



NVIDIA RTX Video SDK

Programming Guide

Version 1.0.2

Document History

SWE-GPUDRV-002-PGRF

Version	Date	Description of Change
1.0.0	Nov 2023	Alpha pre-release
1.0.1	Apr 2024	Pre-release
1.0.2	June 2024	Initial Release

Table of Contents

Abstract	5
Chapter 1. Introduction.....	6
Chapter 2. Getting Started	7
2.1 Requirements.....	7
2.1.1 General Requirements.....	7
2.1.2 Windows.....	7
2.2 Build the Samples.....	7
2.2.1 Windows.....	7
2.3 Run the Samples.....	8
2.3.1 Windows.....	9
2.3.1.1 TrueHDRDemo Parameters	9
2.3.1.2 VSRDemo Parameters	10
2.3.1.3 TrueHDR_VSR_Demo Parameters.....	10
Chapter 3. Integrating RTX Video with Your Application.....	11
3.1 Thread Safety.....	11
3.2 Direct Integration of RTX Video for All Platforms.....	12
3.2.1 Adding RTX Video to Your Application.....	12
3.2.1.1 Windows.....	12
3.2.2 Initializing RTX Video	13
3.2.2.1 General Initialization	13
3.2.2.2 Verifying Availability of RTX Video Features on Your System.....	15
3.2.3 Setting and Obtaining Parameters	16
3.2.4 Scratch Buffer Setup.....	17
3.2.5 Feature Creation.....	18
3.2.6 Feature Evaluation	20
3.2.7 Vulkan Resource Wrapper.....	22
3.2.8 Feature Disposal	23
3.2.9 Shutdown	24
3.3 RTX Video DX11 and DX12 Wrapper Files.....	25
3.3.1 TrueHDR Wrapper Functions.....	25
3.3.2 VSR Wrapper Functions.....	26
3.3.3 DX12 Sync Requirements	27
3.3.4 Code Flow Modifications for Application.....	27
3.3.4.1 TrueHDR used with DX11 DXGI or DirectComposition.....	27
3.3.4.2 TrueHDR used with DX11 PresentationManager	28
3.3.4.3 VSR used with DX11 DXGI or DirectComposition.....	28

3.3.4.4	VSR used with DX11 PresentationManager	29
3.3.4.5	TrueHDR Used with DX12 and DGXI Present	29
3.3.4.6	VSR Used with DX12 and DGXI Present	30
3.3.5	Using Both VSR and TrueHDR	31
3.4	Sample RTX Video API	31
3.4.1	RTX Video API Functions	31
3.4.2	RTX Video API Struct Definitions	32
3.4.3	Sync and Flow Requirements	33
3.5	Running and Verifying RTX Video SDK Features	33
Chapter 4.	Distributing RTX Video Features with Your Application	34
Chapter 5.	RTX Video Features	35
5.1	TrueHDR: Convert Surface from SDR to HDR	35
5.1.1	TrueHDR Evaluation Parameter Struct Definitions	37
5.1.2	Sample Code for TrueHDR DX11	38
5.1.2.1	Using DX11 PresentationManager with TrueHDR	39
5.1.3	Sample Code for TrueHDR DX12	40
5.2	VSR: Upscale SDR Surface with Artifact Reduction	42
5.2.1	VSR Evaluation Parameter Struct Definitions	43
5.2.2	Sample Code for VSR DX11	44
5.2.2.1	Using DX11 PresentationManager with VSR	45
5.2.3	Sample Code for VSR DX12	46
Chapter 6.	Troubleshooting / FAQ	48
6.1	Common Sample Build Errors	48
6.2	Common Issues When Running Samples	49
6.3	General Issues When Using the SDK	49
Licenses & Copyright Notices		51
FFmpeg		51
VULKAN		52

List of Tables

Table 1. AI Features	34
Table 2. TrueHDR Supported Parameters	35
Table 3. VSR Supported Parameters	42

Abstract

This RTX Video SDK Programming Guide provides a detailed overview about how you can integrate and distribute RTX Video SDK features with your application. The Programming Guide also provides sample code to help you achieve these goals.

The following features are included in this SDK:

- > [TrueHDR](#): Convert Surface from SDR to HDR.
- > [VSR](#): Video Super Resolution, Upscale SDR Surface with Artifact Reduction.

Chapter 1. Introduction

NVIDIA RTX Video SDK makes it easy for you to integrate pre-built AI video-based features into your applications. NVIDIA will be adding new features and updating the existing ones over time. When an existing feature is updated, the underlying NGX infrastructure will update the feature on all clients that use it.

There are three main components that make up the system:

- > **RTX Video SDK:** Provides DirectX 11, DirectX 12, and Vulkan APIs for applications to access the AI video features. (DX11/12 versions refer to Windows only.)
- > **NGX Core Runtime:** The NGX Core Runtime is a system component that determines which shared library (DLL) to load for each feature and application. The NGX runtime module is always installed on the end-user's system as part of the NVIDIA Graphics Driver if supported NVIDIA RTX hardware is detected. During an "advanced driver installation" the module is called "NGX Core."
- > **NGX Update Module:** Updates to RTX Video features that are managed by the NGX Core Runtime itself. When an application initiates an RTX Video feature, the runtime calls the NGX Update Module to check for new versions of the feature that is in use. If a newer version is found, the NGX Update Module downloads and swaps the DLL for the calling application.

Chapter 2. Getting Started

2.1 Requirements

2.1.1 General Requirements

- > Sign an NDA with NVIDIA for accessing the RTX Video SDK.
- > The RTX Video SDK is distributed in a ZIP file that can be unpacked and placed on your desired development system.
 - Set the `NV_RTX_VIDEO_SDK` environment variable to the RTX Video SDK directory.
- > An NVIDIA RTX GPU (Quadro, Titan or GeForce)
- > The latest [NVIDIA Graphics Driver](#), minimum r550.58
 - Avoid Insider Preview drivers.

2.1.2 Windows

- > PC with Windows 10 v20H1 (May 2020 Update 64-bit) or newer.
- > The minimum development environment for integrating the RTX Video SDK into your application is Microsoft Visual Studio 2019 with Windows 10 SDK 10.0.19041.0.
 - Building the samples also requires C++ MFC and C++ ATL, which can be retroactively installed with the Visual Studio installer under **Individual Components**.

2.2 Build the Samples

2.2.1 Windows

To build the SDK Samples on Windows, open the RTX Video SDK Samples solution file in `<RTX_VIDEO_SDK>\samples\` with Visual Studio.

Select **Release x64** from the configuration drop down.



Note: It is possible that the Windows SDK version or the Microsoft tools versions in the Visual Studio project files may not be the exact ones on your development system. This is

common, as there are many different install options for Visual Studio; the error will be of the form:

```
error MSB8036: The Windows SDK version 10.0.15063.0 was not found. Install the required version of Windows SDK or change the SDK version in the project property pages or by right-clicking the solution and selecting "Retarget solution".
```

Simply use the Visual Studio settings (i.e., right-click the solution and selecting **Retarget solution**) to retarget the projects to use the VS2019 SDK or later and tools that are on your system. The samples should be compatible.

Next, select **Build Solution** from the **Build** menu. The samples will build to the `x64Release` folder. To step through the source code in the debugger while executing the samples, select the **Debug x64** configuration instead. In this case, the built samples are placed into `x64Debug`.

Finally, copy the RTX Video SDK feature DLLs from `<RTX_VIDEO_SDK>\bin\Windows\x64\dev` to the build folder. The feature binaries will be picked up from the same directory as the running application unless otherwise specified during [SDK initialization](#).

In the current version of the RTX Video SDK, expect to see the following executables and libraries:

- > `TrueHDRDemo.exe`: Video TrueHDR applied to SDR input.
- > `VSRDemo.exe`: Video Super Resolution applied to input.
- > `TrueHDR_VSR_Demo.exe`: Video TrueHDR and VSR applied to input.
- > `RTX_Video_API.lib`: Static library for simple integration with the [Sample RTX Video API](#).

2.3 Run the Samples

The samples use an uncompressed file as input the needs to be generated. Here is an example using FFmpeg.



Note: FFmpeg is a third-party program. For more info on how to get and use FFmpeg, visit <https://ffmpeg.org/>



Note: The generated file name must be in the following form:

```
fmt_width_height_pitch_numFrames_xxxx.yuv  
NV12_1280_720_1280_120_star_sky.yuv
```

- > Use input format NV12. Pitch equals width.
- > Use FFmpeg to downscale a video file if it is high resolution.
- > Use `-1` on scale for same ratio (note use 704 instead of 720 for 720p):

```
ffmpeg.exe -i "file.mp4" -vf scale=1280:-1 scaled1280.mp4
```
- > To capture the first 120 frames to NV12:

```
ffmpeg -i scaled1280.mp4 -c:v rawvideo -pix_fmt nv12 -start_number 0 -frames:v 120  
NV12_1280_720_1280_120_fname.yuv
```


2.3.1 Windows

The samples will build three executables located in the <RTX VIDEO SDK>\samples\x64Release directory:

- > TrueHDRDemo.exe: Video TrueHDR applied to SDR input.
- > VSRDemo.exe: Video Super Resolution applied to input.
- > TrueHDR_VSR_Demo.exe: Video TrueHDR and VSR applied to input.



Note: Be sure to copy the feature runtime DLLs in <RTX VIDEO SDK>\bin\Windows\x64\dev to the same folder as the sample executables.

The sample apps are all command line executables and operate in a similar way. Run the executables with no arguments or -? to see the options. -i for input file is required. For TrueHDR_VSR_Demo.exe at least one feature selection is also required (-thdr and/or -vsr)

These are some example runs:

```
// run the TrueHDR demo with 120 frames of 720p input
TrueHDRDemo.exe -i NV12_1280_720_1280_120_star_sky.yuv

// run the VSR demo with 100 frames of 480p input scaled up to 1920x1080
VSRDemo.exe -i NV12_640_480_640_100_test.yuv -size 1920 1080

// run TrueHDR+VSR demo with 120 frames of 1080p input
TrueHDR_VSR_Demo.exe -I NV12_1920_1080_1920_120_car_chase.yuv -vsr -thdr
```

2.3.1.1 TrueHDRDemo Parameters

```
-i <file>    // required: use <input file> name: 4cc_width_height_pitch_frames_xxxx.xxx
-dx12        // use dx12 (default dx11)
-fp16        // use fp16 output (default abgr10)
-size w h    // display at this size
-nativeSize  // display at source size
-fps n       // frames per second (1-60)
-head n      // head on display adapter (default 0)
-?           // usage
```

2.3.1.2 VSRDemo Parameters

```
-i <file>    // required: use <input file> name: 4cc_width_height_pitch_frames_xxxx.xxx
-dx12        // use dx12 (default dx11)
-size w h    // display at this size
-nativeSize  // display at source size
-fps n       // frames per second (1-60)
-head n      // head on display adapter (default 0)
-?          // usage
```

2.3.1.3 TrueHDR_VSR_Demo Parameters

```
-i <file>          // required: use <file> name: 4cc_width_height_pitch_frames_xxx.xxx
-vsr              // use vsr (-thdr or -vsr required)
-thdr            // use truehdr (-thdr or -vsr required)
-vsrQuality n     // vsr quality level 1 to 4 (default 1)
-thdrContrast n   // truehdr contrast 0 to 200 (default 100)
-thdrSaturation n // truehdr saturation 0 to 200 (default 100)
-thdrMiddleGray n // truehdr middle gray 10 to 100 (default 50)
-thdrMaxLuminance n // truehdr monitor max luminance 400 to 2000 (default from monitor)
-head n          // head on display adapter (default 0)
-dx12           // use dx12 (default dx11)
-size w h       // display at this size
-nativeSize     // display at source size
-fps n          // frames per second (1-60)
-head n         // head on display adapter (default 0)
-?             // usage
```

Chapter 3. Integrating RTX Video with Your Application

For a simple DirectX 11 and DirectX 12 integration, use the RTX Video API sample that distills SDK usage into a single set of create, evaluate, and shutdown calls. This is covered in [Sample RTX Video API](#).

