NVJPEG

DA-06762-001_v10.1 | April 2019

nvJPEG Libary Guide

TABLE OF CONTENTS

Chapter 1. Introduction 1
1.1. nvJPEG Decoder 1
1.2. nvJPEG Encoder 2
Chapter 2. JPEG Decoding 3
2.1. Using JPEG Decoding 3
2.1.1. Single Image Decoding 3
2.1.3. Batched Image Decoding 6
2.1.4. Single Phase 6
2.1.5. Multiple Phases 7
2.2. nvJPEG Type Declarations 7
2.2.1. nvJPEG Device Memory Allocator Interface 7
2.2.2. nvJPEG Host Pinned Memory Allocator Interface 8
2.2.3. nvJPEG Opaque Library Handle Struct 8
2.2.4. nvJPEG Opaque JPEG Decoding State Handle 8
2.2.5. nvJPEG Output Pointer Struct 9
2.2.6. nvJPEG Backend 9
2.3. nvJPEG API Reference 9
2.3.1. nvJPEG Helper API Reference 9
2.3.1.1. nvjpegGetProperty() 9
2.3.1.2. nvjpegCreate() 10
2.3.1.3. nvjpegCreateSimple() 11
2.3.1.4. nvjpegCreateEx() 11
2.3.1.5. nvjpegDestroy() 12
2.3.1.6. nvjpegSetDeviceMemoryPadding() 12
2.3.1.7. nvjpegGetDeviceMemoryPadding() 13
2.3.1.8. nvjpegSetPinnedMemoryPadding() 13
2.3.1.9. nvjpegGetPinnedMemoryPadding() 13
2.3.1.10. nvjpegJpegStateCreate() 14
2.3.1.11. nvjpegJpegStateDestroy() 14
2.3.2. Retrieve Encoded Image Information API 14
2.3.2.1. nvjpegGetImageInfo() 15
2.3.3. Decode API -- Single Phase 15
2.3.3.1. nvjpegDecode() 15
2.3.3.2. nvjpegDecodeBatchedInitialize() 16
2.3.3.3. nvjpegDecodeBatched() 17
2.3.4. Decode API -- Multiple Phases 18
2.3.4.1. nvjpegDecodePhaseOne() 18
2.3.4.2. nvjpegDecodePhaseTwo() 19
2.3.4.3. nvjpegDecodePhaseThree() 19
2.3.4.4. nvjpegDecodeBatchedPhaseOne() 20

2.3.5. nvjpeg-api-return-codes....22
2.3.6. nvjpeg-chroma-subsampling....23
2.3.7. Reference Documents....23
2.4. Examples of nvJPEG....23
Chapter 3. JPEG Encoding....25
3.1. Using the Encoder....25
3.1.1. Encoding the Parameters....25
3.1.2. Encoding the State....25
3.1.3. Encoding the Image....26
3.1.3.1. nvjpegEncodeYUV....26
3.1.3.2. nvjpegEncodeImage....26
3.1.4. Retrieving the Compressed Stream....27
3.1.5. JPEG Encoding Example....27
3.2. nvJPEG Encoder Type Declarations....28
3.2.1. nvjpegInputFormat_t....29
3.2.2. nvjpegEncoderState_t....29
3.2.3. nvjpegEncoderParams_t....29
3.3. nvJPEG Encoder Helper API Reference....29
3.3.1. nvjpegEncoderStateCreate()....29
3.3.2. nvjpegEncoderStateDestroy()....30
3.3.3. nvjpegEncoderParamsCreate()....30
3.3.4. nvjpegEncoderParamsDestroy()....30
3.3.5. nvjpegEncoderParamsSetQuality()....31
3.3.6. nvjpegEncoderParamsSetOptimizedHuffman()....31
3.3.7. nvjpegEncoderParamsSetSamplingFactors()....31
3.4. nvJPEG Encoder API Reference....32
3.4.1. nvjpegEncodeGetBufferSize()....32
3.4.2. nvjpegEncodeYUV()....33
3.4.3. nvjpegEncodeImage()....33
3.4.4. nvjpegEncodeRetrieveBitstream()....34

Chapter 1. INTRODUCTION

1.1. nvJPEG Decoder

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.

