NVJPEG

DA-06762-001_v0.1.4 | August 2018

nvJPEG Libary Guide

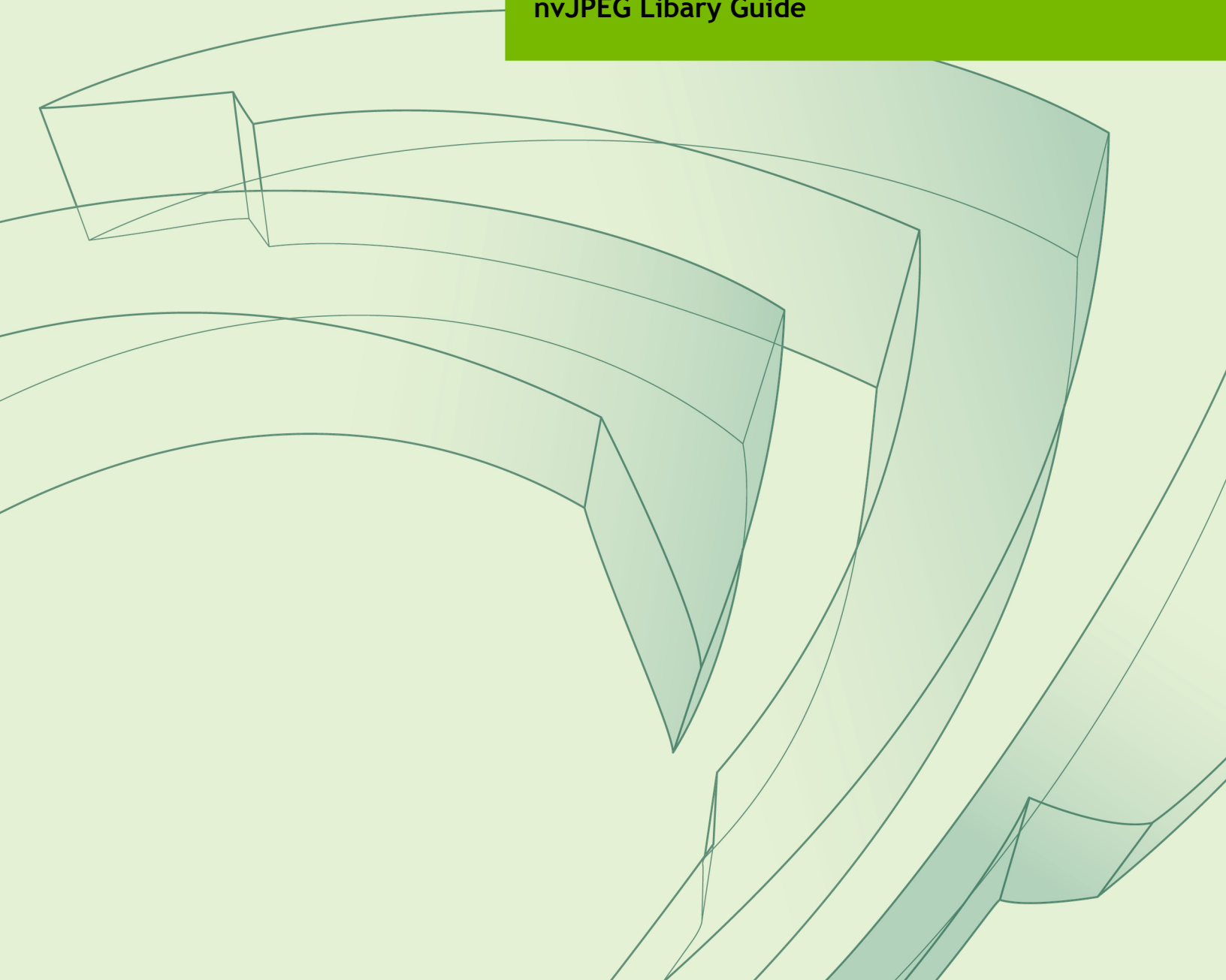

TABLE OF CONTENTS

Chapter 1. INTRODUCTION

The nvJPEG 1.0 library provides high-performance, GPU accelerated JPEG decoding functionality for image formats commonly used in deep learning and hyperscale multimedia applications. The library offers single and batched JPEG decoding capabilities which efficiently utilize the available GPU resources for optimum performance; and the flexibility for users to manage the memory allocation needed for decoding.

The nvJPEG library enables the following functions: use the JPEG image data stream as input; retrieve the width and height of the image from the data stream, and use this retrieved information to manage the GPU memory allocation and the decoding. A dedicated API is provided for retrieving the image information from the raw JPEG image data stream.

Tip Throughout this document, the terms “CPU” and “Host” are used synonymously. Similarly, the terms “GPU” and “Device” are synonymous.