If more control over the DirectX 11 and DirectX 12 integration is needed, there are wrapper classes available. Those are covered in [RTX Video DX11 and DX12 Wrapper Files](#).

The RTX Video SDK uses NGX under the hood. Direct integration with NGX is covered in [Direct Integration of RTX Video for All Platforms](#). Direct integration can be used on all supported APIs and platforms.

Before starting any integration, the following additional technical requirements should be considered:

- > DX11 and DX12 destination resources must be UAV (Unordered Access View)
 - The Sample RTX Video API handles this case.
- > In multithreaded scenarios, access to the SDK should be serialized.
 - Additional DX API multithreading setup will be needed during manual integration.

3.1 Thread Safety

The underlying NGX API is **not** thread safe. The client application must ensure that thread safety is enforced as needed. Invoking NGX APIs from multiple threads can result in unpredictable behavior.

For features that use DirectX, the NGX API preserves the state of the immediate D3D11 context. However, that is **not** the case with D3D12 command lists. For all applications using DirectX 12 and D3D12 command lists, the client application must manage the command list state as needed.

3.2 Direct Integration of RTX Video for All Platforms



Note: When doing a direct integration, it is still advisable to reference the wrapper classes as an example of a correct integration. The wrappers exemplify the correct inputs, outputs, and calling order of the SDK functions. See `<RTX_VIDEO_SDK>\common\ngx_wrapper`

3.2.1 Adding RTX Video to Your Application

The RTX Video SDK comes with several header files. These files are in the `<RTX_VIDEO_SDK>\include` folder. Of the header files, there are four that are of note:

- > `nvsdk_ngx_helpers_truehdr.h` – header file for TrueHDR on DX11 and DX12.
- > `nvsdk_ngx_helpers_truehdr_vk.h` – header file for TrueHDR on Vulkan
- > `nvsdk_ngx_helpers_vsr.h` – header file for VSR on DX11 and DX12
- > `nvsdk_ngx_helpers_vsr_vk.h` – header file for VSR on Vulkan

Include the features of your choice. The remaining header files are included by these four headers as needed. They do not need to be manually added.



Note: The samples also include the `_defs.h` headers for clarity. This is not required.

3.2.1.1 Windows

In addition to including the header files, your project should also link against:

- > `nvsdk_ngx_s.lib` – if static runtime library (/MT) linking is used in your project, or
- > `nvsdk_ngx_d.lib` – if dynamic runtime library (/MD) linking is used in your project.
- > Both files are in the `<RTX_VIDEO_SDK>\lib\x64` folder (NGX is provided as a **64bit only** library).



Note: Depending on how the project is linked, a pragma library import of the appropriate library may be required in your source files.

```
#pragma comment( lib, "nvsdk_ngx_s.lib")
#pragma comment( lib, "nvsdk_ngx_d.lib")
```

During development, copy the feature runtime libraries `nvngx_*.dll` from `<RTX_VIDEO_SDK>\bin\Windows\x64\` to the same folder as your executable or the folder specified in `InFeatureInfo` during SDK init. For information about how the RTX Video SDK should be distributed with your application, see [Distributing RTX Video Features with Your Application](#).

3.2.2 Initializing RTX Video

3.2.2.1 General Initialization

The RTX Video SDK supports the D3D11, D3D12 and Vulkan APIs. For more information, see the [RTX Video Features section](#), which describes the APIs that each feature supports.

All calls in the RTX Video SDK are similar for the supported APIs (D3D11, D3D12 and Vulkan) so all features can be initialized using code that matches or is very similar to the following sample code. Ensure that you use calls that match the API that your application is using. Any interop between APIs, for example, D3D11 to Vulkan, must be handled by the application outside the RTX VIDEO SDK.

To initialize an SDK instance, use one of the following methods (more details in `nvsdk_ngx.h` and `nvsdk_ngx_vk.h` in the <RTX VIDEO SDK>\include folder):

```
// NVSDK NGX_Init
// -----
//
// InApplicationId:
//     Unique Id provided by NVIDIA. 0 Default unless otherwise requested from NVIDIA.
//
// InApplicationDataPath:
//     Folder to store logs and other temporary files (write access required). Relative
//     paths are ok.
//
// InDevice: [d3d11/12 only]
//     DirectX device to use
//
// InInstance/InPD, InDevice: [vk only]
//     Vulkan Instance, PhysicalDevice, and Device to use
//
// InGIPA/InGDPA: [vk only]
//     Vulkan function pointers to vkGetInstanceProcAddr and vkGetDeviceProcAddr
//
// InFeatureInfo:
//     Contains information common to all features, presently only a list of all paths
//     feature dlls can be located, other than the default path - application directory.
//
// InSDKVersion:
//     For RTX Video SDK, leave this as default value NVSDK NGX_Version_API.
//
// DESCRIPTION:
//     Initializes new SDK instance.

NVSDK NGX_Result NVSDK_CONV NVSDK NGX_D3D11_Init(
    unsigned long long InApplicationId,
    const wchar_t *InApplicationDataPath,
    ID3D11Device *InDevice,
    const NVSDK NGX_FeatureCommonInfo *InFeatureInfo = nullptr,
```

```

    NVSDK_NGX_Version InSDKVersion = NVSDK_NGX_Version_API);

NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_D3D12_Init(
    unsigned long long InApplicationId,
    const wchar_t *InApplicationDataPath,
    ID3D12Device *InDevice,
    const NVSDK_NGX_FeatureCommonInfo *InFeatureInfo = nullptr,
    NVSDK_NGX_Version InSDKVersion = NVSDK_NGX_Version_API);

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_VULKAN_Init(
    unsigned long long InApplicationId,
    const wchar_t *InApplicationDataPath,
    VkInstance InInstance,
    VkPhysicalDevice InPD,
    VkDevice InDevice,
    PFN_vkGetInstanceProcAddr InGIPA = nullptr,
    PFN_vkGetDeviceProcAddr InGDPA = nullptr,
    const NVSDK_NGX_FeatureCommonInfo *InFeatureInfo = nullptr,
    NVSDK_NGX_Version InSDKVersion = NVSDK_NGX_Version_API);

```

With the following relevant struct definitions in `nvsdk_ngx_defs.h`:

```

typedef enum NVSDK_NGX_Version {
    NVSDK_NGX_Version_API = NVSDK_NGX_VERSION_API_MACRO
} NVSDK_NGX_Version;

typedef struct NVSDK_NGX_FeatureCommonInfo
{
    // List of all paths in descending order of search sequence to locate a feature dll in,
    // other than the default path - application folder.
    NVSDK_NGX_PathListInfo PathListInfo;

    // Used internally by NGX, introduced in SDK version 0x0000013
    NVSDK_NGX_FeatureCommonInfo_Internal* InternalData;

    // Fields below were introduced in SDK version 0x0000014
    NVSDK_NGX_LoggingInfo LoggingInfo;
} NVSDK_NGX_FeatureCommonInfo;

```



Note: All the NGX calls have an Application ID as the first parameter which is used for features that have application specific tuning. Please contact your NVIDIA developer relations manager to obtain the correct application ID (`InApplicationId`) for your application. Use `0` until NVIDIA has assigned you an application ID.

Once an SDK instance is created, check for availability of the desired features by getting your system capability parameters. This is detailed in the next section.

3.2.2.2 Verifying Availability of RTX Video Features on Your System

Successful initialization indicates that the target system can run RTX Video features. However, each feature can have additional dependencies, for example, minimum driver version. Therefore, it is good practice to check if the specific feature you would like to use is available.

For this purpose, NGX provides an `NVSDK_NGX_Parameter` interface which can be used to query read-only parameters provided by NGX runtime and can be obtained using the following API:

```
// NVSDK_NGX_GetCapabilityParameters
// -----
//
// OutParameters:
//     The parameters interface populated with NGX and feature capabilities
//
// DESCRIPTION:
//     This interface allows the app to create a new parameter map
//     pre-populated with NGX capabilities and available features.
//     The output parameter map can also be used for any purpose
//     parameter maps output by NVSDK_NGX_AllocateParameters can be used for
//     but it is not recommended to use NVSDK_NGX_GetCapabilityParameters
//     unless querying NGX capabilities and available features
//     due to the overhead associated with pre-populating the parameter map.
//     Parameter maps output by NVSDK_NGX_GetCapabilityParameters must NOT be freed
//     using the free/delete operator; to free a parameter map output
//     by NVSDK_NGX_GetCapabilityParameters, NVSDK_NGX_DestroyParameters should be
//     used. Unlike with NVSDK_NGX_GetParameters, parameter maps allocated with
//     NVSDK_NGX_GetCapabilityParameters
//     must be destroyed by the app using NVSDK_NGX_DestroyParameters.
//     This function may return NVSDK_NGX_Result_FAIL_OutOfDate if an older driver,
//     which does not support this API call is being used. This function may only be
//     called after a successful call into NVSDK_NGX_Init.
//     If NVSDK_NGX_GetCapabilityParameters returns NVSDK_NGX_Result_FAIL_OutOfDate,
//     NVSDK_NGX_GetParameters may be used as a fallback, to get a parameter map
//     pre-populated with NGX capabilities and available features.

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_D3D11_GetCapabilityParameters(
    NVSDK_NGX_Parameter** OutParameters);

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_D3D12_GetCapabilityParameters(
    NVSDK_NGX_Parameter** OutParameters);

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_VULKAN_GetCapabilityParameters(
    NVSDK_NGX_Parameter** OutParameters);
```

Once the parameters have been queried, the parameter handle should be saved for later use in capability queries, create calls, and evaluate calls.

For example, to check if `NVSDK NGX_Feature_VSR` is available on your system while using D3D12, issue the following:

```
int VSRSupported = 0;
NVSDK NGX_Parameter *Params = nullptr;

// Instantiate Params for later use in Get query and CREATE_VSR
NVSDK NGX_D3D12_GetCapabilityParameters( &Params );

// Get if VSR is available right now
Params->Get( NVSDK NGX_Parameter_VSR_Available, &VSRSupported );

if( VSRSupported )
{
    // OK to create and use VSR feature
}
```

The most common reasons for a feature to not be supported are incompatible drivers or missing feature DLLs. See the [FAQ](#) for more details.



Note: Not all NGX features always have D3D11/D3D12 and Vulkan implementations. Check the return codes or read the [documentation for the feature](#) to determine which implementation is available.

3.2.3 Setting and Obtaining Parameters



Note: The RTX Video SDK provides helper functions in the `nvsdk_ngx_helper_*.h` headers that abstract away the NGX Parameter interface in favor for setting parameters using create and evaluate input structs. See the struct definitions in [RTX Video Features](#). This step can be skipped if using the provided helper functions.

