# NVIDIA Performance Primitives (NPP)

Version 9.0

August 18, 2017

# **Contents**

1	NVI	IDIA Performance Primitives	1
	1.1	What is NPP?	2
	1.2	Documentation	2
	1.3	Technical Specifications	2
	1.4	Files	3
		1.4.1 Header Files	3
		1.4.2 Library Files	3
	1.5	Supported NVIDIA Hardware	4
2	Gen	neral API Conventions	5
	2.1	Memory Management	6
		2.1.1 Scratch Buffer and Host Pointer	6
	2.2	Function Naming	7
	2.3	Integer Result Scaling	7
	2.4	Rounding Modes	8
		2.4.1 Rounding Mode Parameter	8
3	Sign	nal-Processing Specific API Conventions	9
	3.1	Signal Data	10
		3.1.1 Parameter Names for Signal Data	10
		3.1.1.1 Source Signal Pointer	10
		3.1.1.2 Destination Signal Pointer	10
		3.1.1.3 In-Place Signal Pointer	10
		3.1.2 Signal Data Alignment Requirements	11
		3.1.3 Signal Data Related Error Codes	11
	3.2	Signal Length	11
		3.2.1 Length Related Error Codes	11
4	Ima	ging-Processing Specific API Conventions	13

ii CONTENTS

	4.1	Functi	on Naming	4
	4.2	Image	Data	4
		4.2.1	Line Step	5
		4.2.2	Parameter Names for Image Data	5
			4.2.2.1 Passing Source-Image Data	5
			4.2.2.2 Passing Destination-Image Data	6
			4.2.2.3 Passing In-Place Image Data	8
			4.2.2.4 Passing Mask-Image Data	8
			4.2.2.5 Passing Channel-of-Interest Data	8
		4.2.3	Image Data Alignment Requirements	8
		4.2.4	Image Data Related Error Codes	9
	4.3	Region	n-of-Interest (ROI)	9
		4.3.1	ROI Related Error Codes	9
	4.4	Maske	d Operation	0
	4.5	Chann	el-of-Interest API	0
		4.5.1	Select-Channel Source-Image Pointer	0
		4.5.2	Select-Channel Source-Image	0
		4.5.3	Select-Channel Destination-Image Pointer	0
	4.6	Source	-Image Sampling	1
		4.6.1	Point-Wise Operations	1
		4.6.2	Neighborhood Operations	1
			4.6.2.1 Mask-Size Parameter	1
			4.6.2.2 Anchor-Point Parameter	2
			4.6.2.3 Sampling Beyond Image Boundaries	2
5	Mod	lula Ind	ex 23	2
3	5.1	lule Ind	es	
	3.1	Modul	2.	J
6	Data	a Struct	ure Index 25	5
	6.1	Data S	tructures	5
7	Mod	lula Da	cumentation 2'	7
,				
	7.1			
		7.1.1	<b>P</b>	
		7.1.2	Function Documentation	
			7.1.2.2 nppGetGpuDeviceProperties	
			7.1.2.3 nppGetGpuName	ð

CONTENTS

,	7.1.2.4	nppGetGpuNumSMs	28
,	7.1.2.5	nppGetLibVersion	29
,	7.1.2.6	nppGetMaxThreadsPerBlock	29
,	7.1.2.7	nppGetMaxThreadsPerSM	29
,	7.1.2.8	nppGetStream	29
,	7.1.2.9	nppGetStreamMaxThreadsPerSM	29
,	7.1.2.10	nppGetStreamNumSMs	29
,	7.1.2.11	nppSetStream	30
7.2 NPP Typ	pe Definit	ions and Constants	31
7.2.1	Define Do	ocumentation	37
,	7.2.1.1	NPP_HOG_MAX_BINS_PER_CELL	37
,	7.2.1.2	NPP_HOG_MAX_BLOCK_SIZE	37
,	7.2.1.3	NPP_HOG_MAX_CELL_SIZE	37
,	7.2.1.4	NPP_HOG_MAX_CELLS_PER_DESCRIPTOR	38
,	7.2.1.5	NPP_HOG_MAX_DESCRIPTOR_LOCATIONS_PER_CALL	38
,	7.2.1.6	NPP_HOG_MAX_OVERLAPPING_BLOCKS_PER_DESCRIPTOR .	38
,	7.2.1.7	NPP_MAX_16S	38
,	7.2.1.8	NPP_MAX_16U	38
,	7.2.1.9	NPP_MAX_32S	38
,	7.2.1.10	NPP_MAX_32U	38
,	7.2.1.11	NPP_MAX_64S	38
,	7.2.1.12	NPP_MAX_64U	38
,	7.2.1.13	NPP_MAX_8S	38
,	7.2.1.14	NPP_MAX_8U	38
,	7.2.1.15	NPP_MAXABS_32F	39
,	7.2.1.16	NPP_MAXABS_64F	39
,	7.2.1.17	NPP_MIN_16S	39
,	7.2.1.18	NPP_MIN_16U	39
,	7.2.1.19	NPP_MIN_32S	39
,	7.2.1.20	NPP_MIN_32U	39
,	7.2.1.21	NPP_MIN_64S	39
,	7.2.1.22	NPP_MIN_64U	39
,	7.2.1.23	NPP_MIN_8S	39
,	7.2.1.24	NPP_MIN_8U	39
,	7.2.1.25	NPP_MINABS_32F	39
,	7.2.1.26	NPP_MINABS_64F	40

iv CONTENTS

	7.2.2	Enumera	tion Type Documentation	40
		7.2.2.1	NppCmpOp	40
		7.2.2.2	NppGpuComputeCapability	40
		7.2.2.3	NppHintAlgorithm	41
		7.2.2.4	NppiAlphaOp	41
		7.2.2.5	NppiAxis	41
		7.2.2.6	NppiBayerGridPosition	41
		7.2.2.7	NppiBorderType	42
		7.2.2.8	NppiDifferentialKernel	42
		7.2.2.9	NppiHuffmanTableType	42
		7.2.2.10	NppiInterpolationMode	42
		7.2.2.11	NppiMaskSize	43
		7.2.2.12	NppiNorm	43
		7.2.2.13	NppRoundMode	43
		7.2.2.14	NppStatus	44
		7.2.2.15	NppsZCType	46
7.3	Basic I	NPP Data '	Types	47
	7.3.1	Typedef 1	Documentation	48
		7.3.1.1	Npp16s	48
		7.3.1.2	Npp16u	48
		7.3.1.3	Npp32f	48
		7.3.1.4	Npp32fc	48
		7.3.1.5	Npp32s	48
		7.3.1.6	Npp32sc	49
		7.3.1.7	Npp32u	49
		7.3.1.8	Npp32uc	49
		7.3.1.9	Npp64f	49
		7.3.1.10	Npp64fc	49
		7.3.1.11	Npp64s	49
		7.3.1.12	Npp64sc	49
		7.3.1.13	Npp64u	49
		7.3.1.14	Npp8s	49
		7.3.1.15	Npp8u	49
	7.3.2	Function	Documentation	49
		7.3.2.1	align	49
		7.3.2.2	align	50

CONTENTS

	7.3.3	Variable	Documentation	50
		7.3.3.1	Npp16sc	50
		7.3.3.2	Npp16uc	50
		7.3.3.3	Npp8uc	50
7.4	Compr	ression .		51
	7.4.1	Detailed	Description	52
	7.4.2	Typedef 1	Documentation	53
		7.4.2.1	NppiDecodeHuffmanSpec	53
		7.4.2.2	NppiEncodeHuffmanSpec	53
	7.4.3	Function	Documentation	53
		7.4.3.1	nppiDecodeHuffmanScanHost_JPEG_8u16s_P1R	53
		7.4.3.2	nppiDecodeHuffmanScanHost_JPEG_8u16s_P3R	53
		7.4.3.3	nppiDecodeHuffmanSpecFreeHost_JPEG	54
		7.4.3.4	nppiDecodeHuffmanSpecGetBufSize_JPEG	54
		7.4.3.5	nppiDecodeHuffmanSpecInitAllocHost_JPEG	55
		7.4.3.6	nppiDecodeHuffmanSpecInitHost_JPEG	55
		7.4.3.7	nppiEncodeHuffmanGetSize	55
		7.4.3.8	nppiEncodeHuffmanScan_JPEG_8u16s_P1R	56
		7.4.3.9	nppiEncodeHuffmanScan_JPEG_8u16s_P3R	56
		7.4.3.10	nppiEncodeHuffmanSpecFree_JPEG	57
		7.4.3.11	nppiEncodeHuffmanSpecGetBufSize_JPEG	57
		7.4.3.12	nppiEncodeHuffmanSpecInit_JPEG	58
		7.4.3.13	nppiEncodeHuffmanSpecInitAlloc_JPEG	58
		7.4.3.14	nppiEncodeOptimizeHuffmanGetSize	58
		7.4.3.15	nppiEncodeOptimizeHuffmanScan_JPEG_8u16s_P1R	59
		7.4.3.16	nppiEncodeOptimizeHuffmanScan_JPEG_8u16s_P3R	59
7.5	Quanti	zation Fun	nctions	61
	7.5.1	Typedef 1	Documentation	62
		7.5.1.1	NppiDCTState	62
	7.5.2	Function	Documentation	62
		7.5.2.1	nppiDCTFree	62
		7.5.2.2	nppiDCTInitAlloc	62
		7.5.2.3	nppiDCTQuant16Fwd8x8LS_JPEG_8u16s_C1R_NEW	63
		7.5.2.4	nppiDCTQuant16Inv8x8LS_JPEG_16s8u_C1R_NEW	63
		7.5.2.5	nppiDCTQuantFwd8x8LS_JPEG_8u16s_C1R	64
		7.5.2.6	nppiDCTQuantFwd8x8LS_JPEG_8u16s_C1R_NEW	64

vi CONTENTS

			7.5.2.7	nppiDCTQuantInv8x8LS_JPEG_16s8u_C1R	65
			7.5.2.8	nppiDCTQuantInv8x8LS_JPEG_16s8u_C1R_NEW	65
			7.5.2.9	nppiQuantFwdRawTableInit_JPEG_8u	66
			7.5.2.10	nppiQuantFwdTableInit_JPEG_8u16u	66
			7.5.2.11	nppiQuantInvTableInit_JPEG_8u16u	66
0	D-4-	C4	D		<b>60</b>
8			ure Docur		69
	8.1			Struct Reference	69
		8.1.1		Description	69
		8.1.2		cumentation	69
			8.1.2.1	im	69
			8.1.2.2	im	70
			8.1.2.3	re	70
			8.1.2.4	re	70
	8.2	NPP_A		Struct Reference	71
		8.2.1	Detailed	Description	71
		8.2.2	Field Do	cumentation	71
			8.2.2.1	im	71
			8.2.2.2	im	71
			8.2.2.3	im	71
			8.2.2.4	re	72
			8.2.2.5	re	72
			8.2.2.6	re	72
	8.3	NppiH	laarBuffer	Struct Reference	73
		8.3.1	Field Do	cumentation	73
			8.3.1.1	haarBuffer	73
			8.3.1.2	haarBufferSize	73
	8.4	NppiH	[aarClassifi	ier_32f Struct Reference	74
		8.4.1	Field Do	cumentation	74
			8.4.1.1	classifiers	74
			8.4.1.2	classifierSize	74
			8.4.1.3	classifierStep	74
			8.4.1.4	counterDevice	74
			8.4.1.5	numClassifiers	74
	8.5	NppiH	OGConfig	Struct Reference	75
		8.5.1	_	Description	75
		8.5.2		cumentation	75

CONTENTS vii

		8.5.2.1	cellSize	 	75
		8.5.2.2	detectionWindowSize	 	75
		8.5.2.3	histogramBlockSize	 	75
		8.5.2.4	nHistogramBins	 	75
8.6	NppiPo	oint Struct	Reference	 	76
	8.6.1	Detailed 1	Description	 	76
	8.6.2	Field Doo	cumentation	 	76
		8.6.2.1	$x \ldots \ldots \ldots \ldots \ldots$	 	76
		8.6.2.2	y	 	76
8.7	NppiRe	ect Struct I	Reference	 	77
	8.7.1	Detailed 1	Description	 	77
	8.7.2	Field Doo	cumentation	 	77
		8.7.2.1	height	 	77
		8.7.2.2	width	 	77
		8.7.2.3	$x \ldots \ldots \ldots \ldots$	 	77
		8.7.2.4	y	 	77
8.8	NppiSi	ze Struct F	Reference	 	78
	8.8.1	Detailed 1	Description	 	78
	8.8.2	Field Doo	cumentation	 	78
		8.8.2.1	height	 	78
		8.8.2.2	width	 	78
8.9	NppLil	brary Versio	on Struct Reference	 	79
	8.9.1	Field Doo	cumentation	 	79
		8.9.1.1	build	 	79
		8.9.1.2	major	 	79
		8.9.1.3	minor	 	79
8.10	NppPo		truct Reference		80
	8.10.1	Detailed 1	Description	 	80
	8.10.2	Field Doo	cumentation	 	80
		8.10.2.1	rho	 	80
		8.10.2.2	theta	 	80

# Chapter 1

# **NVIDIA Performance Primitives**

Note: The static NPP libraries depend on a common thread abstraction layer library called cuLIBOS (libculibos.a) that is now distributed as part of the toolkit. Consequently, cuLIBOS must be provided to the linker when the static library is being linked against. To minimize library loading and CUDA runtime startup times it is recommended to use the static library(s) whenever possible. To improve loading and runtime performance when using dynamic libraries, NPP 9.0 has deprecated the full sized nppi library and replaced it with a full set of nppi sub-libraries. Linking to only the sub-libraries that contain functions that your application uses can significantly improve load time and runtime startup performance. Some nppi functions make calls to other nppi and/or npps functions internally so you may need to link to a few extra libraries depending on what function calls your application makes. The nppi sub-libraries are split into sections corresponding to the way that nppi header files are split. This list of sub-libraries is as follows:

```
nppial arithmetic and logical operation functions in nppi_arithmetic_and_logical_operations.h
nppicc color conversion and sampling functions in nppi_color_conversion.h
nppicom JPEG compression and decompression functions in nppi_compression_functions.h
nppidei data exchange and initialization functions in nppi_data_exchange_and_initialization.h
nppif filtering and computer vision functions in nppi_filter_functions.h
nppig geometry transformation functions found in nppi_geometry_transforms.h
nppim morphological operation functions found in nppi_morphological_operations.h
nppist statistics and linear transform in nppi_statistics_functions.h and nppi_linear_transforms.
nppisu memory support functions in nppi_support_functions.h
nppitc threshold and compare operation functions in nppi_threshold_and_compare_operations.h
```

For example, on Linux, to compile a small application foo using NPP against the dynamic library, the following command can be used:

```
nvcc foo.c -lnppi -o foo
```

Whereas to compile against the static NPP library, the following command has to be used:

```
nvcc foo.c -lnppi_static -lculibos -o foo
```

It is also possible to use the native host C++ compiler. Depending on the host operating system, some additional libraries like pthread or dl might be needed on the linking line. The following command on Linux is suggested:

```
g++ foo.c -lnppi_static -lculibos -lcudart_static -lpthread -ldl
-I <cuda-toolkit-path>/include -L <cuda-toolkit-path>/lib64 -o foo
```

NPP is a stateless API, as of NPP 6.5 the ONLY state that NPP remembers between function calls is the current stream ID, i.e. the stream ID that was set in the most recent nppSetStream call and a few bits

of device specific information about that stream. The default stream ID is 0. If an application intends to use NPP with multiple streams then it is the responsibility of the application to call nppSetStream whenever it wishes to change stream IDs. Several NPP functions may call other NPP functions internally to complete their functionality. For this reason it is recommended that cudaDeviceSynchronize (or at least cudaStreamSynchronize) be called before making an nppSetStream call to change to a new stream ID. This will insure that any internal function calls that have not yet occurred will be completed using the current stream ID before it changes to a new ID. Calling cudaDeviceSynchronize frequently call kill performance so minimizing the frequency of these calls is critical for good performance. It is not necessary to call cudaDeviceSynchronize for stream management while the same stream ID is used for multiple NPP calls. All NPP functions should be thread safe except for the following functions:

```
nppiDCTQuantFwd8x8LS_JPEG_8u16s_C1R
nppiDCTQuantInv8x8LS_JPEG_16s8u_C1R
```

#### 1.1 What is NPP?

NVIDIA NPP is a library of functions for performing CUDA accelerated processing. The initial set of functionality in the library focuses on imaging and video processing and is widely applicable for developers in these areas. NPP will evolve over time to encompass more of the compute heavy tasks in a variety of problem domains. The NPP library is written to maximize flexibility, while maintaining high performance.

NPP can be used in one of two ways:

- A stand-alone library for adding GPU acceleration to an application with minimal effort. Using this route allows developers to add GPU acceleration to their applications in a matter of hours.
- A cooperative library for interoperating with a developer's GPU code efficiently.

Either route allows developers to harness the massive compute resources of NVIDIA GPUs, while simultaneously reducing development times.

#### 1.2 Documentation

- General API Conventions
- Signal-Processing Specific API Conventions
- Imaging-Processing Specific API Conventions

## 1.3 Technical Specifications

Supported Platforms:

- Microsoft Windows 7, 8, and 10 (64-bit and 32-bit)
- Microsoft Windows Vista (64-bit and 32-bit)
- Linux (Centos, Ubuntu, and several others) (64-bit and 32-bit)
- Mac OS X (64-bit)
- Android on Arm (32-bit and 64-bit)

1.4 Files 3

#### 1.4 Files

NPP is comprises the following files:

#### 1.4.1 Header Files

- nppdefs.h
- nppcore.h
- nppi::h
- npps::h
- nppversion.h
- npp::h

All those header files are located in the CUDA Toolkit's

/include/

directory.

#### 1.4.2 Library Files

Starting with Version 5.5 NPP's functionality is now split up into 3 distinct library groups:

- A core library (NPPC) containing basic functionality from the npp.h header files as well as functionality shared by the other two libraries.
- The image processing library NPPI. Any functions from the nppi.h header file (or the various header files named "nppi\_xxx.h" are bundled into the NPPI library.
- The signal processing library NPPS. Any function from the npps.h header file (or the various header files named "npps\_xxx.h" are bundled into the NPPS library.

On the Windows platform the NPP stub libraries are found in the CUDA Toolkit's library directory:

```
/lib/nppc.lib
/lib/nppial.lib
/lib/nppicc.lib
/lib/nppicom.lib
/lib/nppidei.lib
/lib/nppif.lib
/lib/nppig.lib
```

```
/lib/nppim.lib
/lib/nppist.lib
/lib/nppisu.lib
/lib/nppitc.lib
/lib/npps.lib
```

The matching DLLs are located in the CUDA Toolkit's binary directory. Example

```
/bin/nppial64_90_<build_no>.dll // Dynamic image-processing library for 64-bit Windows.
```

On Linux and Mac platforms the dynamic libraries are located in the lib directory

```
/lib/libnppc.so.9.0.<br/>
// NPP dynamic core library for Linux /lib/libnpps.9.0.dylib // NPP dynamic signal processing library for Mac
```

# 1.5 Supported NVIDIA Hardware

NPP runs on all CUDA capable NVIDIA hardware. For details please see http://www.nvidia.com/object/cuda\_learn\_products.html

# **Chapter 2**

# **General API Conventions**

### 2.1 Memory Management

The design of all the NPP functions follows the same guidelines as other NVIDIA CUDA libraries like cuFFT and cuBLAS. That is that all pointer arguments in those APIs are device pointers.

This convention enables the individual developer to make smart choices about memory management that minimize the number of memory transfers. It also allows the user the maximum flexibility regarding which of the various memory transfer mechanisms offered by the CUDA runtime is used, e.g. synchronous or asynchronous memory transfers, zero-copy and pinned memory, etc.

The most basic steps involved in using NPP for processing data is as follows:

1. Transfer input data from the host to device using

```
cudaMemCpy(...)
```

- 2. Process data using one or several NPP functions or custom CUDA kernels
- 3. Transfer the result data from the device to the host using

```
cudaMemCpy(...)
```

#### 2.1.1 Scratch Buffer and Host Pointer

Some primitives of NPP require additional device memory buffers (scratch buffers) for calculations, e.g. signal and image reductions (Sum, Max, Min, MinMax, etc.). In order to give the NPP user maximum control regarding memory allocations and performance, it is the user's responsibility to allocate and delete those temporary buffers. For one this has the benefit that the library will not allocate memory unbeknownst to the user. It also allows developers who invoke the same primitive repeatedly to allocate the scratch only once, improving performance and potential device-memory fragmentation .