1.2. nvJPEG Encoder

The encoding functions of the nvJPEG library perform GPU-accelerated compression of user's image data to the JPEG bitstream. User can provide input data in a number of formats and colorspaces, and control the encoding process with parameters. Encoding functionality will allocate temporary buffers using user-provided memory allocator.

Before calling the encoding functions the user should perform a few prerequisite steps using the helper functions described in nvJPEG Encoder Helper API Reference.

Chapter 2. JPEG DECODING

2.1. Using JPEG Decoding

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

2.1.1. 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 one of the helper functions **`nvjpegCreateSimple() or nvjpegCreateEx()`**.
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);`**
   - **`[DEPRECATED] nvjpegStatus_t nvjpegCreate(nvjpegBackend_t backend, nvjpegHandle_t *handle , nvjpeg_dev_allocator allocator);`**
   - **`nvjpegStatus_t nvjpegCreateSimple(nvjpegHandle_t *handle);`**
   - **`nvjpegStatus_t nvjpegCreateEx(nvjpegBackend_t backend, nvjpegDevAllocator_t *dev_allocator, nvjpegPinnedAllocator_t *pinned_allocator, unsigned int flags, nvjpegHandle_t *handle);`**
   - **`nvjpegStatus_t nvjpegDestroy(nvjpegHandle_t handle);`**
   - **`nvjpegStatus_t nvjpegJpegStateCreate(nvjpegHandle_t handle, nvjpegJpegState_t *jpeg_handle);`**
   - **`nvjpegStatus_t nvjpegJpegStateDestroy(nvjpegJpegState handle);`**

- Other helper functions such as **nvjpegSet*()** and **nvjpegGet*()** can be used to configure the library functionality on per-handle basis. Refer to the helper API reference for more details.

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.

2.1.2. 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.

2.1.3. 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.

2.1.4. 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.

2.1.5. 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.

2.2. nvJPEG Type Declarations

2.2.1. nvJPEG Device 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()`** or **`nvjpegCreateEx()`** function is set as a pointer to the above **`nvjpegDevAllocator_t`**

structure, then this structure is used for allocating and releasing the device 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()** or **nvjpegCreateEx()** function is set to NULL, then the default memory allocation functions **cudaMalloc()** and **cudaFree()** will be used. When using **nvjpegCreateSimple()** function to create library handle the default device memory allocator will be used.

2.2.2. nvJPEG Host Pinned Memory Allocator Interface

```
typedef int (*tPinnedMalloc)(void**, size_t, unsigned int flags);
typedef int (*tPinnedFree)(void*);
typedef struct
{
tPinnedMalloc pinned_malloc;
tPinnedFree pinned_free;
} nvjpegPinnedAllocator_t;
```

When the **nvjpegPinnedAllocator_t *allocator** parameter in the **nvjpegCreateEx()** function is set as a pointer to the above **nvjpegPinnedAllocator_t** structure, then this structure will be used for allocating and releasing host pinned memory for copying data to/from device. The function prototypes for the memory allocation and memory freeing functions are similar to the **cudaHostAlloc()** and **cudaFreeHost()** functions. They will return 0 in case of success, and non-zero otherwise.

However, if the **nvjpegPinnedAllocator_t *allocator** parameter in the **nvjpegCreateEx()** function is set to NULL, then the default memory allocation functions **cudaHostAlloc()** and **cudaFreeHost()** will be used. When using **nvjpegCreate()** or **nvjpegCreateSimple()** function to create library handle, the default host pinned memory allocator will be used.

2.2.3. 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.

2.2.4. 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**) .

2.2.5. 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.