Each feature requires certain parameters to be set-up prior to feature creation; others must be passed to feature evaluation. All parameters may be passed to both creation and evaluation if desired, but creation parameters will be ignored in evaluation and vice-versa. Parameters that are required by creation are set at creation time and cannot be changed per-evaluation. Setting of parameters to be sent to creation and evaluation are both accomplished by using the `NVSDK NGX_Parameter` interface.

Parameters are specified as {name,value} pairs, for example:

```
NVSDK NGX_Parameter *Params = nullptr;
NVSDK NGX_VULKAN_GetParameters( &Params );
Params->Set( NVSDK NGX_Parameter_Width, 100 );
```

For more details on which parameters should be set for creation and evaluation of specific features, see [RTX Video Features](#).

3.2.4 Scratch Buffer Setup



Note: In the case of TrueHDR and VSR, no scratch buffer is needed. This step can be skipped, but it is covered here for completeness.

After parameters are set, for each feature, check how much GPU scratch memory is required by calling the following:

```
// NVSDK_NGX_GetScratchBufferSize
// -----
//
// InFeatureId:
//     AI feature in question
//
// InParameters:
//     Parameters used by the feature to help estimate scratch buffer size
//
// OutSizeInBytes:
//     Number of bytes needed for the scratch buffer for the specified feature.
//
// DESCRIPTION:
//     SDK needs a buffer of a certain size provided by the client in
//     order to initialize AI feature. Once feature is no longer
//     needed buffer can be released. It is safe to reuse the same
//     scratch buffer for different features as long as minimum size
//     requirement is met for all features. Please note that some
//     features might not need a scratch buffer so return size of 0
//     is completely valid.

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_D3D11_GetScratchBufferSize(
    NVSDK_NGX_Feature InFeatureId,
    const NVSDK_NGX_Parameter *InParameters,
    size_t *OutSizeInBytes);

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_D3D12_GetScratchBufferSize(
    NVSDK_NGX_Feature InFeatureId,
    const NVSDK_NGX_Parameter *InParameters,
    size_t *OutSizeInBytes);

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_VULKAN_GetScratchBufferSize(
    NVSDK_NGX_Feature InFeatureId,
    const NVSDK_NGX_Parameter *InParameters,
    size_t *OutSizeInBytes);
```

The application is responsible for allocating a scratch buffer of the requested size and passing it as a parameter when creating a specific feature (more details can be found in the source code examples in the RTX Video SDK).



Note: It is acceptable for the SDK to return 0 as a required scratch buffer size for a specific feature.

3.2.5 Feature Creation

When all necessary parameters are set, create the feature. Use the helper functions included in the corresponding `nvsdk_ngx_helpers_*.h` for the desired feature. For example, creating TrueHDR will use the following:

```
static inline NVSDK NGX_Result NGX_D3D11_CREATE_TRUEHDR_EXT(
    ID3D11DeviceContext *pInCtx,           // Command context
    NVSDK NGX_Handle **ppOutHandle,        // Handle for new feature
    NVSDK NGX_Parameter *pInParams,        // From NVSDK NGX_GetCapabilityParameters
    NVSDK NGX_Feature_Create_Params *pTrueHDRCreateParams) // Unused by RTX Video SDK

static inline NVSDK NGX_Result NGX_D3D12_CREATE_TRUEHDR_EXT(
    ID3D12GraphicsCommandList* InCmdList, // Command list
    unsigned int InCreationNodeMask,       // Multi GPU only (default 1)
    unsigned int InVisibilityNodeMask,     // Multi GPU only (default 1)
    NVSDK NGX_Handle** ppOutHandle,        // Handle for new feature
    NVSDK NGX_Parameter* pInParams,        // From NVSDK NGX_GetCapabilityParameters
    NVSDK NGX_Feature_Create_Params* pTrueHDRCreateParams) // Unused by RTX Video SDK

static inline NVSDK NGX_Result NGX_VULKAN_CREATE_TRUEHDR_EXT1(
    VkDevice InDevice,                     // Vulkan device
    VkCommandBuffer InCmdList,             // Vulkan command buffer
    unsigned int InCreationNodeMask,       // Multi GPU only (default 1)
    unsigned int InVisibilityNodeMask,     // Multi GPU only (default 1)
    NVSDK NGX_Handle **ppOutHandle,        // Handle for new feature
    NVSDK NGX_Parameter *pInParams,        // From NVSDK NGX_GetCapabilityParameters
    NVSDK NGX_Feature_Create_Params *pTrueHDRCreateParams) // Unused by RTX Video SDK
```

With the following relevant struct definitions in `nvsdk_ngx_defs.h` and `nvsdk_ngx_params.h`:

```
typedef struct NVSDK NGX_Parameter; // Opaque definition

typedef struct NVSDK NGX_Handle {
    unsigned int Id;
} NVSDK NGX_Handle;

typedef struct NVSDK NGX_Feature_Create_Params // Unused. See Evaluate params instead.
{
    ...
} NVSDK NGX_Feature_Create_Params;
```

Alternatively, directly using the NGX create is an option:

```
// NVSDK NGX_CreateFeature
// -----
```

```

//
// InCmdList:[d3d12 only]
//     Command list to use to execute GPU commands. Must be:
//     - Open and recording
//     - With node mask including the device provided in NVSDK_NGX_D3D12_Init
//     - Execute on non-copy command queue.
// InDevCtx: [d3d11 only]
//     Device context to use to execute GPU commands
//
// InFeatureID:
//     AI feature to initialize
//
// InParameters:
//     List of parameters
//
// OutHandle:
//     Handle which uniquely identifies the feature. If feature with
//     provided parameters already exists the "already exists" error code is returned.
//
// DESCRIPTION:
//     Each feature needs to be created before it can be used.
//     Refer to the sample code to find out which input parameters
//     are needed to create specific feature.
//     CreateFeature() creates feature on single existing Device
//     CreateFeature1() creates feature on the specified Device
//

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_D3D11_CreateFeature(
    ID3D11DeviceContext *InDevCtx,
    NVSDK_NGX_Feature InFeatureID,
    const NVSDK_NGX_Parameter *InParameters,
    NVSDK_NGX_Handle **OutHandle);

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_D3D12_CreateFeature(
    ID3D12GraphicsCommandList *InCmdList,
    NVSDK_NGX_Feature InFeatureID,
    const NVSDK_NGX_Parameter *InParameters,
    NVSDK_NGX_Handle **OutHandle);

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_VULKAN_CreateFeature(
    VkCommandBuffer InCmdBuffer,
    NVSDK_NGX_Feature InFeatureID,
    NVSDK_NGX_Parameter *InParameters,
    NVSDK_NGX_Handle **OutHandle);

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_VULKAN_CreateFeature1(
    VkDevice InDevice,
    VkCommandBuffer InCmdList,
    NVSDK_NGX_Feature InFeatureID,
    NVSDK_NGX_Parameter *InParameters,

```

```
NVSDK NGX_Handle **OutHandle);
```



Note: Create a `NVSDK NGX_Handle*` to store the feature instance for later reference in the evaluate step. Feature handles do not use reference counting. If a request is made to create a new feature with parameters matching an existing feature, the SDK will return an **already exists** error code.

3.2.6 Feature Evaluation

Features are evaluated by executing inference on specific deep learning models. This can be achieved by calling the helper functions from the corresponding `nvsdk_ngx_helpers_*.h` for the desired feature. For example, evaluating VSR will use the following:

```
static inline NVSDK NGX_Result NGX_D3D11_EVALUATE_VSR_EXT(
    ID3D11DeviceContext *pInCtx,      // Command context
    NVSDK NGX_Handle *pInHandle,      // Handle for created feature
    NVSDK NGX_Parameter *pInParams,   // From NVSDK NGX_GetCapabilityParameters
    NVSDK NGX_D3D11_VSR_Eval_Params *pInVSR Eval_Params) // Populated param struct

static inline NVSDK NGX_Result NGX_D3D12_EVALUATE_VSR_EXT(
    ID3D12GraphicsCommandList *pInCmdList, // Command list
    NVSDK NGX_Handle *pInHandle,           // Handle for created feature
    NVSDK NGX_Parameter *pInParams,       // From NVSDK NGX_GetCapabilityParameters
    NVSDK NGX_D3D12_VSR_Eval_Params *pInVSR Eval_Params) // Populated param struct

static inline NVSDK NGX_Result NGX_VULKAN_EVALUATE_VSR_EXT(
    VkCommandBuffer InCmdList,          // Vulkan command buffer
    NVSDK NGX_Handle *pInHandle,        // Handle for created feature
    NVSDK NGX_Parameter *pInParams,     // From NVSDK NGX_GetCapabilityParameters
    NVSDK NGX_VK_VSR_Eval_Params *pInVSR Eval_Params) // Populated param struct
```

With evaluate parameters detailed in the [RTX Video Features](#) and the remaining relevant struct definitions in `nvsdk_ngx_defs.h` and `nvsdk_ngx_params.h`:

```
typedef struct NVSDK NGX_Parameter; // Opaque definition

typedef struct NVSDK NGX_Handle {
    unsigned int Id;
} NVSDK NGX_Handle;
```

Alternatively, the original NGX evaluate can be used:

```
// NVSDK NGX_EvaluateFeature
// -----
//
// InCmdList:[d3d12 only]
// Command list to use to execute GPU commands. Must be:
// - Open and recording
// - With node mask including the device provided in NVSDK NGX_D3D12_Init
```

```

//      - Execute on non-copy command queue.
// InDevCtx: [d3d11 only]
//      Device context to use to execute GPU commands
//
// InFeatureHandle:
//      Handle representing feature to be evaluated
//
// InParameters:
//      List of parameters required to evaluate feature
//
// InCallback:
//      Optional callback for features which might take longer
//      to execute. If specified SDK will call it with progress
//      values in range 0.0f - 1.0f. Client application can indicate
//      that evaluation should be cancelled by setting OutShouldCancel
//      to true.
//
// DESCRIPTION:
//      Evaluates given feature using the provided parameters and
//      pre-trained NN. Please note that for most features
//      it can be beneficial to pass as many input buffers and parameters
//      as possible (for example provide all render targets like color,
//      albedo, normals, depth etc)
//
typedef void (NVSDK_CONV *PFN_NVSDK NGX_ProgressCallback)(float InCurrentProgress, bool
&OutShouldCancel);

NVSDK NGX_API NVSDK NGX_Result NVSDK_CONV NVSDK NGX_D3D11_EvaluateFeature(
    ID3D11DeviceContext *InDevCtx,
    const NVSDK NGX_Handle *InFeatureHandle,
    const NVSDK NGX_Parameter *InParameters,
    PFN_NVSDK NGX_ProgressCallback InCallback = nullptr);

NVSDK NGX_API NVSDK NGX_Result NVSDK_CONV NVSDK NGX_D3D12_EvaluateFeature(
    ID3D12GraphicsCommandList *InCmdList,
    const NVSDK NGX_Handle *InFeatureHandle,
    const NVSDK NGX_Parameter *InParameters,
    PFN_NVSDK NGX_ProgressCallback InCallback = nullptr);

NVSDK NGX_API NVSDK NGX_Result NVSDK_CONV NVSDK NGX_VULKAN_EvaluateFeature(
    VkCommandBuffer InCmdList,
    const NVSDK NGX_Handle *InFeatureHandle,
    const NVSDK NGX_Parameter *InParameters,
    PFN_NVSDK NGX_ProgressCallback InCallback = NULL);

```



Caution: DirectX: NGX will modify D3D12 command list state; therefore, the calling process should save or restore its own D3D state before and after calling the NGX evaluate feature.