Scratch-buffer memory is unstructured and may be passed to the primitive in uninitialized form. This allows for reuse of the same scratch buffers with any primitive require scratch memory, as long as it is sufficiently sized.

The minimum scratch-buffer size for a given primitive (e.g. nppsSum\_32f()) can be obtained by a companion function (e.g. nppsSumGetBufferSize\_32f()). The buffer size is returned via a host pointer as allocation of the scratch-buffer is performed via CUDA runtime host code.

An example to invoke signal sum primitive and allocate and free the necessary scratch memory:

```
// pSrc, pSum, pDeviceBuffer are all device pointers.
Npp32f * pSrc;
Npp32f * pSum;
Npp8u * pDeviceBuffer;
int nLength = 1024;

// Allocate the device memroy.
cudaMalloc((void **)(&pSrc), sizeof(Npp32f) * nLength);
nppsSet_32f(1.0f, pSrc, nLength);
cudaMalloc((void **)(&pSum), sizeof(Npp32f) * 1);

// Compute the appropriate size of the scratch-memory buffer int nBufferSize;
nppsSumGetBufferSize_32f(nLength, &nBufferSize);
// Allocate the scratch buffer
cudaMalloc((void **)(&pDeviceBuffer), nBufferSize);
// Call the primitive with the scratch buffer
```

2.2 Function Naming 7

```
nppsSum_32f(pSrc, nLength, pSum, pDeviceBuffer);
Npp32f nSumHost;
cudaMemcpy(&nSumHost, pSum, sizeof(Npp32f) * 1, cudaMemcpyDeviceToHost);
printf("sum = %f\n", nSumHost); // nSumHost = 1024.0f;

// Free the device memory
cudaFree(pSrc);
cudaFree(pDeviceBuffer);
cudaFree(pSum);
```

## 2.2 Function Naming

Since NPP is a C API and therefore does not allow for function overloading for different data-types the NPP naming convention addresses the need to differentiate between different flavors of the same algorithm or primitive function but for various data types. This disambiguation of different flavors of a primitive is done via a suffix containing data type and other disambiguating information.

In addition to the flavor suffix, all NPP functions are prefixed with by the letters "npp". Primitives belonging to NPP's image-processing module add the letter "i" to the npp prefix, i.e. are prefixed by "nppi". Similarly signal-processing primitives are prefixed with "npps".

The general naming scheme is:

npp<module info><PrimitiveName>\_<data-type info>[\_<additional flavor info>]((parameter list>)

The data-type information uses the same names as the Basic NPP Data Types. For example the data-type information "8u" would imply that the primitive operates on Npp8u data.

If a primitive consumes different type data from what it produces, both types will be listed in the order of consumed to produced data type.

Details about the "additional flavor information" is provided for each of the NPP modules, since each problem domain uses different flavor information suffixes.

# 2.3 Integer Result Scaling

NPP signal processing and imaging primitives often operate on integer data. This integer data is usually a fixed point fractional representation of some physical magnitue (e.g. luminance). Because of this fixed-point nature of the representation many numerical operations (e.g. addition or multiplication) tend to produce results exceeding the original fixed-point range if treated as regular integers.

In cases where the results exceed the original range, these functions clamp the result values back to the valid range. E.g. the maximum positive value for a 16-bit unsigned integer is 32767. A multiplication operation of 4 \* 10000 = 40000 would exceed this range. The result would be clamped to be 32767.

To avoid the level of lost information due to clamping most integer primitives allow for result scaling. Primitives with result scaling have the "Sfs" suffix in their name and provide a parameter "nScaleFactor" that controls the amount of scaling. Before the results of an operation are clamped to the valid output-data range by multiplying them with 2-nScaleFactor.

Example: The primitive nppsSqr\_8u\_Sfs() computes the square of 8-bit unsigned sample values in a signal (1D array of values). The maximum value of a 8-bit value is 255. The square of  $255^2=65025$  which would be clamped to 255 if no result scaling is performed. In order to map the maximum value of 255 to 255 in the result, one would specify an integer result scaling factor of 8, i.e. multiply each result with  $2^{-8}=\frac{1}{2^8}=\frac{1}{256}$ . The final result for a signal value of 255 being squared and scaled would be:

$$255^2 \cdot 2^{-8} = 254.00390625$$

8 General API Conventions

which would be rounded to a final result of 254.

A medium gray value of 128 would result in

$$128^2 * 2^{-8} = 64$$

# 2.4 Rounding Modes

Many NPP functions require converting floating-point values to integers. The NppRoundMode enum lists NPP's supported rounding modes. Not all primitives in NPP that perform rounding as part of their functionality allow the user to specify the round-mode used. Instead they use NPP's default rounding mode, which is NPP\_RND\_FINANCIAL.

### 2.4.1 Rounding Mode Parameter

A subset of NPP functions performing rounding as part of their functionality do allow the user to specify which rounding mode is used through a parameter of the NppRoundMode type.

# **Chapter 3**

# **Signal-Processing Specific API Conventions**

## 3.1 Signal Data

Signal data is passed to and from NPPS primitives via a pointer to the signal's data type.

The general idea behind this fairly low-level way of passing signal data is ease-of-adoption into existing software projects:

• Passing the data pointer rather than a higher-level signal struct allows for easy adoption by not requiring a specific signal representation (that could include total signal size offset, or other additional information). This avoids awkward packing and unpacking of signal data from the host application to an NPP specific signal representation.

#### 3.1.1 Parameter Names for Signal Data

There are three general cases of image-data passing throughout NPP detailed in the following sections.

Those are signals consumed by the algorithm.

#### 3.1.1.1 Source Signal Pointer

The source signal data is generally passed via a pointer named

```
pSrc
```

The source signal pointer is generally defined constant, enforcing that the primitive does not change any image data pointed to by that pointer. E.g.

```
nppsPrimitive_32s(const Npp32s * pSrc, ...)
```

In case the primitive consumes multiple signals as inputs the source pointers are numbered like this:

```
pSrc1, pScr2, ...
```

#### 3.1.1.2 Destination Signal Pointer

The destination signal data is generally passed via a pointer named

```
pDst
```

In case the primitive consumes multiple signals as inputs the source pointers are numbered like this:

```
pDst1, pDst2, ...
```

#### 3.1.1.3 In-Place Signal Pointer

In the case of in-place processing, source and destination are served by the same pointer and thus pointers to in-place signal data are called:

```
pSrcDst
```

3.2 Signal Length

### 3.1.2 Signal Data Alignment Requirements

NPP requires signal sample data to be naturally aligned, i.e. any pointer

```
NppType * p;
```

to a sample in a signal needs to fulfill:

```
assert(p % sizeof(p) == 0);
```

#### 3.1.3 Signal Data Related Error Codes

All NPPI primitives operating on signal data validate the signal-data pointer for proper alignment and test that the point is not null.

Failed validation results in one of the following error codes being returned and the primitive not being executed:

- NPP NULL POINTER ERROR is returned if the image-data pointer is 0 (NULL).
- NPP\_ALIGNMENT\_ERROR if the signal-data pointer address is not a multiple of the signal's data-type size.

### 3.2 Signal Length

The vast majority of NPPS functions take a

```
nLength
```

parameter that tells the primitive how many of the signal's samples starting from the given data pointer are to be processed.

#### 3.2.1 Length Related Error Codes

All NPPS primitives taking a length parameter validate this input.

Failed validation results in the following error code being returned and the primitive not being executed:

• NPP\_SIZE\_ERROR is returned if the length is negative.

12	Signal-Processing Specific API Conventions

# **Chapter 4**

# **Imaging-Processing Specific API Conventions**

### 4.1 Function Naming

Image processing related functions use a number of suffixes to indicate various different flavors of a primitive beyond just different data types. The flavor suffix uses the following abbreviations:

- "A" if the image is a 4 channel image this indicates the result alpha channel is not affected by the primitive.
- "Cn" the image consists of n channel packed pixels, where n can be 1, 2, 3 or 4.
- "Pn" the image consists of n separate image planes, where n can be 1, 2, 3 or 4.
- "C" (following the channel information) indicates that the primitive only operates on one of the color channels, the "channel-of-interest". All other output channels are not affected by the primitive.
- "I" indicates that the primitive works "in-place". In this case the image-data pointer is usually named "pSrcDst" to indicate that the image data serves as source and destination at the same time.
- "M" indicates "masked operation". These types of primitives have an additional "mask image" as as input. Each pixel in the destination image corresponds to a pixel in the mask image. Only pixels with a corresponding non-zero mask pixel are being processed.
- "R" indicates the primitive operates only on a rectangular "region-of-interest" or "ROI". All ROI primitives take an additional input parameter of type NppiSize, which specifies the width and height of the rectangular region that the primitive should process. For details on how primitives operate on ROIs see: Region-of-Interest (ROI).
- "Sfs" indicates the result values are processed by fixed scaling and saturation before they're written
  out.

The suffixes above always appear in alphabetical order. E.g. a 4 channel primitive not affecting the alpha channel with masked operation, in place and with scaling/saturation and ROI would have the postfix: "AC4IMRSfs".

# 4.2 Image Data

Image data is passed to and from NPPI primitives via a pair of parameters:

- 1. A pointer to the image's underlying data type.
- 2. A line step in bytes (also sometimes called line stride).

The general idea behind this fairly low-level way of passing image data is ease-of-adoption into existing software projects:

- Passing a raw pointer to the underlying pixel data type, rather than structured (by color) channel pixel
  data allows usage of the function in a wide variety of situations avoiding risky type cast or expensive
  image data copies.
- Passing the data pointer and line step individually rather than a higher-level image struct again allows for easy adoption by not requiring a specific image representation and thus avoiding awkward packing and unpacking of image data from the host application to an NPP specific image representation.

4.2 Image Data

#### **4.2.1** Line Step

The line step (also called "line stride" or "row step") allows lines of oddly sized images to start on well-aligned addresses by adding a number of unused bytes at the ends of the lines. This type of line padding has been common practice in digital image processing for a long time and is not particular to GPU image processing.

The line step is the number of bytes in a line **including the padding.** An other way to interpret this number is to say that it is the number of bytes between the first pixel of successive rows in the image, or generally the number of bytes between two neighboring pixels in any column of pixels.

The general reason for the existence of the line step it is that uniformly aligned rows of pixel enable optimizations of memory-access patterns.

Even though all functions in NPP will work with arbitrarily aligned images, best performance can only be achieved with well aligned image data. Any image data allocated with the NPP image allocators or the 2D memory allocators in the CUDA runtime, is well aligned.

Particularly on older CUDA capable GPUs it is likely that the performance decrease for misaligned data is substantial (orders of magnitude).

All image data passed to NPPI primitives requires a line step to be provided. It is important to keep in mind that this line step is always specified in terms of bytes, not pixels.

#### 4.2.2 Parameter Names for Image Data

There are three general cases of image-data passing throughout NPP detailed in the following sections.

#### 4.2.2.1 Passing Source-Image Data

Those are images consumed by the algorithm.

#### 4.2.2.1.1 Source-Image Pointer

The source image data is generally passed via a pointer named

```
pSrc
```

The source image pointer is generally defined constant, enforcing that the primitive does not change any image data pointed to by that pointer. E.g.

```
nppiPrimitive_32s_C1R(const Npp32s * pSrc, ...)
```

In case the primitive consumes multiple images as inputs the source pointers are numbered like this:

```
pSrc1, pScr2, ...
```

#### 4.2.2.1.2 Source-Planar-Image Pointer Array

The planar source image data is generally passed via an array of pointers named

```
pSrc[]
```

The planar source image pointer array is generally defined a constant array of constant pointers, enforcing that the primitive does not change any image data pointed to by those pointers. E.g.

```
nppiPrimitive_8u_P3R(const Npp8u * const pSrc[3], ...)
```

Each pointer in the array points to a different image plane.

#### 4.2.2.1.3 Source-Planar-Image Pointer

The multiple plane source image data is passed via a set of pointers named

```
pSrc1, pSrc2, ...
```

The planar source image pointer is generally defined as one of a set of constant pointers with each pointer pointing to a different input image plane.

#### 4.2.2.1.4 Source-Image Line Step

The source image line step is the number of bytes between successive rows in the image. The source image line step parameter is

```
nSrcStep
```

or in the case of multiple source images

```
nSrcStep1, nSrcStep2, ...
```

#### 4.2.2.1.5 Source-Planar-Image Line Step Array

The source planar image line step array is an array where each element of the array contains the number of bytes between successive rows for a particular plane in the input image. The source planar image line step array parameter is

```
rSrcStep[]
```

#### 4.2.2.1.6 Source-Planar-Image Line Step

The source planar image line step is the number of bytes between successive rows in a particular plane of the multiplane input image. The source planar image line step parameter is

```
nSrcStep1, nSrcStep2, ...
```

#### 4.2.2.2 Passing Destination-Image Data

Those are images produced by the algorithm.

4.2 Image Data 17

#### 4.2.2.2.1 Destination-Image Pointer

The destination image data is generally passed via a pointer named

```
pDst
```

In case the primitive generates multiple images as outputs the destination pointers are numbered like this:

```
pDst1, pDst2, ...
```

#### 4.2.2.2. Destination-Planar-Image Pointer Array

The planar destination image data pointers are generally passed via an array of pointers named

```
pDst[]
```

Each pointer in the array points to a different image plane.

#### 4.2.2.2.3 Destination-Planar-Image Pointer

The destination planar image data is generally passed via a pointer to each plane of a multiplane output image named

```
pDst1, pDst2, ...
```

#### 4.2.2.2.4 Destination-Image Line Step

The destination image line step parameter is

```
nDstStep
```

or in the case of multiple destination images

```
nDstStep1, nDstStep2, ...
```

#### 4.2.2.2.5 Destination-Planar-Image Line Step Array

The destination planar image line step array is an array where each element of the array contains the number of bytes between successive rows for a particular plane in the output image. The destination planar image line step array parameter is

```
rDstStep[]
```

#### 4.2.2.2.6 Destination-Planar-Image Line Step

The destination planar image line step is the number of bytes between successive rows for a particular plane in a multiplane output image. The destination planar image line step parameter is

```
nDstStep1, nDstStep2, ...
```

#### 4.2.2.3 Passing In-Place Image Data

#### 4.2.2.3.1 In-Place Image Pointer

In the case of in-place processing, source and destination are served by the same pointer and thus pointers to in-place image data are called:

pSrcDst

#### 4.2.2.3.2 In-Place-Image Line Step

The in-place line step parameter is

nSrcDstStep

#### 4.2.2.4 Passing Mask-Image Data

Some image processing primitives have variants supporting Masked Operation.

#### 4.2.2.4.1 Mask-Image Pointer

The mask-image data is generally passed via a pointer named

pMask

#### 4.2.2.4.2 Mask-Image Line Step

The mask-image line step parameter is

nMaskStep

#### 4.2.2.5 Passing Channel-of-Interest Data

Some image processing primitives support Channel-of-Interest API.

#### 4.2.2.5.1 Channel\_of\_Interest Number

The channel-of-interest data is generally an integer (either 1, 2, or 3):

nCOI

#### **4.2.3** Image Data Alignment Requirements

NPP requires pixel data to adhere to certain alignment constraints: For 2 and 4 channel images the following alignment requirement holds: data\_pointer % (#channels \* sizeof(channel type)) == 0. E.g. a 4 channel image with underlying type Npp8u (8-bit unsigned) would require all pixels to fall on addresses that are multiples of 4 (4 channels \* 1 byte size).

As a logical consequence of all pixels being aligned to their natural size the image line steps of 2 and 4 channel images also need to be multiples of the pixel size.

1 and 3 channel images only require that pixel pointers are aligned to the underlying data type, i.e. pData % sizof(data type) == 0. And consequentially line steps are also held to this requirement.

#### **4.2.4** Image Data Related Error Codes

All NPPI primitives operating on image data validate the image-data pointer for proper alignment and test that the point is not null. They also validate the line stride for proper alignment and guard against the step being less or equal to 0. Failed validation results in one of the following error codes being returnd and the primitive not being executed:

- NPP\_STEP\_ERROR is returned if the data step is 0 or negative.
- NPP\_NOT\_EVEN\_STEP\_ERROR is returned if the line step is not a multiple of the pixel size for 2 and 4 channel images.
- NPP\_NULL\_POINTER\_ERROR is returned if the image-data pointer is 0 (NULL).
- NPP\_ALIGNMENT\_ERROR if the image-data pointer address is not a multiple of the pixel size for 2 and 4 channel images.

### 4.3 Region-of-Interest (ROI)

In practice processing a rectangular sub-region of an image is often more common than processing complete images. The vast majority of NPP's image-processing primitives allow for processing of such sub regions also referred to as regions-of-interest or ROIs.

All primitives supporting ROI processing are marked by a "R" in their name suffix. In most cases the ROI is passed as a single NppiSize struct, which provides the with and height of the ROI. This raises the question how the primitive knows where in the image this rectangle of (width, height) is located. The "start pixel" of the ROI is implicitly given by the image-data pointer. I.e. instead of explicitly passing a pixel coordinate for the upper-left corner (lowest memory address), the user simply offsets the image-data pointers to point to the first pixel of the ROI.

In practice this means that for an image (pSrc, nSrcStep) and the start-pixel of the ROI being at location (x, y), one would pass

```
pSrcOffset = pSrc + y * nSrcStep + x * PixelSize;
```

as the image-data source to the primitive. PixelSize is typically computed as

PixelSize = NumberOfColorChannels \* sizeof(PixelDataType).

E.g. for a pimitive like nppiSet\_16s\_C4R() we would have

- NumberOfColorChannels == 4;
- sizeof(Npp16s) == 2;
- and thus PixelSize = 4 \* 2 = 8;

#### 4.3.1 ROI Related Error Codes

All NPPI primitives operating on ROIs of image data validate the ROI size and image's step size. Failed validation results in one of the following error codes being returned and the primitive not being executed:

- NPP\_SIZE\_ERROR is returned if either the ROI width or ROI height are negative.
- NPP\_STEP\_ERROR is returned if the ROI width exceeds the image's line step. In mathematical terms (widthROI \* PixelSize) > nLinStep indicates an error.

### 4.4 Masked Operation

Some primitive support masked operation. An "M" in the suffix of those variants indicates masked operation. Primitives supporting masked operation consume an additional input image provided via a Mask-Image Pointer and Mask-Image Line Step. The mask image is interpreted by these primitives as a boolean image. The values of type Npp8u are interpreted as boolean values where a values of 0 indicates false, any non-zero values true.

Unless otherwise indicated the operation is only performed on pixels where its spatially corresponding mask pixel is true (non-zero). E.g. a masked copy operation would only copy those pixels in the ROI that have corresponding non-zero mask pixels.

#### 4.5 Channel-of-Interest API

Some primitives allow restricting operations to a single channel of interest within a multi-channel image. These primitives are suffixed with the letter "C" (after the channel information, e.g. nppiCopy\_-8u\_C3CR(...). The channel-of-interest is generally selected by offsetting the image-data pointer to point directly to the channel- of-interest rather than the base of the first pixel in the ROI. Some primitives also explicitly specify the selected channel number and pass it via an integer, e.g. nppiMean\_StdDev\_8u\_-C3CR(...).

#### 4.5.1 Select-Channel Source-Image Pointer

This is a pointer to the channel-of-interest within the first pixel of the source image. E.g. if pSrc is the pointer to the first pixel inside the ROI of a three channel image. Using the appropriate select-channel copy primitive one could copy the second channel of this source image into the first channel of a destination image given by pDst by offsetting the pointer by one:

```
nppiCopy_8u_C3CR(pSrc + 1, nSrcStep, pDst, nDstStep, oSizeROI);
```

#### 4.5.2 Select-Channel Source-Image

Some primitives allow the user to select the channel-of-interest by specifying the channle number (nCOI). This approach is typically used in the image statistical functions. For example,

```
nppiMean_StdDev_8u_C3CR(pSrc, nSrcStep, oSizeROI, nCOI, pDeviceBuffer, pMean, pStdDev );
```

The channel-of-interest number can be either 1, 2, or 3.