2.2.6. nvJPEG Backend

```
typedef enum {
 NVJPEG_BACKEND_DEFAULT = 0,
 NVJPEG_BACKEND_HYBRID = 1,
 NVJPEG_BACKEND_GPU_HYBRID = 2
} nvjpegBackend_t;
```

The **nvjpegBackend_t** enum is used to select either default back-end by default, or use GPU decoding for baseline JPEG images, or use CPU for Huffman decoding.

Member	Description
NVJPEG_BACKEND_DEFAULT	Default back-end is selected internally
NVJPEG_BACKEND_HYBRID	Uses CPU for Huffman decoding
NVJPEG_BACKEND_GPU_HYBRID	The function **nvjpegDecodeBatched** will use GPU decoding for the baseline JPEG images with interleaved scan when batch size is greater than 100. For other JPEG types it uses CPU. Other nvJPEG decode APIs will continue to use CPU for Huffman decode.

2.3. nvJPEG API Reference

This section describes the nvJPEG API.

2.3.1. nvJPEG Helper API Reference

The nvJPEG helper functions are used for initializing.

2.3.1.1. 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.

2.3.1.2. nvjpegCreate()

Allocates and initializes the library handle.

This function is deprecated. Use either **nvjpegCreateSimple()** or **nvjpegCreateEx()** functions to create 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.

2.3.1.3. nvjpegCreateSimple()

Allocates and initializes the library handle, with default codec implementations selected by library and default memory allocators.

Signature:

```
nvjpegStatus_t nvjpegCreateSimple(nvjpegHandle_t *handle);
```

Parameters:

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

Returns:

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

2.3.1.4. nvjpegCreateEx()

Allocates and initializes the library handle using the provided arguments.

Signature:

```
nvjpegStatus_t nvjpegCreateEx(nvjpegBackend_t backend,
          nvjpegDevAllocator_t *dev_allocator,
          nvjpegPinnedAllocator_t *pinned_allocator,
          unsigned int flags,
          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 of the functions called with this handle. If this is set to “DEFAULT" then it will choose one of underlying algorithms automatically.
nvjpegDevAllocator_t *dev_allocator	Input	Host	Device memory allocator. See **nvjpegDevAllocator_t** structure description. If NULL is provided, then the default CUDA

			runtime functions **cudaMalloc()** and **cudaFree()** will be used.
nvjpegPinnedAllocator_t *pinned_allocator	Input	Host	Pinned host memory allocator. See **nvjpegPinnedAllocator_t** structure description. If NULL is provided, then the default CUDA runtime functions **cudaHostAlloc()** and **cudaFreeHost()** will be used.
nvjpegHandle_t *handle	Input/Output	Host	The library handle.

Returns:

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

2.3.1.5. 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.

2.3.1.6. nvjpegSetDeviceMemoryPadding()

Use the provided padding for all device memory allocations with specified library handle. A large number will help to amortize the need for device memory reallocations when needed.

Signature:

```
nvjpegStatus_t nvjpegSetDeviceMemoryPadding(
 size_t padding,
 nvjpegHandle_t handle);
```

Parameters:

Parameter	Input / Output	Memory	Description
size_t padding	Input	Host	Device memory padding to use for all further device memory allocations.
nvjpegHandle_t *handle	Input/Output	Host	The library handle.

Returns:

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

2.3.1.7. nvjpegGetDeviceMemoryPadding()

Retrieve the device memory padding that is currently used for the specified library handle.

Signature:

```
nvjpegStatus_t nvjpegGetDeviceMemoryPadding(
 size_t *padding,
 nvjpegHandle_t handle);
```

Parameters:

Parameter	Input / Output	Memory	Description
size_t *padding	Output	Host	Device memory padding that is currently used for device memory allocations.
nvjpegHandle_t *handle	Input/Output	Host	The library handle.

Returns:

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

2.3.1.8. nvjpegSetPinnedMemoryPadding()

Use the provided padding for all pinned host memory allocations with specified library handle. A large number will help to amortize the need for pinned host memory reallocations when needed.

Signature:

```
nvjpegStatus_t nvjpegSetPinnedMemoryPadding(
 size_t padding,
 nvjpegHandle_t handle);
```

Parameters:

Parameter	Input / Output	Memory	Description
size_t padding	Input	Host	Pinned host memory padding to use for all further pinned host memory allocations.
nvjpegHandle_t handle	Input/Output	Host	The library handle.

Returns:

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

2.3.1.9. nvjpegGetPinnedMemoryPadding()

Retrieve the pinned host memory padding that is currently used for specified library handle.

Signature:

```
nvjpegStatus_t nvjpegGetPinnedMemoryPadding(
 size_t *padding,
 nvjpegHandle_t handle);
```

Parameters:

Parameter	Input / Output	Memory	Description
size_t *padding	Output	Host	Pinned host memory padding that is currently used for pinned host memory allocations.
nvjpegHandle_t handle	Input/Output	Host	The library handle.

Returns:

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

2.3.1.10. 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.

2.3.1.11. 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.

2.3.2. Retrieve Encoded Image Information API

The helper functions for retrieving the encoded image information.

2.3.2.1. 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.

2.3.3. Decode API -- Single Phase

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

2.3.3.1. 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.

2.3.3.2. 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.

2.3.3.3. 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.

2.3.4. 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.

2.3.4.1. 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.

2.3.4.2. 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.

2.3.4.3. 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.

2.3.4.4. 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.

2.3.4.5. 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.

2.3.4.6. 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.

2.3.5. 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.

2.3.6. 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;
```

2.3.7. Reference Documents

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

2.4. Examples of nvJPEG

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.

Chapter 3. JPEG ENCODING

This section describes the encoding functions of the nvJPEG Library.

3.1. Using the Encoder

The user should perform the below prerequisite steps before calling the nvJPEG encoding functions. See also nvJPEG Encoder Helper API Reference.

3.1.1. Encoding the Parameters

The user should create an encoding parameters structure with **`nvjpegEncoderParamsCreate()`** function. The function will be initialized with default parameters. User can use an appropriate **`nvjpegEncoderParamsSet*()`** function to set a specific parameter.

The quality parameter can be set, using **`nvjpegEncoderParamsSetQuality()`** function, to an integer value between 1 and 100, and this quality parameter will be used as a base for generating the JPEG quantization tables.

The parameters structure should be passed to compression functions.

The encoding parameters structure can be reused to compress multiple images simultaneously, but no changes to the parameters should be made during the ongoing encoding, or encoding result will be undefined.

3.1.2. Encoding the State

The user should create the encoding state structure using **`nvjpegEncoderStateCreate()`** function. This function will hold intermediate buffers for the encoding process. This state should be passed to the compression functions.

The encoding state structure can be reused to encode a series of images, but no encoding should be performed on multiple images with the same encoding state at the same time - otherwise result of the encodings will be undefined.

3.1.3. Encoding the Image

The nvJPEG library provides a few interfaces for compressing the image in different formats and colorspaces. See below.

3.1.3.1. nvjpegEncodeYUV

Input for this function is an image in YUV colorspace. See nvjpegEncodeYUV(). The **source** argument should be filled with the corresponding YUV planar data. The **chroma_subsampling** argument should have the chroma subsampling of the input data. If the chroma subsampling in the encoding parameters is the same as input chroma subsampling, then the user's input data will be directly used in the JPEG compression. Otherwise chroma will be resampled to match the chroma subsampling of the encoding parameters.

Input data should be provided with respect to the subsampling factors. That is, the chrominance image planes should have sizes aligned to the corresponding subsamplings. For example:

- Image dimensions: 123x321
- Input chroma subsampling is: NVJPEG_CSS_410
- Chroma subsampling factor for this chroma subsampling: is 4x2
- Given the above, the encoder library expects the user to provide:
  - Y plane with size: 123 x 321
  - Cb and Cr plane with size: 31 x 161

3.1.3.2. nvjpegEncodeImage

See nvjpegEncodeImage(). Input for this function, i.e., how data should be provided in the **source** argument, is determined by the **input_format** argument. For the interleaved formats (ending with **I**) only the first channel is used. For the non-interleaved formats, all the channels in the input format are used.