In this case, you must manually populate the `NVSDK NGX_Parameter` instance using the setters defined in `nvsdk_ngx_params.h`. For a full list of valid parameters, see the

corresponding `nvsdk_ngx_helpers_*.h` for the desired feature and take note of `NVSDK NGX_Parameter_Set*` usage.

3.2.7 Vulkan Resource Wrapper

The Vulkan resource wrapper passes metadata to the SDK which is otherwise not available. This wrapper structure must be used as an argument instead of directly passing in Vulkan resources. The following is the wrapper struct, and creation function signature:

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/
// NVSDK_NGX_ImageViewInfo_VK [Vulkan only]
// Contains ImageView-specific metadata.
// ImageView:
//     The VkImageView resource.
//
// Image:
//     The VkImage associated to this VkImageView.
//
// SubresourceRange:
//     The VkImageSubresourceRange associated to this VkImageView.
//
// Format:
//     The format of the resource.
//
// Width:
//     The width of the resource.
//
// Height:
//     The height of the resource.
//
typedef struct NVSDK_NGX_ImageViewInfo_VK {
    VkImageView ImageView;
    VkImage Image;
    VkImageSubresourceRange SubresourceRange;
    VkFormat Format;
    unsigned int Width;
    unsigned int Height;
} NVSDK_NGX_ImageViewInfo_VK;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/
// NVSDK_NGX_BufferInfo_VK [Vulkan only]
// Contains Buffer-specific metadata.
// Buffer
//     The VkBuffer resource.
//
// SizeInBytes:
//     The size of the resource (in bytes).
//

```

```

typedef struct NVSDK NGX_BufferInfo_VK {
    VkBuffer Buffer;
    unsigned int SizeInBytes;
} NVSDK NGX_BufferInfo_VK;

//
// NVSDK NGX_Resource_VK [Vulkan only]
//
// ImageViewInfo:
//     The VkImageView resource, and VkImageView-specific metadata. A NVSDK NGX_Resource_VK
//     can only have one of ImageViewInfo or BufferInfo.
//
// BufferInfo:
//     The VkBuffer Resource, and VkBuffer-specific metadata. A NVSDK NGX_Resource_VK can
//     only have one of ImageViewInfo or BufferInfo.
//
// Type:
//     Whether or this resource is a VkImageView or a VkBuffer.
//
// ReadWrite:
//     True if the resource is available for read and write access.
//     For VkBuffer resources: VkBufferUsageFlags includes
//     VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT or VK_BUFFER_USAGE_STORAGE_BUFFER_BIT
//     For VkImage resources: VkImageUsageFlags for associated VkImage includes
//     VK_IMAGE_USAGE_STORAGE_BIT
//
typedef struct NVSDK NGX_Resource_VK {
    union {
        NVSDK NGX_ImageViewInfo_VK ImageViewInfo;
        NVSDK NGX_BufferInfo_VK BufferInfo;
    } Resource;
    NVSDK NGX_Resource_VK_Type Type;
    bool ReadWrite;
} NVSDK NGX_Resource_VK;

```

3.2.8 Feature Disposal

When a feature is no longer needed, it should be released by calling the following method:

```

// NVSDK NGX_Release
// -----
//
// InHandle:
//     Handle to feature to be released
//
// DESCRIPTION:
//     Releases feature with a given handle.
//     Handles are not reference counted so

```

```
//      after this call it is invalid to use provided handle.

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_D3D11_ReleaseFeature(
    NVSDK_NGX_Handle *InHandle);

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_D3D12_ReleaseFeature(
    NVSDK_NGX_Handle *InHandle);

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_VULKAN_ReleaseFeature(
    NVSDK_NGX_Handle *InHandle);
```

Once released, the feature handle cannot be used any longer.

3.2.9 Shutdown

Before shutdown, release any parameter structs created through `AllocateParameters` or `GetCapabilityParameters`. This does not need to be done if using the deprecated `GetParameters` call.

```
// NVSDK_NGX_DestroyParameters
// -----
//
// InParameters:
//      The parameters interface to be destroyed
//
// DESCRIPTION:
//      This interface allows the app to destroy the parameter map passed in. Once
//      NVSDK_NGX_DestroyParameters is called on a parameter map, it
//      must not be used again.
//      NVSDK_NGX_DestroyParameters must not be called on any parameter map returned
//      by NVSDK_NGX_GetParameters; NGX will manage the lifetime of those
//      parameter maps.
//      This function may return NVSDK_NGX_Result_FAIL_OutOfDate if an older driver, which
//      does not support this API call is being used. This function may only be called
//      after a successful call into NVSDK_NGX_Init.

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_D3D11_DestroyParameters(
    NVSDK_NGX_Parameter* InParameters);

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_D3D12_DestroyParameters(
    NVSDK_NGX_Parameter* InParameters);

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_VULKAN_DestroyParameters(
    NVSDK_NGX_Parameter* InParameters);
```

Once parameters are released, you should release the SDK instance and all resources allocated with the following method:

```
// NVSDK_NGX_Shutdown
// -----
```



```
//
// DESCRIPTION:
//     Shuts down the current SDK instance and releases all resources.
//

NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_D3D11_Shutdown1();
NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_D3D12_Shutdown1();
NVSDK_NGX_API NVSDK_NGX_Result NVSDK_CONV NVSDK_NGX_VULKAN_Shutdown1();
```

3.3 RTX Video DX11 and DX12 Wrapper Files



Note: Both the VSRDemo and TrueHDRDemo samples are implemented using the wrapper files. For a reference to a correct implementation, see <RTX_VIDEO_SDK>\samples\TrueHDR and <RTX_VIDEO_SDK>\samples\VSR.

As part of the RTX Video SDK samples, there are NGX Wrapper classes that provide a simple integration for using NGX. Currently, there is support for both DX11 and DX12. You can either use the classes directly or adapt the code into your application. These files have information in the header with details about how to use the classes and what additional code is needed to be added to your application. They are in the <RTX_VIDEO_SDK>\samples\common\ngx_wrapper directory. The files are:

```
CDx11NGXTrueHDR.h, CDx11NGXTrueHDR.cpp,
CDx12NGXTrueHDR.h, CDx12NGXTrueHDR.cpp,
CDx11NGXVSR.h, CDx11NGXVSR.cpp,
CDx12NGXVSR.h, CDx12NGXVSR.cpp,
CDx1xNGX_common.h
```

While NGX is not thread-safe, these classes do initialize DX11 and DX12 API multithreading to aid multithread efforts. There are some additional requirements for DX12 thread safety that are needed in your application. Those are detailed in the header file.

For these features, input resources must be in a RGB format. A typical video app does this:

```
video decode -> vpb1t to RGB -> TrueHDR to HDR swap back buffer -> Present
```

Here are the items that add to your project after installing the RTX Video SDK.

1. Add include path for \$(NV_RTX_VIDEO_SDK)\include
2. Add lib path for \$(NV_RTX_VIDEO_SDK)\lib\Windows\x64
3. Add lib nvsdk_ngx_s.lib for static lib, nvsdk_ngx_d.lib for dynamic lib
#pragma comment(lib, "nvsdk_ngx_s.lib") // RTX Video SDK

3.3.1 TrueHDR Wrapper Functions

For DX11 TrueHDR, these are the functions your application calls.

NVIDIA CONFIDENTIAL

NVIDIA RTX Video SDK

SWE-GPUDRV-002-PGRF | 25

```

HRESULT CreateFeature(ID3D11Device* pD3D11Device);
HRESULT EvaluateFeature(ID3D11Texture2D* pDstTexture2d, RECT rcOutput,
    ID3D11Texture2D* pSrcTexture2d, RECT rcInput,
    UINT Contrast,        // 0 to 200 for HDR Contrast    (default 100)
    UINT Saturation,      // 0 to 200 for HDR Saturation (default 100)
    UINT MiddleGray,      // 10 to 100 for HDR MiddleGray (default 50)
    UINT MaxLuminance); // 400 to 2000 for MaxLuminance (default 1000)

void ReleaseFeature();

```

For DX12 TrueHDR, these are the functions your application calls.

```

HRESULT CreateFeature(ID3D12Device* pD3D12Device,
    UINT uGPUNodeMask, UINT uGPUVisibleNodeMask);

HRESULT EvaluateFeature(ID3D12Resource* pDstResource,
    CDx12SyncObject* pDstSyncObj, RECT rcOutput,
    ID3D12Resource* pSrcResource,
    CDx12SyncObject* pSrcSyncObj, RECT rcInput,
    UINT Contrast,        // 0 to 200 for HDR Contrast    (default 100)
    UINT Saturation,      // 0 to 200 for HDR Saturation (default 100)
    UINT MiddleGray,      // 10 to 100 for HDR MiddleGray (default 50)
    UINT MaxLuminance); // 400 to 2000 for MaxLuminance (default 1000)

void ReleaseFeature();

```

3.3.2 VSR Wrapper Functions

For DX11 VSR, these are the functions your application calls.

```

HRESULT CreateFeature(ID3D11Device* pD3D11Device);

HRESULT EvaluateFeature(ID3D11Texture2D* pDstTexture2d, RECT rcOutput,
    ID3D11Texture2D* pSrcTexture2d, RECT rcInput,
    int Quality);

void ReleaseFeature();

```

For DX12 VSR, these are the functions your application calls.

```

HRESULT CreateFeature(ID3D12Device* pD3D12Device,
    UINT uGPUNodeMask, UINT uGPUVisibleNodeMask);

HRESULT EvaluateFeature(ID3D12Resource* pDstResource,
    CDx12SyncObject* pDstSyncObj, RECT rcOutput,
    ID3D12Resource* pSrcResource,
    CDx12SyncObject* pSrcSyncObj, RECT rcInput,
    int Quality); // VSR quality 0:bicubic 1-4:4 is highest

```

