VRWORKS 360 LOW-LEVEL VIDEO

PG-08684-001_v01.1 | December 2017

Programming Guide

DOCUMENT CHANGE HISTORY

PG-08684-001_v01.1

Version	Date	Description of Change
01	2017-08-04	Initial release
01.1	2017-12-01	Release 1.1

TABLE OF CONTENTS

Chapter 1. INTRODUCTION TO VRWORKS 360 LOW-LEVEL VIDEO

The NVIDIA Video Stitch SDK (NVSS) is a low-level image processing API for stitching sets of images, typically captured with a camera rig, into 360-degree panoramas. NVSS is part of the NVIDIA VRWorks™ 360 Video SDK.

Input is a set of 32-bit RGBX images that need to be stored in device memory managed by the SDK. Output consists of equirectangular panoramas in the same pixel format, in device memory managed by the SDK. In case of monoscopic stitching, the output is a single panorama, whereas with stereoscopic pipeline the output includes two panoramas in omnidirectional stereo projection, one for each eye.

The stitching process is GPU-accelerated, and optimized for real-time performance. The stereoscopic pipeline is scalable across multiple GPUs. The stitching operation is asynchronous and the client application is responsible for synchronization when reading the output buffer.

Chapter 2. USING NVSS

This section describes the typical execution flow when NVSS is used for 360-degree panorama stitching.

2.1 SETUP

All enumerations, structures, and functions with the `nvssVideo` prefix are defined in the `nvss_video.h` header and are specific to this component. Enumerations and structures with the `nvstitch` prefix are shared with other components, such as the high-level SDK. These enumerations and structures are defined in common headers, for example, `nvstitch_common.h` and `nvstitch_common_video.h`.

The NVSS pipeline can utilize any subset of compatible GPUs[1] available on the system[2]. The user controls which GPUs will be used for stitching by passing an array of device indices. To identify compatible GPUs in a general case, the application can use the `cudaGetDeviceCount()` and `cudaGetDeviceProperties()` functions to check the compute version of each GPU, as demonstrated in the sample application.

All stitching operations are done through a stitching pipeline instance, which is managed using the `nvssVideoHandle` object. Most stitcher properties are set at initialization, using the `nvssVideoStitcherProperties_t` structure. Here, the application can specify parameters such as output size, stitching pipeline type, etc. The array of GPU indices described above is also passed through this structure.

A detailed and accurate description of the camera rig is also required to initialize and run a stitcher instance because the relative alignment of cameras is dictated by their

[1] GPUs with compute capability 5.2 or higher

[2] Monoscopic implementations use only a single GPU.

intrinsic and extrinsic parameters. A structure of type `nvstitchVideoRigProperties_t` is used for this purpose, and includes specification of intrinsic and extrinsic parameters of each camera in the rig, as well as additional information on the rig itself. This structure can be initialized manually if all values are known in the user application. Alternatively, utility functions included in the SDK can fully initialize the structure from an XML file, such as the file included in the \footage directory.

Because using an accurate rig description significantly improves the quality of the output panorama, we recommend calibrating the initial rig properties using the `nvcalib` component included in this SDK. The result of the calibration process is an `nvstitchVideoRigProperties_t` structure that can be directly used to initialize the stitcher instance. For more information on the `nvcalib` component, refer to *VRWorks 360 Calibration Programming Guide* (PG-08666-001).

After the structures containing the stitcher and rig properties are initialized, an instance can be created using `nvssVideoCreateInstance()`. The handle is returned through the output parameter. If an error occurs in the initialization, an error code is returned, and the handle parameter is not set.

2.2 OPERATIONS

A stitching loop consists of copying the input frames into the input buffers, calling the stitch function, and copying from the output buffer when the output is ready. Stitching is asynchronous, so the application must ensure proper CUDA synchronization of the output buffer access. NVSS provides an API that enables this functionality without loss of performance.

The user can directly access the input buffers after using `nvssVideoGetInputBuffer()` to get their description in the form of `nvstitchImageBuffer_t` objects. To avoid unnecessary copying, input buffers are stored and processed by the same GPU. The user application can identify which GPU holds which input buffer by calling `nvssVideoGetInputDevice().` Additionally, to facilitate optimal performance in the application, `nvssVideoGetInputStream()` can be used to determine the CUDA stream with which the input ingestion should be synchronized. The use of `nvssVideoGetInputStream()` means that the application does not need to use broad CUDA synchronization, such as `cudaStreamSynchronize(cudaStreamDefault)` or `cudaDeviceSynchronize(),` which often leads to performance loss. With NVSS, an asynchronous copy of the next input frame in the corresponding stream has guaranteed correctness.