The nvJPEG library supports the following:

JPEG options:

- Baseline and Progressive JPEG decoding
- 8 bits per pixel
- Huffman bitstream decoding
- 3 color channels (YCbCr) or 1 color channel (Grayscale)
- 8- and 16-bit quantization tables
- The following chroma subsampling for the 3 color channels Y, Cb, Cr (Y, U, V):
  - 4:4:4
  - 4:2:2
  - 4:2:0
  - 4:4:0
  - 4:1:1 and
  - 4:1:0

Features:

- Hybrid decoding using both the CPU (i.e., host) and the GPU (i.e., device).
- Input to the library is in the host memory, and the output is in the GPU memory.
- Single image and batched image decoding.
- Single phase and multiple phases decoding.
- Color space conversion.
- User-provided memory manager for the device allocations.

Chapter 2. USING THE NVJPEG LIBRARY

The nvJPEG library provides functions for both the decoding of a single image, and batched decoding of multiple images.

2.1. Single Image Decoding

For single-image decoding you provide the data size and a pointer to the file data, and the decoded image is placed in the output buffer.

To use the nvJPEG library, start by calling the helper functions for initialization.

1. Create nvJPEG library handle with the helper function **`nvjpegCreate()`**.
2. Create JPEG state with the helper function **`nvjpegJpegStateCreate()`**. See nvJPEG Type Declarations and **`nvjpegJpegStateCreate()`**.

   Below is the list of helper functions available in the nvJPEG library:

   - **`nvjpegStatus_t nvjpegGetProperty(libraryPropertyType type, int *value);`**
   - **`nvjpegStatus_t nvjpegCreate(nvjpegHandle_t *handle , nvjpeg_dev_allocator allocator);`**
   - **`nvjpegStatus_t nvjpegDestroy(nvjpegHandle_t handle);`**
   - **`nvjpegStatus_t nvjpegJpegStateCreate(nvjpegHandle_t handle, nvjpegJpegState_t *jpeg_handle);`**
   - **`nvjpegStatus_t nvjpegJpegStateDestroy(nvjpegJpegState handle);`**
3. Retrieve the width and height information from the JPEG-encoded image by using the **`nvjpegGetImageInfo()`** function. See also **`nvjpegGetImageInfo()`**.

   Below is the signature of **`nvjpegGetImageInfo()`** function:

```
nvjpegStatus_t nvjpegGetImageInfo(
  nvjpegHandle_t              handle,
  const unsigned char         *data,
  size_t                      length,
  int                         *nComponents,
  nvjpegChromaSubsampling_t   *subsampling,
  int                         *widths,
  int                         *heights);
```

For each image to be decoded, pass the JPEG data pointer and data length to the above function. The **nvjpegGetImageInfo()** function is thread safe.

4. One of the outputs of the above **nvjpegGetImageInfo()** function is **nvjpegChromaSubsampling_t**. This parameter is an enum type, and its enumerator list is composed of the chroma subsampling property retrieved from the JPEG image. See nvJPEG Chroma Subsampling.
5. Use the **nvjpegDecode()** function in the nvJPEG library to decode this single JPEG image. See the signature of this function below:

```
nvjpegStatus_t nvjpegDecode(
  nvjpegHandle_t          handle,
  nvjpegJpegState_t       jpeg_handle,
  const unsigned char     *data,
  size_t                  length,
  nvjpegOutputFormat_t    output_format,
  nvjpegImage_t           *destination,
  cudaStream_t            stream);
```

In the above **nvjpegDecode()** function, the parameters **nvjpegOutputFormat_t**, **nvjpegImage_t**, and **cudaStream_t** can be used to set the output behavior of the **nvjpegDecode()** function. You provide the **cudaStream_t** parameter to indicate the stream to which your asynchronous tasks are submitted.

6. **The nvjpegOutputFormat_t parameter:**

The **nvjpegOutputFormat_t** parameter can be set to one of the **output_format** settings below:

output_format	Meaning
NVJPEG_OUTPUT_UNCHANGED	Return the decoded image planar format.
NVJPEG_OUTPUT_RGB	Convert to planar RGB.
NVJPEG_OUTPUT_BGR	Convert to planar BGR.
NVJPEG_OUTPUT_RGBI	Convert to interleaved RGB.
NVJPEG_OUTPUT_BGRI	Convert to interleaved BGR.
NVJPEG_OUTPUT_Y	Return the Y component only.
NVJPEG_OUTPUT_YUV	Return in the YUV planar format.

For example, if the output_format is set to **NVJPEG_OUTPUT_Y** or **NVJPEG_OUTPUT_RGBI**, or **NVJPEG_OUTPUT_BGRI** then the output is written only to channel[0], and the other channels are not touched.

Alternately, in the case of planar output, the data is written to the corresponding channels of the **nvjpegImage_t** destination structure.

Finally, in the case of grayscale JPEG and RGB output, the luminance is used to create the grayscale RGB.

7. As mentioned above, an important benefit of the **nvjpegGetImageInfo()** function is the ability to utilize the image information retrieved from the the input JPEG image to allocate proper GPU memory for your decoding operation.