```
void ReleaseFeature();
```

3.3.3 DX12 Sync Requirements

DX12 requires sync on resources for separate command list executions. This is done with a fence; each command list issues a wait on before starting to issue commands and a signal after the execution of the command list. An example of a `CDx12SyncObject` class supporting generalized fence usage is at the end of `CDx12NGXTrueHDR.cpp`. This is the general usage:

1. Insert wait for resource fences before starting decode/rendering.
2. Signal on the command queue when done with decode/rendering.
3. Pass the source/dest fences into `EvaluateFeature`.
4. It will insert waits before starting `EvaluateFeature`, and signals when done.
5. Insert wait before present and signal after doing present.

3.3.4 Code Flow Modifications for Application

It is likely that the RTX Video SDK is being integrated into an existing application. Below, we have outlined the necessary modifications to an existing DX11 or DX12 pipeline to support RTX Video SDK features.

3.3.4.1 TrueHDR used with DX11 DXGI or DirectComposition

Initialization:

- > Create `ID3D11Device` and `ID3D11Context`.
- > Call `CreateFeature` to verify it is supported.
- > Create `DXGI_FORMAT_R8G8B8A8_UNORM` `ID3D11Texture2D` for source input to feature.
 - For an array, create one resource for each entry in the array (array of textures).
 - Set an input rect to define you source size.
- > Create a swap chain in HDR, either in `DXGI_FORMAT_R10G10B10A2_UNORM` or `DXGI_FORMAT_R16G16B16A16_FLOAT`.
 - Have several backbuffers, use `DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL` or `DXGI_SWAP_EFFECT_FLIP_DISCARD`.
 - Set an output rect to define a destination size.
- > If possible, create output buffers with `D3D11_BIND_UNORDERED_ACCESS`.

Execution:

- > Render into source RGB.
 - For Video apps with YUV surfaces, use `VPBlt` from the YUV to the source RGB.
- > Call `EvaluateFeature` with DX11 pointers,
 - source RGB, source rect,
 - dest HDR, dest rect,

- other TrueHDR parameters.



Note: EvaluateFeature uses ID3D10Multithread Enter/Leave for the device's critical section to ensure thread safety.

- > Present dest HDR back buffer.

Shutdown:

- > Call ReleaseFeature from a shutdown function, not the class destructor.
 - There is a critical section used by NGX that the runtime deletes in a destructor.

3.3.4.2 TrueHDR used with DX11 PresentationManager

PresentationManager works similarly except there is no swapchain.

Rather than passing in decode render target array, follow this instead.

- > Create an array of HDR textures at the size you want to present.
 - Use D3D11_BIND_UNORDERED_ACCESS.
 - Create one resource for each entry in the array (array of textures).
- > Pass that array to create of the PresentationManger to use for adding buffers and presentation surfaces.

3.3.4.3 VSR used with DX11 DXGI or DirectComposition

Initialization:

- > Create ID3D11Device and ID3D11Context.
- > Call CreateFeature to verify it is supported.
- > Create DXGI_FORMAT_R8G8B8A8_UNORM ID3D11Texture2D for source input to feature.
 - For an array, create one resource for each entry in the array (array of textures).
 - Set an input rect to define you source size.
- > Create a swap chain in DXGI_FORMAT_R8G8B8A8_UNORM.
 - Have several backbuffers.
 - Set an output rect to define a destination size.
- > If possible, create output buffers with D3D11_BIND_UNORDERED_ACCESS.

Execution:

- > Render into source RGB.
 - For Video apps with YUV surfaces, use VPB1t from the YUV to the source RGB.
- > Call EvaluateFeature with DX11 pointers,
 - source RGB, source rect,
 - dest RGB, dest rect,
 - quality.



Note: `EvaluateFeature` uses `ID3D10Multithread` `Enter/Leave` for the device's critical section to ensure thread safety.

- > Present dest RGB back buffer.

Shutdown:

- > Call `ReleaseFeature` from a shutdown function, not the class destructor.
 - There is a critical section used by NGX that the runtime deletes in a destructor.

3.3.4.4 VSR used with DX11 PresentationManager

`PresentationManager` works similarly except there is no swapchain.

Instead of passing in decode render target array do this.

- > Create an array of RGB textures at the size you want to present.
 - Use `D3D11_BIND_UNORDERED_ACCESS`.
 - Create one resource for each entry in the array (array of textures).
- > Pass that array to create of the `PresentationManager` to use for adding buffers and presentation surfaces.

3.3.4.5 TrueHDR Used with DX12 and DXGI Present

Initialization:

- > Create `ID3D12Device` and determine GPU Node Mask and Visible Node Mask.
- > Call `CreateFeature` to verify it is supported.
- > Create `DXGI_FORMAT_R8G8B8A8_UNORM` `ID3D12Resource` for source input to feature.
 - For an array, create one resource for each entry in the array (array of textures).
- > Set an input rect to define you source size.
- > Create a swap chain in HDR, either in `DXGI_FORMAT_R10G10B10A2_UNORM` or `DXGI_FORMAT_R16G16B16A16_FLOAT`.
 - Have several backbuffers, use `DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL` or `DXGI_SWAP_EFFECT_FLIP_DISCARD`.
 - Set an output rect to define a destination size.
- > If possible, create output buffers with `D3D11_BIND_UNORDERED_ACCESS`.

Execution:

- > Render into source RGB.
 - For Video apps with YUV surfaces, use `VPB1t` from the YUV to the source RGB.
- > In the `CommandQueue` used for rendering, issue a signal.
- > Call `EvaluateFeature` with DX12 pointers,
 - source RGB, source sync object, source rect,
 - dest HDR, dest sync object, dest rect,
 - other TrueHDR parameters.



Note: Evaluate will insert a wait into its `CommandQueue` for both source and dest sync objects, then insert a signal into both.

- > Present dest HDR back buffer.
- > In the `CommandQueue` used for present:
 - Issue a wait on both sync objects before the present.
 - Then issue a wait on both sync objects after the present.

Shutdown:

- > Call `ReleaseFeature` from a shutdown function, not the class destructor.
 - There is a critical section used by NGX that the runtime deletes in a destructor.

3.3.4.6 VSR Used with DX12 and DGXI Present

Initialization:

- > Create `ID3D12Device` and determine GPU Node Mask and Visible Node Mask.
- > Call `CreateFeature` to verify it is supported.
- > Create `DXGI_FORMAT_R8G8B8A8_UNORM` `ID3D12Resource` for source input to feature.
 - For an array, create one resource for each entry in the array (array of textures).
 - Set an input rect to define you source size.
- > Create a swap chain in `DXGI_FORMAT_R8G8B8A8_UNORM`.
 - Have several backbuffers.
 - Set an output rect to define a destination size.
- > If possible, create output buffers with `D3D11_BIND_UNORDERED_ACCESS`.

Execution:

- > Render into source RGB.
 - For Video apps with YUV surfaces, use `VPB1t` from the YUV to the source RGB.
- > In the `CommandQueue` used for rendering, issue a signal.
- > Call `EvaluateFeature` with DX12 pointers,
 - source RGB, source sync object, source rect,
 - dest RGB, dest sync object, dest rect,
 - quality.



Note: Evaluate will insert a wait into its `CommandQueue` for both source and dest sync objects, then insert a signal into both.

- > Present dest RGB back buffer.
- > In the `CommandQueue` used for present:
 - Issue a wait on both sync objects before the present.
 - Then issue a wait on both sync objects after the present.

Shutdown:

- > Call `ReleaseFeature` from a shutdown function, not the class destructor.
 - There is a critical section used by NGX that the runtime deletes in a destructor.

3.3.5 Using Both VSR and TrueHDR

VSR and TrueHDR can be run sequentially in that order. TrueHDR must come after VSR, since TrueHDR outputs to HDR, while VSR does not support HDR input. Create an array of RGB textures to serve as an intermediate between VSR output and TrueHDR input.

3.4 Sample RTX Video API



Note: The `TrueHDR_VSR_Demo` is implemented using the Sample RTX Video API. For an implementation reference, see `<RTX_VIDEO_SDK>\samples\TrueHDR_VSR_Demo`.

For use cases that require less fine control over the SDK internals, a simplified API is available. Currently there is support for both DX11 and DX12. The sample API consists of the following files:

```
utils.h
rtx_video_api.h
rtx_video_api_dx11_impl.cpp
rtx_video_api_dx12_impl.cpp
```

To add these to your project, add the following:

1. Add include path for `$(NV_RTX_VIDEO_SDK)\samples\RTX_Video_API`
2. Add source files `rtx_video_api_dx11_impl.cpp` and/or `rtx_video_api_dx12_impl.cpp`

Add or remove graphics APIs as needed. Like the wrapper classes, the API is a sample and can be further modified to suit your individual needs.

The sample RTX Video API can also be built as a static library. Build the library from [Build the Samples](#). Confirm the `NV_RTX_VIDEO_SDK` is pointing to the RTX Video SDK top-level directory before building. Then, include the following lib and header.

```
RTX_Video_API.lib
rtx_video_api.h
```

3.4.1 RTX Video API Functions

Like the wrapper classes, calling is separated into create, evaluate, and shutdown functions. A key difference is that the API simplifies using VSR and THDR into one set of API calls and handles VSR+THDR cases for you. These are the functions your application calls.

```
API_B00L rtx_video_api_dx11_create(ID3D11Device* pD3DDevice, API_B00L
```

```

        THDREnable, API_BOOL VSREnable);
API_BOOL rtx_video_api_dx11_evaluate(ID3D11Texture2D* pInput, ID3D11Texture2D* pOutput,
        API_RECT inputRect, API_RECT outputRect,
        API_VSR_Setting* pVSRSetting,
        API_THDR_Setting* pTHDRSetting);
void rtx_video_api_dx11_shutdown();

API_BOOL rtx_video_api_dx12_create(ID3D12Device* pD3DDevice, uint32_t uGPUNodeMask,
        uint32_t uGPUVisibleNodeMask, API_BOOL THDREnable,
        API_BOOL VSREnable);
API_BOOL rtx_video_api_dx12_evaluate(ID3D12Resource* pInput, ID3D12Resource* pOutput,
        ID3D12Fence* pInFence, uint64_t& qwInFenceValue,
        ID3D12Fence* pOutFence, uint64_t& qwOutFenceValue,
        API_RECT inputRect, API_RECT outputRect,
        API_VSR_Setting* pVSRSetting, API_THDR_Setting*
        pTHDRSetting);
void rtx_video_api_dx12_shutdown();

```

3.4.2 RTX Video API Struct Definitions

The functions in the previous section make use of structs defined in `<RTX_VIDEO SDK>\samples\RTX_Video_API\rtx_video_api.h`.

```

typedef struct
{
    uint32_t QualityLevel;
} API_VSR_Setting;

typedef struct
{
    uint32_t Contrast;
    uint32_t Saturation;
    uint32_t MiddleGray;
    uint32_t MaxLuminance;
} API_THDR_Setting;

typedef unsigned int API_BOOL;
#define API_BOOL_FAIL      0
#define API_BOOL_SUCCESS   1

typedef struct
{
    uint32_t left;
    uint32_t top;
    uint32_t right;
    uint32_t bottom;
} API_RECT;

```


3.4.3 Sync and Flow Requirements

Sync and flow requirements are the same as those with the wrapper class. See [DX12 Sync Requirements](#) and [Code Flow](#).

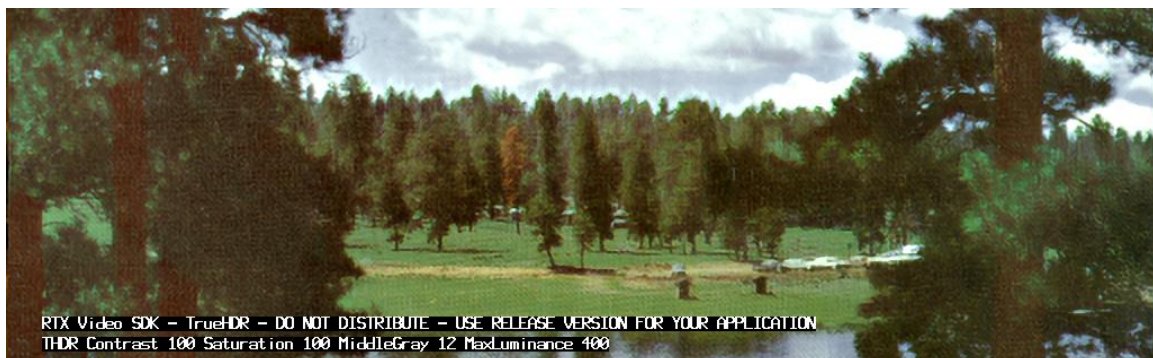
3.5 Running and Verifying RTX Video SDK Features

To run the RTX Video SDK features, copy the desired features DLLs from <RTX_VIDEO_SDK>\bin\Windows\x64\dev to the build folder. The feature binaries will be picked up from the same directory as the running application unless otherwise specified during [SDK initialization](#) in NVSDK_NGX_FeatureCommonInfo.



Note: For information on when to use the \rel binaries instead, see [Distributing RTX Video Features](#).

If the feature is taking effect, you should see watermarks like the following in the lower left corner of the frame.



Chapter 4. Distributing RTX Video Features with Your Application

The RTX Video SDK contains feature runtime DLLs for development and release purposes. These are in the `<RTX VIDEO SDK>\bin\Windows\x64` folder and should be installed in the same folder as your application's executable or path specified in `NVSDK_NGX_FeatureCommonInfo`.

DLLs in `<RTX VIDEO SDK>\bin\Windows\x64\dev` should be used for development. These are usually watermarked and display useful debug information.

For release, use the DLLs in the `<RTX VIDEO SDK>\bin\Windows\x64\rel` folder. These binaries are optimized for release and generally do not contain a watermark. If a "PreRelease" watermark is visible, contact your NVIDIA developer relations manager to discuss distribution.

If there is application specific feature tuning required, please contact your NVIDIA developer relations manager to obtain the correct application ID (`InApplicationId`) for your application. Use `0` until NVIDIA has assigned you an application ID. See [Initializing RTX Video](#) for application ID usage.



Note: You need to distribute only DLLs for the features your application is using. For example, if your application is only using TrueHDR, then you should only include `nvngx_truehdr.dll` with the rest of your binary files.

Table 1. AI Features

AI Feature	NGX Feature DLLs
AI TrueHDR	<code>nvngx_truehdr.dll</code>
AI VSR	<code>nvngx_vsr.dll</code>

The installer for your application should treat these DLLs in the same way as other components to your application and remove them on uninstall.

Chapter 5. RTX Video Features

5.1 TrueHDR: Convert Surface from SDR to HDR

What does this feature do?

The TrueHDR feature inputs a RGB SDR surface, performs debanding, and outputs it as a HDR surface.

TrueHDRDemoNGX Sample

The `TrueHDRDemoNGX` sample application included in the RTX Video SDK demonstrates the conversion of SDR to TrueHDR. It inputs a YUV or RGB surface and outputs HDR. A second window shows the surface without TrueHDR. The `ReadMe.txt` details how to generate YUV source files by using FFmpeg.

The following command reads a YUV file and displays it repeatedly.