### 4.5.3 Select-Channel Destination-Image Pointer

This is a pointer to the channel-of-interest within the first pixel of the destination image. E.g. if pDst is the pointer to the first pixel inside the ROI of a three channel image. Using the appropriate select-channel

copy primitive one could copy data into the second channel of this destination image from the first channel of a source image given by pSrc by offseting the destination pointer by one:

```
nppiCopy_8u_C3CR(pSrc, nSrcStep, pDst + 1, nDstStep, oSizeROI);
```

### 4.6 Source-Image Sampling

A large number of NPP image-processing functions consume at least one source image and produce an output image (e.g. nppiAddC\_8u\_C1RSfs() or nppiFilterBox\_8u\_C1R()). All NPP functions falling into this category also operate on ROIs (see Region-of-Interest (ROI)) which for these functions should be considered to describe the destination ROI. In other words the ROI describes a rectangular region in the destination image and all pixels inside of this region are being written by the function in question.

In order to use such functions successfully it is important to understand how the user defined destination ROI affects which pixels in the input image(s) are being read by the algorithms. To simplify the discussion of ROI propagation (i.e. given a destination ROI, what are the ROIs in the source(s)), it makes sense to distinguish two major cases:

- 1. Point-Wise Operations: These are primitives like nppiAddC\_8u\_C1RSfs(). Each output pixel requires exactly one input pixel to be read.
- 2. Neighborhood Operations: These are primitives like nppiFilterBox\_8u\_C1R(), which require a group of pixels from the source image(s) to be read in order to produce a single output.

#### 4.6.1 Point-Wise Operations

As mentioned above, point-wise operations consume a single pixel from the input image (or a single pixel from each input image, if the operation in question has more than one input image) in order to produce a single output pixel.

#### 4.6.2 Neighborhood Operations

In the case of neightborhood operations a number of input pixels (a "neighborhood" of pixels) is read in the input image (or images) in order to compute a single output pixel. All of the functions for image\_filtering\_functions and image\_morphological\_operations are neighborhood operations.

Most of these functions have parameters that affect the size and relative location of the neighborhood: a mask-size structure and an achor-point structure. Both parameters are described in more detail in the next subsections.

#### 4.6.2.1 Mask-Size Parameter

Many NPP neighborhood operations allow the user to specify the size of the neighborhood via a parameter usually named oMaskSize of type NppiSize. In those cases the neighborhood of pixels read from the source(s) is exactly the size of the mask. Assuming the mask is anchored at location (0, 0) (see Anchor-Point Parameter below) and has a size of (w, h), i.e.

```
assert(oMaskSize.w == w);
assert(oMaskSize.h == h);
assert(oAnchor.x == 0);
assert(oAnchor.y == 0);
```

a neighborhood operation would read the following source pixels in order to compute destiation pixel  $D_{i,j}$ :

```
S_{i,j} S_{i,j+1} ... S_{i,j+w-1} S_{i+1,j} S_{i+1,j+1} ... S_{i+1,j+w-1} ... S_{i+1,j+w-1} ... S_{i+h-1,j} S_{i+h-1,j+1} ... S_{i+h-1,j+w-1}
```

#### 4.6.2.2 Anchor-Point Parameter

Many NPP primitives perforing neighborhood operations allow the user to specify the relative location of the neighborhood via a parameter usually named oAnchor of type NppiPoint. Using the anchor a developer can chose the position of the mask (see Mask-Size Parameter) relative to current pixel index.

Using the same example as in Mask-Size Parameter, but this time with an anchor position of (a, b):

```
assert (oMaskSize.w == w);
assert (oMaskSize.h == h);
assert (oAnchor.x == a);
assert (oAnchor.y == b);
```

the following pixels from the source image would be read:

```
S_{i-a,j-b} S_{i-a,j-b+1} ... S_{i-a,j-b+w-1} S_{i-a+1,j-b} S_{i-a+1,j-b+1} ... S_{i-a+1,j-b+w-1} ... S_{i-a+1,j-b+w-1} ... S_{i-a+h-1,j-b+w-1}
```

#### 4.6.2.3 Sampling Beyond Image Boundaries

NPP primitives in general and NPP neighborhood operations in particular require that all pixel locations read and written are valid and within the boundaries of the respective images. Sampling outside of the defined image data regions results in undefined behavior and may lead to system instabilty.

This poses a problem in practice: when processing full-size images one cannot choose the destination ROI to be the same size as the source image. Because neigborhood operations read pixels from an enlarged source ROI, the destination ROI must be shrunk so that the expanded source ROI does not exceed the source image's size.

For cases where this "shrinking" of the destination image size is unacceptable, NPP provides a set of border-expanding Copy primitives. E.g. nppiCopyConstBorder\_8u\_C1R(), nppiCopyReplicateBorder\_8u\_C1R() and nppiCopyWrapBorder\_8u\_C1R(). The user can use these primitives to "expand" the source image's size using one of the three expansion modes. The expanded image can then be safely passed to a neighborhood operation producing a full-size result.

# **Chapter 5**

# **Module Index**

# 5.1 Modules

	_			_		_		
Here	10	9	liet	Ωť	all	mod	111	Pe

NPP Core	27
NPP Type Definitions and Constants	31
Basic NPP Data Types	47
Compression	51
Quantization Functions	61

24 Module Index

# Chapter 6

# **Data Structure Index**

# **6.1 Data Structures**

Here are the data structures with brief descriptions:

NPP_ALIGN_16 (Complex Number This struct represents a long long complex number)	69
NPP_ALIGN_8 (Complex Number This struct represents an unsigned int complex number)	71
NppiHaarBuffer	73
NppiHaarClassifier_32f	74
NppiHOGConfig (The NppiHOGConfig structure defines the configuration parameters for the	
HOG descriptor: )	75
NppiPoint (2D Point )	76
NppiRect (2D Rectangle This struct contains position and size information of a rectangle in two	
space)	77
NppiSize (2D Size This struct typically represents the size of a a rectangular region in two space)	78
NppLibrary Version	<del>7</del> 9
NppPointPolar (2D Polar Point )	80

26 Data Structure Index

## **Chapter 7**

## **Module Documentation**

## 7.1 NPP Core

Basic functions for library management, in particular library version and device property query functions.

## **Functions**

- const NppLibrary Version \* nppGetLibVersion (void) Get the NPP library version.
- NppGpuComputeCapability nppGetGpuComputeCapability (void)
   What CUDA compute model is supported by the active CUDA device?
- int nppGetGpuNumSMs (void)

Get the number of Streaming Multiprocessors (SM) on the active CUDA device.

• int nppGetMaxThreadsPerBlock (void)

Get the maximum number of threads per block on the active CUDA device.

• int nppGetMaxThreadsPerSM (void)

Get the maximum number of threads per SM for the active GPU.

• int nppGetGpuDeviceProperties (int \*pMaxThreadsPerSM, int \*pMaxThreadsPerBlock, int \*pNumberOfSMs)

Get the maximum number of threads per SM, maximum threads per block, and number of SMs for the active GPU

- const char \* nppGetGpuName (void)
  - Get the name of the active CUDA device.
- cudaStream\_t nppGetStream (void)

Get the NPP CUDA stream.

• unsigned int nppGetStreamNumSMs (void)

Get the number of SMs on the device associated with the current NPP CUDA stream.

• unsigned int nppGetStreamMaxThreadsPerSM (void)

Get the maximum number of threads per SM on the device associated with the current NPP CUDA stream.

• void nppSetStream (cudaStream\_t hStream)

Set the NPP CUDA stream.

## 7.1.1 Detailed Description

Basic functions for library management, in particular library version and device property query functions.

## 7.1.2 Function Documentation

## 7.1.2.1 NppGpuComputeCapability nppGetGpuComputeCapability (void)

What CUDA compute model is supported by the active CUDA device?

Before trying to call any NPP functions, the user should make a call this function to ensure that the current machine has a CUDA capable device.

### **Returns:**

An enum value representing if a CUDA capable device was found and what level of compute capabilities it supports.

## 7.1.2.2 int nppGetGpuDeviceProperties (int \* pMaxThreadsPerSM, int \* pMaxThreadsPerBlock, int \* pNumberOfSMs)

Get the maximum number of threads per SM, maximum threads per block, and number of SMs for the active GPU.

## **Returns:**

cudaSuccess for success, -1 for failure

#### 7.1.2.3 const char\* nppGetGpuName (void)

Get the name of the active CUDA device.

## **Returns:**

Name string of the active graphics-card/compute device in a system.

### 7.1.2.4 int nppGetGpuNumSMs (void)

Get the number of Streaming Multiprocessors (SM) on the active CUDA device.

#### **Returns:**

Number of SMs of the default CUDA device.

7.1 NPP Core 29

### 7.1.2.5 const NppLibraryVersion\* nppGetLibVersion (void)

Get the NPP library version.

#### **Returns:**

A struct containing separate values for major and minor revision and build number.

## 7.1.2.6 int nppGetMaxThreadsPerBlock (void)

Get the maximum number of threads per block on the active CUDA device.

#### **Returns:**

Maximum number of threads per block on the active CUDA device.

### 7.1.2.7 int nppGetMaxThreadsPerSM (void)

Get the maximum number of threads per SM for the active GPU.

#### **Returns:**

Maximum number of threads per SM for the active GPU

## 7.1.2.8 cudaStream\_t nppGetStream (void)

Get the NPP CUDA stream.

NPP enables concurrent device tasks via a global stream state varible. The NPP stream by default is set to stream 0, i.e. non-concurrent mode. A user can set the NPP stream to any valid CUDA stream. All CUDA commands issued by NPP (e.g. kernels launched by the NPP library) are then issed to that NPP stream.

### 7.1.2.9 unsigned int nppGetStreamMaxThreadsPerSM (void)

Get the maximum number of threads per SM on the device associated with the current NPP CUDA stream.

NPP enables concurrent device tasks via a global stream state varible. The NPP stream by default is set to stream 0, i.e. non-concurrent mode. A user can set the NPP stream to any valid CUDA stream. All CUDA commands issued by NPP (e.g. kernels launched by the NPP library) are then issed to that NPP stream. This call avoids a cudaGetDeviceProperties() call.

## 7.1.2.10 unsigned int nppGetStreamNumSMs (void)

Get the number of SMs on the device associated with the current NPP CUDA stream.

NPP enables concurrent device tasks via a global stream state varible. The NPP stream by default is set to stream 0, i.e. non-concurrent mode. A user can set the NPP stream to any valid CUDA stream. All CUDA commands issued by NPP (e.g. kernels launched by the NPP library) are then issed to that NPP stream. This call avoids a cudaGetDeviceProperties() call.

## 7.1.2.11 void nppSetStream (cudaStream\_t hStream)

Set the NPP CUDA stream.

## See also:

nppGetStream()

## 7.2 NPP Type Definitions and Constants

### **Data Structures**

- struct NppLibraryVersion
- struct NppiPoint

2D Point

• struct NppPointPolar

2D Polar Point

• struct NppiSize

2D Size This struct typically represents the size of a a rectangular region in two space.

• struct NppiRect

2D Rectangle This struct contains position and size information of a rectangle in two space.

• struct NppiHOGConfig

The NppiHOGConfig structure defines the configuration parameters for the HOG descriptor:.

- struct NppiHaarClassifier\_32f
- struct NppiHaarBuffer

## **Modules**

• Basic NPP Data Types

### **Defines**

- #define NPP\_MIN\_8U ( 0 )
  - Minimum 8-bit unsigned integer.
- #define NPP MAX 8U (255)

Maximum 8-bit unsigned integer.

• #define NPP\_MIN\_16U ( 0 )

Minimum 16-bit unsigned integer.

• #define NPP\_MAX\_16U (65535)

Maximum 16-bit unsigned integer.

• #define NPP\_MIN\_32U ( 0 )

Minimum 32-bit unsigned integer.

• #define NPP\_MAX\_32U ( 4294967295U )

Maximum 32-bit unsigned integer.

• #define NPP\_MIN\_64U ( 0 )

Minimum 64-bit unsigned integer.

```
• #define NPP_MAX_64U ( 18446744073709551615ULL )
     Maximum 64-bit unsigned integer.
• #define NPP_MIN_8S (-127 - 1)
     Minimum 8-bit signed integer.
• #define NPP MAX 8S (127)
     Maximum 8-bit signed integer.
• #define NPP MIN 16S (-32767 - 1)
     Minimum 16-bit signed integer.
• #define NPP_MAX_16S ( 32767 )
     Maximum 16-bit signed integer.
• #define NPP_MIN_32S (-2147483647 - 1)
     Minimum 32-bit signed integer.
• #define NPP_MAX_32S ( 2147483647 )
     Maximum 32-bit signed integer.
• #define NPP_MAX_64S ( 9223372036854775807LL )
     Maximum 64-bit signed integer.
• #define NPP_MIN_64S (-9223372036854775807LL - 1)
     Minimum 64-bit signed integer.
• #define NPP_MINABS_32F ( 1.175494351e-38f )
     Smallest positive 32-bit floating point value.
• #define NPP MAXABS 32F ( 3.402823466e+38f )
     Largest positive 32-bit floating point value.
• #define NPP_MINABS_64F ( 2.2250738585072014e-308 )
     Smallest positive 64-bit floating point value.
• #define NPP_MAXABS_64F ( 1.7976931348623158e+308 )
     Largest positive 64-bit floating point value.
• #define NPP HOG MAX CELL SIZE (16)
     max horizontal/vertical pixel size of cell.
• #define NPP HOG MAX BLOCK SIZE (64)
     max horizontal/vertical pixel size of block.
• #define NPP_HOG_MAX_BINS_PER_CELL (16)
     max number of histogram bins.
```

• #define NPP\_HOG\_MAX\_CELLS\_PER\_DESCRIPTOR (256)

max number of cells in a descriptor window.

- #define NPP\_HOG\_MAX\_OVERLAPPING\_BLOCKS\_PER\_DESCRIPTOR (256) max number of overlapping blocks in a descriptor window.
- #define NPP\_HOG\_MAX\_DESCRIPTOR\_LOCATIONS\_PER\_CALL (128) max number of descriptor window locations per function call.

## **Enumerations**

```
• enum NppiInterpolationMode {
 NPPI_INTER_UNDEFINED = 0,
 NPPI_INTER_NN = 1,
 NPPI_INTER_LINEAR = 2,
 NPPI INTER CUBIC = 4,
 NPPI_INTER_CUBIC2P_BSPLINE,
 NPPI_INTER_CUBIC2P_CATMULLROM,
 NPPI_INTER_CUBIC2P_B05C03,
 NPPI_INTER_SUPER = 8,
 NPPI_INTER_LANCZOS = 16,
 NPPI_INTER_LANCZOS3_ADVANCED = 17,
 NPPI_SMOOTH_EDGE = (1 << 31)}
    Filtering methods.
• enum NppiBayerGridPosition {
 NPPI_BAYER_BGGR = 0,
 NPPI_BAYER_RGGB = 1,
 NPPI_BAYER_GBRG = 2,
 NPPI_BAYER_GRBG = 3 }
    Bayer Grid Position Registration.
• enum NppiMaskSize {
 NPP_MASK_SIZE_1_X_3,
 NPP_MASK_SIZE_1_X_5,
 NPP\_MASK\_SIZE\_3\_X\_1 = 100,
 NPP_MASK_SIZE_5_X_1,
 NPP\_MASK\_SIZE\_3\_X\_3 = 200,
 NPP_MASK_SIZE_5_X_5,
 NPP\_MASK\_SIZE\_7\_X\_7 = 400,
 NPP\_MASK\_SIZE\_9\_X\_9 = 500,
 NPP\_MASK\_SIZE\_11\_X\_11 = 600,
 NPP_MASK_SIZE_{13}X_{13} = 700,
 NPP\_MASK\_SIZE\_15\_X\_15 = 800
```

Fixed filter-kernel sizes.

```
• enum NppiDifferentialKernel {
 NPP_FILTER_SOBEL,
 NPP_FILTER_SCHARR }
    Differential Filter types.
• enum NppStatus {
 NPP_NOT_SUPPORTED_MODE_ERROR = -9999,
 NPP_INVALID_HOST_POINTER_ERROR = -1032,
 NPP_INVALID_DEVICE_POINTER_ERROR = -1031,
 NPP_LUT_PALETTE_BITSIZE_ERROR = -1030,
 NPP_ZC_MODE_NOT_SUPPORTED_ERROR = -1028,
 NPP_NOT_SUFFICIENT_COMPUTE_CAPABILITY = -1027,
 NPP TEXTURE BIND ERROR = -1024,
 NPP_WRONG_INTERSECTION_ROI_ERROR = -1020,
 NPP_HAAR_CLASSIFIER_PIXEL_MATCH_ERROR = -1006,
 NPP\_MEMFREE\_ERROR = -1005,
 NPP\_MEMSET\_ERROR = -1004,
 NPP\_MEMCPY\_ERROR = -1003,
 NPP\_ALIGNMENT\_ERROR = -1002,
 NPP_CUDA_KERNEL_EXECUTION_ERROR = -1000,
 NPP_ROUND_MODE_NOT_SUPPORTED_ERROR = -213,
 NPP_QUALITY_INDEX_ERROR = -210,
 NPP_RESIZE_NO_OPERATION_ERROR = -201,
 NPP OVERFLOW ERROR = -109,
 NPP_NOT_EVEN_STEP_ERROR = -108,
 NPP_HISTOGRAM_NUMBER_OF_LEVELS_ERROR = -107,
 NPP_LUT_NUMBER_OF_LEVELS_ERROR = -106,
 NPP_CORRUPTED_DATA_ERROR = -61,
 NPP\_CHANNEL\_ORDER\_ERROR = -60,
 NPP_ZERO_MASK_VALUE_ERROR = -59,
 NPP_QUADRANGLE_ERROR = -58,
 NPP_RECTANGLE_ERROR = -57,
 NPP COEFFICIENT ERROR = -56,
 NPP_NUMBER_OF_CHANNELS_ERROR = -53,
 NPP\_COI\_ERROR = -52,
 NPP DIVISOR ERROR = -51,
 NPP_CHANNEL_ERROR = -47,
 NPP\_STRIDE\_ERROR = -37,
 NPP\_ANCHOR\_ERROR = -34,
 NPP\_MASK\_SIZE\_ERROR = -33,
```

```
NPP_RESIZE_FACTOR_ERROR = -23,
 NPP_INTERPOLATION_ERROR = -22,
 NPP_MIRROR_FLIP_ERROR = -21,
 NPP\_MOMENT\_00\_ZERO\_ERROR = -20,
 NPP_THRESHOLD_NEGATIVE_LEVEL_ERROR = -19,
 NPP\_THRESHOLD\_ERROR = -18,
 NPP_CONTEXT_MATCH_ERROR = -17,
 NPP_FFT_FLAG_ERROR = -16,
 NPP FFT ORDER ERROR = -15,
 NPP\_STEP\_ERROR = -14,
 NPP_SCALE_RANGE_ERROR = -13,
 NPP_DATA_TYPE_ERROR = -12,
 NPP_OUT_OFF_RANGE_ERROR = -11,
 NPP_DIVIDE_BY_ZERO_ERROR = -10,
 NPP_MEMORY_ALLOCATION_ERR = -9,
 NPP_NULL_POINTER_ERROR = -8,
 NPP_RANGE_ERROR = -7,
 NPP\_SIZE\_ERROR = -6,
 NPP_BAD_ARGUMENT_ERROR = -5,
 NPP_NO_MEMORY_ERROR = -4,
 NPP_NOT_IMPLEMENTED_ERROR = -3,
 NPP ERROR = -2,
 NPP\_ERROR\_RESERVED = -1,
 NPP_NO_ERROR = 0,
 NPP_SUCCESS = NPP_NO_ERROR,
 NPP NO OPERATION WARNING = 1,
 NPP_DIVIDE_BY_ZERO_WARNING = 6,
 NPP_AFFINE_QUAD_INCORRECT_WARNING = 28,
 NPP_WRONG_INTERSECTION_ROI_WARNING = 29,
 NPP WRONG INTERSECTION QUAD WARNING = 30,
 NPP_DOUBLE_SIZE_WARNING = 35,
 NPP_MISALIGNED_DST_ROI_WARNING = 10000 }
    Error Status Codes.
• enum NppGpuComputeCapability {
 NPP_CUDA_UNKNOWN_VERSION = -1,
 NPP\_CUDA\_NOT\_CAPABLE = 0,
 NPP\_CUDA\_1\_0 = 100,
 NPP_CUDA_1_1 = 110,
 NPP\_CUDA\_1\_2 = 120,
 NPP\_CUDA\_1\_3 = 130,
```

```
NPP_CUDA_2_0 = 200,
 NPP\_CUDA\_2\_1 = 210,
 NPP_CUDA_3_0 = 300,
 NPP_CUDA_3_2 = 320,
 NPP\_CUDA\_3\_5 = 350,
 NPP\_CUDA\_3\_7 = 370,
 NPP\_CUDA\_5\_0 = 500,
 NPP_CUDA_5_2 = 520,
 NPP\_CUDA\_5\_3 = 530,
 NPP_CUDA_6_0 = 600,
 NPP\_CUDA\_6\_1 = 610,
 NPP_CUDA_6_2 = 620,
 NPP\_CUDA\_6\_3 = 630,
 NPP_CUDA_7_0 = 700 }
enum NppiAxis {
 NPP_HORIZONTAL_AXIS,
 NPP_VERTICAL_AXIS,
 NPP BOTH AXIS }
• enum NppCmpOp {
 NPP_CMP_LESS,
 NPP_CMP_LESS_EQ,
 NPP_CMP_EQ,
 NPP_CMP_GREATER_EQ,
 NPP_CMP_GREATER }
• enum NppRoundMode {
 NPP_RND_NEAR,
 NPP_ROUND_NEAREST_TIES_TO_EVEN = NPP_RND_NEAR,
 NPP_RND_FINANCIAL,
 NPP_ROUND_NEAREST_TIES_AWAY_FROM_ZERO = NPP_RND_FINANCIAL,
 NPP_RND_ZERO,
 NPP_ROUND_TOWARD_ZERO = NPP_RND_ZERO }
    Rounding Modes.
• enum NppiBorderType {
 NPP\_BORDER\_UNDEFINED = 0,
 NPP_BORDER_NONE = NPP_BORDER_UNDEFINED,
 NPP_BORDER_CONSTANT = 1,
 NPP_BORDER_REPLICATE = 2,
 NPP\_BORDER\_WRAP = 3,
 NPP_BORDER_MIRROR = 4 }
```

```
• enum NppHintAlgorithm {
 NPP_ALG_HINT_NONE,
 NPP_ALG_HINT_FAST,
 NPP_ALG_HINT_ACCURATE }
• enum NppiAlphaOp {
 NPPI_OP_ALPHA_OVER,
 NPPI_OP_ALPHA_IN,
 NPPI_OP_ALPHA_OUT,
 NPPI_OP_ALPHA_ATOP,
 NPPI_OP_ALPHA_XOR,
 NPPI_OP_ALPHA_PLUS,
 NPPI_OP_ALPHA_OVER_PREMUL,
 NPPI_OP_ALPHA_IN_PREMUL,
 NPPI_OP_ALPHA_OUT_PREMUL,
 NPPI_OP_ALPHA_ATOP_PREMUL,
 NPPI_OP_ALPHA_XOR_PREMUL,
 NPPI_OP_ALPHA_PLUS_PREMUL,
 NPPI OP ALPHA PREMUL }
• enum NppsZCType {
 nppZCR,
 nppZCXor,
 nppZCC }
• enum NppiHuffmanTableType {
 nppiDCTable,
 nppiACTable }
• enum NppiNorm {
 nppiNormInf = 0,
 nppiNormL1 = 1,
 nppiNormL2 = 2 }
```