The **nvjpegGetImageInfo()** function returns the **widths**, **heights** and **nComponents** parameters.

```
nvjpegStatus_t nvjpegGetImageInfo(
  nvjpegHandle_t              handle,
  const unsigned char         *data,
  size_t                      length,
  int                         *nComponents,
  nvjpegChromaSubsampling_t   *subsampling,
  int                         *widths,
  int                         *heights);
```

You can use the retrieved parameters, **widths**, **heights** and **nComponents**, to calculate the required size for the output buffers, either for a single decoded JPEG, or for every decoded JPEG in a batch.

To optimally set the **destination** parameter for the **nvjpegDecode()** function, use the following guidelines:

For the output_format: NVJPEG_OUTPUT_Y	**destination.pitch[0] should be at least:** width[0]	**destination.channel[0] should be at least of size:** destination.pitch[0]*height[0]
For the output_format	**destination.pitch[c] should be at least:**	**destination.channel[c] should be at least of size:**
NVJPEG_OUTPUT_YUV	width[c] for c = 0, 1, 2	destination.pitch[c]*height[c] for c = 0, 1, 2
NVJPEG_OUTPUT_RGB and NVJPEG_OUTPUT_BGR	width[0] for c = 0, 1, 2	destination.pitch[0]*height[0] for c = 0, 1, 2
NVJPEG_OUTPUT_RGBI and NVJPEG_OUTPUT_BGRI	width[0]*3	destination.pitch[0]*height[0]
NVJPEG_OUTPUT_UNCHANGED	width[c] for c = [0, nComponents - 1]	destination.pitch[c]*height[c] for c = [0, nComponents - 1]

8. Ensure that the **nvjpegImage_t** structure (or structures, in the case of batched decode) is filled with the pointers and pitches of allocated buffers. The **nvjpegImage_t** structure that holds the output pointers is defined as follows:

```
typedef struct
{
    unsigned char * channel[NVJPEG_MAX_COMPONENT];
    unsigned int pitch[NVJPEG_MAX_COMPONENT];
} nvjpegImage_t;
```

NVJPEG_MAX_COMPONENT is the maximum number of color components the nvJPEG library supports in the current release. For generic images, this is the maximum number of encoded channels that the library is able to decompress.

9. Finally, when you call the **nvjpegDecode()** function with the parameters as described above, the **nvjpegDecode()** function fills the output buffers with the decoded data.

2.2. Decode by Phases

Alternately, you can decode a single image in multiple phases. This gives you flexibility in controlling the flow, and optimizing the decoding process.

To decode an image in multiple phases, follow these steps:

1. Just as when you are decoding in a single phase, create the JPEG state with the helper function **`nvjpegJpegStateCreate()`**.
2. Next, call the functions in the sequence below (see Decode API -- Multiple Phases.)
   - **`nvjpegDecodePhaseOne()`**
   - **`nvjpegDecodePhaseTwo()`**
   - **`nvjpegDecodePhaseThree()`**
3. At the conclusion of the third phase, the **`nvjpegDecodePhaseThree()`** function writes the decoded output at the memory location pointed to by its **`*destination`** parameter.

2.3. Batched Image Decoding

For the batched image decoding you provide pointers to multiple file data in the memory, and also provide the buffer sizes for each file data. The nvJPEG library will decode these multiple images, and will place the decoded data in the output buffers that you specified in the parameters.

2.4. Single Phase

For batched image decoding in single phase, follow these steps:

1. Call **`nvjpegDecodeBatchedInitialize()`** function to initialize the batched decoder. Specify the batch size in the **`batch_size`** parameter. See **`nvjpegDecodeBatchedInitialize()`**.
2. Next, call **`nvjpegDecodeBatched()`** for each new batch. Make sure to pass the parameters that are correct to the specific batch of images. If the size of the batch changes, or if the batch decoding fails, then call the **`nvjpegDecodeBatchedInitialize()`** function again.

2.5. Multiple Phases

To decode a batch of images in multiple phases, follow these steps:

This is the only case where the JPEG state could be used by multiple threads at the same time.

1. Create the JPEG state with the helper function **`nvjpegJpegStateCreate()`**.
2. Call the **`nvjpegDecodeBatchedInitialize()`** function to initialize the batched decoder. Specify the batch size in the **`batch_size`** parameter, and specify the **`max_cpu_threads`** parameter to set the maximum number of CPU threads that work on single batch.
3. Batched processing is done by calling the functions for the specific phases in sequence:

- In the first phase, call **`nvjpegDecodePhaseOne()`** for each image in the batch, according to the index of the image in the batch. Note that this could be done using multiple threads. If multiple threads are used then the thread index in the range [0, max_cpu_threads-1] should be provided to the **`nvjpegDecodeBatchedPhaseOne()`** function. Before proceeding to the next phase, ensure that the **`nvjpegDecodePhaseOne()`** calls for every image have finished.
- Next, call **`nvjpegDecodePhaseTwo()`**.
- Finally, call **`nvjpegDecodePhaseThree()`**.