For example, if the user has interleaved the RGB image of size **W x H**, stored continuously, and the pointer to it is **pImage**, then **source** should be:

- `source.channel[0] = pImage`
- `source.pitch[0] = W*3`

When the same the image is stored in planar format, with image planes pointers stored continuously in the array **pImage[3]**, then **source** should be:

- `source.channel[0] = pImage[0]`
- `source.channel[1] = pImage[1]`
- `source.channel[2] = pImage[2]`

The **pitch** values for each channel in the **source** parameter should be set accordingly to the data layout.

The nvJPEG library will perform the color transformation to the YCbCr, and will compress the result.

3.1.4. Retrieving the Compressed Stream

Often it is not feasible to accurately predict the final compressed data size of the final JPEG stream for any input data and parameters. The nvJPEG library, while encoding, will calculate the size of the final stream, allocate temporary buffer in the encoder state and save the compressed data in the encoding state's buffer. In order to get final compressed JPEG stream, the user should provide the memory buffer large enough to store this compressed data. There are two options for how to do this:

1. Use the upper bound on compressed JPEG stream size for the given parameters and image dimensions:
    a. Use the **`nvjpegEncodeRetrieveBitstream()`** function to retrieve the maximum possible JPEG stream size at any given time.
    b. Allocate the memory buffer at any given time.
    c. Encode the image using one of the encoding functions.
    d. Retrieve the compressed JPEG stream from the encoder state after successful encoding, using the **`nvjpegEncodeRetrieveBitstream()`** and the allocated buffer.
2. Wait for the encoding to complete, and retrieve the exact size of required buffer, as below:
    a. Encode the image using one of the encoding functions.
    b. Use the **`nvjpegEncodeRetrieveBitstream()`** function to retrieve the size in bytes of the compressed JPEG stream.
    c. Allocate the memory buffer of at least this size.
    d. Use the **`nvjpegEncodeRetrieveBitstream()`** function to populate your buffer with the compressed JPEG stream.

As the same encoding image state can be reused to compress a series of images, the **`nvjpegEncodeRetrieveBitstream()`** function will return the result for the last compressed image.

3.1.5. JPEG Encoding Example

See below the example code, and the block diagram shown in Figure 1, for encoding with nvJPEG Encoder.