## 7.2.1 Define Documentation

## 7.2.1.1 #define NPP\_HOG\_MAX\_BINS\_PER\_CELL (16)

max number of histogram bins.

## 7.2.1.2 #define NPP\_HOG\_MAX\_BLOCK\_SIZE (64)

max horizontal/vertical pixel size of block.

## 7.2.1.3 #define NPP\_HOG\_MAX\_CELL\_SIZE (16)

max horizontal/vertical pixel size of cell.

### 7.2.1.4 #define NPP\_HOG\_MAX\_CELLS\_PER\_DESCRIPTOR (256)

max number of cells in a descriptor window.

## 7.2.1.5 #define NPP\_HOG\_MAX\_DESCRIPTOR\_LOCATIONS\_PER\_CALL (128)

max number of descriptor window locations per function call.

### 7.2.1.6 #define NPP\_HOG\_MAX\_OVERLAPPING\_BLOCKS\_PER\_DESCRIPTOR (256)

max number of overlapping blocks in a descriptor window.

## 7.2.1.7 #define NPP\_MAX\_16S ( 32767 )

Maximum 16-bit signed integer.

## 7.2.1.8 #define NPP\_MAX\_16U ( 65535 )

Maximum 16-bit unsigned integer.

## 7.2.1.9 #define NPP\_MAX\_32S ( 2147483647 )

Maximum 32-bit signed integer.

## 7.2.1.10 #define NPP\_MAX\_32U ( 4294967295U )

Maximum 32-bit unsigned integer.

## $7.2.1.11 \quad \text{\#define NPP\_MAX\_64S} \ (\ 9223372036854775807LL \ )$

Maximum 64-bit signed integer.

## 7.2.1.12 #define NPP\_MAX\_64U ( 18446744073709551615ULL )

Maximum 64-bit unsigned integer.

## 7.2.1.13 #define NPP\_MAX\_8S ( 127 )

Maximum 8-bit signed integer.

## 7.2.1.14 #define NPP\_MAX\_8U ( 255 )

Maximum 8-bit unsigned integer.

## 7.2.1.15 #define NPP\_MAXABS\_32F ( 3.402823466e+38f )

Largest positive 32-bit floating point value.

## 7.2.1.16 #define NPP\_MAXABS\_64F ( 1.7976931348623158e+308 )

Largest positive 64-bit floating point value.

## 7.2.1.17 #define NPP\_MIN\_16S (-32767 - 1)

Minimum 16-bit signed integer.

## 7.2.1.18 #define NPP MIN 16U(0)

Minimum 16-bit unsigned integer.

## 7.2.1.19 #define NPP\_MIN\_32S (-2147483647 - 1 )

Minimum 32-bit signed integer.

## 7.2.1.20 #define NPP\_MIN\_32U ( 0 )

Minimum 32-bit unsigned integer.

## 7.2.1.21 #define NPP\_MIN\_64S (-9223372036854775807LL - 1)

Minimum 64-bit signed integer.

## 7.2.1.22 #define NPP\_MIN\_64U ( 0 )

Minimum 64-bit unsigned integer.

## 7.2.1.23 #define NPP\_MIN\_8S (-127 - 1)

Minimum 8-bit signed integer.

## **7.2.1.24** #define NPP\_MIN\_8U ( 0 )

Minimum 8-bit unsigned integer.

## 7.2.1.25 #define NPP\_MINABS\_32F ( 1.175494351e-38f )

Smallest positive 32-bit floating point value.

Copyright ©2009-2017 NVIDIA Corporation

### 7.2.1.26 #define NPP\_MINABS\_64F ( 2.2250738585072014e-308 )

Smallest positive 64-bit floating point value.

## 7.2.2 Enumeration Type Documentation

## 7.2.2.1 enum NppCmpOp

#### **Enumerator:**

NPP\_CMP\_LESS
NPP\_CMP\_LESS\_EQ
NPP\_CMP\_EQ
NPP\_CMP\_GREATER\_EQ
NPP\_CMP\_GREATER

## 7.2.2.2 enum NppGpuComputeCapability

#### **Enumerator:**

NPP\_CUDA\_UNKNOWN\_VERSION Indicates that the compute-capability query failed. NPP\_CUDA\_NOT\_CAPABLE Indicates that no CUDA capable device was found. NPP\_CUDA\_1\_0 Indicates that CUDA 1.0 capable device is machine's default device. NPP CUDA 1 1 Indicates that CUDA 1.1 capable device is machine's default device. NPP CUDA 1 2 Indicates that CUDA 1.2 capable device is machine's default device. NPP\_CUDA\_1\_3 Indicates that CUDA 1.3 capable device is machine's default device. **NPP\_CUDA\_2\_0** Indicates that CUDA 2.0 capable device is machine's default device. NPP\_CUDA\_2\_1 Indicates that CUDA 2.1 capable device is machine's default device. **NPP CUDA 3 0** Indicates that CUDA 3.0 capable device is machine's default device. NPP CUDA 3 2 Indicates that CUDA 3.2 capable device is machine's default device. NPP\_CUDA\_3\_5 Indicates that CUDA 3.5 capable device is machine's default device. NPP CUDA 3 7 Indicates that CUDA 3.7 capable device is machine's default device. NPP\_CUDA\_5\_0 Indicates that CUDA 5.0 capable device is machine's default device. NPP\_CUDA\_5\_2 Indicates that CUDA 5.2 capable device is machine's default device. NPP CUDA 5 3 Indicates that CUDA 5.3 capable device is machine's default device. **NPP\_CUDA\_6\_0** Indicates that CUDA 6.0 capable device is machine's default device. NPP CUDA 6 1 Indicates that CUDA 6.1 capable device is machine's default device. NPP\_CUDA\_6\_2 Indicates that CUDA 6.2 capable device is machine's default device. NPP\_CUDA\_6\_3 Indicates that CUDA 6.3 capable device is machine's default device. NPP\_CUDA\_7\_0 Indicates that CUDA 7.0 or better is machine's default device.

## 7.2.2.3 enum NppHintAlgorithm

#### **Enumerator:**

NPP\_ALG\_HINT\_NONE

NPP\_ALG\_HINT\_FAST

NPP\_ALG\_HINT\_ACCURATE

## 7.2.2.4 enum NppiAlphaOp

### **Enumerator:**

NPPI\_OP\_ALPHA\_OVER

NPPI\_OP\_ALPHA\_IN

NPPI\_OP\_ALPHA\_OUT

NPPI\_OP\_ALPHA\_ATOP

NPPI\_OP\_ALPHA\_XOR

NPPI\_OP\_ALPHA\_PLUS

NPPI\_OP\_ALPHA\_OVER\_PREMUL

NPPI\_OP\_ALPHA\_IN\_PREMUL

NPPI\_OP\_ALPHA\_OUT\_PREMUL

NPPI\_OP\_ALPHA\_ATOP\_PREMUL

NPPI\_OP\_ALPHA\_XOR\_PREMUL

NPPI\_OP\_ALPHA\_XOR\_PREMUL

NPPI\_OP\_ALPHA\_PLUS\_PREMUL

NPPI\_OP\_ALPHA\_PLUS\_PREMUL

NPPI\_OP\_ALPHA\_PREMUL

## 7.2.2.5 enum NppiAxis

## **Enumerator:**

NPP\_HORIZONTAL\_AXIS

NPP\_VERTICAL\_AXIS

NPP\_BOTH\_AXIS

## 7.2.2.6 enum NppiBayerGridPosition

Bayer Grid Position Registration.

## **Enumerator:**

NPPI\_BAYER\_BGGR Default registration position.

NPPI\_BAYER\_RGGB

NPPI\_BAYER\_GBRG

NPPI\_BAYER\_GRBG

### 7.2.2.7 enum NppiBorderType

#### **Enumerator:**

NPP\_BORDER\_UNDEFINED
NPP\_BORDER\_NONE
NPP\_BORDER\_CONSTANT
NPP\_BORDER\_REPLICATE
NPP\_BORDER\_WRAP
NPP\_BORDER\_MIRROR

## 7.2.2.8 enum NppiDifferentialKernel

Differential Filter types.

### **Enumerator:**

```
NPP_FILTER_SOBEL
NPP_FILTER_SCHARR
```

## 7.2.2.9 enum NppiHuffmanTableType

#### **Enumerator:**

```
nppiDCTable DC Table.nppiACTable AC Table.
```

## 7.2.2.10 enum NppiInterpolationMode

Filtering methods.

#### **Enumerator:**

```
NPPI_INTER_UNDEFINED

NPPI_INTER_NN Nearest neighbor filtering.

NPPI_INTER_LINEAR Linear interpolation.

NPPI_INTER_CUBIC Cubic interpolation.

NPPI_INTER_CUBIC2P_BSPLINE Two-parameter cubic filter (B=1, C=0).

NPPI_INTER_CUBIC2P_CATMULLROM Two-parameter cubic filter (B=0, C=1/2).

NPPI_INTER_CUBIC2P_B05C03 Two-parameter cubic filter (B=1/2, C=3/10).

NPPI_INTER_SUPER Super sampling.

NPPI_INTER_LANCZOS Lanczos filtering.

NPPI_INTER_LANCZOS3_ADVANCED Generic Lanczos filtering with order 3.

NPPI_SMOOTH_EDGE Smooth edge filtering.
```

## 7.2.2.11 enum NppiMaskSize

Fixed filter-kernel sizes.

#### **Enumerator:**

```
NPP_MASK_SIZE_1_X_3
NPP_MASK_SIZE_1_X_5
NPP_MASK_SIZE_3_X_1
NPP_MASK_SIZE_5_X_1
NPP_MASK_SIZE_3_X_3
NPP_MASK_SIZE_5_X_5
NPP_MASK_SIZE_7_X_7
NPP_MASK_SIZE_9_X_9
NPP_MASK_SIZE_11_X_11
NPP_MASK_SIZE_11_X_11
NPP_MASK_SIZE_13_X_13
NPP_MASK_SIZE_15_X_15
```

## 7.2.2.12 enum NppiNorm

#### **Enumerator:**

```
nppiNormInf maximumnppiNormL1 sumnppiNormL2 square root of sum of squares
```

## 7.2.2.13 enum NppRoundMode

Rounding Modes.

The enumerated rounding modes are used by a large number of NPP primitives to allow the user to specify the method by which fractional values are converted to integer values. Also see Rounding Modes.

For NPP release 5.5 new names for the three rounding modes are introduced that are based on the naming conventions for rounding modes set forth in the IEEE-754 floating-point standard. Developers are encouraged to use the new, longer names to be future proof as the legacy names will be deprecated in subsequent NPP releases.

#### **Enumerator:**

NPP\_RND\_NEAR Round to the nearest even integer.

All fractional numbers are rounded to their nearest integer. The ambiguous cases (i.e. <integer>.5) are rounded to the closest even integer. E.g.

- roundNear(0.5) = 0
- roundNear(0.6) = 1
- roundNear(1.5) = 2
- roundNear(-1.5) = -2

NPP\_ROUND\_NEAREST\_TIES\_TO\_EVEN Alias name for NPP\_RND\_NEAR.

**NPP\_RND\_FINANCIAL** Round according to financial rule.

All fractional numbers are rounded to their nearest integer. The ambiguous cases (i.e. <integer>.5) are rounded away from zero. E.g.

- roundFinancial(0.4) = 0
- roundFinancial(0.5) = 1
- roundFinancial(-1.5) = -2

NPP\_ROUND\_NEAREST\_TIES\_AWAY\_FROM\_ZERO Alias name for NPP\_RND\_-FINANCIAL.

NPP\_RND\_ZERO Round towards zero (truncation).

All fractional numbers of the form <integer>.<decimals> are truncated to <integer>.

- roundZero(1.5) = 1
- roundZero(1.9) = 1
- roundZero(-2.5) = -2

NPP\_ROUND\_TOWARD\_ZERO Alias name for NPP\_RND\_ZERO.

## 7.2.2.14 enum NppStatus

Error Status Codes.

Almost all NPP function return error-status information using these return codes. Negative return codes indicate errors, positive return codes indicate warnings, a return code of 0 indicates success.

## **Enumerator:**

NPP\_NOT\_SUPPORTED\_MODE\_ERROR

NPP\_INVALID\_HOST\_POINTER\_ERROR

NPP\_INVALID\_DEVICE\_POINTER\_ERROR

NPP LUT PALETTE BITSIZE ERROR

NPP\_ZC\_MODE\_NOT\_SUPPORTED\_ERROR ZeroCrossing mode not supported.

NPP\_NOT\_SUFFICIENT\_COMPUTE\_CAPABILITY

NPP\_TEXTURE\_BIND\_ERROR

NPP\_WRONG\_INTERSECTION\_ROI\_ERROR

NPP\_HAAR\_CLASSIFIER\_PIXEL\_MATCH\_ERROR

NPP\_MEMFREE\_ERROR

NPP\_MEMSET\_ERROR

NPP\_MEMCPY\_ERROR

NPP ALIGNMENT ERROR

NPP\_CUDA\_KERNEL\_EXECUTION\_ERROR

NPP\_ROUND\_MODE\_NOT\_SUPPORTED\_ERROR Unsupported round mode.

NPP\_QUALITY\_INDEX\_ERROR Image pixels are constant for quality index.

NPP\_RESIZE\_NO\_OPERATION\_ERROR One of the output image dimensions is less than 1 pixel.

NPP\_OVERFLOW\_ERROR Number overflows the upper or lower limit of the data type.

NPP\_NOT\_EVEN\_STEP\_ERROR Step value is not pixel multiple.

**NPP\_HISTOGRAM\_NUMBER\_OF\_LEVELS\_ERROR** Number of levels for histogram is less than 2.

NPP\_LUT\_NUMBER\_OF\_LEVELS\_ERROR Number of levels for LUT is less than 2.

NPP\_CORRUPTED\_DATA\_ERROR Processed data is corrupted.

NPP\_CHANNEL\_ORDER\_ERROR Wrong order of the destination channels.

NPP\_ZERO\_MASK\_VALUE\_ERROR All values of the mask are zero.

NPP\_QUADRANGLE\_ERROR The quadrangle is nonconvex or degenerates into triangle, line or point.

NPP\_RECTANGLE\_ERROR Size of the rectangle region is less than or equal to 1.

NPP\_COEFFICIENT\_ERROR Unallowable values of the transformation coefficients.

NPP\_NUMBER\_OF\_CHANNELS\_ERROR Bad or unsupported number of channels.

NPP\_COI\_ERROR Channel of interest is not 1, 2, or 3.

NPP\_DIVISOR\_ERROR Divisor is equal to zero.

NPP\_CHANNEL\_ERROR Illegal channel index.

**NPP\_STRIDE\_ERROR** Stride is less than the row length.

NPP\_ANCHOR\_ERROR Anchor point is outside mask.

NPP\_MASK\_SIZE\_ERROR Lower bound is larger than upper bound.

NPP RESIZE FACTOR ERROR

NPP INTERPOLATION ERROR

NPP\_MIRROR\_FLIP\_ERROR

NPP\_MOMENT\_00\_ZERO\_ERROR

NPP\_THRESHOLD\_NEGATIVE\_LEVEL\_ERROR

NPP\_THRESHOLD\_ERROR

NPP\_CONTEXT\_MATCH\_ERROR

NPP\_FFT\_FLAG\_ERROR

NPP\_FFT\_ORDER\_ERROR

NPP\_STEP\_ERROR Step is less or equal zero.

NPP\_SCALE\_RANGE\_ERROR

NPP\_DATA\_TYPE\_ERROR

NPP\_OUT\_OFF\_RANGE\_ERROR

NPP\_DIVIDE\_BY\_ZERO\_ERROR

NPP MEMORY ALLOCATION ERR

NPP\_NULL\_POINTER\_ERROR

NPP\_RANGE\_ERROR

NPP\_SIZE\_ERROR

NPP\_BAD\_ARGUMENT\_ERROR

NPP\_NO\_MEMORY\_ERROR

NPP\_NOT\_IMPLEMENTED\_ERROR

NPP ERROR

NPP ERROR RESERVED

**NPP\_NO\_ERROR** Error free operation.

**NPP\_SUCCESS** Successful operation (same as NPP\_NO\_ERROR).

NPP\_NO\_OPERATION\_WARNING Indicates that no operation was performed.

**NPP\_DIVIDE\_BY\_ZERO\_WARNING** Divisor is zero however does not terminate the execution.

**NPP\_AFFINE\_QUAD\_INCORRECT\_WARNING** Indicates that the quadrangle passed to one of affine warping functions doesn't have necessary properties.

First 3 vertices are used, the fourth vertex discarded.

NPP\_WRONG\_INTERSECTION\_ROI\_WARNING The given ROI has no interestion with either the source or destination ROI.

Thus no operation was performed.

NPP\_WRONG\_INTERSECTION\_QUAD\_WARNING The given quadrangle has no intersection with either the source or destination ROI.

Thus no operation was performed.

NPP\_DOUBLE\_SIZE\_WARNING Image size isn't multiple of two.

Indicates that in case of 422/411/420 sampling the ROI width/height was modified for proper processing.

NPP\_MISALIGNED\_DST\_ROI\_WARNING Speed reduction due to uncoalesced memory accesses warning.