4. If you have another batch of images of the same size to process, then repeat from 3.

Chapter 3. NVJPEG TYPE DECLARATIONS

3.1. nvJPEG Memory Allocator Interface

```
typedef int (*tDevMalloc)(void**, size_t);
typedef int (*tDevFree)(void*);
typedef struct
  {
    tDevMalloc dev_malloc;
    tDevFree dev_free;
  } nvjpegDevAllocator_t;
```

Users can tell the library to use their own device memory allocator. The function prototypes for the memory allocation and memory freeing functions are similar to the **cudaMalloc()** and **cudaFree()** functions. They should return 0 in case of success, and non-zero otherwise. A pointer to the **nvjpegDevAllocator_t** structure, with properly filled fields, should be provided to the **nvjpegCreate()** function. NULL is accepted, in which case the default memory allocation functions **cudaMalloc()** and **cudaFree()** is used.

When the **nvjpegDevAllocator_t *allocator** parameter in the **nvjpegCreate()** function is set as a pointer to the above **nvjpegDevAllocator_t** structure, then this structure is used for allocating and releasing memory. The function prototypes for the memory allocation and memory freeing functions are similar to the **cudaMalloc()** and **cudaFree()** functions. They should return 0 in case of success, and non-zero otherwise.

However, if the **nvjpegDevAllocator_t *allocator** parameter in the **nvjpegCreate()** function is set to NULL, then the default memory allocation functions **cudaMalloc()** and **cudaFree()** will be used.

3.2. nvJPEG Opaque Library Handle Struct

```
struct nvjpegHandle;
typedef struct nvjpegHandle* nvjpegHandle_t;
```

The library handle is used in any consecutive nvJPEG library calls, and should be initialized first.

The library handle is thread safe, and can be used by multiple threads simultaneously.

3.3. nvJPEG Opaque JPEG Decoding State Handle

```
struct nvjpegJpegState;
typedef struct nvjpegJpegState* nvjpegJpegState_t;
```

The **nvjpegJpegState** structure stores the temporary JPEG information. It should be initialized before any usage. This JPEG state handle can be reused after being used in another decoding. The same JPEG handle should be used across the decoding phases for the same image or batch. Multiple threads are allowed to share the JPEG state handle only when processing same batch during first phase (**nvjpegDecodePhaseOne**) .

3.4. nvJPEG Output Pointer Struct

```
typedef struct
  {
    unsigned char * channel[NVJPEG_MAX_COMPONENT];
    unsigned int pitch[NVJPEG_MAX_COMPONENT];
  } nvjpegImage_t;
```

The **nvjpegImage_t** struct holds the pointers to the output buffers, and holds the corresponding strides of those buffers for the image decoding.

See Single Image Decoding on how to set up the **nvjpegImage_t** struct.

Chapter 4.
NVJPEG API REFERENCE

This section describes the nvJPEG API.

4.1. nvJPEG Helper API Reference

The nvJPEG helper functions are used for initializing.

4.1.1. nvjpegGetProperty()

Gets the numeric value for the major or minor version, or the patch level, of the nvJPEG library.

Signature:

```
nvjpegStatus_t nvjpegGetProperty(
 libraryPropertyType   type,
 int                   *value);
```

Parameters:

Parameter	Input / Output	Memory	Description
`libraryPropertyType type`	Input	Host	One of the supported **`libraryPropertyType`** values, that is, MAJOR_VERSION, MINOR_VERSION or PATCH_LEVEL.
`int *value`	Output	Host	The numeric value corresponding to the specific **`libraryPropertyType`** requested.

Returns:

`nvjpegStatus_t` - An error code as specified in nvJPEG API Return Codes.

4.1.2. nvjpegCreate()

Allocates and initializes the library handle.

Signature:

```
nvjpegStatus_t nvjpegCreate(
 nvjpegBackend_t   backend,
 nvjpegDevAllocator_t   *allocator,
 nvjpegHandle_t   *handle);
```

Parameters:

Parameter	Input / Output	Memory	Description
nvjpegBackend_t backend	Input	Host	A backend parameter for the library. This backend will be used for all the functions called with this handle. If this is set to DEFAULT then it automatically chooses one of the underlying algorithms.
nvjpegDevAllocator_t *allocator	Input	Host	Device memory allocator. See **nvjpegDevAllocator_t** structure description. If NULL is provided, then the default CUDA runtime **cudaMalloc()** and **cudaFree()** functions will be used.
nvjpegHandle_t *handle	Input/Output	Host	The library handle.

The **nvjpegBackend_t** parameter is an **enum** type, with the below enumerated list values:

```
typedef enum {
 NVJPEG_BACKEND_DEFAULT = 0,
 NVJPEG_BACKEND_HYBRID = 1,
} nvjpegBackend_t;
```

Returns:

nvjpegStatus_t - An error code as specified in nvJPEG API Return Codes.

4.1.3. nvjpegDestroy()

Releases the library handle.

Signature:

```
nvjpegStatus_t nvjpegDestroy(nvjpegHandle_t handle);
```

Parameters:

Parameter	Input / Output	Memory	Description
nvjpegHandle_t handle	Input/Output	Host	The library handle to release.

