2(float3 SpecularColor, float alpha, float NoV)
{
    NoV = abs(NoV);
    // [Ray Tracing Gems, Chapter 32]
    float4 X;
    X.x = 1.f;
    X.y = NoV;
    X.z = NoV * NoV;
    X.w = NoV * X.z;
    float4 Y;
    Y.x = 1.f;
    Y.y = alpha;
    Y.z = alpha * alpha;
    Y.w = alpha * Y.z;
    float2x2 M1 = float2x2(0.99044f, -1.28514f, 1.29678f, -0.755907f);
    float3x3 M2 = float3x3(1.f, 2.92338f, 59.4188f, 20.3225f, -27.0302f,
222.592f, 121.563f, 626.13f, 316.627f);
    float2x2 M3 = float2x2(0.0365463f, 3.32707, 9.0632f, -9.04756);
    float3x3 M4 = float3x3(1.f, 3.59685f, -1.36772f, 9.04401f, -16.3174f,
9.22949f, 5.56589f, 19.7886f, -20.2123f);
    float bias = dot(mul(M1, X.xy), Y.xy) * rcp(dot(mul(M2, X.xyw), Y.xyw));
    float scale = dot(mul(M3, X.xy), Y.xy) * rcp(dot(mul(M4, X.xzw), Y.xyw));
    // This is a hack for specular reflectance of 0
    bias *= saturate(SpecularColor.g * 50);
    return mad(SpecularColor, max(0, scale), max(0, bias));
}

outSpecularAlbedo = EnvBRDFApprox2( GBuffer.SpecularAlbedo, GBuffer.fRoughness
* GBuffer.fRoughness, NdotV );
```

Third Party Software

Linux Driver Compatibility

The Linux driver no longer adheres to a strict minimum driver version compatibility model that can be queried by

`NVSDK NGX_Parameter_SuperSamplingDenoising_MinDriverVersionMajor`.

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that the customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA, the NVIDIA logo, Deep Learning, DLSS, DLSS-RR are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Copyright

© 2025 NVIDIA CORPORATION and affiliates. All rights reserved.