## 7.2.2.15 enum NppsZCType

### **Enumerator:**

nppZCR sign change
nppZCXor sign change XOR
nppZCC sign change count\_0

## 7.3 Basic NPP Data Types

## **Data Structures**

• struct NPP\_ALIGN\_8

Complex Number This struct represents an unsigned int complex number.

• struct NPP\_ALIGN\_16

Complex Number This struct represents a long long complex number.

## **Typedefs**

• typedef unsigned char Npp8u 8-bit unsigned chars

• typedef signed char Npp8s 8-bit signed chars

• typedef unsigned short Npp16u

16-bit unsigned integers

• typedef short Npp16s

16-bit signed integers

• typedef unsigned int Npp32u 32-bit unsigned integers

• typedef int Npp32s

32-bit signed integers

• typedef unsigned long long Npp64u 64-bit unsigned integers

• typedef long long Npp64s 64-bit signed integers

• typedef float Npp32f

32-bit (IEEE) floating-point numbers

• typedef double Npp64f
64-bit floating-point numbers

• typedef struct NPP\_ALIGN\_8 Npp32uc

Complex Number This struct represents an unsigned int complex number.

• typedef struct NPP\_ALIGN\_8 Npp32sc

Complex Number This struct represents a signed int complex number.

• typedef struct NPP\_ALIGN\_8 Npp32fc

Complex Number This struct represents a single floating-point complex number.

• typedef struct NPP\_ALIGN\_16 Npp64sc

Complex Number This struct represents a long long complex number.

• typedef struct NPP\_ALIGN\_16 Npp64fc

Complex Number This struct represents a double floating-point complex number.

## **Functions**

• struct \_\_align\_\_ (2)

Complex Number This struct represents an unsigned char complex number.

• struct \_\_align\_\_ (4)

Complex Number This struct represents an unsigned short complex number.

## **Variables**

- Npp8uc
- Npp16uc
- Npp16sc

## 7.3.1 Typedef Documentation

## 7.3.1.1 typedef short Npp16s

16-bit signed integers

## 7.3.1.2 typedef unsigned short Npp16u

16-bit unsigned integers

## 7.3.1.3 typedef float Npp32f

32-bit (IEEE) floating-point numbers

## 7.3.1.4 typedef struct NPP\_ALIGN\_8 Npp32fc

Complex Number This struct represents a single floating-point complex number.

## 7.3.1.5 typedef int Npp32s

32-bit signed integers

## 7.3.1.6 typedef struct NPP\_ALIGN\_8 Npp32sc

Complex Number This struct represents a signed int complex number.

## 7.3.1.7 typedef unsigned int Npp32u

32-bit unsigned integers

## 7.3.1.8 typedef struct NPP\_ALIGN\_8 Npp32uc

Complex Number This struct represents an unsigned int complex number.

## 7.3.1.9 typedef double Npp64f

64-bit floating-point numbers

## 7.3.1.10 typedef struct NPP\_ALIGN\_16 Npp64fc

Complex Number This struct represents a double floating-point complex number.

## 7.3.1.11 typedef long long Npp64s

64-bit signed integers

## 7.3.1.12 typedef struct NPP\_ALIGN\_16 Npp64sc

Complex Number This struct represents a long long complex number.

## 7.3.1.13 typedef unsigned long long Npp64u

64-bit unsigned integers

## 7.3.1.14 typedef signed char Npp8s

8-bit signed chars

## 7.3.1.15 typedef unsigned char Npp8u

8-bit unsigned chars

### 7.3.2 Function Documentation

## **7.3.2.1 struct \_\_align\_\_ (4)** [read]

Complex Number This struct represents an unsigned short complex number.

Complex Number This struct represents a short complex number.

- < Real part
- < Imaginary part
- < Real part
- < Imaginary part

## **7.3.2.2 struct \_\_align\_\_(2)** [read]

Complex Number This struct represents an unsigned char complex number.

- < Real part
- < Imaginary part

## 7.3.3 Variable Documentation

- 7.3.3.1 Npp16sc
- 7.3.3.2 Npp16uc
- 7.3.3.3 Npp8uc

7.4 Compression 51

## 7.4 Compression

Image compression primitives.

## **Modules**

• Quantization Functions

## **Typedefs**

- typedef struct NppiDecodeHuffmanSpec NppiDecodeHuffmanSpec
- typedef struct NppiEncodeHuffmanSpec NppiEncodeHuffmanSpec

## **Functions**

• NppStatus nppiDecodeHuffmanSpecGetBufSize\_JPEG (int \*pSize)

Returns the length of the NppiDecodeHuffmanSpec structure.

NppStatus nppiDecodeHuffmanSpecInitHost\_JPEG (const Npp8u \*pRawHuffmanTable, NppiHuffmanTableType eTableType, NppiDecodeHuffmanSpec \*pHuffmanSpec)

Creates a Huffman table in a format that is suitable for the decoder on the host.

 NppStatus nppiDecodeHuffmanSpecInitAllocHost\_JPEG (const Npp8u \*pRawHuffmanTable, NppiHuffmanTableType eTableType, NppiDecodeHuffmanSpec \*\*ppHuffmanSpec)

Allocates memory and creates a Huffman table in a format that is suitable for the decoder on the host.

NppStatus nppiDecodeHuffmanSpecFreeHost\_JPEG (NppiDecodeHuffmanSpec \*pHuffmanSpec)

Frees the host memory allocated by nppiDecodeHuffmanSpecInitAllocHost\_JPEG.

NppStatus nppiDecodeHuffmanScanHost\_JPEG\_8u16s\_P1R (const Npp8u \*pSrc, Npp32s nLength, Npp32s restartInterval, Npp32s Ss, Npp32s Se, Npp32s Ah, Npp32s Al, Npp16s \*pDst, Npp32s nDstStep, NppiDecodeHuffmanSpec \*pHuffmanTableDC, NppiDecodeHuffmanSpec \*pHuffmanTableAC, NppiSize oSizeROI)

Huffman Decoding of the JPEG decoding on the host.

NppStatus nppiDecodeHuffmanScanHost\_JPEG\_8u16s\_P3R (const Npp8u \*pSrc, Npp32s nLength, Npp32s nRestartInterval, Npp32s nSs, Npp32s nSe, Npp32s nAh, Npp32s nAl, Npp16s \*apDst[3], Npp32s aDstStep[3], NppiDecodeHuffmanSpec \*apHuffmanACTable[3], NppiDecodeHuffmanSpec \*apHuffmanACTable[3], NppiSize aSizeROI[3])

Huffman Decoding of the JPEG decoding on the host.

• NppStatus nppiEncodeHuffmanSpecGetBufSize\_JPEG (int \*pSize)

Returns the length of the NppiEncodeHuffmanSpec structure.

NppStatus nppiEncodeHuffmanSpecInit\_JPEG (const Npp8u \*pRawHuffmanTable, NppiHuffmanTableType eTableType, NppiEncodeHuffmanSpec \*pHuffmanSpec)

Creates a Huffman table in a format that is suitable for the encoder.

• NppStatus nppiEncodeHuffmanSpecInitAlloc\_JPEG (const Npp8u \*pRawHuffmanTable, Nppi-HuffmanTableType eTableType, NppiEncodeHuffmanSpec \*\*ppHuffmanSpec)

Allocates memory and creates a Huffman table in a format that is suitable for the encoder.

• NppStatus nppiEncodeHuffmanSpecFree\_JPEG (NppiEncodeHuffmanSpec \*pHuffmanSpec)

Frees the memory allocated by nppiEncodeHuffmanSpecInitAlloc\_JPEG.

NppStatus nppiEncodeHuffmanScan\_JPEG\_8u16s\_P1R (const Npp16s \*pSrc, Npp32s nSrc-Step, Npp32s nRestartInterval, Npp32s nSs, Npp32s nSe, Npp32s nAh, Npp32s nAl, Npp8u \*pDst, Npp32s \*nLength, NppiEncodeHuffmanSpec \*pHuffmanTableDC, NppiEncodeHuffmanSpec \*pHuffmanTableAC, NppiSize oSizeROI, Npp8u \*pTempStorage)

Huffman Encoding of the JPEG Encoding.

NppStatus nppiEncodeHuffmanScan\_JPEG\_8u16s\_P3R (Npp16s \*apSrc[3], Npp32s aSrcStep[3], Npp32s nRestartInterval, Npp32s nSs, Npp32s nSe, Npp32s nAh, Npp32s nAl, Npp8u \*pDst, Npp32s \*nLength, NppiEncodeHuffmanSpec \*apHuffmanDCTable[3], NppiEncodeHuffmanSpec \*apHuffmanACTable[3], NppiSize aSizeROI[3], Npp8u \*pTempStorage)

Huffman Encoding of the JPEG Encoding.

NppStatus nppiEncodeOptimizeHuffmanScan\_JPEG\_8u16s\_P1R (const Npp16s \*pSrc, Npp32s nSrcStep, Npp32s nRestartInterval, Npp32s nSs, Npp32s nSe, Npp32s nAh, Npp32s nAl, Npp8u \*pDst, Npp32s \*pLength, Npp8u \*hpCodesDC, Npp8u \*hpTableDC, Npp8u \*hpCodesAC, Npp8u \*hpTableAC, NppiEncodeHuffmanSpec \*pHuffmanDCTable, NppiEncodeHuffmanSpec \*pHuffmanACTable, NppiSize oSizeROI, Npp8u \*pTempStorage)

Optimize Huffman Encoding of the JPEG Encoding.

NppStatus nppiEncodeOptimizeHuffmanScan\_JPEG\_8u16s\_P3R (Npp16s \*apSrc[3], Npp32s aSrcStep[3], Npp32s nRestartInterval, Npp32s nSs, Npp32s nSe, Npp32s nAh, Npp32s nAl, Npp8u \*pDst, Npp32s \*pLength, Npp8u \*hpCodesDC[3], Npp8u \*hpTableDC[3], Npp8u \*hpCodesAC[3], Npp8u \*hpTableAC[3], NppiEncodeHuffmanSpec \*apHuffmanDCTable[3], NppiEncodeHuffmanSpec \*apHuffmanACTable[3], NppiSize oSizeROI[3], Npp8u \*pTempStorage)

Optimize Huffman Encoding of the JPEG Encoding.

- NppStatus nppiEncodeHuffmanGetSize (NppiSize oSize, int nChannels, size\_t \*pBufSize)

  Calculates the size of the temporary buffer for baseline Huffman encoding.
- NppStatus nppiEncodeOptimizeHuffmanGetSize (NppiSize oSize, int nChannels, int \*pBufSize)

  Calculates the size of the temporary buffer for optimize Huffman coding.

## 7.4.1 Detailed Description

Image compression primitives.

The JPEG standard defines a flow of level shift, DCT and quantization for forward JPEG transform and inverse level shift, IDCT and de-quantization for inverse JPEG transform. This group has the functions for both forward and inverse functions.

These functions can be found in the nppicom library. Linking to only the sub-libraries that you use can significantly save link time, application load time, and CUDA runtime startup time when using dynamic libraries.

7.4 Compression 53

## 7.4.2 Typedef Documentation

- 7.4.2.1 typedef struct NppiDecodeHuffmanSpec NppiDecodeHuffmanSpec
- 7.4.2.2 typedef struct NppiEncodeHuffmanSpec NppiEncodeHuffmanSpec

### 7.4.3 Function Documentation

7.4.3.1 NppStatus nppiDecodeHuffmanScanHost\_JPEG\_8u16s\_P1R (const Npp8u \* pSrc, Npp32s nLength, Npp32s restartInterval, Npp32s Ss, Npp32s Se, Npp32s Ah, Npp32s Al, Npp16s \* pDst, Npp32s nDstStep, NppiDecodeHuffmanSpec \* pHuffmanTableDC, NppiDecodeHuffmanSpec \* pHuffmanTableAC, NppiSize oSizeROI)

Huffman Decoding of the JPEG decoding on the host.

Input is expected in byte stuffed huffman encoded JPEG scan and output is expected to be 64x1 macro blocks.

#### **Parameters:**

```
pSrc Byte-stuffed huffman encoded JPEG scan.
nLength Byte length of the input.
restartInterval Restart Interval, see JPEG standard.
Ss Start Coefficient, see JPEG standard.
Se End Coefficient, see JPEG standard.
Ah Bit Approximation High, see JPEG standard.
Al Bit Approximation Low, see JPEG standard.
pDst Destination-Image Pointer.
nDstStep Destination-Image Line Step.
pHuffmanTableDC DC Huffman table.
pHuffmanTableAC AC Huffman table.
oSizeROI Region-of-Interest (ROI).
```

## **Returns:**

Error codes:

- NPP\_SIZE\_ERROR For negative input height/width or not a multiple of 8 width/height.
- NPP\_STEP\_ERROR If input image width is not multiple of 8 or does not match ROI.
- NPP\_NULL\_POINTER\_ERROR If the destination pointer is 0.
- 7.4.3.2 NppStatus nppiDecodeHuffmanScanHost\_JPEG\_8u16s\_P3R (const Npp8u \* pSrc, Npp32s nLength, Npp32s nRestartInterval, Npp32s nSs, Npp32s nSe, Npp32s nAh, Npp32s nAl, Npp16s \* apDst[3], Npp32s aDstStep[3], NppiDecodeHuffmanSpec \* apHuffmanDCTable[3], NppiDecodeHuffmanSpec \* apHuffmanACTable[3], NppiSize aSizeROI[3])

Huffman Decoding of the JPEG decoding on the host.

Input is expected in byte stuffed huffman encoded JPEG scan and output is expected to be 64x1 macro blocks.

#### **Parameters:**

```
pSrc Byte-stuffed huffman encoded JPEG scan.
nLength Byte length of the input.
nRestartInterval Restart Interval, see JPEG standard.
nSs Start Coefficient, see JPEG standard.
nSe End Coefficient, see JPEG standard.
nAh Bit Approximation High, see JPEG standard.
nAl Bit Approximation Low, see JPEG standard.
apDst Destination-Image Pointer.
aDstStep Destination-Image Line Step.
apHuffmanDCTable DC Huffman tables.
apHuffmanACTable AC Huffman tables.
aSizeROI Region-of-Interest (ROI).
```

#### **Returns:**

Error codes:

- NPP\_SIZE\_ERROR For negative input height/width or not a multiple of 8 width/height.
- NPP\_STEP\_ERROR If input image width is not multiple of 8 or does not match ROI.
- NPP\_NULL\_POINTER\_ERROR If the destination pointer is 0.

## 7.4.3.3 NppStatus nppiDecodeHuffmanSpecFreeHost\_JPEG (NppiDecodeHuffmanSpec \* pHuffmanSpec)

Frees the host memory allocated by nppiDecodeHuffmanSpecInitAllocHost\_JPEG.

#### **Parameters:**

pHuffmanSpec Pointer to the Huffman table for the decoder

## 7.4.3.4 NppStatus nppiDecodeHuffmanSpecGetBufSize\_JPEG (int \* pSize)

Returns the length of the NppiDecodeHuffmanSpec structure.

## **Parameters:**

pSize Pointer to a variable that will receive the length of the NppiDecodeHuffmanSpec structure.

## **Returns:**

Error codes:

• NPP\_NULL\_POINTER\_ERROR If one of the pointers is 0.

7.4 Compression 55

# 7.4.3.5 NppStatus nppiDecodeHuffmanSpecInitAllocHost\_JPEG (const Npp8u \* pRawHuffmanTable, NppiHuffmanTableType eTableType, NppiDecodeHuffmanSpec \*\* ppHuffmanSpec)

Allocates memory and creates a Huffman table in a format that is suitable for the decoder on the host.

#### **Parameters:**

```
pRawHuffmanTable Huffman table formated as specified in the JPEG standard.eTableType Enum specifying type of table (nppiDCTable or nppiACTable).ppHuffmanSpec Pointer to returned pointer to the Huffman table for the decoder
```

#### **Returns:**

Error codes:

• NPP\_NULL\_POINTER\_ERROR If one of the pointers is 0.

## 7.4.3.6 NppStatus nppiDecodeHuffmanSpecInitHost\_JPEG (const Npp8u \* pRawHuffmanTable, NppiHuffmanTableType eTableType, NppiDecodeHuffmanSpec \* pHuffmanSpec)

Creates a Huffman table in a format that is suitable for the decoder on the host.

## **Parameters:**

```
pRawHuffmanTable Huffman table formated as specified in the JPEG standard.eTableType Enum specifying type of table (nppiDCTable or nppiACTable).pHuffmanSpec Pointer to the Huffman table for the decoder
```

## **Returns:**

Error codes:

• NPP\_NULL\_POINTER\_ERROR If one of the pointers is 0.

## 7.4.3.7 NppStatus nppiEncodeHuffmanGetSize (NppiSize oSize, int nChannels, size\_t \* pBufSize)

Calculates the size of the temporary buffer for baseline Huffman encoding.

#### See also:

```
nppiEncodeHuffmanScan_JPEG_8u16s_P1R(), nppiEncodeHuffmanScan_JPEG_8u16s_P3R().
```

## **Parameters:**

```
oSize Image Dimension.nChannels Number of channels in the image.pBufSize Pointer to variable that returns the size of the temporary buffer.
```

## **Returns:**

NPP\_SUCCESS Indicates no error. Any other value indicates an error or a warning NPP\_SIZE\_ERROR Indicates an error condition if any image dimension has zero or negative value NPP\_NULL\_POINTER\_ERROR Indicates an error condition if pBufSize pointer is NULL

7.4.3.8 NppStatus nppiEncodeHuffmanScan\_JPEG\_8u16s\_P1R (const Npp16s \* pSrc, Npp32s nSrcStep, Npp32s nRestartInterval, Npp32s nSs, Npp32s nSe, Npp32s nAh, Npp32s nAl, Npp8u \* pDst, Npp32s \* nLength, NppiEncodeHuffmanSpec \* pHuffmanTableDC, NppiEncodeHuffmanSpec \* pHuffmanTableAC, NppiSize oSizeROI, Npp8u \* pTempStorage)

Huffman Encoding of the JPEG Encoding.

Input is expected to be 64x1 macro blocks and output is expected as byte stuffed huffman encoded JPEG scan.

#### **Parameters:**

```
pSrc Destination-Image Pointer.
nSrcStep Destination-Image Line Step.
nRestartInterval Restart Interval, see JPEG standard. Currently only values <=0 are supported.
nSs Start Coefficient, see JPEG standard.
nSe End Coefficient, see JPEG standard.
nAh Bit Approximation High, see JPEG standard.
nAl Bit Approximation Low, see JPEG standard.
pDst Byte-stuffed huffman encoded JPEG scan.
nLength Byte length of the huffman encoded JPEG scan.
pHuffmanTableDC DC Huffman table.
pHuffmanTableAC AC Huffman table.
oSizeROI Region-of-Interest (ROI).
pTempStorage Temporary storage.</pre>
```

## **Returns:**

## Error codes:

- NPP\_SIZE\_ERROR For negative input height/width or not a multiple of 8 width/height.
- NPP\_STEP\_ERROR If input image width is not multiple of 8 or does not match ROI.
- NPP NULL POINTER ERROR If the destination pointer is 0.
- NPP\_NOT\_SUFFICIENT\_COMPUTE\_CAPABILITY If the device has compute capability < 2.0.
- 7.4.3.9 NppStatus nppiEncodeHuffmanScan\_JPEG\_8u16s\_P3R (Npp16s \* apSrc[3], Npp32s aSrcStep[3], Npp32s nRestartInterval, Npp32s nSs, Npp32s nSe, Npp32s nAh, Npp32s nAl, Npp8u \* pDst, Npp32s \* nLength, NppiEncodeHuffmanSpec \* apHuffmanDCTable[3], NppiEncodeHuffmanSpec \* apHuffmanACTable[3], NppiSize aSizeROI[3], Npp8u \* pTempStorage)

Huffman Encoding of the JPEG Encoding.

Input is expected to be 64x1 macro blocks and output is expected as byte stuffed huffman encoded JPEG scan.

#### **Parameters:**

apSrc Destination-Image Pointer.