Returns:

nvjpegStatus_t - An error code as specified in nvJPEG API Return Codes.

4.1.4. nvjpegJpegStateCreate()

Allocates and initializes the internal structure required for the JPEG processing.

Signature:

```
nvjpegStatus_t nvjpegJpegStateCreate(
 nvjpegHandle_t      handle,
 nvjpegJpegState_t   *jpeg_handle);
```

Parameters:

Parameter	Input / Output	Memory	Description
nvjpegHandle_t handle	Input	Host	The library handle.
nvjpegJpegState_t *jpeg_handle	Input/Output	Host	The image state handle.

Returns:

nvjpegStatus_t - An error code as specified in nvJPEG API Return Codes.

4.1.5. nvjpegJpegStateDestroy()

Releases the image internal structure.

Signature:

```
nvjpegStatus_t nvjpegJpegStateDestroy(nvjpegJpegState handle);
```

Parameters:

Parameter	Input / Output	Memory	Description
nvjpegJpegState handle	Input/Output	Host	The image state handle.

Returns:

nvjpegStatus_t - An error code as specified in nvJPEG API Return Codes.

4.2. Retrieve Encoded Image Information API

The helper functions for retrieving the encoded image information.

4.2.1. nvjpegGetImageInfo()

Decodes the JPEG header and retrieves the basic information about the image.

Signature:

```
nvjpegStatus_t nvjpegGetImageInfo(
 nvjpegHandle_t              handle,
 const unsigned char         *data,
 size_t                      length,
 int                         *nComponents,
 nvjpegChromaSubsampling_t   *subsampling,
 int                         *widths,
 int                         *heights);
```

Parameters:

Parameter	Input / Output	Memory	Description
`nvjpegHandle_t handle`	Input	Host	The library handle.
`const unsigned char *data`	Input	Host	Pointer to the encoded data.
`size_t length`	Input	Host	Size of the encoded data in bytes.
`int *nComponents`	Output	Host	Chroma subsampling for the 1- or 3- channel encoding.
`int *widths`	Output	Host	Pointer to the first element of array of size NVJPEG_MAX_COMPONENT, where the width of each channel (up to NVJPEG_MAX_COMPONENT) will be saved. If the channel is not encoded, then the corresponding value would be zero.
`int *heights`	Output	Host	Pointer to the first element of array of size NVJPEG_MAX_COMPONENT, where the height of each channel (up to NVJPEG_MAX_COMPONENT) will be saved. If the channel is not encoded, then the corresponding value would be zero.

Returns:

`nvjpegStatus_t` - An error code as specified in nvJPEG API Return Codes.

4.3. Decode API -- Single Phase

Functions for decoding single image or batched images in a single phase.

4.3.1. nvjpegDecode()

Decodes a single image, and writes the decoded image in the desired format to the output buffers. This function is asynchronous with respect to the host. All GPU tasks for this function will be submitted to the provided stream.

Signature:

```
nvjpegStatus_t nvjpegDecode(
 nvjpegHandle_t           handle,
 nvjpegJpegState_t        jpeg_handle,
 const unsigned char      *data,
 size_t                   length,
 nvjpegOutputFormat_t     output_format,
 nvjpegImage_t            *destination,
 cudaStream_t             stream);
```

Parameters:

Parameter	Input / Output	Memory	Description
nvjpegHandle_t handle	Input	Host	The library handle.
nvjpegJpegState_t jpeg_handle	Input	Host	The image state handle.
const unsigned char *data	Input	Host	Pointer to the encoded data.
size_t length	Input	Host	Size of the encoded data in bytes.
nvjpegOutputFormat_t output_format	Input	Host	Format in which the decoded output will be saved.
nvjpegImage_t *destination	Input/Output	Host/ Device	Pointer to the structure that describes the output destination. This structure should be on the host (CPU), but the pointers in this structure should be pointing to the device (i.e., GPU) memory. See **nvjpegImage_t.**
cudaStream_t stream	Input	Host	The CUDA stream where all of the GPU work will be submitted.

Returns:

nvjpegStatus_t - An error code as specified in nvJPEG API Return Codes.

4.3.2. nvjpegDecodeBatchedInitialize()

This function initializes the batched decoder state. The initialization parameters include the batch size, the maximum number of CPU threads, and the specific output format in which the decoded image will be saved. This function should be called once, prior to decoding the batches of images. Any currently running batched decoding should be finished before calling this function.

Signature:

```
nvjpegStatus_t nvjpegDecodeBatchedInitialize(
  nvjpegHandle_t           handle,
  nvjpegJpegState_t        jpeg_handle,
  int                      batch_size,
  int                      max_cpu_threads,
  nvjpegOutputFormat_t     output_format);
```

Parameters:

Parameter	Input / Output	Memory	Description

nvjpegHandle_t handle	Input	Host	The library handle.
nvjpegJpegState_t jpeg_handle	Input	Host	The image state handle.
int batch_size	Input	Host	Batch size.
int max_cpu_threads	Input	Host	Maximum number of CPU threads that can participate in decoding a batch.
nvjpegOutputFormat_t output_format	Input	Host	Format in which the decoded output will be saved.