```
TrueHDRDemoNGX.exe -i NV12_1280_720_1280_120_star_sky.yuv
```

Table 2. TrueHDR Supported Parameters

Creation Parameter	Description
<code>pD3DDevice</code>	DX Device created by application (ID3D11Device or ID3D12Device)
Evaluation Parameter	Description
<code>Output</code>	HDR surface to for TrueHDR output (ID3D11Texture2D or ID3D12Resource)
<code>Input</code>	RGB surface source for TrueHDR input (ID3D11Texture2D or ID3D12Resource)
<code>RectOutput</code>	Defines top, left, bottom, right in pixels for TrueHDR output to be put in the output surface
<code>RectInput</code>	Defines top, left, bottom, right in pixels for TrueHDR input to be read from the input surface
<code>pDstSyncObject</code>	DX12 Only: Sync object with the fence for the Output surface
<code>pDstSyncObject</code>	DX12 Only: Sync object with the fence for the Input surface

Contrast	<p>0 to 200 for HDR Contrast (default 100), adjusts the difference between lights and darks</p> 
Saturation	<p>0 to 200 for HDR Saturation (default 100), adjusts color intensity</p> 
MiddleGray	<p>10 to 100 for HDR MiddleGray (default 50), adjusts average brightness</p> 
MaxLuminance	<p>400 to 2000 for Monitor MaxLuminance (default 1000), adjusts peak brightness in nits</p> 

5.1.1 TrueHDR Evaluation Parameter Struct Definitions

TrueHDR parameters can be found in <RTX_VIDEO_SDK>\include\nv sdk_ngx_helpers_thdr*.h

```
typedef struct NVSDK NGX_D3D11_TRUEHDR_Eval_Params
{
    ID3D11Resource*      pInput;
    ID3D11Resource*      pOutput;
    NVSDK_NGX_Coordinates InputSubrectTL;
    NVSDK_NGX_Dimensions InputSubrectBR;
    NVSDK_NGX_Coordinates OutputSubrectTL;
    NVSDK_NGX_Dimensions OutputSubrectBR;
    unsigned int          Contrast;
    unsigned int          Saturation;
    unsigned int          MiddleGray;
    unsigned int          MaxLuminance;
} NVSDK_NGX_D3D11_TRUEHDR_Eval_Params;

typedef struct NVSDK_NGX_D3D12_TRUEHDR_Eval_Params
{
    ID3D12Resource*      pInput;
    ID3D12Resource*      pOutput;
    NVSDK_NGX_Coordinates InputSubrectTL;
    NVSDK_NGX_Dimensions InputSubrectBR;
    NVSDK_NGX_Coordinates OutputSubrectTL;
    NVSDK_NGX_Dimensions OutputSubrectBR;
    unsigned int          Contrast;
    unsigned int          Saturation;
    unsigned int          MiddleGray;
    unsigned int          MaxLuminance;
} NVSDK_NGX_D3D12_TRUEHDR_Eval_Params;

typedef struct NVSDK_NGX_VK_TRUEHDR_Eval_Params
{
    NVSDK_NGX_Resource_VK* pInput;
    NVSDK_NGX_Resource_VK* pOutput;
    NVSDK_NGX_Coordinates InputSubrectTL;
    NVSDK_NGX_Dimensions InputSubrectBR;
    NVSDK_NGX_Coordinates OutputSubrectTL;
    NVSDK_NGX_Dimensions OutputSubrectBR;
    unsigned int          Contrast;
    unsigned int          Saturation;
    unsigned int          MiddleGray;
    unsigned int          MaxLuminance;
} NVSDK_NGX_VK_TRUEHDR_Eval_Params;

typedef struct NVSDK_NGX_Coordinates
{

```

```

    unsigned int X;
    unsigned int Y;
} NVSDK NGX_Coordinates;

typedef struct NVSDK NGX_Dimensions
{
    unsigned int Width;
    unsigned int Height;
} NVSDK NGX_Dimensions;

```

5.1.2 Sample Code for TrueHDR DX11

The following code snippet shows how to use the TrueHDR feature with the NGX DX11 Wrapper Class.

```

////////////////////
// In .h
class CDx11NGXTrueHDR*      m_pCNGXTrueHDR      = nullptr;
RECT                        m_rcNGXInput          = {};
RECT                        m_rcNGXOutput         = {};
ID3D11Texture2D*           m_pNGXInputBuffer     = nullptr;

////////////////////
// In Initialization:
// create feature after creating DX11Device
m_pCNGXTrueHDR = new CDx11NGXTrueHDR;
hr = m_pCNGXTrueHDR->CreateFeature(m_pD3D11Device);
if (FAILED(hr))
{
    SafeDelete(m_pCNGXTrueHDR);
    if (FAILED(hr)) return hr;
}

// create swap chain with HDR format
scd.Format = (bUseABGR10) ? DXGI_FORMAT_R10G10B10A2_UNORM :
                DXGI_FORMAT_R16G16B16A16_FLOAT;

// set swap chain color space
if (scd.Format == DXGI_FORMAT_R16G16B16A16_FLOAT)
    m_pSwapChain->SetColorSpace1(DXGI_COLOR_SPACE_RGB_FULL_G10_NONE_P709);
if (scd.Format == DXGI_FORMAT_R10G10B10A2_UNORM)
    m_pSwapChain->SetColorSpace1(DXGI_COLOR_SPACE_RGB_FULL_G2084_NONE_P2020);

// set input size for NGX
m_rcNGXInput = { 0, 0, (LONG)scd.Width, (LONG)scd.Height };
// set output size for NGX
m_rcNGXOutput = { 0, 0, (LONG)scd.Width, (LONG)scd.Height };
// set vpBlit output size to input size of NGX
m_rcVpOutput = m_rcNGXInput;
// set vpBlit output format to DXGI_FORMAT_R8G8B8A8_UNORM
m_FmtVpOutput = DXGI_FORMAT_R8G8B8A8_UNORM;

```

```

// create a resource for output from vpBlt for input to NGX.
// Use format   DXGI_FORMAT_R8G8B8A8_UNORM and
                D3D12_RESOURCE_FLAG_ALLOW_UNORDERED_ACCESS
// create a texture2d for output from vpBlt for input to NGX. Use format
hr = CreateTexture2D(m_rcNGXInput.right, m_rcNGXInput.bottom, 1,
                    DXGI_FORMAT_R8G8B8A8_UNORM, &m_pNGXInputBuffer);

// optionally get monitor max luminance and verify it is in hdr mode
DXGI_OUTPUT_DESC1 DxgiOutputDesc = {};
pDXGIOutput6->GetDesc1(&DxgiOutputDesc);
m_uMaxLuminance = (UINT)((DxgiOutputDesc.MaxLuminance >= 400.0f) ?
                        DxgiOutputDesc.MaxLuminance : 1000);
if (DxgiOutputDesc.ColorSpace != DXGI_COLOR_SPACE_RGB_FULL_G2084_NONE_P2020)
    return E_FAIL; //MonitorNotInHDRMode

//////////
// In Update BackBuffer for Present
// VpBlt from decode output to NGX input buffer, RGB apps skip this.
hr = VpBlt(pDecodeRTInUse, m_pNGXInputBuffer);
// Apply TrueHDR with EvaluateFeature
hr = m_pCNGXTrueHDR->EvaluateFeature(pSwapBackBuffer,
                                    m_rcNGXOutput,
                                    m_pNGXInputBuffer,
                                    m_rcNGXInput,
                                    Contrast, Saturation,
                                    MiddleGray, MaxLuminance);

//////////
// In Deinitialization (call prior to destructor:
if (m_pCNGXTrueHDR)
{
    m_pCNGXTrueHDR->ReleaseFeature();
    SafeDelete(m_pCNGXTrueHDR);
}

```

5.1.2.1 Using DX11 PresentationManager with TrueHDR

When using the `PresentationManager`, instead of using a swap chain, typically the array of textures for the decode render targets is passed directly to the `PresentationManager`.

```

hr = m_PresentationManager->AddBufferFromResource(pResource,
                                                  &m_paPresentationBuffer[n]);

```

Then, to present the buffer after it is updated, the present start time and the indexed buffer to present are provided to the `PresentationManager`.

```

hr = m_PresentationSurface->SetBuffer(m_paPresentationBuffer[PicInfo.uPicIndex]);
m_PresentationManager->SetTargetTime(m_presentAtTime);
hr = m_PresentationManager->Present();

```

For TrueHDR, create an array of textures in the HDR format desired with the same number as the decode render targets. They need to be created as displayable.

```

    for (UINT n = 0; n < m_NumDecodeSurfaces && hr == S_OK; n++)
    {
        // Use format    hdrFMT = DXGI_FORMAT_R10G10B10A2_UNORM
                        (or DXGI_FORMAT_R16G16B16A16_FLOAT)
        // Use bind      D3D11_BIND_UNORDERED_ACCESS
        // create a texture2d for output from vpBlt for input to NGX. Use format
        hr = CreateTexture2D(m_rcNGXOutput.right, m_rcNGXOutput.bottom, 1,
                            hdrFMT, & m_pPresentATSurfaceArray[n]);
    }

//////////
// To prep for PresentationManager Present
// VpBlt from decode output to NGX input buffer, RGB apps skip this.
hr = VpBlt(pDecodeRTInUse[n], m_pNGXInputBuffer);
// Apply TrueHDR with EvaluateFeature
hr = m_pCNGXTrueHDR->EvaluateFeature(m_pPresentATSurfaceArray[n],
                                     m_rcNGXOutput,
                                     m_pNGXInputBuffer,
                                     m_rcNGXInput,
                                     Contrast, Saturation,
                                     MiddleGray, MaxLuminance);

```

5.1.3 Sample Code for TrueHDR DX12

The following code snippet shows how to use the TrueHDR feature with the NGX DX12 Wrapper Class.