7.4 Compression 57

```
aSrcStep Destination-Image Line Step.
nRestartInterval Restart Interval, see JPEG standard. Currently only values <=0 are supported.
nSs Start Coefficient, see JPEG standard.
nSe End Coefficient, see JPEG standard.
nAh Bit Approximation High, see JPEG standard.
nAl Bit Approximation Low, see JPEG standard.
pDst Byte-stuffed huffman encoded JPEG scan.
nLength Byte length of the huffman encoded JPEG scan.
apHuffmanDCTable DC Huffman tables.
apHuffmanACTable AC Huffman tables.
aSizeROI Region-of-Interest (ROI).
pTempStorage Temporary storage.</pre>
```

### **Returns:**

Error codes:

- NPP\_SIZE\_ERROR For negative input height/width or not a multiple of 8 width/height.
- NPP\_STEP\_ERROR If input image width is not multiple of 8 or does not match ROI.
- NPP\_NULL\_POINTER\_ERROR If the destination pointer is 0.
- NPP\_NOT\_SUFFICIENT\_COMPUTE\_CAPABILITY If the device has compute capability < 2.0.

## 7.4.3.10 NppStatus nppiEncodeHuffmanSpecFree\_JPEG (NppiEncodeHuffmanSpec \* pHuffmanSpec)

Frees the memory allocated by nppiEncodeHuffmanSpecInitAlloc\_JPEG.

## **Parameters:**

pHuffmanSpec Pointer to the Huffman table for the encoder

## 7.4.3.11 NppStatus nppiEncodeHuffmanSpecGetBufSize\_JPEG (int \* pSize)

Returns the length of the NppiEncodeHuffmanSpec structure.

#### **Parameters:**

pSize Pointer to a variable that will receive the length of the NppiEncodeHuffmanSpec structure.

## **Returns:**

Error codes:

• NPP\_NULL\_POINTER\_ERROR If one of the pointers is 0.

## 7.4.3.12 NppStatus nppiEncodeHuffmanSpecInit\_JPEG (const Npp8u \* pRawHuffmanTable, NppiHuffmanTableType eTableType, NppiEncodeHuffmanSpec \* pHuffmanSpec)

Creates a Huffman table in a format that is suitable for the encoder.

#### **Parameters:**

```
pRawHuffmanTable Huffman table formated as specified in the JPEG standard.eTableType Enum specifying type of table (nppiDCTable or nppiACTable).pHuffmanSpec Pointer to the Huffman table for the decoder
```

### **Returns:**

Error codes:

• NPP\_NULL\_POINTER\_ERROR If one of the pointers is 0.

## 7.4.3.13 NppStatus nppiEncodeHuffmanSpecInitAlloc\_JPEG (const Npp8u \* pRawHuffmanTable, NppiHuffmanTableType eTableType, NppiEncodeHuffmanSpec \*\* ppHuffmanSpec)

Allocates memory and creates a Huffman table in a format that is suitable for the encoder.

#### **Parameters:**

```
pRawHuffmanTable Huffman table formated as specified in the JPEG standard.eTableType Enum specifying type of table (nppiDCTable or nppiACTable).ppHuffmanSpec Pointer to returned pointer to the Huffman table for the encoder
```

## **Returns:**

Error codes:

• NPP\_NULL\_POINTER\_ERROR If one of the pointers is 0.

## 7.4.3.14 NppStatus nppiEncodeOptimizeHuffmanGetSize (NppiSize oSize, int nChannels, int \* pBufSize)

Calculates the size of the temporary buffer for optimize Huffman coding.

#### Parameters:

```
oSize Image Dimension.nChannels Number of channels in the image.pBufSize Pointer to variable that returns the size of the temporary buffer.
```

#### **Returns:**

NPP\_SUCCESS Indicates no error. Any other value indicates an error or a warning NPP\_SIZE\_ERROR Indicates an error condition if any image dimension has zero or negative value NPP\_NULL\_POINTER\_ERROR Indicates an error condition if pBufSize pointer is NULL

7.4 Compression 59

7.4.3.15 NppStatus nppiEncodeOptimizeHuffmanScan\_JPEG\_8u16s\_P1R (const Npp16s \* pSrc, Npp32s nSrcStep, Npp32s nRestartInterval, Npp32s nSs, Npp32s nSe, Npp32s nAh, Npp32s nAl, Npp8u \* pDst, Npp32s \* pLength, Npp8u \* hpCodesDC, Npp8u \* hpTableDC, Npp8u \* hpCodesAC, Npp8u \* hpTableAC, NppiEncodeHuffmanSpec \* pHuffmanDCTable, NppiEncodeHuffmanSpec \* pHuffmanACTable, NppiSize oSizeROI, Npp8u \* pTempStorage)

Optimize Huffman Encoding of the JPEG Encoding.

Input is expected to be 64x1 macro blocks and output is expected as byte stuffed huffman encoded JPEG scan.

#### **Parameters:**

```
pSrc Destination-Image Pointer.
nSrcStep Destination-Image Line Step.
nRestartInterval Restart Interval, see JPEG standard. Currently only values <=0 are supported.
nSs Start Coefficient, see JPEG standard.
nSe End Coefficient, see JPEG standard.
nAh Bit Approximation High, see JPEG standard.
nAl Bit Approximation Low, see JPEG standard.
pDst Byte-stuffed huffman encoded JPEG scan.
pLength Pointer to the byte length of the huffman encoded JPEG scan.
hpCodesDC Host pointer to the code of the huffman tree for DC component.
hpTableDC Host pointer to the table of the huffman tree for DC component.
hpCodesAC Host pointer to the code of the huffman tree for AC component.
hpTableAC Host pointer to the table of the huffman tree for AC component.
pHuffmanDCTable DC Huffman table.
pHuffmanACTable AC Huffman table.
oSizeROI Region-of-Interest (ROI).
pTempStorage Temporary storage.
```

#### **Returns:**

## Error codes:

- NPP\_SIZE\_ERROR For negative input height/width or not a multiple of 8 width/height.
- NPP\_STEP\_ERROR If input image width is not multiple of 8 or does not match ROI.
- NPP\_NULL\_POINTER\_ERROR If the destination pointer is 0.
- NPP\_NOT\_SUFFICIENT\_COMPUTE\_CAPABILITY If the device has compute capability < 2.0.
- 7.4.3.16 NppStatus nppiEncodeOptimizeHuffmanScan\_JPEG\_8u16s\_P3R (Npp16s \* apSrc[3], Npp32s aSrcStep[3], Npp32s nRestartInterval, Npp32s nSs, Npp32s nSe, Npp32s nSe, Npp32s nAh, Npp32s nAl, Npp8u \* pDst, Npp32s \* pLength, Npp8u \* hpCodesDC[3], Npp8u \* hpTableDC[3], Npp8u \* hpTableAC[3], NppiEncodeHuffmanSpec \* apHuffmanDCTable[3], NppiEncodeHuffmanSpec \* apHuffmanACTable[3], NppiSize oSizeROI[3], Npp8u \* pTempStorage)

Optimize Huffman Encoding of the JPEG Encoding.

Input is expected to be 64x1 macro blocks and output is expected as byte stuffed huffman encoded JPEG scan

#### **Parameters:**

```
apSrc Destination-Image Pointer.
aSrcStep Destination-Image Line Step.
nRestartInterval Restart Interval, see JPEG standard. Currently only values <=0 are supported.
nSs Start Coefficient, see JPEG standard.
nSe End Coefficient, see JPEG standard.
nAh Bit Approximation High, see JPEG standard.
nAl Bit Approximation Low, see JPEG standard.
pDst Byte-stuffed huffman encoded JPEG scan.
pLength Pointer to the byte length of the huffman encoded JPEG scan.
hpCodesDC Host pointer to the code of the huffman tree for DC component.
hpTableDC Host pointer to the table of the huffman tree for DC component.
hpCodesAC Host pointer to the code of the huffman tree for AC component.
hpTableAC Host pointer to the table of the huffman tree for AC component.
apHuffmanDCTable DC Huffman tables.
apHuffmanACTable AC Huffman tables.
oSizeROI Region-of-Interest (ROI).
pTempStorage Temporary storage.
```

## **Returns:**

#### Error codes:

- NPP\_SIZE\_ERROR For negative input height/width or not a multiple of 8 width/height.
- NPP\_STEP\_ERROR If input image width is not multiple of 8 or does not match ROI.
- NPP\_NULL\_POINTER\_ERROR If the destination pointer is 0.
- NPP\_NOT\_SUFFICIENT\_COMPUTE\_CAPABILITY If the device has compute capability < 2.0.

## 7.5 Quantization Functions

## **Typedefs**

typedef struct NppiDCTState NppiDCTState

## **Functions**

NppStatus nppiQuantFwdRawTableInit\_JPEG\_8u (Npp8u \*hpQuantRawTable, int nQualityFactor)

Apply quality factor to raw 8-bit quantization table.

NppStatus nppiQuantFwdTableInit\_JPEG\_8u16u (const Npp8u \*hpQuantRawTable, Npp16u \*hpQuantFwdRawTable)

Initializes a quantization table for nppiDCTQuantFwd8x8LS\_JPEG\_8u16s\_C1R().

• NppStatus nppiQuantInvTableInit\_JPEG\_8u16u (const Npp8u \*hpQuantRawTable, Npp16u \*hpQuantFwdRawTable)

Initializes a quantization table for nppiDCTQuantInv8x8LS\_JPEG\_16s8u\_C1R().

 NppStatus nppiDCTQuantFwd8x8LS\_JPEG\_8u16s\_C1R (const Npp8u \*pSrc, int nSrcStep, Npp16s \*pDst, int nDstStep, const Npp16u \*pQuantFwdTable, NppiSize oSizeROI)

Forward DCT, quantization and level shift part of the JPEG encoding.

NppStatus nppiDCTQuantInv8x8LS\_JPEG\_16s8u\_C1R (const Npp16s \*pSrc, int nSrcStep, Npp8u \*pDst, int nDstStep, const Npp16u \*pQuantInvTable, NppiSize oSizeROI)

Inverse DCT, de-quantization and level shift part of the JPEG decoding.

• NppStatus nppiDCTInitAlloc (NppiDCTState \*\*ppState)

Initializes DCT state structure and allocates additional resources.

• NppStatus nppiDCTFree (NppiDCTState \*pState)

Frees the additional resources of the DCT state structure.

NppStatus nppiDCTQuantFwd8x8LS\_JPEG\_8u16s\_C1R\_NEW (const Npp8u \*pSrc, int nSrcStep, Npp16s \*pDst, int nDstStep, const Npp8u \*pQuantizationTable, NppiSize oSizeROI, NppiDCTState \*pState)

Forward DCT, quantization and level shift part of the JPEG encoding.

NppStatus nppiDCTQuantInv8x8LS\_JPEG\_16s8u\_C1R\_NEW (const Npp16s \*pSrc, int nSrcStep, Npp8u \*pDst, int nDstStep, const Npp8u \*pQuantizationTable, NppiSize oSizeROI, NppiDCTState \*pState)

Inverse DCT, de-quantization and level shift part of the JPEG decoding.

 NppStatus nppiDCTQuant16Fwd8x8LS\_JPEG\_8u16s\_C1R\_NEW (const Npp8u \*pSrc, int nSrc-Step, Npp16s \*pDst, int nDstStep, const Npp16u \*pQuantizationTable, NppiSize oSizeROI, NppiD-CTState \*pState)

Forward DCT, quantization and level shift part of the JPEG encoding, 16-bit short integer.

NppStatus nppiDCTQuant16Inv8x8LS\_JPEG\_16s8u\_C1R\_NEW (const Npp16s \*pSrc, int nSrc-Step, Npp8u \*pDst, int nDstStep, const Npp16u \*pQuantizationTable, NppiSize oSizeROI, NppiD-CTState \*pState)

Inverse DCT, de-quantization and level shift part of the JPEG decoding, 16-bit short integer.

## 7.5.1 Typedef Documentation

## 7.5.1.1 typedef struct NppiDCTState NppiDCTState

## 7.5.2 Function Documentation

## 7.5.2.1 NppStatus nppiDCTFree (NppiDCTState \* pState)

Frees the additional resources of the DCT state structure.

#### See also:

nppiDCTInitAlloc

#### **Parameters:**

pState Pointer to DCT state structure.

## **Returns:**

NPP\_SUCCESS Indicates no error. Any other value indicates an error or a warning NPP\_SIZE\_ERROR Indicates an error condition if any image dimension has zero or negative value NPP\_NULL\_POINTER\_ERROR Indicates an error condition if pState pointer is NULL

## 7.5.2.2 NppStatus nppiDCTInitAlloc (NppiDCTState \*\* ppState)

Initializes DCT state structure and allocates additional resources.

## See also:

 $nppiDCTQuantFwd8x8LS\_JPEG\_8u16s\_C1R\_NEW(), \qquad nppiDCTQuantInv8x8LS\_JPEG\_16s8u\_-C1R\_NEW.$ 

## **Parameters:**

ppState Pointer to pointer to DCT state structure.

#### **Returns:**

NPP\_SUCCESS Indicates no error. Any other value indicates an error or a warning NPP\_SIZE\_ERROR Indicates an error condition if any image dimension has zero or negative value NPP\_NULL\_POINTER\_ERROR Indicates an error condition if pBufSize pointer is NULL

# 7.5.2.3 NppStatus nppiDCTQuant16Fwd8x8LS\_JPEG\_8u16s\_C1R\_NEW (const Npp8u \* pSrc, int nSrcStep, Npp16s \* pDst, int nDstStep, const Npp16u \* pQuantizationTable, NppiSize oSizeROI, NppiDCTState \* pState)

Forward DCT, quantization and level shift part of the JPEG encoding, 16-bit short integer.

Input is expected in 8x8 macro blocks and output is expected to be in 64x1 macro blocks. The new version of the primitive takes the ROI in image pixel size and works with DCT coefficients that are in zig-zag order.

#### **Parameters:**

```
pSrc Source-Image Pointer.
nSrcStep Source-Image Line Step.
pDst Destination-Image Pointer.
nDstStep Image width in pixels x 8 x sizeof(Npp16s).
pQuantizationTable Quantization Table in zig-zag order.
oSizeROI Region-of-Interest (ROI).
pState Pointer to DCT state structure. This structure must be initialized allocated and initialized using nppiDCTInitAlloc().
```

#### **Returns:**

Error codes:

- NPP\_SIZE\_ERROR For negative input height/width or not a multiple of 8 width/height.
- NPP\_STEP\_ERROR If input image width is not multiple of 8 or does not match ROI.
- NPP\_NULL\_POINTER\_ERROR If the destination pointer is 0.

# 7.5.2.4 NppStatus nppiDCTQuant16Inv8x8LS\_JPEG\_16s8u\_C1R\_NEW (const Npp16s \* pSrc, int nSrcStep, Npp8u \* pDst, int nDstStep, const Npp16u \* pQuantizationTable, NppiSize oSizeROI, NppiDCTState \* pState)

Inverse DCT, de-quantization and level shift part of the JPEG decoding, 16-bit short integer.

Input is expected in 64x1 macro blocks and output is expected to be in 8x8 macro blocks. The new version of the primitive takes the ROI in image pixel size and works with DCT coefficients that are in zig-zag order.

#### Parameters:

```
pSrc Source-Image Pointer.
nSrcStep Image width in pixels x 8 x sizeof(Npp16s).
pDst Destination-Image Pointer.
nDstStep Destination-Image Line Step.
pQuantizationTable Quantization Table in zig-zag order.
oSizeROI Region-of-Interest (ROI).
pState Pointer to DCT state structure. This structure must be initialized allocated and initialized using nppiDCTInitAlloc().
```

#### **Returns:**

Error codes:

- NPP\_SIZE\_ERROR For negative input height/width or not a multiple of 8 width/height.
- NPP STEP ERROR If input image width is not multiple of 8 or does not match ROI.
- NPP\_NULL\_POINTER\_ERROR If the destination pointer is 0.

64 Module Documentation

# 7.5.2.5 NppStatus nppiDCTQuantFwd8x8LS\_JPEG\_8u16s\_C1R (const Npp8u \* pSrc, int nSrcStep, Npp16s \* pDst, int nDstStep, const Npp16u \* pQuantFwdTable, NppiSize oSizeROI)

Forward DCT, quantization and level shift part of the JPEG encoding.

Input is expected in 8x8 macro blocks and output is expected to be in 64x1 macro blocks.

#### **Parameters:**

```
pSrc Source-Image Pointer.
nSrcStep Source-Image Line Step.
pDst Destination-Image Pointer.
nDstStep Destination-Image Line Step.
pQuantFwdTable Forward quantization tables for JPEG encoding created using nppiQuantInvTableInit_JPEG_8u16u().
oSizeROI Region-of-Interest (ROI).
```

#### **Returns:**

Error codes:

- NPP\_SIZE\_ERROR For negative input height/width or not a multiple of 8 width/height.
- NPP\_STEP\_ERROR If input image width is not multiple of 8 or does not match ROI.
- NPP\_NULL\_POINTER\_ERROR If the destination pointer is 0.

# 7.5.2.6 NppStatus nppiDCTQuantFwd8x8LS\_JPEG\_8u16s\_C1R\_NEW (const Npp8u \* pSrc, int nSrcStep, Npp16s \* pDst, int nDstStep, const Npp8u \* pQuantizationTable, NppiSize oSizeROI, NppiDCTState \* pState)

Forward DCT, quantization and level shift part of the JPEG encoding.

Input is expected in 8x8 macro blocks and output is expected to be in 64x1 macro blocks. The new version of the primitive takes the ROI in image pixel size and works with DCT coefficients that are in zig-zag order.

#### **Parameters:**

```
pSrc Source-Image Pointer.
nSrcStep Source-Image Line Step.
pDst Destination-Image Pointer.
nDstStep Image width in pixels x 8 x sizeof(Npp16s).
pQuantizationTable Quantization Table in zig-zag order.
oSizeROI Region-of-Interest (ROI).
pState Pointer to DCT state structure. This structure must be initialized allocated and initialized using nppiDCTInitAlloc().
```

#### **Returns:**

Error codes:

- NPP\_SIZE\_ERROR For negative input height/width or not a multiple of 8 width/height.
- NPP STEP ERROR If input image width is not multiple of 8 or does not match ROI.
- NPP\_NULL\_POINTER\_ERROR If the destination pointer is 0.

# 7.5.2.7 NppStatus nppiDCTQuantInv8x8LS\_JPEG\_16s8u\_C1R (const Npp16s \* pSrc, int nSrcStep, Npp8u \* pDst, int nDstStep, const Npp16u \* pQuantInvTable, NppiSize oSizeROI)

Inverse DCT, de-quantization and level shift part of the JPEG decoding.

Input is expected in 64x1 macro blocks and output is expected to be in 8x8 macro blocks.

#### **Parameters:**

```
pSrc Source-Image Pointer.
nSrcStep Image width in pixels x 8 x sizeof(Npp16s).
pDst Destination-Image Pointer.
nDstStep Image width in pixels x 8 x sizeof(Npp16s).
pQuantInvTable Inverse quantization tables for JPEG decoding created using nppiQuantInvTableInit_JPEG_8u16u().
oSizeROI Region-of-Interest (ROI).
```

#### **Returns:**

Error codes:

- NPP\_SIZE\_ERROR For negative input height/width or not a multiple of 8 width/height.
- NPP\_STEP\_ERROR If input image width is not multiple of 8 or does not match ROI.
- NPP\_NULL\_POINTER\_ERROR If the destination pointer is 0.

# 7.5.2.8 NppStatus nppiDCTQuantInv8x8LS\_JPEG\_16s8u\_C1R\_NEW (const Npp16s \* pSrc, int nSrcStep, Npp8u \* pDst, int nDstStep, const Npp8u \* pQuantizationTable, NppiSize oSizeROI, NppiDCTState \* pState)

Inverse DCT, de-quantization and level shift part of the JPEG decoding.

Input is expected in 64x1 macro blocks and output is expected to be in 8x8 macro blocks. The new version of the primitive takes the ROI in image pixel size and works with DCT coefficients that are in zig-zag order.

#### **Parameters:**

```
pSrc Source-Image Pointer.
nSrcStep Image width in pixels x 8 x sizeof(Npp16s).
pDst Destination-Image Pointer.
nDstStep Destination-Image Line Step.
pQuantizationTable Quantization Table in zig-zag order.
oSizeROI Region-of-Interest (ROI).
pState Pointer to DCT state structure. This structure must be initialized allocated and initialized using nppiDCTInitAlloc().
```

#### **Returns:**

Error codes:

- NPP\_SIZE\_ERROR For negative input height/width or not a multiple of 8 width/height.
- NPP STEP ERROR If input image width is not multiple of 8 or does not match ROI.
- NPP\_NULL\_POINTER\_ERROR If the destination pointer is 0.