Returns:

nvjpegStatus_t - An error code as specified in nvJPEG API Return Codes.

4.3.3. nvjpegDecodeBatched()

Decodes the batch of images, and writes them to the buffers described in the **destination** parameter in a format provided to **nvjpegDecodeBatchedInitialize()** function. This function is asynchronous with respect to the host. All GPU tasks for this function will be submitted to the provided stream.

Signature:

```
nvjpegStatus_t nvjpegDecodeBatched(
  nvjpegHandle_t              handle,
  nvjpegJpegState_t           jpeg_handle,
  const unsigned char *const *data,
  const size_t               *lengths,
  nvjpegImage_t              *destinations,
  cudaStream_t                stream);
```

Parameters:

Parameter	Input / Output	Memory	Description
nvjpegHandle_t handle	Input	Host	The library handle.
nvjpegJpegState_t jpeg_handle	Input	Host	The image state handle.
const unsigned char *const *data	Input	Host	Pointer to the first element of array of the input data. The size of the array is assumed to be batch_size provided to **nvjpegDecodeBatchedInitialize()** batch initialization function.
const size_t *lengths	Input	Host	Pointer to the first element of array of input sizes. Size of array is assumed to be batch_size provided to **nvjpegDecodeBatchedInitialize()**, the batch initialization function.
nvjpegImage_t *destinations	Input/ Output	Host/ Device	Pointer to the first element of array of output descriptors. The size of array is assumed to be batch_size provided to **nvjpegDecodeBatchedInitialize()**,

			the batch initialization function. See also **nvjpegImage_t**.
cudaStream_t stream	Input	Host	The CUDA stream where all the GPU work will be submitted.

Returns:

nvjpegStatus_t - An error code as specified in nvJPEG API Return Codes.

4.4. Decode API -- Multiple Phases

The nvJPEG library provides an ability to control the decoding process in phases. In the simple case of a single-image decode you can split the decoding into phases. For decoding multiple images, you can overlap the decoding phases of separate images within a single thread. Finally, for the batched decode you can use multiple threads to split the host tasks. Synchronization between phases should be handled with CUDA events and CUDA stream synchronization mechanisms, by the user.

Note that first phases are synchronous with the respect to the host, while the second and third phases are asynchronous--for both single image and batched decode.

4.4.1. nvjpegDecodePhaseOne()

The first phase of a single-image decode. You provide all the inputs, and the nvJPEG library performs any required preprocessing on the host. Any previous calls to **nvjpegDecodePhaseOne()** and **nvjpegDecodePhaseTwo()** with the same **nvjpeg_handle** parameter should be finished prior to this call.

Signature:

```
nvjpegStatus_t nvjpegDecodePhaseOne(
 nvjpegHandle_t          handle,
 nvjpegJpegState_t       jpeg_handle,
 const unsigned char     *data,
 size_t                  length,
 nvjpegOutputFormat_t    output_format,
 cudaStream_t            stream);
```

Parameters:

Parameter	Input / Output	Memory	Description
nvjpegHandle_t handle	Input	Host	The library handle.
nvjpegJpegState_t jpeg_handle	Input	Host	The image state handle.
const unsigned char *data	Input	Host	Pointer to the encoded stream.
size_t length	Input	Host	Size of the encoded stream.
nvjpegOutputFormat_t output_format	Input	Host	Format in which the decoded image will be saved.
cudaStream_t stream	Input	Host	The CUDA stream where all the GPU work will be submitted.

Returns:

nvjpegStatus_t - An error code as specified in nvJPEG API Return Codes.

4.4.2. nvjpegDecodePhaseTwo()

In this second phase of the decoding process, the GPU (that is, the device) is involved. The decoding task is transferred to the device memory. Any required preprocessing is performed on the device. Any previous calls to **nvjpegDecodePhaseTwo()** and **nvjpegDecodePhaseThree()** with the same **jpeg_handle** parameter should be finished prior to this call.

Signature:

```
nvjpegStatus_t nvjpegDecodePhaseTwo(
  nvjpegHandle_t       handle,
  nvjpegJpegState_t    jpeg_handle,
  cudaStream_t         stream);
```

Parameters:

Parameter	Input / Output	Memory	Description
nvjpegHandle_t handle	Input	Host	The library handle.
nvjpegJpegState_t jpeg_handle	Input	Host	The image state handle.
cudaStream_t stream	Input	Host	The CUDA stream where all the GPU work will be submitted.

Returns:

nvjpegStatus_t - An error code as specified in nvJPEG API Return Codes.

4.4.3. nvjpegDecodePhaseThree()

In this third phase of the decoding process, the decoded image is written to the output, in the specified decoding format.

If the same **jpeg_handle** is shared for decoding multiple images simultaneously, then these multiple images should be of the same **output_format**.