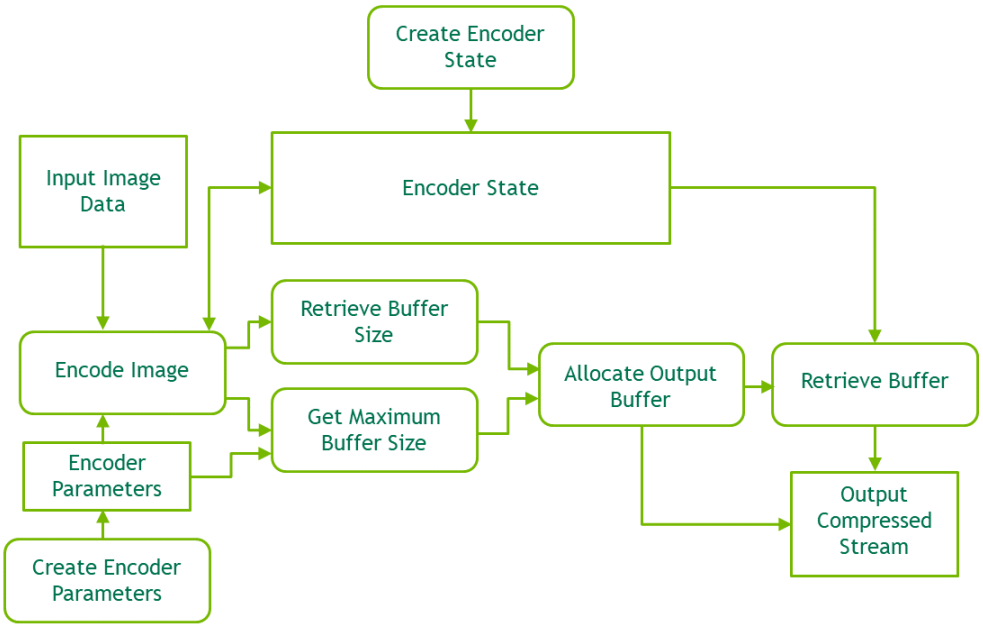

Figure 1 JPEG Encoding Using nvJPEG Encoder

```
nvjpegHandle_t nv_handle;
nvjpegEncoderState_t nv_enc_state;
nvjpegEncoderParams_t nv_enc_params;
cudaStream_t stream;

// initialize nvjpeg structures
nvjpegCreateSimple(&nv_handle);
nvjpegEncoderStateCreate(nv_handle, &nv_enc_state, stream);
nvjpegEncoderParamsCreate(nv_handle, &nv_enc_params, stream);

nvjpegImage_t nv_image;
// Fill nv_image with image data, let’s say 640x480 image in RGB format

// Compress image
nvjpegEncodeImage(nv_handle, nv_enc_state, nv_enc_params,
    &nv_image, NVJPEG_INPUT_RGB, 640, 480, stream);

// get compressed stream size
size_t length;
nvjpegEncodeRetrieveBitstream(nv_handle, nv_enc_state, NULL, &length, stream);
// get stream itself
cudaStreamSynchronize(stream);
std::vector<char> jpeg(length);
nvjpegEncodeRetrieveBitstream(nv_handle, nv_enc_state, jpeg.data(), &length, 0);

// write stream to file
cudaStreamSynchronize(stream);
std::ofstream output_file(“test.jpg”, std::ios::out | std::ios::binary);
output_file.write(jpeg.data(), length);
output_file.close();
```

3.2. nvJPEG Encoder Type Declarations

This section describes the nvJPEG Encoder Type Declarations.

3.2.1. nvjpegInputFormat_t

```
typedef enum
{
    NVJPEG_INPUT_RGB       = 3,
    NVJPEG_INPUT_BGR       = 4,
    NVJPEG_INPUT_RGBI      = 5,
    NVJPEG_INPUT_BGRI      = 6
} nvjpegInputFormat_t;
```

The **nvjpegInputFormat_t** enum is used to select the color model and pixel format of the input image. It is used for conversion to YCbCr during encoding.

Member	Description
NVJPEG_INPUT_RGB	Input image is in RGB color model. Pixel format is RGB.
NVJPEG_INPUT_BGR	Input image is in RGB color model. Pixel format is BGR.
NVJPEG_INPUT_RGBI	Input image is in RGB color model. Pixel format is interleaved RGB.
NVJPEG_INPUT_BGRI	Input image is in RGB color model. Pixel format is interleaved BGR.

3.2.2. nvjpegEncoderState_t

The **nvjpegEncoderState_t** is a structure that stores intermdiate buffers and variables used for compression.

3.2.3. nvjpegEncoderParams_t

The **nvjpegEncoderParams_t** is a structure that stores JPEG encode parameters.

3.3. nvJPEG Encoder Helper API Reference

The nvJPEG Encoder helper functions are used for initializing.

3.3.1. nvjpegEncoderStateCreate()

Creates encoder state that stores intermediate buffers used in compression.

Signature:

```
nvjpegStatus_t nvjpegEncoderStateCreate(
 nvjpegHandle_t handle,
 nvjpegEncoderState_t *encoder_state,
 cudaStream_t stream);
```

Parameters:

Parameter	Input / Output	Memory	Description
handle	Input	Host	Library handle
encoder_state	Output	Host	Pointer to the encoder state structure, where the new state will be placed.

stream	Inputt	Host	CUDA stream where all the required device operations will be placed.

3.3.2. nvjpegEncoderStateDestroy()

Destroys the encoder state.