66 Module Documentation

# 7.5.2.9 NppStatus nppiQuantFwdRawTableInit\_JPEG\_8u (Npp8u \* hpQuantRawTable, int nQualityFactor)

Apply quality factor to raw 8-bit quantization table.

This is effectively and in-place method that modifies a given raw quantization table based on a quality factor. Note that this method is a host method and that the pointer to the raw quantization table is a host pointer.

#### Parameters:

```
hpQuantRawTable Raw quantization table.nQualityFactor Quality factor for the table. Range is [1:100].
```

#### **Returns:**

Error code: NPP NULL POINTER ERROR is returned if hpQuantRawTable is 0.

# 7.5.2.10 NppStatus nppiQuantFwdTableInit\_JPEG\_8u16u (const Npp8u \* hpQuantRawTable, Npp16u \* hpQuantFwdRawTable)

Initializes a quantization table for nppiDCTQuantFwd8x8LS\_JPEG\_8u16s\_C1R().

The method creates a 16-bit version of the raw table and converts the data order from zigzag layout to original row-order layout since raw quantization tables are typically stored in zigzag format.

This method is a host method. It consumes and produces host data. I.e. the pointers passed to this function must be host pointers. The resulting table needs to be transferred to device memory in order to be used with nppiDCTQuantFwd8x8LS\_JPEG\_8u16s\_C1R() function.

#### **Parameters:**

hpQuantRawTable Host pointer to raw quantization table as returned by nppiQuantFwdRawTableInit\_JPEG\_8u(). The raw quantization table is assumed to be in zigzag order.

hpQuantFwdRawTable Forward quantization table for use with nppiDCTQuantFwd8x8LS\_JPEG\_-8u16s\_C1R().

#### **Returns:**

Error code: NPP\_NULL\_POINTER\_ERROR pQuantRawTable is 0.

# 7.5.2.11 NppStatus nppiQuantInvTableInit\_JPEG\_8u16u (const Npp8u \* hpQuantRawTable, Npp16u \* hpQuantFwdRawTable)

Initializes a quantization table for nppiDCTQuantInv8x8LS\_JPEG\_16s8u\_C1R().

The nppiDCTQuantFwd8x8LS\_JPEG\_8u16s\_C1R() method uses a quantization table in a 16-bit format allowing for faster processing. In addition it converts the data order from zigzag layout to original row-order layout. Typically raw quantization tables are stored in zigzag format.

This method is a host method and consumes and produces host data. I.e. the pointers passed to this function must be host pointers. The resulting table needs to be transferred to device memory in order to be used with nppiDCTQuantFwd8x8LS\_JPEG\_8u16s\_C1R() function.

# **Parameters:**

hpQuantRawTable Raw quantization table.hpQuantFwdRawTable Inverse quantization table.

### **Returns:**

NPP\_NULL\_POINTER\_ERROR pQuantRawTable or pQuantFwdRawTable is0.

68 Module Documentation

# **Chapter 8**

# **Data Structure Documentation**

# 8.1 NPP\_ALIGN\_16 Struct Reference

Complex Number This struct represents a long long complex number.

```
#include <nppdefs.h>
```

### **Data Fields**

• Npp64s re

Real part.

• Npp64s im

Imaginary part.

• Npp64f re

Real part.

• Npp64f im

Imaginary part.

# 8.1.1 Detailed Description

Complex Number This struct represents a long long complex number.

Complex Number This struct represents a double floating-point complex number.

### **8.1.2** Field Documentation

### 8.1.2.1 Npp64f NPP\_ALIGN\_16::im

Imaginary part.

# 8.1.2.2 Npp64s NPP\_ALIGN\_16::im

Imaginary part.

# 8.1.2.3 Npp64f NPP\_ALIGN\_16::re

Real part.

# 8.1.2.4 Npp64s NPP\_ALIGN\_16::re

Real part.

The documentation for this struct was generated from the following file:

# 8.2 NPP\_ALIGN\_8 Struct Reference

Complex Number This struct represents an unsigned int complex number.

```
#include <nppdefs.h>
```

#### **Data Fields**

- Npp32u re

  Real part.
- Npp32u im

  Imaginary part.
- Npp32s re

  Real part.
- Npp32s im

  Imaginary part.
- Npp32f re

  Real part.
- Npp32f im

  Imaginary part.

# **8.2.1** Detailed Description

Complex Number This struct represents an unsigned int complex number.

Complex Number This struct represents a single floating-point complex number.

Complex Number This struct represents a signed int complex number.

#### **8.2.2** Field Documentation

#### 8.2.2.1 Npp32f NPP\_ALIGN\_8::im

Imaginary part.

### 8.2.2.2 Npp32s NPP\_ALIGN\_8::im

Imaginary part.

### 8.2.2.3 Npp32u NPP\_ALIGN\_8::im

Imaginary part.

# 8.2.2.4 Npp32f NPP\_ALIGN\_8::re

Real part.

# 8.2.2.5 Npp32s NPP\_ALIGN\_8::re

Real part.

# 8.2.2.6 Npp32u NPP\_ALIGN\_8::re

Real part.

The documentation for this struct was generated from the following file:

# 8.3 NppiHaarBuffer Struct Reference

#include <nppdefs.h>

# **Data Fields**

- int haarBufferSize size of the buffer
- Npp32s \* haarBuffer buffer

#### **8.3.1** Field Documentation

# 8.3.1.1 Npp32s\* NppiHaarBuffer::haarBuffer

buffer

# 8.3.1.2 int NppiHaarBuffer::haarBufferSize

size of the buffer

The documentation for this struct was generated from the following file:

# 8.4 NppiHaarClassifier\_32f Struct Reference

#include <nppdefs.h>

# **Data Fields**

- int numClassifiers

  number of classifiers
- Npp32s \* classifiers

  packed classifier data 40 bytes each
- size\_t classifierStep
- NppiSize classifierSize
- Npp32s \* counterDevice

#### **8.4.1** Field Documentation

### 8.4.1.1 Npp32s\* NppiHaarClassifier\_32f::classifiers

packed classifier data 40 bytes each

- 8.4.1.2 NppiSize NppiHaarClassifier\_32f::classifierSize
- 8.4.1.3 size\_t NppiHaarClassifier\_32f::classifierStep
- 8.4.1.4 Npp32s\* NppiHaarClassifier\_32f::counterDevice
- 8.4.1.5 int NppiHaarClassifier\_32f::numClassifiers

number of classifiers

The documentation for this struct was generated from the following file:

# 8.5 NppiHOGConfig Struct Reference

The NppiHOGConfig structure defines the configuration parameters for the HOG descriptor:.

```
#include <nppdefs.h>
```

#### **Data Fields**

• int cellSize square cell size (pixels).

• int histogramBlockSize square histogram block size (pixels).

• int nHistogramBins required number of histogram bins.

 NppiSize detectionWindowSize detection window size (pixels).

### 8.5.1 Detailed Description

The NppiHOGConfig structure defines the configuration parameters for the HOG descriptor:.

#### 8.5.2 Field Documentation

#### 8.5.2.1 int NppiHOGConfig::cellSize

square cell size (pixels).

#### 8.5.2.2 NppiSize NppiHOGConfig::detectionWindowSize

detection window size (pixels).

#### 8.5.2.3 int NppiHOGConfig::histogramBlockSize

square histogram block size (pixels).

# 8.5.2.4 int NppiHOGConfig::nHistogramBins

required number of histogram bins.

The documentation for this struct was generated from the following file:

# 8.6 NppiPoint Struct Reference

### 2D Point

```
#include <nppdefs.h>
```

### **Data Fields**

• int x

*x-coordinate.* 

• int y

y-coordinate.

# **8.6.1** Detailed Description

2D Point

### **8.6.2** Field Documentation

### 8.6.2.1 int NppiPoint::x

x-coordinate.

# 8.6.2.2 int NppiPoint::y

y-coordinate.

The documentation for this struct was generated from the following file:

# 8.7 NppiRect Struct Reference

2D Rectangle This struct contains position and size information of a rectangle in two space.

```
#include <nppdefs.h>
```

#### **Data Fields**

• int x

x-coordinate of upper left corner (lowest memory address).

• int y

y-coordinate of upper left corner (lowest memory address).

• int width

Rectangle width.

• int height

Rectangle height.

### 8.7.1 Detailed Description

2D Rectangle This struct contains position and size information of a rectangle in two space.

The rectangle's position is usually signified by the coordinate of its upper-left corner.

#### **8.7.2** Field Documentation

#### 8.7.2.1 int NppiRect::height

Rectangle height.

### 8.7.2.2 int NppiRect::width

Rectangle width.

#### 8.7.2.3 int NppiRect::x

x-coordinate of upper left corner (lowest memory address).

### 8.7.2.4 int NppiRect::y

y-coordinate of upper left corner (lowest memory address).

The documentation for this struct was generated from the following file:

# 8.8 NppiSize Struct Reference

2D Size This struct typically represents the size of a a rectangular region in two space.

```
#include <nppdefs.h>
```

### **Data Fields**

• int width

Rectangle width.

• int height

Rectangle height.

# 8.8.1 Detailed Description

2D Size This struct typically represents the size of a a rectangular region in two space.

#### **8.8.2** Field Documentation

### 8.8.2.1 int NppiSize::height

Rectangle height.

### 8.8.2.2 int NppiSize::width

Rectangle width.

The documentation for this struct was generated from the following file:

# 8.9 NppLibraryVersion Struct Reference

#include <nppdefs.h>

# **Data Fields**

• int major

Major version number.

• int minor

Minor version number.

• int build

Build number.

#### **8.9.1** Field Documentation

# 8.9.1.1 int NppLibraryVersion::build

Build number.

This reflects the nightly build this release was made from.

# 8.9.1.2 int NppLibraryVersion::major

Major version number.

# 8.9.1.3 int NppLibraryVersion::minor

Minor version number.

The documentation for this struct was generated from the following file:

# 8.10 NppPointPolar Struct Reference

### 2D Polar Point

#include <nppdefs.h>

### **Data Fields**

- Npp32f rho
- Npp32f theta

# 8.10.1 Detailed Description

2D Polar Point

### 8.10.2 Field Documentation

8.10.2.1 Npp32f NppPointPolar::rho

8.10.2.2 Npp32f NppPointPolar::theta

The documentation for this struct was generated from the following file:

# **Index**

align	NPP_ALIGN_8, 71
npp_basic_types, 49, 50	image_compression
	nppiDecodeHuffmanScanHost_JPEG
Basic NPP Data Types, 47	8u16s_P1R, 53
build	nppiDecodeHuffmanScanHost_JPEG
NppLibrary Version, 79	8u16s_P3R, 53
	NppiDecodeHuffmanSpec, 53
cellSize	nppiDecodeHuffmanSpecFreeHost_JPEG, 54
NppiHOGConfig, 75	nppiDecodeHuffmanSpecGetBufSize_JPEG,
classifiers	54
NppiHaarClassifier_32f, 74	nppiDecodeHuffmanSpecInitAllocHost
classifierSize	JPEG, 54
NppiHaarClassifier_32f, 74	nppiDecodeHuffmanSpecInitHost_JPEG, 55
classifierStep	nppiEncodeHuffmanGetSize, 55
NppiHaarClassifier_32f, 74	nppiEncodeHuffmanScan_JPEG_8u16s_P1R
Compression, 51	55
core_npp	nppiEncodeHuffmanScan_JPEG_8u16s_P3R
nppGetGpuComputeCapability, 28	56
nppGetGpuDeviceProperties, 28	NppiEncodeHuffmanSpec, 53
nppGetGpuName, 28	nppiEncodeHuffmanSpecFree_JPEG, 57
nppGetGpuNumSMs, 28	nppiEncodeHuffmanSpecGetBufSize_JPEG,
nppGetLibVersion, 28	57
nppGetMaxThreadsPerBlock, 29	nppiEncodeHuffmanSpecInit_JPEG, 57
nppGetMaxThreadsPerSM, 29	nppiEncodeHuffmanSpecInitAlloc_JPEG, 58
nppGetStream, 29	nppiEncodeOptimizeHuffmanGetSize, 58
nppGetStreamMaxThreadsPerSM, 29	nppiEncodeOptimizeHuffmanScan_JPEG
nppGetStreamNumSMs, 29	8u16s_P1R, 58
nppSetStream, 29	nppiEncodeOptimizeHuffmanScan_JPEG
counterDevice	8u16s_P3R, 59
NppiHaarClassifier_32f, 74	image_quantization
1	nppiDCTFree, 62
detectionWindowSize	nppiDCTInitAlloc, 62
NppiHOGConfig, 75	nppiDCTQuant16Fwd8x8LS_JPEG_8u16s
haarBuffer	C1R_NEW, 62
NppiHaarBuffer, 73	nppiDCTQuant16Inv8x8LS_JPEG_16s8u
haarBufferSize	C1R_NEW, 63
NppiHaarBuffer, 73	nppiDCTQuantFwd8x8LS_JPEG_8u16s
	C1R, 63
height NppiRect, 77	nppiDCTQuantFwd8x8LS_JPEG_8u16s
NppiSize, 78	C1R_NEW, 64
histogramBlockSize	nppiDCTQuantInv8x8LS_JPEG_16s8u_C1R
NppiHOGConfig, 75	64
ryphriodcomig, 73	nppiDCTQuantInv8x8LS_JPEG_16s8u
im	C1R_NEW, 65
NPP ALIGN 16 69	NppiDCTState 62

nppiQuantFwdRawTableInit_JPEG_8u, 65	typedefs_npp, 41
nppiQuantFwdTableInit_JPEG_8u16u, 66	NPP_ALG_HINT_NONE
	typedefs_npp, 41
nppiQuantInvTableInit_JPEG_8u16u, 66	
	NPP_ALIGNMENT_ERROR
major	typedefs_npp, 44
NppLibrary Version, 79	NPP_ANCHOR_ERROR
minor	typedefs_npp, 45
NppLibrary Version, 79	
ryppelorary version, 79	NPP_BAD_ARGUMENT_ERROR
II' . D'	typedefs_npp, 45
nHistogramBins	NPP_BORDER_CONSTANT
NppiHOGConfig, 75	typedefs_npp, 42
NPP Core, 27	NPP_BORDER_MIRROR
NPP Type Definitions and Constants, 31	typedefs_npp, 42
Npp16s	
* *	NPP_BORDER_NONE
npp_basic_types, 48	typedefs_npp, 42
Npp16sc	NPP_BORDER_REPLICATE
npp_basic_types, 50	typedefs_npp, 42
Npp16u	NPP_BORDER_UNDEFINED
npp_basic_types, 48	
**	typedefs_npp, 42
Npp16uc	NPP_BORDER_WRAP
npp_basic_types, 50	typedefs_npp, 42
Npp32f	NPP_BOTH_AXIS
npp_basic_types, 48	typedefs_npp, 41
Npp32fc	NPP_CHANNEL_ERROR
npp_basic_types, 48	
	typedefs_npp, 45
Npp32s	NPP_CHANNEL_ORDER_ERROR
npp_basic_types, 48	typedefs_npp, 45
Npp32sc	NPP_CMP_EQ
npp_basic_types, 48	typedefs_npp, 40
Npp32u	NPP_CMP_GREATER
* *	
npp_basic_types, 49	typedefs_npp, 40
Npp32uc	NPP_CMP_GREATER_EQ
npp_basic_types, 49	typedefs_npp, 40
Npp64f	NPP CMP LESS
npp_basic_types, 49	typedefs_npp, 40
Npp64fc	NPP_CMP_LESS_EQ
* *	_
npp_basic_types, 49	typedefs_npp, 40
Npp64s	NPP_COEFFICIENT_ERROR
npp_basic_types, 49	typedefs_npp, 45
Npp64sc	NPP_COI_ERROR
npp_basic_types, 49	typedefs_npp, 45
Npp64u	NPP_CONTEXT_MATCH_ERROR
npp_basic_types, 49	typedefs_npp, 45
Npp8s	NPP_CORRUPTED_DATA_ERROR
npp_basic_types, 49	typedefs_npp, 45
Npp8u	NPP_CUDA_1_0
npp_basic_types, 49	typedefs_npp, 40
Npp8uc	NPP_CUDA_1_1
npp_basic_types, 50	typedefs_npp, 40
NPP_AFFINE_QUAD_INCORRECT_WARNING	NPP_CUDA_1_2
typedefs_npp, 46	typedefs_npp, 40
NPP_ALG_HINT_ACCURATE	NPP_CUDA_1_3
typedefs_npp, 41	typedefs_npp, 40
NPP_ALG_HINT_FAST	NPP_CUDA_2_0

typedefs_npp, 40	typedefs_npp, 42
NPP_CUDA_2_1	NPP_HAAR_CLASSIFIER_PIXEL_MATCH ERROR
typedefs_npp, 40 NPP_CUDA_3_0	typedefs_npp, 44
typedefs_npp, 40	NPP_HISTOGRAM_NUMBER_OF_LEVELS
NPP_CUDA_3_2	ERROR
typedefs_npp, 40	typedefs_npp, 44
NPP_CUDA_3_5	NPP_HORIZONTAL_AXIS
typedefs_npp, 40	typedefs_npp, 41
NPP_CUDA_3_7	NPP_INTERPOLATION_ERROR
typedefs_npp, 40	typedefs_npp, 45
NPP_CUDA_5_0	NPP_INVALID_DEVICE_POINTER_ERROR
typedefs_npp, 40	typedefs_npp, 44
NPP_CUDA_5_2	NPP_INVALID_HOST_POINTER_ERROR
typedefs_npp, 40	typedefs_npp, 44
NPP_CUDA_5_3	NPP_LUT_NUMBER_OF_LEVELS_ERROR
typedefs_npp, 40	typedefs_npp, 45
NPP_CUDA_6_0	NPP_LUT_PALETTE_BITSIZE_ERROR
typedefs_npp, 40	typedefs_npp, 44
NPP_CUDA_6_1	NPP_MASK_SIZE_11_X_11
typedefs_npp, 40	typedefs_npp, 43
NPP_CUDA_6_2	NPP_MASK_SIZE_13_X_13
typedefs_npp, 40	typedefs_npp, 43
NPP_CUDA_6_3	NPP_MASK_SIZE_15_X_15
typedefs_npp, 40	typedefs_npp, 43
NPP_CUDA_7_0	NPP_MASK_SIZE_1_X_3
typedefs_npp, 40	typedefs_npp, 43
NPP_CUDA_KERNEL_EXECUTION_ERROR	NPP_MASK_SIZE_1_X_5
typedefs_npp, 44	typedefs_npp, 43
NPP_CUDA_NOT_CAPABLE	NPP_MASK_SIZE_3_X_1
typedefs_npp, 40	typedefs_npp, 43
NPP_CUDA_UNKNOWN_VERSION	NPP_MASK_SIZE_3_X_3
typedefs_npp, 40	typedefs_npp, 43
NPP_DATA_TYPE_ERROR	NPP_MASK_SIZE_5_X_1
typedefs_npp, 45	typedefs_npp, 43
NPP_DIVIDE_BY_ZERO_ERROR	NPP_MASK_SIZE_5_X_5
typedefs_npp, 45	typedefs_npp, 43
NPP_DIVIDE_BY_ZERO_WARNING	NPP_MASK_SIZE_7_X_7
typedefs_npp, 46	typedefs_npp, 43
NPP_DIVISOR_ERROR	NPP_MASK_SIZE_9_X_9
typedefs_npp, 45	typedefs_npp, 43
NPP_DOUBLE_SIZE_WARNING	NPP_MASK_SIZE_ERROR
typedefs_npp, 46 NPP_ERROR	typedefs_npp, 45
	NPP_MEMCPY_ERROR
typedefs_npp, 45 NPP_ERROR_RESERVED	typedefs_npp, 44 NPP_MEMFREE_ERROR
typedefs_npp, 45	typedefs_npp, 44
NPP_FFT_FLAG_ERROR	NPP_MEMORY_ALLOCATION_ERR
typedefs_npp, 45	typedefs_npp, 45
NPP_FFT_ORDER_ERROR	NPP_MEMSET_ERROR
typedefs_npp, 45	typedefs_npp, 44
NPP_FILTER_SCHARR	NPP_MIRROR_FLIP_ERROR
typedefs_npp, 42	typedefs_npp, 45
NPP_FILTER_SOBEL	NPP_MISALIGNED_DST_ROI_WARNING
_	