Signature:

```
nvjpegStatus_t nvjpegDecodePhaseThree(
  nvjpegHandle_t       handle,
  nvjpegJpegState_t    jpeg_handle,
  nvjpegImage_t        *destination,
  cudaStream_t         stream);
```

Parameters:

Parameter	Input / Output	Memory	Description
nvjpegHandle_t handle	Input	Host	The library handle.
nvjpegJpegState_t jpeg_handle	Input	Host	The image state handle.

nvjpegImage_t *destination	Input/Output	Host/ Device	Pointer to the structure that describes the output destination. This structure should be on host, but the pointers in this structure should be pointing to the device memory. See **nvjpegImage_t** description for details.
cudaStream_t stream	Input	Host	The CUDA stream where all the GPU work will be submitted.

Returns:

nvjpegStatus_t - An error code as specified in nvJPEG API Return Codes.

4.4.4. nvjpegDecodeBatchedPhaseOne()

This first phase of the batched decoding should be called separately for each image in the batch. The batch initialization API, with appropriate batch parameters, should be called prior to starting the task with the batch.

If the batch parameters (batch size, number of threads, output format) did not change, then there is no need to initialize the batch again before starting the task.

It is possible to use multiple threads to split this first phase of the task. In which case, each thread should have a unique index. Provide the index of the image in the batch, and use the same JPEG decoding state parameter.

The thread index for the batch should be in the range of **[0, max_cpu_threads-1]**. The image index should be in the range of **[0, batch_size-1]**. Any previous calls to **nvjpegDecodeBatchedPhaseOne()** and **nvjpegDecodeBatchedPhaseTwo()** on a different batch with the same JPEG state handle parameter should be completed prior to this call.

Signature:

```
nvjpegStatus_t nvjpegDecodeBatchedPhaseOne(
  nvjpegHandle_t          handle,
  nvjpegJpegState_t       jpeg_handle,
  const unsigned char     *data,
  size_t                  length,
  int                     image_idx,
  int                     thread_idx,
  cudaStream_t            stream);
```

Parameters:

Parameter	Input / Output	Memory	Description
nvjpegHandle_t handle	Input	Host	The library handle.
nvjpegJpegState_t jpeg_handle	Input	Host	The image state handle.
const unsigned char *data	Input	Host	Pointer to the encoded stream.
size_t length	Input	Host	Size of the encoded stream.

int image_idx	Input	Host	Image index in the batch. Should be in the range from 0 to **batch_size-1**.
int thread_idx	Input	Host	Thread index that calls this phase. Should be in the range from 0 to **max_cpu_threads-1**.
cudaStream_t stream	Input	Host	The CUDA stream where all the GPU work will be submitted.

Returns:

nvjpegStatus_t - An error code as specified in nvJPEG API Return Codes.

4.4.5. nvjpegDecodeBatchedPhaseTwo()

This phase should be called once per batch. It should be called only after the **nvjpegDecodeBatchedPhaseOne()** calls for every image in the batch have finished. Any prior calls to **nvjpegDecodeBatchedPhaseTwo()** and **nvjpegDecodeBatchedPhaseThree()** for other batches with the same JPEG state handle parameter should be finished prior this call.

Signature:

```
nvjpegStatus_t nvjpegDecodeBatchedPhaseTwo(
 nvjpegHandle_t          handle,
 nvjpegJpegState_t       jpeg_handle,
 cudaStream_t            stream);
```

Parameters:

Parameter	Input / Output	Memory	Description
nvjpegHandle_t handle	Input	Host	The library handle.
nvjpegJpegState_t jpeg_handle	Input	Host	The image state handle.
cudaStream_t stream	Input	Host	The CUDA stream where all the GPU work will be submitted.

Returns:

nvjpegStatus_t - An error code as specified in nvJPEG API Return Codes.

4.4.6. nvjpegDecodeBatchedPhaseThree()

This phase should be called once per batch. It should be called only after the **nvjpegDecodeBatchedPhaseTwo()** call for the same batch has finished.

Between a call to **nvjpegDecodeBatchedPhaseTwo()** and a call to this function, no calls are allowed to **nvjpegDecodeBatchedPhaseTwo()** or **nvjpegDecodeBatchedPhaseThree()** for any other batch with the same JPEG state handle parameter.

Signature:

```
nvjpegStatus_t nvjpegDecodeBatchedPhaseThree(
 nvjpegHandle_t            handle,
 nvjpegJpegState_t         jpeg_handle,
 nvjpegImage_t             *destinations,
 cudaStream_t              stream);
```

Parameters:

Parameter	Input / Output	Memory	Description
nvjpegHandle_t handle	Input	Host	The library handle.
nvjpegJpegState_t jpeg_handle	Input	Host	The image state handle.
nvjpegImage_t *destinations	Input/Output	Host/ Device	Pointer to the first element of the array of output descriptors. The size of the array is assumed to be the **batch_size** parameter that was provided to the batch initialization function. See **nvjpegImage_t** description for details.
cudaStream_t stream	Input	Host	The CUDA stream to which all the GPU tasks will be submitted.