```

//////////
// In .h
CDx12NGXTrueHDR*    m_CNGXTrueHDR           = nullptr;
RECT                m_rcNGXInput            = {};
RECT                m_rcNGXOutput           = {};
ID3D12Resource*     m_NGXInputBuffer        = nullptr;
CDx12SyncObject*    m_NGXInputSyncObj      = nullptr;

//////////
// In Initialization:
// create feature after creating DX12Device
m_pCNGXTrueHDR = new CDx12NGXTrueHDR;
hr = m_pCNGXTrueHDR->CreateFeature(m_pD3D12Device,
                                   m_GPUNodeMask, m_GPUVisibleNodeMask);

if (FAILED(hr))
{
    SafeDelete(m_pCNGXTrueHDR);
    if (FAILED(hr)) return hr;
}

```



```

// create sync object
m_NGXInputSyncObj = new CDx12SyncObject(m_D3D12Device);

// create swap chain with HDR format
scd.Format = (bUseABGR10) ? DXGI_FORMAT_R10G10B10A2_UNORM :
                    DXGI_FORMAT_R16G16B16A16_FLOAT;
// set swap chain color space
if (scd.Format == DXGI_FORMAT_R16G16B16A16_FLOAT)
    m_pSwapChain->SetColorSpace1(DXGI_COLOR_SPACE_RGB_FULL_G10_NONE_P709);
if (scd.Format == DXGI_FORMAT_R10G10B10A2_UNORM)
    m_pSwapChain->SetColorSpace1(DXGI_COLOR_SPACE_RGB_FULL_G2084_NONE_P2020);

// set input size for NGX
m_rcNGXInput = { 0, 0, (LONG)scd.Width, (LONG)scd.Height };
// set output size for NGX
m_rcNGXOutput = { 0, 0, (LONG)scd.Width, (LONG)scd.Height };
// set vpBlt output size to input size of NGX
m_rcVpOutput = m_rcNGXInput;
// set vpBlt output format to DXGI_FORMAT_R8G8B8A8_UNORM
m_FmtVpOutput = DXGI_FORMAT_R8G8B8A8_UNORM;

// create a resource for output from vpBlt for input to NGX.
// Use format DXGI_FORMAT_R8G8B8A8_UNORM and
// D3D12_RESOURCE_FLAG_ALLOW_UNORDERED_ACCESS
D3D12_RESOURCE_DESC textureDesc =
    CD3DX12_RESOURCE_DESC::Tex2D(DXGI_FORMAT_R8G8B8A8_UNORM,
        m_rcNGXInput.right, m_rcNGXInput.bottom, 1, 1, 1, 0,
        D3D12_RESOURCE_FLAG_ALLOW_UNORDERED_ACCESS);
CD3DX12_HEAP_PROPERTIES heapProps =
    CD3DX12_HEAP_PROPERTIES(D3D12_HEAP_TYPE_DEFAULT,
        m_GPUNodeMask, m_GPUVisibleNodeMask);
hr = m_D3D12Device->CreateCommittedResource(&heapProps, D3D12_HEAP_FLAG_NONE,
    &textureDesc, D3D12_RESOURCE_STATE_COMMON, nullptr,
    IID_PPV_ARGS(&m_NGXInputBuffer));

// optionally get monitor max luminance and verify it is in hdr mode
DXGI_OUTPUT_DESC1 DxgiOutputDesc = {};
pDXGIOutput6->GetDesc1(&DxgiOutputDesc);
m_uMaxLuminance = (UINT)((DxgiOutputDesc.MaxLuminance >= 400.0f) ?
    DxgiOutputDesc.MaxLuminance : 1000);
if (DxgiOutputDesc.ColorSpace != DXGI_COLOR_SPACE_RGB_FULL_G2084_NONE_P2020)
    return E_FAIL; //MonitorNotInHDRMode

//////////
// In Update BackBuffer for Present
// VpBlt from decode output to NGX input buffer, RGB apps skip this.
hr = VpBlt(pDecoderRTInUse, pSrcSyncObj, m_pNGXInputBuffer, m_NGXInputSyncObj);
// Apply TrueHDR with EvaluateFeature
hr = m_pCNGXTrueHDR->EvaluateFeature(pSwapBackBuffer, pSwapSyncObj,
    m_rcNGXOutput,

```

```

        m_pNGXInputBuffer, m_NGXInputSyncObj,
        m_rcNGXInput,
        Contrast, Saturation,

        MiddleGray, MaxLuminance);

//////////
// In Deinitialization (call prior to destructor:
    if (m_pCNGXTrueHDR)
    {
        m_pCNGXTrueHDR->ReleaseFeature();
        SafeDelete(m_pCNGXTrueHDR);
    }

```

5.2 VSR: Upscale SDR Surface with Artifact Reduction

What does this feature do?

The VSR feature inputs a RGB SDR surface and upscales, sharpens, and deblocks it to an output RGB surface.

VSR Sample

The VSR Demo sample application included in the RTX Video SDK demonstrates the conversion of VSR to an RGB SDR surface.

The following command reads a YUV file and displays it repeatedly.

```
VSRDemo.exe -i NV12_1280_720_1280_120_star_sky.yuv
```

Table 3. VSR Supported Parameters

Creation Parameter	Description
pD3DDevice	DX Device created by application (ID3D11Device or ID3D12Device)
Evaluation Parameter	Description
Output	HDR surface to for TrueHDR output (ID3D11Texture2D or ID3D12Resource)
Input	RGB surface source for TrueHDR input (ID3D11Texture2D or ID3D12Resource)
RectOutput	Defines top, left, bottom, right in pixels for VSR output to be put in the output surface
RectInput	Defines top, left, bottom, right in pixels for VSR input to be read from the input surface
pDstSyncObject	DX12 Only: Sync object with the fence for the Output surface
pSrcSyncObject	DX12 Only: Sync object with the fence for the Input surface

Quality	<p>NVSDK_NGX_VSR_QualityLevel enumeration where 0 is bicubic, 1 to 4 are VSR quality levels. 4 is best quality but longest runtime.</p>  <p>(c) copyright 2008, Blender Foundation / www.bigbuckbunny.org</p>
---------	---

5.2.1 VSR Evaluation Parameter Struct Definitions

VSR parameters can be found in `nvsdk_ngx_helpers_vsr*.h` and `nvsdk_ngx_defs_vsr.h` in `<RTX VIDEO SDK>\include\`.

```
typedef struct NVSDK_NGX_D3D11_VSR_Eval_Params
{
    ID3D11Resource*      pInput;
    ID3D11Resource*      pOutput;
    NVSDK_NGX_Coordinates InputSubrectBase;
    NVSDK_NGX_Dimensions InputSubrectSize;
    NVSDK_NGX_Coordinates OutputSubrectBase;
    NVSDK_NGX_Dimensions OutputSubrectSize;
    NVSDK_NGX_VSR_QualityLevel QualityLevel;
} NVSDK_NGX_D3D11_VSR_Eval_Params;

typedef struct NVSDK_NGX_D3D12_VSR_Eval_Params
{
    ID3D12Resource*      pInput;
    ID3D12Resource*      pOutput;
    NVSDK_NGX_Coordinates InputSubrectBase;
    NVSDK_NGX_Dimensions InputSubrectSize;
    NVSDK_NGX_Coordinates OutputSubrectBase;
    NVSDK_NGX_Dimensions OutputSubrectSize;
    NVSDK_NGX_VSR_QualityLevel QualityLevel;
} NVSDK_NGX_D3D12_VSR_Eval_Params;

typedef struct NVSDK_NGX_VK_VSR_Eval_Params
{
    NVSDK_NGX_Resource_VK* pInput;
    NVSDK_NGX_Resource_VK* pOutput;
}
```

```

    NVSDK_NGX_Coordinates      InputSubrectBase;
    NVSDK_NGX_Dimensions       InputSubrectSize;
    NVSDK_NGX_Coordinates      OutputSubrectBase;
    NVSDK_NGX_Dimensions       OutputSubrectSize;
    NVSDK_NGX_VSR_QualityLevel QualityLevel;
} NVSDK_NGX_VK_VSR_Eval_Params;

typedef enum NVSDK_NGX_VSR_QualityLevel
{
    NVSDK_NGX_VSR_Quality_Bicubic = 0,
    NVSDK_NGX_VSR_Quality_Low     = 1,
    NVSDK_NGX_VSR_Quality_Medium  = 2,
    NVSDK_NGX_VSR_Quality_High    = 3,
    NVSDK_NGX_VSR_Quality_Ultra   = 4,
} NVSDK_NGX_VSR_QualityLevel;

typedef struct NVSDK_NGX_Coordinates
{
    unsigned int X;
    unsigned int Y;
} NVSDK_NGX_Coordinates;

typedef struct NVSDK_NGX_Dimensions
{
    unsigned int Width;
    unsigned int Height;
} NVSDK_NGX_Dimensions;

```

5.2.2 Sample Code for VSR DX11

The following code snippet shows how to use the VSR feature with the NGX DX11 Wrapper Class.

```

//////////
// In .h
class CDx11NGXVSR*          m_pCNGXVSR          = nullptr;
RECT                        m_rcNGXInput          = {};
RECT                        m_rcNGXOutput         = {};
ID3D11Texture2D*           m_pNGXInputBuffer     = nullptr;

//////////
// In Initialization:
    // create feature after creating DX11Device
    m_pCNGXVSR = new CDx11NGXVSR;
    hr = m_pCNGXVSR->CreateFeature(m_pD3D11Device);
    if (FAILED(hr))
    {
        SafeDelete(m_pCNGXVSR);
        return hr;
    }

```

```

// create swap chain with RGB format
scd.Format = DXGI_FORMAT_R8G8B8A8_UNORM;

// set input size for NGX
m_rcNGXInput = { 0, 0, (LONG)srcWidth, (LONG)srcHeight };
// set output size for NGX
m_rcNGXOutput = { 0, 0, (LONG)scd.Width, (LONG)scd.Height };
// set vpBlit output size to input size of NGX
m_rcVpOutput = m_rcNGXInput;
// set vpBlit output format to DXGI_FORMAT_R8G8B8A8_UNORM
m_FmtVpOutput = DXGI_FORMAT_R8G8B8A8_UNORM;

// create a resource for output from vpBlit for input to NGX.
// Use format   DXGI_FORMAT_R8G8B8A8_UNORM and
//              D3D12_RESOURCE_FLAG_ALLOW_UNORDERED_ACCESS
// create a texture2d for output from vpBlit for input to NGX. Use format
hr = CreateTexture2D(m_rcNGXInput.right, m_rcNGXInput.bottom, 1,
                    DXGI_FORMAT_R8G8B8A8_UNORM, &m_pNGXInputBuffer);

//////////
// In Update BackBuffer for Present
// VpBlit from decode output to NGX input buffer, RGB apps skip this.
hr = VpBlit(pDecoderRTInUse, m_pNGXInputBuffer);
// Apply VSR with EvaluateFeature
hr = m_pCNGXVSR->EvaluateFeature(pSwapBackBuffer, m_rcNGXOutput,
                                m_pNGXInputBuffer, m_rcNGXInput,
                                quality);

//////////
// In Deinitialization (call prior to destructor):
if (m_pCNGXVSR)
{
    m_pCNGXVSR->ReleaseFeature();
    SafeDelete(m_pCNGXVSR);
}

```

5.2.2.1 Using DX11 PresentationManager with VSR

When using the PresentationManager, instead of using a swap chain, typically the array of textures for the decode render targets is passed directly to the PresentationManager.