After the input frame copies are issued, the application calls `nvssVideoStitch()` to generate the output panoramas. This function returns before the CUDA work is complete. If synchronous behavior is desired, the application can call `cudaStreamSynchronize(cudaStreamDefault)` or `cudaDeviceSynchronize()` afterwards. We do not recommend this approach for real time applications. Instead, the application should use the functions described below to ensure correct and optimal behavior.

The API for handling the output buffers mirrors the API used for input buffers - `nvssVideoGetOutputBuffer()`, `nvssVideoGetOutputDevice()`, and `nvssVideoGetInputStream()` need to be used to access output in an optimal fashion. If a monoscopic pipeline is used, there is one output buffer. For stereoscopic stitching, there are two output buffers, one for each eye.

2.3 CLEANUP

After the client application has finished stitching, it needs to call `nvssVideoDestroyInstance()` to release the resources held by the stitcher instance. The input stitcher handle parameter is not modified, but its use after the `nvssVideoDestroyInstance()` call yields undefined behavior.

Chapter 3. VRWORKS 360 LOW-LEVEL VIDEO API REFERENCE

3.1 ENUMERATIONS

These enumerations are defined in the header file `nvss_video.h`.

3.1.1 nvssVideoStitcherFormat

This enumeration is used in `nvssVideoStitcherProperties_t` to specify the output pixel format.

Enumerator	**Description**
`NVSS_STITCHER_FORMAT_RGBA8UI = 0`	32-bit RGB format with 8-bit channels

3.2 STRUCTURES

These structures are defined in the header file `nvss_video.h`.

3.2.1 nvssVideoStitcherProperties_st

3.2.1.1 Description

Property set used to create a stitcher instance.

```
typedef struct nvssVideoStitcherProperties_st
{
    uint32_t                        version;
    nvssVideoStitcherFormat         format;
    nvstitchStitcherPipelineType    pipeline;
    nvstitchStitcherQuality         quality;
    nvstitchPanoramaProjectionType projection;
    uint32_t                        pano_width;
    float                           stereo_ipd;
    float                           feather_width;
    uint32_t                        num_gpus;
    const int*                      ptr_gpus;
    int                             min_dist;
}
nvssVideoStitcherProperties_t;
```

3.2.1.2 Members

`version`

Type: `uint32_t`

Structure version.

`format`

Type: `nvssVideoStitcherFormat`

The color format.

`pipeline`

Type: `nvstitchStitcherPipelineType`

The type of stitching (monoscopic or stereoscopic).

`quality`

Type: `nvstitchStitcherQuality`

The stitching quality preset, which affects the output quality and execution time.

`projection`

Type: `nvstitchPanoramaProjectionType`

The projection type of the 360-degree panorama output.

`pano_width`

Type: `uint32_t`

The width of the output panorama. The height of the output panorama will be half of its width for a stereoscopic pipeline and equal to the width for a monoscopic pipeline. If the panorama width is odd, its height will be rounded up.

`stereo_ipd`

Type: `float`

The target interpupillary distance in cm for the output pair of stereoscopic panoramas. The recommended value for most use cases is 6.3, and this value will be used if the input parameter is negative.

`feather_width`

Type: `float`

Width of the blending region in pixels used for monoscopic stitching.

`num_gpus`

Type: `uint32_t`

The number of GPUs to be used. This member cannot be zero. For a monoscopic pipeline, only a single GPU can be used.

`ptr_gpus`

Type: `const int*`

An array of CUDA device indices; cannot be null.

`min_dist`

Type: `int`

The minimum required distance in cm of objects from camera rig.

3.3 FUNCTIONS

3.3.1 nvssVideoCreateInstance

This function creates an instance from the input properties and returns a handle to the newly created instance. The handle is used as a parameter in subsequent API calls.

```
extern nvstitchResult NVSTITCHAPI nvssVideoCreateInstance(
          const nvssVideoStitcherProperties_t* stitcher_props,
          const nvstitchVideoRigProperties_t*  rig_props,
          nvssVideoHandle*                     handle
);
```

Parameters

`stitcher_props [in]`

Type: `nvssVideoStitcherProperties_t*`

Pointer to the set of stitcher properties.