Signature:

```
nvjpegStatus_t nvjpegEncoderStateDestroy(
 nvjpegEncoderState_t encoder_state);
```

Parameters:

Parameter	Input / Output	Memory	Description
encoder_state	Input/Output	Host	Encoder state structure that will be released.

3.3.3. nvjpegEncoderParamsCreate()

Creates the structure that holds the compression parameters.

Signature:

```
nvjpegStatus_t nvjpegEncoderParamsCreate(
 nvjpegHandle_t handle,
 nvjpegEncoderParams_t *encoder_params,
 cudaStream_t stream);
```

Parameters:

Parameter	Input / Output	Memory	Description
handle	Input	Host	Library handle
encoder_params	Output	Host	Pointer to the location where the new parameters structure will be placed.
stream	Inputt	Host	CUDA stream where all the required device operations will be placed.

3.3.4. nvjpegEncoderParamsDestroy()

Destroys the encoder parameters structure.

Signature:

```
nvjpegEncoderParamsDestroy(
 nvjpegEncoderParams_t encoder_params);
```

Parameters:

Parameter	Input / Output	Memory	Description
encoder_params	Input/Output	Host	Encoder params structure that will be released.

3.3.5. nvjpegEncoderParamsSetQuality()

Sets the parameter quality in the encoder parameters structure.

Signature:

```
nvjpegStatus_t nvjpegEncoderParamsSetQuality(
 nvjpegEncoderParams_t encoder_params,
 const int quality,
 cudaStream_t stream);
```

Parameters:

Parameter	Input / Output	Memory	Description
encoder_params	Input/Output	Host	Encoder params structure handle.
quality	Input	Host	Integer value of quality between 1 and 100, where 100 is the highest quality. Default value is 70.
stream	Inputt	Host	CUDA stream where all the required device operations will be placed.

3.3.6. nvjpegEncoderParamsSetOptimizedHuffman()

Sets whether or not to use optimized Huffman. Using optimized Huffman produces smaller JPEG bitstream sizes with the same quality, but with slower performance.

Signature:

```
nvjpegStatus_t nvjpegEncoderParamsSetOptimizedHuffman(
 nvjpegEncoderParams_t encoder_params,
 const int optimized,
 cudaStream_t stream);
```

Parameters:

Parameter	Input / Output	Memory	Description
encoder_params	Input/Output	Host	Encoder params structure handle.
optimized	Input	Host	If this value is 0 then non-optimized Huffman will be used. Otherwise optimized version will be used. Default value is 0.
stream	Inputt	Host	CUDA stream where all the required device operations will be placed.

3.3.7. nvjpegEncoderParamsSetSamplingFactors()

Sets which chroma subsampling will be used for JPEG compression.

Signature:

```
nvjpegStatus_t nvjpegEncoderParamsSetSamplingFactors(
 nvjpegEncoderParams_t encoder_params,
 const nvjpegChromaSubsampling_t chroma_subsampling,
 cudaStream_t stream);
```

Parameters:

Parameter	Input / Output	Memory	Description
encoder_params	Input/Output	Host	Encoder params structure handle.
chroma_subsampling	Input	Host	Chroma subsampling that will be used for JPEG compression. If the input is in YUV color model and **chroma_subsampling** is different from the subsampling factors of source image, then the NVJPEG library will convert subsampling to the value of **chroma_subsampling**. Default value is 4:4:4.
stream	Inputt	Host	CUDA stream where all the required device operations will be placed.

3.4. nvJPEG Encoder API Reference

This section describes the nvJPEG Encoder API.

3.4.1. nvjpegEncodeGetBufferSize()

Returns the maximum possible buffer size that is needed to store the compressed JPEG stream, for the given input parameters.

Signature:

```
nvjpegStatus_t nvjpegEncodeGetBufferSize(
 nvjpegHandle_t handle,
 const nvjpegEncoderParams_t encoder_params,
 int image_width,
 int image_height,
 size_t *max_stream_length);
```

Parameters:

Parameter	Input / Output	Memory	Description
handle	Input	Host	Library handle
encoder_params	Input/Output	Host	Encoder parameters structure handle.
image_width	Input	Host	Input image width.
image_height	Input	Host	Input image height.

stream	Inputt	Host	CUDA stream where all the required device operations will be placed.