```

hr = m_PresentationManager->AddBufferFromResource(pResource,
                                                  &m_paPresentationBuffer[n]);

```

Then, to present the buffer after it is updated, the present start time and the indexed buffer to present are provided to the PresentationManager.

```

hr = m_PresentationSurface->SetBuffer(m_paPresentationBuffer[PicInfo.uPicIndex]);
m_PresentationManager->SetTargetTime(m_presentAtTime);

```

```
hr = m_PresentationManager->Present();
```

For VSR, create an array of textures in the RGB format desired with the same number as the decode render targets. They need to be created as displayable.

```
for (UINT n = 0; n < m_NumDecodeSurfaces && hr == S_OK; n++)
{
    // Use format    rgbFMT = DXGI_FORMAT_R8G8B8A8_UNORM
    // Use bind      D3D11_BIND_UNORDERED_ACCESS
    // create a texture2d for output from vpBlt for input to NGX. Use format
    hr = CreateTexture2D(m_rcNGXOutput.right, m_rcNGXOutput.bottom, 1,
                        rgbFMT, & m_pPresentATSurfaceArray[n]);
}

//////////
// To prep for PresentationManager Present
// VpBlt from decode output to NGX input buffer, RGB apps skip this.
hr = VpBlt(pDecoderTInUse[n], m_pNGXInputBuffer);
// Apply VSR with EvaluateFeature
hr = m_pCNGXTrueHDR->EvaluateFeature(m_pPresentATSurfaceArray[n],
                                    m_rcNGXOutput,
                                    m_pNGXInputBuffer,
                                    m_rcNGXInput,
                                    quality);
```

5.2.3 Sample Code for VSR DX12

The following code snippet shows how to use the VSR feature with the NGX DX12 Wrapper Class.

```
//////////
// In .h
class CDx11NGXVSR*    m_pCNGXVSR                = nullptr;
RECT                  m_rcNGXInput                = {};
RECT                  m_rcNGXOutput               = {};
ID3D12Resource*       m_NGXInputBuffer            = nullptr;
CDx12SyncObject*      m_NGXInputSyncObj           = nullptr;

//////////
// In Initialization:
// create feature after creating DX12Device
m_pCNGXVSR = new CDx12NGXVSR;
hr = m_pCNGXVSR->CreateFeature(m_pD3D12Device);
if (FAILED(hr))
{
    SafeDelete(m_pCNGXVSR);
    return hr;
}
```

```

// create sync object
m_NGXInputSyncObj = new CDx12SyncObject(m_D3D12Device);

// create swap chain with RGB format
scd.Format = DXGI_FORMAT_R8G8B8A8_UNORM;

// set input size for NGX
m_rcNGXInput = { 0, 0, (LONG)srcWidth, (LONG)srcHeight };
// set output size for NGX
m_rcNGXOutput = { 0, 0, (LONG)scd.Width, (LONG)scd.Height };
// set vpBlt output size to input size of NGX
m_rcVpOutput = m_rcNGXInput;
// set vpBlt output format to DXGI_FORMAT_R8G8B8A8_UNORM
m_FmtVpOutput = DXGI_FORMAT_R8G8B8A8_UNORM;

// create a resource for output from vpBlt for input to NGX.
// Use format DXGI_FORMAT_R8G8B8A8_UNORM and
D3D12_RESOURCE_FLAG_ALLOW_UNORDERED_ACCESS
D3D12_RESOURCE_DESC textureDesc =
    CD3DX12_RESOURCE_DESC::Tex2D(DXGI_FORMAT_R8G8B8A8_UNORM,
    m_rcNGXInput.right, m_rcNGXInput.bottom, 1, 1, 1, 0,
    D3D12_RESOURCE_FLAG_ALLOW_UNORDERED_ACCESS);
CD3DX12_HEAP_PROPERTIES heapProps =
    CD3DX12_HEAP_PROPERTIES(D3D12_HEAP_TYPE_DEFAULT,
    m_GPUNodeMask, m_GPUVisibleNodeMask);
hr = m_D3D12Device->CreateCommittedResource(&heapProps, D3D12_HEAP_FLAG_NONE,
    &textureDesc, D3D12_RESOURCE_STATE_COMMON, nullptr,
    IID_PPV_ARGS(&m_NGXInputBuffer));

//////////
// In Update BackBuffer for Present
// VpBlt from decode output to NGX input buffer, RGB apps skip this.
hr = VpBlt(pDecoderTInUse, pSrcSyncObj, m_pNGXInputBuffer, m_NGXInputSyncObj);
// Apply VSR with EvaluateFeature
hr = m_pCNGXVSR->EvaluateFeature(pSwapBackBuffer, m_rcNGXOutput,
    m_pNGXInputBuffer, m_rcNGXInput,
    quality);

//////////
// In Deinitialization (call prior to destructor):
if (m_pCNGXVSR)
{
    m_pCNGXVSR->ReleaseFeature();
    SafeDelete(m_pCNGXVSR);
}

```

Chapter 6. Troubleshooting / FAQ

6.1 Common Sample Build Errors

error MSB8036: The Windows SDK version 10.0.15063.0 was not found. Install the required version of Windows SDK or change the SDK version in the project property pages or by right-clicking the solution and selecting "Retarget solution".

Simply use the Visual Studio settings (i.e., right-click the solution and selecting **Retarget solution**) to retarget the projects to use the VS2019 SDK or later and tools that are on your system. The samples should be compatible.

error MSB8041: MFC libraries are required for this project. Install them from the Visual Studio installer (Individual Components tab) for any toolsets and architectures being used.

Get C++ MFC matching currently installed build tools version. For v143 build tools, install "C++ MFC for latest v143 build tools (x86 & x64)". On existing Visual Studio installs, this can be done by starting the Visual Studio Installer and selecting "Modify", then search "Individual components". On new installs, directly browse "Individual components".

fatal error C1083: Cannot open include file: 'nvsdk_ngx_defs.h': No such file or directory

The samples use the NV_RTX_VIDEO_SDK environment variable to locate the SDK. Make sure the variable is set as either a system or user environment variable. Ensure the path contains no spaces and points to the RTX Video SDK root directory (not \bin or \include). Verify both the variable name and value do not contain leading or trailing spaces.

error LNK2019: unresolved external symbol NVSDK NGX_Parameter_SetUI referenced in function "enum NVSDK NGX_Result __cdecl NGX_D3D11_EVALUATE_TRUEHDR_EXT(struct ID3D11DeviceContext *,struct NVSDK NGX_Handle *,struct NVSDK NGX_Parameter *,struct NVSDK NGX_D3D11_TRUEHDR_Eval_Params *)"

The linker is having trouble finding the RTX Video SDK libraries. Make sure \$(NV_RTX_VIDEO_SDK)\lib\Windows\x64 is added in the Linker "Additional Library Directories" and one of the following is added to your project source files.

```
#pragma comment( lib, "nvsdk_ngx_s.lib")
#pragma comment( lib, "nvsdk_ngx_d.lib")
```

error LNK2001: unresolved external symbol __imp_RegCloseKey

Some machines require explicitly adding windows libraries through pragma as well. These are some common ones that have caused issues:

```
#pragma comment( lib, "dxgi" )
#pragma comment( lib, "d3d11" )
#pragma comment( lib, "user32.lib" )
```

6.2 Common Issues When Running Samples

The sample exits while printing usage parameters.

Generally, this means one of the required parameters is missing or an input is malformed. For more detailed usage information, refer to the ReadME.md packaged with the samples.

The sample exits silently without further feedback.

This usually means that there is an input file problem or SDK runtime problem.

Confirm the input file name and format. The samples are sensitive to input file name. They must be of the form `fmt_width_height_pitch_numFrames_xxxx.xxx`.

Also, confirm the SDK feature DLLs are copied to the sample app location. **This must be done manually.**

Refer to the ReadME.md packaged with the samples for more detailed usage information. Also, consider starting the Visual Studio debugger to go line by line and find the failure point.

Which window is the RTX Video SDK output?

The SDK output is always the top window for the `TrueHDRDemo`, `VSRDemo`, and `TrueHDR_VSR_Demo`.

6.3 General Issues When Using the SDK

NVSDK NGX_Result_FAIL_PlatformError from SDK Init.

Typically, this means there was a problem with the device or device pointer provided. Make sure the device pointer is valid.

Sometimes a system reboot can resolve the issues, especially after a new driver install.

NVSDK NGX_Result_FAIL_FeatureNotSupported from SDK Init.

This occurs if GPU or driver is not supported. RTX Video SDK supports Turing 20xx+ GPUs and 550.58+ drivers.

In rare cases, this error can also be caused by an inability to call the NVIDIA API. You can verify the Nvidia API is working on your device by confirming NVIDIA Control Panel can start and show your device settings.

Sometimes a system reboot can resolve the issues, especially after a new driver install.

NVSDK NGX_Result_FAIL_UnsupportedParameter when trying to get feature available parameter.

Confirm the SDK feature DLLs are copied to your app location or your specified directory from Init.

NVSDK NGX_Result_FAIL_UnsupportedFormat from SDK Evaluate.

Either the input or output resource format is incorrect. For DX11, verify that both input and output are Texture2D resources. For both DX11 and DX12, also verify that the destination resource is UAV (Unordered Access View).

NVSDK NGX_Result_FAIL_PlatformError from SDK Evaluate.

An internal SDK error occurred that prevented evaluation. Please report this to NVIDIA.

What is the difference between rel and dev feature binaries?

The `dev` binaries should be used only during application development. They are less optimized and show a watermark in the lower right corner with feature details.

The `rel` binaries are the binaries that should be shipped with the application. On release versions of the RTX Video SDK, these binaries will have no watermark. See [Distributing RTX Video Features](#).

How do I get started without an existing application?

The samples are a good place to get familiar with both DX11, DX12 code flow, and the RTX Video SDK. For more information on DX11 and DX12, see the following Microsoft tutorial docs:

<https://learn.microsoft.com/en-us/windows/win32/direct3d11/dx-graphics-overviews>

<https://learn.microsoft.com/en-us/windows/win32/direct3d12/directx-12-programming-guide>

There currently is not reference Vulkan implementation, but a good place to start is the Vulkan Tutorial that introduces initializing Vulkan devices and working with images:

https://vulkan-tutorial.com/Texture_mapping/Images

Since Vulkan resource handles do not contain some necessary information, NGX requires a [wrapper around the Vulkan resources](#) before being passed into NGX.

Licenses & Copyright Notices

NGX, its libraries, features, and/or sample code make use of the following licensed products:

FFmpeg

License URL

<https://ffmpeg.org/legal.html>

License Text

<http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html>

Copyright (C) 2013-2020 Fabrice Bellard

FFmpeg is a trademark of Fabrice Bellard, originator of the FFmpeg project.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

VULKAN

Copyright (c) 2015-2019 LunarG, Inc.

Copyright (c) 2015-2019 Valve Corporation

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

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.

Customer 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 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 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, NGX, and RTX 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

© 2023-2024 NVIDIA CORPORATION and affiliates. All rights reserved.