`rig_props [in]`

Type: `nvstitchVideoRigProperties_t*`

Pointer to the set of camera rig properties.

`handle [out]`

Type: `nvssVideoHandle*`

Pointer to the handle of the newly created stitcher instance.

3.3.1.1 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS`
- `NVSTITCH_ERROR_INVALID_VERSION`
- `NVSTITCH_ERROR_BAD_PARAMETER`
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_OUT_OF_SYSTEM_MEMORY`
- `NVSTITCH_ERROR_OUT_OF_VIDEO_MEMORY`
- `NVSTITCH_VIDEO_UNSUPPORTED_RIG`

3.3.2 nvssVideoCreateInstanceWithMapping

This function creates an instance from the input properties and camera projection maps. A handle to the newly created instance is returned via the stitcher parameter. The handle is used as a parameter in subsequent API calls.

```
extern nvstitchResult NVSTITCHAPI nvssVideoCreateInstanceWithMapping(
        const nvssVideoStitcherProperties_t* stitcher_props,
        const nvstitchCameraMapping_t*       cam_mappings,
        nvssVideoHandle*                     handle
);
```

3.3.2.1 Parameters

`stitcher_props [in]`

Type: `nvssVideoStitcherProperties_t*`

Pointer to the set of stitcher properties.

`cam_mappings [in]`

Type: `nvstitchCameraMapping_t*`

Pointer to the structure containing projection maps.

`handle [out]`

Type: `nvssVideoHandle*`

Pointer to the handle of the newly created stitcher instance.

3.3.2.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS`
- `NVSTITCH_ERROR_INVALID_VERSION`
- `NVSTITCH_ERROR_BAD_PARAMETER`
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_OUT_OF_SYSTEM_MEMORY`
- `NVSTITCH_ERROR_OUT_OF_VIDEO_MEMORY`

3.3.3 nvssVideoDestroyInstance

This function destroys an instance. The input `nvssVideoHandle` parameter is not modified, but it becomes invalid after this function is called.

```
extern nvstitchResult NVSTITCHAPI
nvssVideoDestroyInstance(nvssVideoHandle handle);
```

3.3.3.1 Parameters

`Handle [in]`

Type: `nvssVideoHandle`

Stitcher instance handle to be destroyed.

3.3.3.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS`
- `NVSTITCH_ERROR_NULL_POINTER`

3.3.4 nvssVideoGetInputBuffer

This function populates the image buffer structure with the address in device memory of the input buffer for the given camera index. The input data should be copied to here with `cudaMemcpy`.

```
extern nvstitchResult NVSTITCHAPI nvssVideoGetInputBuffer(
       nvssVideoHandle        handle,
       uint32_t               camera,
       nvstitchImageBuffer_t* buffer
);
```

3.3.4.1 Parameters

`handle [in]`

Type: `nvssVideoHandle`

Stitcher instance handle.

`camera [in]`

Type: `uint32_t`

Camera index.

`buffer [out]`

Type: `nvstitchImageBuffer_t*`

Pointer to the output buffer structure.

3.3.4.2 Returns Value

Returns one of the following values:

- `NVSTITCH_SUCCESS`
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_BAD_PARAMETER`
- `NVSTITCH_ERROR_BAD_STATE`

3.3.5 nvssVideoGetInputStream

This function returns the CUDA stream that should be used when the input buffer is written to, to ensure correct synchronization. Alternatively, the default stream can be used, but using the default stream can lead to performance loss.

```
extern nvstitchResult NVSTITCHAPI nvssVideoGetInputStream(
       nvssVideoHandle      handle,
       uint32_t             camera,
       struct CUstream_st** stream
);
```

3.3.5.1 Parameters

`handle [in]`

Type: `nvssVideoHandle`

Stitcher instance handle.

`camera [in]`

Type: `uint32_t`

Camera index.

`stream [out]`

Type: `struct CUstream_st**`

Pointer to the output CUDA stream.

3.3.5.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS`
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_BAD_PARAMETER`
- `NVSTITCH_ERROR_BAD_STATE`

3.3.6 nvssVideoGetInputDevice

This function returns the device on which the input buffer of the specified camera is stored. The return value is a pointer to the device index, as associated in the CUDA device enumeration.

```
extern nvstitchResult NVSTITCHAPI nvssVideoGetInputDevice(
        nvssVideoHandle handle,
        uint32_t        camera,
        int*            device
);
```

3.3.6.1 Parameters

`handle [in]`

Type: `nvssVideoHandle`

Stitcher instance handle

`camera [in]`

Type: `uint32_t`

Camera index.