3.4.2. nvjpegEncodeYUV()

Compresses the image in YUV colorspace to JPEG stream using the provided parameters, and stores it in the state structure.

Signature:

```
nvjpegStatus_t nvjpegEncodeYUV(
 nvjpegHandle_t handle,
 nvjpegEncoderState_t encoder_state,
 const nvjpegEncoderParams_t encoder_params,
 const nvjpegImage_t *source,
 nvjpegChromaSubsampling_t chroma_subsampling,
 int image_width,
 int image_height,
 cudaStream_t stream);
```

Parameters:

Parameter	Input / Output	Memory	Description
handle	Input	Host	Library handle
encoder_state	Input/Output	Host	Internal structure that holds the temporary buffers required for the compression and also stores the final compressed JPEG stream.
encoder_params	Input	Host	Encoder parameters structure handle.
source	Input	Host	Pointer to the **nvjpeg** structure that holds the device pointers to the **Y, U(Cb) and V(Cr)** image planes and the respective strides.
chroma_subsampling	Input	Host	Chroma subsampling of the input data.
image_width	Input	Host	Input image width.
image_height	Input	Host	Input image height.
stream	Inputt	Host	CUDA stream where all the required device operations will be placed.

3.4.3. nvjpegEncodeImage()

Compresses the image in the provided format to the JPEG stream using the provided parameters, and stores it in the state structure.

Signature:

```
nvjpegStatus_t nvjpegEncodeImage(
 nvjpegHandle_t handle,
 nvjpegEncoderState_t encoder_state,
 const nvjpegEncoderParams_t encoder_params,
 const nvjpegImage_t *source,
 nvjpegInputFormat_t input_format,
 int image_width,
 int image_height,
 cudaStream_t stream);
```

Parameters:

Parameter	Input / Output	Memory	Description
handle	Input	Host	Library handle
encoder_state	Input/Output	Host	Internal structure that holds the temporary buffers required for the compression and also stores the final compressed JPEG stream.
encoder_params	Input	Host	Encoder parameters structure handle.
source	Input	Host	Pointer to the **nvjpeg** structure that holds the device pointers to the **Y, U(Cb) and V(Cr)** image planes and the respective strides.
input_format	Input	Host	Value of **nvjpegInputFormat_t** type that describes the input data.
image_width	Input	Host	Input image width.
image_height	Input	Host	Input image height.
stream	Input	Host	CUDA stream where all the required device operations will be placed.

3.4.4. nvjpegEncodeRetrieveBitstream()

Retrieves the compressed stream from the encoder state that was previously used in one of the encoder functions.

- If **data** parameter is NULL then the encoder will return compressed stream size in the **length** parameter.
- If **data** is not NULL then the provided **length** parameter should contain the **data** buffer size.
- If the provided **length** is less than compressed stream size, then an error will be returned. Otherwise the compressed stream will be stored in the **data** buffer and the actual compressed buffer size will be stored in the **length** parameter.

Signature:

```
nvjpegStatus_t nvjpegEncodeRetrieveBitstream(
 nvjpegHandle_t handle,
 nvjpegEncoderState_t encoder_state,
 unsigned char *data,
 size_t *length,
 cudaStream_t stream);
```

Parameters:

Parameter	Input / Output	Memory	Description
handle	Input	Host	Library handle
encoder_state	Input/Output	Host	The **encoder_state** that was previously used in one of the encoder functions.
data	Input/Output	Host	Pointer to the buffer in the host memory where the compressed stream will be stored. Can be NULL (see description).
length	Input/Output	Host	Pointer to the input buffer size. On return the NVJPEG library will store the actual compressed stream size in this parameter.
stream	Inputt	Host	CUDA stream where all the required device operations will be placed.

Notice

Trademarks

Copyright