Returns:

nvjpegStatus_t - An error code as specified in nvJPEG API Return Codes.

4.5. nvjpeg-api-return-codes

The nvJPEG API adheres to the following return codes and their indicators:

```
typedef enum
{
 NVJPEG_STATUS_SUCCESS = 0,
 NVJPEG_STATUS_NOT_INITIALIZED = 1,
 NVJPEG_STATUS_INVALID_PARAMETER = 2,
 NVJPEG_STATUS_BAD_JPEG = 3,
 NVJPEG_STATUS_JPEG_NOT_SUPPORTED = 4,
 NVJPEG_STATUS_ALLOCATOR_FAILURE = 5,
 NVJPEG_STATUS_EXECUTION_FAILED = 6,
 NVJPEG_STATUS_ARCH_MISMATCH = 7,
 NVJPEG_STATUS_INTERNAL_ERROR = 8,
} nvjpegStatus_t;
```

Description of the returned error codes:

Returned Error (Returned Code)	Description
NVJPEG_STATUS_SUCCESS (0)	The API call has finished successfully. Note that many of the calls are asynchronous and some of the errors may be seen only after synchronization.
NVJPEG_STATUS_NOT_INITIALIZED (1)	The library handle was not initialized. A call to **nvjpegCreate()** is required to initialize the handle.

NVJPEG_STATUS_INVALID_PARAMETER (2)	Wrong parameter was passed. For example, a null pointer as input data, or an image index not in the allowed range.
NVJPEG_STATUS_BAD_JPEG (3)	Cannot parse the JPEG stream. Check that the encoded JPEG stream and its size parameters are correct.
NVJPEG_STATUS_JPEG_NOT_SUPPORTED (4)	Attempting to decode a JPEG stream that is not supported by the nvJPEG library.
NVJPEG_STATUS_ALLOCATOR_FAILURE (5)	The user-provided allocator functions, for either memory allocation or for releasing the memory, returned a non-zero code.
NVJPEG_STATUS_EXECUTION_FAILED (6)	Error during the execution of the device tasks.
NVJPEG_STATUS_ARCH_MISMATCH (7)	The device capabilities are not enough for the set of input parameters provided (input parameters such as backend, encoded stream parameters, output format).
NVJPEG_STATUS_INTERNAL_ERROR (8)	Error during the execution of the device tasks.

4.6. nvjpeg-chroma-subsampling

One of the outputs of the **nvjpegGetImageInfo()** API is **nvjpegChromaSubsampling_t**. This parameter is an **enum** type, and its enumerator list comprises of the chroma subsampling property retrieved from the encoded JPEG image. Below are the chroma subsampling types the **nvjpegGetImageInfo()** function currently supports:

```
typedef enum
{
 NVJPEG_CSS_444,
 NVJPEG_CSS_422,
 NVJPEG_CSS_420,
 NVJPEG_CSS_440,
 NVJPEG_CSS_411,
 NVJPEG_CSS_410,
 NVJPEG_CSS_GRAY,
 NVJPEG_CSS_UNKNOWN
} nvjpegChromaSubsampling_t;
```

4.7. Reference Documents

Refer to the JPEG standard: https://jpeg.org/jpeg/

Chapter 5. EXAMPLES

This package contains the library header and a set of libraries—static and shared. Shared libraries (the **libnvjpeg.so** and the respective versioned libraries) have all of the CUDA toolkit dependencies statically linked. However, if you want to link against the static library (**libnvjpeg_static.a**) you also need to link the other dependencies—for example dl, rt and thread libraries.

Example of linking shared library:

```
g++ -Icuda-linux64-nvjpeg/include -lnvjpeg -Lcuda-linux64-nvjpeg/
lib64 my_example.cpp -o my_example
```

Example of linking static library:

```
g++ -Icuda-linux64-nvjpeg/include -lnvjpeg_static -ldl -lrt -pthread
-Lcuda-linux64-nvjpeg/lib64 my_example.cpp -o my_example
```

Example

Below example shows how to use the various nvJPEG APIs.

Compile with the following command from the **examples** folder, assuming CUDA 9.0 is installed at the path **/usr/local/cuda-9.0**:

```
g++ -O3 -m64 nvjpeg_example.cpp -I../include -lnvjpeg -L../lib64 -I/
usr/local/cuda-9.0/include -ldl -lrt -pthread -lcudart -L/usr/local/
cuda-9.0/lib64 -Wl,-rpath=../lib64 -Wl,-rpath=/usr/local/cuda-9.0/
lib64 -o nvjpeg_example
```

The below examples show how to decode the JPEG files using either single or batched API, and write the decoded files as BMP images.

To decode a single image:

```
./nvjpeg_example -i /tmp/my_image.jpg -fmt rgb -o /tmp
```

To decode multiple images in the folder using the batched API in separate phases:

```
./nvjpeg_example -i /tmp/my_images/ -fmt rgb -b 32 -pipelined -
batched -o /tmp
```

Run the command **./nvjpeg_example -h** for the description of the parameters.

Notice

Trademarks

Copyright