`device [out]`

Type: `int*`

Pointer to the device on which the input buffer is stored.

3.3.6.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS`
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_BAD_PARAMETER`
- `NVSTITCH_ERROR_BAD_STATE`

3.3.7 nvssVideoGetOutputBuffer

This function populates the image buffer structure with the address in device memory of the output buffer for the given eye: left or right eye for stereoscopic, or mono for monoscopic. The output data should be copied from here with `cudaMemcpy()`.

```
extern nvstitchResult NVSTITCHAPI nvssVideoGetOutputBuffer(
       nvssVideoHandle        handle,
       nvstitchEye            eye,
       nvstitchImageBuffer_t* buffer
);
```

3.3.7.1 Parameters

`handle [in]`

Type: `nvssVideoHandle`

Stitcher instance handle.

`eye [in]`

Type: `nvstitchEye`

Eye index (left, right, mono).

`buffer [out]`

Type: `nvstitchImageBuffer_t*`

Pointer to the output buffer structure.

3.3.7.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS`
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_BAD_PARAMETER`
- `NVSTITCH_ERROR_BAD_STATE`

3.3.8 nvssVideoGetOutputStream

This function returns the CUDA stream that should be used when the output buffer is read to ensure correct synchronization. Alternatively, the default stream can be used, but using the default stream can lead to performance loss.

```
extern nvstitchResult NVSTITCHAPI nvssVideoGetOutputStream(
       nvssVideoHandle      handle,
       nvstitchEye          eye,
       struct CUstream_st** stream
);
```

3.3.8.1 Parameters

`handle [in]`

Type: `nvssVideoHandle`

Stitcher instance handle.

`eye [in]`

Type: `nvstitchEye`

Eye index (left, right, mono).

`stream [out]`

Type: `struct CUstream_st**`

Pointer to the output CUDA stream.

3.3.8.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS`
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_BAD_PARAMETER`
- `NVSTITCH_ERROR_BAD_STATE`

3.3.9 nvssVideoGetOutputDevice

This function returns the device on which the output buffer is stored.

The return value is a pointer to the device index, as associated in the CUDA device enumeration.

```
extern nvstitchResult NVSTITCHAPI nvssVideoGetOutputDevice(
       nvssVideoHandle handle,
       nvstitchEye     eye,
       int*            device
);
```

3.3.9.1 Parameters

`handle [in]`

Type: `nvssVideoHandle`

Stitcher instance handle.

`eye [in]`

Type: `nvstitchEye`

Eye index (left, right, mono).

`device [out]`

Type: `int*`

Pointer to the output device.

3.3.9.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS`
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_BAD_PARAMETER`
- `NVSTITCH_ERROR_BAD_STATE`

3.3.10 nvssVideoStitch

This function stitches the current content of the input buffers and stores the result in output buffers accessible through `nvssVideoGetOutputBuffer()`. The call is asynchronous, meaning that the function returns before the output is ready. CUDA calls using the default stream or the stream returned by `nvssVideoGetOutputStream()` will synchronize with the output.

```
extern nvstitchResult NVSTITCHAPI nvssVideoStitch(
      nvssVideoHandle handle
);
```

3.3.10.1 Parameters

`handle [in]`

Type: nvssVideoHandle

Stitcher instance handle.

3.3.10.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS`
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_BAD_STATE`

3.3.11 nvssVideoGetLastErrorMessage

After a result other than `NVSTITCH_SUCCESS` is received, call this function to get additional information about the error. This string is ephemeral, and may change or become invalid after another `nvstitch` call, so a copy of the string should be made if persistence is desired.

```
nvstitchResult NVSTITCHAPI nvssVideoGetLastErrorMessage(
               const char**    errorString
);
```

3.3.11.1 Parameters

`errorString [out]`

Type: `const char**`

Pointer to the error string.

3.3.11.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS`
- `NVSTITCH_ERROR_NULL_POINTER`

3.3.12 nvssVideoGetErrorString

This function returns a string describing the `nvstitch` error code passed as a parameter.

```
const char* NVSTITCHAPI nvssVideoGetErrorString(nvstitchResult error);
```

3.3.12.1 Parameters

`error [in]`

Type: `nvstitchResult`

An `nvstitchResult` value for which the string description will be returned.

3.3.12.2 Return Value

Returns an array of null-terminated characters describing the input error code.