typedefs_npp, 46	NPP_SCALE_RANGE_ERROR
NPP_MOMENT_00_ZERO_ERROR	typedefs_npp, 45
typedefs_npp, 45	NPP_SIZE_ERROR
NPP_NO_ERROR	typedefs_npp, 45
typedefs_npp, 45	NPP_STEP_ERROR
NPP_NO_MEMORY_ERROR	typedefs_npp, 45
typedefs_npp, 45	NPP_STRIDE_ERROR
NPP_NO_OPERATION_WARNING	typedefs_npp, 45
typedefs_npp, 45	NPP SUCCESS
NPP_NOT_EVEN_STEP_ERROR	typedefs_npp, 45
typedefs_npp, 44	NPP_TEXTURE_BIND_ERROR
NPP_NOT_IMPLEMENTED_ERROR	
	typedefs_npp, 44
typedefs_npp, 45	NPP_THRESHOLD_ERROR
NPP_NOT_SUFFICIENT_COMPUTE	typedefs_npp, 45
CAPABILITY	NPP_THRESHOLD_NEGATIVE_LEVEL
typedefs_npp, 44	ERROR
NPP_NOT_SUPPORTED_MODE_ERROR	typedefs_npp, 45
typedefs_npp, 44	NPP_VERTICAL_AXIS
NPP_NULL_POINTER_ERROR	typedefs_npp, 41
typedefs_npp, 45	NPP_WRONG_INTERSECTION_QUAD
NPP_NUMBER_OF_CHANNELS_ERROR	WARNING
typedefs_npp, 45	typedefs_npp, 46
NPP_OUT_OFF_RANGE_ERROR	NPP_WRONG_INTERSECTION_ROI_ERROR
typedefs_npp, 45	typedefs_npp, 44
NPP_OVERFLOW_ERROR	NPP_WRONG_INTERSECTION_ROI
typedefs_npp, 44	WARNING
NPP_QUADRANGLE_ERROR	typedefs_npp, 46
typedefs_npp, 45	NPP_ZC_MODE_NOT_SUPPORTED_ERROR
NPP_QUALITY_INDEX_ERROR	typedefs_npp, 44
typedefs_npp, 44	NPP_ZERO_MASK_VALUE_ERROR
NPP_RANGE_ERROR	typedefs_npp, 45
typedefs_npp, 45	NPP_ALIGN_16, 69
NPP_RECTANGLE_ERROR	im, 69
typedefs_npp, 45	re, 70
NPP_RESIZE_FACTOR_ERROR	NPP_ALIGN_8, 71
typedefs_npp, 45	im, 71
NPP_RESIZE_NO_OPERATION_ERROR	re, 71, 72
typedefs_npp, 44	npp_basic_types
NPP_RND_FINANCIAL	align, 49, 50
typedefs_npp, 43	Npp16s, 48
NPP_RND_NEAR	Npp16sc, 50
typedefs_npp, 43	Npp16u, 48
NPP_RND_ZERO	Npp16uc, 50
typedefs_npp, 44	Npp32f, 48
NPP_ROUND_MODE_NOT_SUPPORTED	Npp32fc, 48
ERROR	Npp32s, 48
typedefs_npp, 44	Npp32sc, 48
NPP_ROUND_NEAREST_TIES_AWAY	Npp32u, 49
FROM_ZERO	Npp32uc, 49
typedefs_npp, 44	Npp64f, 49
NPP_ROUND_NEAREST_TIES_TO_EVEN	Npp64fc, 49
typedefs_npp, 43	Npp64s, 49
NPP_ROUND_TOWARD_ZERO	Npp64sc, 49
typedefs_npp, 44	Npp64u, 49
2) Pedero_iiPP,	- 'PP' '6', '/

Npp8s, 49	typedefs_npp, 39
Npp8u, 49	NPP_MINABS_64F
Npp8uc, 50	typedefs_npp, 39
NPP_HOG_MAX_BINS_PER_CELL	NppCmpOp
typedefs_npp, 37	typedefs_npp, 40
NPP_HOG_MAX_BLOCK_SIZE	nppGetGpuComputeCapability
typedefs_npp, 37	core_npp, 28
NPP_HOG_MAX_CELL_SIZE	nppGetGpuDeviceProperties
typedefs_npp, 37	core_npp, 28
NPP_HOG_MAX_CELLS_PER_DESCRIPTOR	nppGetGpuName
typedefs_npp, 37	core_npp, 28
NPP_HOG_MAX_DESCRIPTOR	nppGetGpuNumSMs
LOCATIONS_PER_CALL	core_npp, 28
typedefs_npp, 38	nppGetLibVersion
NPP_HOG_MAX_OVERLAPPING_BLOCKS	core_npp, 28
PER_DESCRIPTOR	nppGetMaxThreadsPerBlock
typedefs_npp, 38	core_npp, 29
NPP MAX 16S	nppGetMaxThreadsPerSM
<del>-</del> -	* *
typedefs_npp, 38	core_npp, 29
NPP_MAX_16U	nppGetStream
typedefs_npp, 38	core_npp, 29
NPP_MAX_32S	nppGetStreamMaxThreadsPerSM
typedefs_npp, 38	core_npp, 29
NPP_MAX_32U	nppGetStreamNumSMs
typedefs_npp, 38	core_npp, 29
NPP_MAX_64S	NppGpuComputeCapability
typedefs_npp, 38	typedefs_npp, 40
NPP_MAX_64U	NppHintAlgorithm
typedefs_npp, 38	typedefs_npp, 40
NPP_MAX_8S	NPPI_BAYER_BGGR
typedefs_npp, 38	typedefs_npp, 41
NPP_MAX_8U	NPPI_BAYER_GBRG
typedefs_npp, 38	typedefs_npp, 41
NPP_MAXABS_32F	NPPI_BAYER_GRBG
typedefs_npp, 38	typedefs_npp, 41
NPP_MAXABS_64F	NPPI_BAYER_RGGB
typedefs_npp, 39	typedefs_npp, 41
NPP_MIN_16S	NPPI_INTER_CUBIC
typedefs_npp, 39	typedefs_npp, 42
NPP_MIN_16U	NPPI_INTER_CUBIC2P_B05C03
typedefs_npp, 39	typedefs_npp, 42
NPP_MIN_32S	NPPI_INTER_CUBIC2P_BSPLINE
typedefs_npp, 39	typedefs_npp, 42
NPP_MIN_32U	NPPI_INTER_CUBIC2P_CATMULLROM
typedefs_npp, 39	typedefs_npp, 42
NPP_MIN_64S	NPPI_INTER_LANCZOS
typedefs_npp, 39	typedefs_npp, 42
NPP_MIN_64U	NPPI_INTER_LANCZOS3_ADVANCED
typedefs_npp, 39	typedefs_npp, 42
NPP_MIN_8S	NPPI_INTER_LINEAR
typedefs_npp, 39	typedefs_npp, 42
NPP_MIN_8U	NPPI_INTER_NN
typedefs_npp, 39	typedefs_npp, 42
NPP_MINABS_32F	NPPI_INTER_SUPER

tunadafa non 42	image_quantization, 63
typedefs_npp, 42 NPPI_INTER_UNDEFINED	nppiDCTQuantFwd8x8LS_JPEG_8u16s_C1R
typedefs_npp, 42	NEW
NPPI_OP_ALPHA_ATOP	image_quantization, 64
typedefs_npp, 41	nppiDCTQuantInv8x8LS_JPEG_16s8u_C1R
NPPI_OP_ALPHA_ATOP_PREMUL	image_quantization, 64
typedefs_npp, 41	nppiDCTQuantInv8x8LS_JPEG_16s8u_C1R
NPPI_OP_ALPHA_IN	NEW
typedefs_npp, 41	image_quantization, 65
NPPI_OP_ALPHA_IN_PREMUL	NppiDCTState
typedefs_npp, 41	image_quantization, 62
NPPI_OP_ALPHA_OUT	nppiDecodeHuffmanScanHost_JPEG_8u16s_P1R
typedefs_npp, 41	image_compression, 53
NPPI_OP_ALPHA_OUT_PREMUL	nppiDecodeHuffmanScanHost_JPEG_8u16s_P3R
typedefs_npp, 41	image_compression, 53
NPPI_OP_ALPHA_OVER	NppiDecodeHuffmanSpec
typedefs_npp, 41	image_compression, 53
NPPI_OP_ALPHA_OVER_PREMUL	nppiDecodeHuffmanSpecFreeHost_JPEG
typedefs_npp, 41	image_compression, 54
NPPI_OP_ALPHA_PLUS	nppiDecodeHuffmanSpecGetBufSize_JPEG
typedefs_npp, 41	image_compression, 54
NPPI_OP_ALPHA_PLUS_PREMUL	nppiDecodeHuffmanSpecInitAllocHost_JPEG
typedefs_npp, 41	image_compression, 54
NPPI_OP_ALPHA_PREMUL	nppiDecodeHuffmanSpecInitHost_JPEG
typedefs_npp, 41	image_compression, 55
NPPI_OP_ALPHA_XOR	NppiDifferentialKernel
typedefs_npp, 41	typedefs_npp, 42
NPPI_OP_ALPHA_XOR_PREMUL	nppiEncodeHuffmanGetSize
typedefs_npp, 41	image_compression, 55
NPPI_SMOOTH_EDGE	nppiEncodeHuffmanScan_JPEG_8u16s_P1R
typedefs_npp, 42	image_compression, 55
nppiACTable	nppiEncodeHuffmanScan_JPEG_8u16s_P3R
typedefs_npp, 42	image_compression, 56
NppiAlphaOp	NppiEncodeHuffmanSpec
typedefs_npp, 41	image_compression, 53
NppiAxis	nppiEncodeHuffmanSpecFree_JPEG
typedefs_npp, 41	image_compression, 57
NppiBayerGridPosition	nppiEncodeHuffmanSpecGetBufSize_JPEG
typedefs_npp, 41	image_compression, 57
NppiBorderType	nppiEncodeHuffmanSpecInit_JPEG
typedefs_npp, 41	image_compression, 57
nppiDCTable	nppiEncodeHuffmanSpecInitAlloc_JPEG
typedefs_npp, 42	image_compression, 58
nppiDCTFree	nppiEncodeOptimizeHuffmanGetSize
image_quantization, 62	image_compression, 58
nppiDCTInitAlloc	nppiEncodeOptimizeHuffmanScan_JPEG_8u16s
image_quantization, 62	P1R
nppiDCTQuant16Fwd8x8LS_JPEG_8u16s_C1R	image_compression, 58
NEW	nppiEncodeOptimizeHuffmanScan_JPEG_8u16s
image_quantization, 62	P3R
nppiDCTQuant16Inv8x8LS_JPEG_16s8u_C1R	image_compression, 59
NEW	NppiHaarBuffer, 73
image_quantization, 63	haarBuffer, 73
nppiDCTQuantFwd8x8LS_JPEG_8u16s_C1R	haarBufferSize, 73

NppiHaarClassifiers, 74 classifierSize, 74 classifierSize, 74 classifierSize, 74 classifierSize, 74 classifierSize, 74 classifierSize, 74 nppiHorGonfig, 75 cellSize, 75 detectionWindowSize, 75 nhistogramBlockSize, 75 nhistogramBlockSize, 75 nhistogramBlockSize, 75 nhistogramBlockSize, 75 nhistogramBlockSize, 75 npiHufmanTableType typedefs_npp, 42 NppiHufmanTableType typedefs_npp, 42 NppiHormanTableType typedefs_npp, 42 NppiNorm typedefs_npp, 42 NppiNorm typedefs_npp, 43 nppiNormIL 1 typedefs_npp, 43 nppiNormIL 2 typedefs_npp, 43 nppiNormIL 4 typedefs_npp, 43 nppiNormIL 5 typedefs_npp, 43 nppiNormIL 6 x, 76 x, 76 y, 76 nppiQuantFwdRawTableInit_JPEG_8u image_quantization, 65 nppiQuantFwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantfwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantfwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantfwTableInit_JPEG_8u16u image_quantization, 66 nppiQuentfwTableInit_PPEG_8u16u image_quantization, 66 nppiQuentfwTableInit_JPEG_8u16u image_quantization, 66 nppidet_s_npp, 43 npiNormIL 1 nppe ALIGN_B, 71, 72 nppintPolar, 80 nppe ALIGN_B, 71, 72 nppintPolar, 80 nppentDolar, 80 nppentDolar, 80 nppentDolar, 80 nppentDolar, 80 nppentDolar, 80 nppentDolar,		
classifierSize, 74 classifierSize, 74 classifierSize, 74 counterDevice, 74 npmClassifiers, 74 NppiHOGConfig, 75 cellSize, 75 detectionWindowSize, 75 nHistogramBiockSize, 75 nHistogramBios, 75 NppiHuffmanTableType typedefs_npp, 42 NppiInterpolationMode typedefs_npp, 42 NppiMsakSize typedefs_npp, 42 NppiMsakSize typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiQuantfwdRawTableInit_JPEG_8u image_quantization, 65 nppiQuantfwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantfwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantfwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantfwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantfwdRawTableInit_JPEG_8u16u image_quantization, 66 nppiQuantfwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantfwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantfwdRawTableInit_JPEG_8u16u image_quantization, 66 nppiQuantfwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantfwdRawTableInit_JPEG_8u16u image_quantizat	NppiHaarClassifier_32f, 74	** **
classifierStep, 74 counterDevice, 74 npmICassifiers, 74 NppiHOGConfig, 75 cellSize, 75 detectionWindowSize, 75 histogramBlockSize, 75 nHistogramBlockSize, 75 nHistogramBlockSize, 75 histogramBlockSize, 75 nHistogramBlockSize, 75 NppiHuffmanTableType typedefs_npp, 42 NppiHoglationMode typedefs_npp, 42 NppiMormL1 typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 NppiPoint, 76 x, 76 y, 76 nppiQuantFwdRawTableInit_JPEG_8u16u image_quantization, 65 nppiQuantIrwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantIrwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantIrwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantIrwdTableInit_JPEG_8u16u image_quantization, 66 NppiRoment typedefs_npp, 43 NppiDoment typedefs_npp, 43 Npp		** **
counterDevice, 74 nppiHOGConfig, 75 cellSize, 75 detectionWindowSize, 75 histogramBlockSize, 75 nHistogramBlockSize, 75 npiHarClassifier, 32f, 74 NppiAlign-16, 70 NPP_ALIGN_16, 70 NPP_ALIGN_16, 70 NPP_ALIGN_16, 70 NPP_ALIGN_18, 70 NPP_ALIGN_18, 70 NPP_AIG, HINT_ACCURATE, 41 NPP_AIG, HINT_NONE, 41 NPP_AIG, HINT_NONE, 41 NPP_AIG, HINT_NONE, 41 NPP_ANCHOR_ERROR, 44 NPP_BORDER_MINT_ERROR, 45 NPP_BORDER_MINT, 42 NPP_BORDER_MINT, 42 NPP_BORDER_MINT, 42 NPP_BORDER_WRAP, 42		**
numClassifiers, 74 NppiHOGConfig, 75 cellSize, 75 detectionWindowSize, 75 nHistogramBlockSize, 70 nppZCXor nPp_AliGN_16, 70 nPp_AliGN_16, 70 nPp_AliGN_16, 70 nPp_AliGN_16, 70 nPp_AliGN_16, 70 nPp_AliGN_16, 70 nPp_AliGN_16,	<u> -</u>	* *
NppiHOGConfig, 75 cellSize, 75 clestize, 75 histogramBlockSize, 75 hippAlidarClassifier, 22f, 74  NppLaliGn 16, 70 NPP_ALIGN 16, 70 NPP_ALIGN 18, 70 NPP_AFFINE_QUAD_INCORRECT WARNING, 46 NPP_ALIGN_BOTORRECT WARNING, 46 NPP_ALIGN_HIT_ACCURATE, 41 NPP_ALIGN_HIT_ACCURATE, 41 NPP_ALIGN_HIT_ACCURATE, 41 NPP_ANCHO_ERROR, 45 NPP_BORDER_CNOK, 42 NPP_BORDER_MIRROR, 42 NPP_BORDER_MIRROR, 42 NPP_BORDER_MIRROR, 42 NPP_BORDER_UNDEFINED, 42 NPP_BO		**
cellSize, 75 detectionWindowSize, 75 histogramBlockSize, 75 nHistogramBlockSize, 75 nPpiHamClassifiers NppiHaarClassifier, 32f, 74 NppiHamClassifiers NppiHaarClassifier, 32f, 74 NppiHamClassifiers NppiHaarClassifier, 32f, 74 NppiHamClassifiers NppiHamClassifier, 32f, 74 NppiHamClassifiers NppiHamClassifier, 32f, 74 NppiHamClassifiers NppiHamClassifier, 32f, 74 Npp	•	* *
detectionWindowSize, 75 histogramBlockSize, 75 nHistogramBlockSize, 75 nHistogramBlockSize, 75 NppiHuffmanTableType typedefs_npp, 42 NppiHarghantableType typedefs_npp, 42 NppiMaskSize NppiMaskSize NppiNorm typedefs_npp, 42 NppiNorm typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 NppiDoint, 76 x, 76 y, 76 nppiQuantFwdTableInit_JPEG_8u image_quantization, 65 nppiQuantIrwTableInit_JPEG_8u16u image_quantization, 66 nppiQuantIrwTableInit_JPEG_8u16u image_quantization, 66 nppiQuantIrwTableInit_JPEG_8u16u image_quantization, 66 NppiRect, 77 height, 77 width, 77 x, 77 y, 77 NppiSize, 78 height, 78 width, 78 NppLibraryVersion, 79 build, 79 major, 79 npinor, 79 NppPointPolar, 80 typedefs_npp, 46 numClassifiers NppiHarcTlassifier_32f, 74 NppiAlarClassifier_30f, 74  numIclassifiers NppiHarcTlassifier_32f, 74 NppiAlarClassifiers NppiHarcTlassifier_32f, 74 Quantization Functions, 61 NPP_ALIGNIE, 70 NPP_ALIGNIE, 80 typedefs_npp, 43 NPP_ALIGNIE, 80 typedefs_npp NPP_ALIGNIE, 80 typedefs_npp NPP_ALG_HINT_ACCURATE, 41 NPP_ALG_HINT_ACCURATE, 42 NPP_BORDER_WARNIN, 46 NPP_ALG_HINT_ACCURATE, 41 NPP_ALG_HINT_ACCURATE, 41 NPP_ALG_HINT_ACCURATE, 41 NPP_ALG_HINT_ACCURATE, 41 NPP_ALG_HINT_ACCURATE, 41 NPP_ALG_HINT_ACCURATE, 42 NPP_BORDER_WARNIN, 44 NPP_ALG_HINT_ACCURATE, 42 NPP_BORDER_WARNIN, 44 NP		** **
histogramBlockSize, 75 NppiHuffmanTableType typedefs_npp, 42 NppiInterpolationMode typedefs_npp, 42 NppiMaskSize typedefs_npp, 42 NppiNormIn typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiNormL6 x, 76 x, 76 y, 76 nppiQuantFwdRawTableInit_JPEG_8u image_quantization, 65 nppiQuantfwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 NppiRect, 77 height, 77 width, 77 width, 77 x, 77 y, 77 NppiSize, 78 height, 78 width, 78 width, 78 NppLibraryVersion, 79 build, 79 major, 79 major, 79 npiorupoudMode typedefs_npp, 43 nppSetStream numClassifiers NppiHaarClassifier_32f, 74 NppiHaarClassifier_32f, 74 Npp=ALIGN_61 NPP_ALIGN_16, 70 NPP_ALIGN_16, 70 NPP_ALIGN_16, 70 NPP_ALIGN_16, 70 NPP_ALIGN_16, 70 NPP_OND_INTORRECT WARNING, 46 NPP_ALIG_HINT_ACCURATE, 41 NPP_ALG_HINT_NONE, 41 NPP_ALG_HINT_NONE, 41 NPP_ALG_HINT_NONE, 41 NPP_ALG_HINT_NONE, 41 NPP_BAD_ARGUMENT_ERROR, 45 NPP_BORDER_CONSTANT, 42 NPP_BORDER_CONSTANT, 42 NPP_BORDER_MIRROR, 42 NPP_BORDER_NONE, 42 NPP_BORDER_REPLICATE, 42 NPP_BORDER_REPLICATE, 42 NPP_BORDER_WAAP, 42 NPP_BORDER_WAAP, 42 NPP_BORDER_WAAP, 42 NPP_BORDER_WAAP, 42 NPP_BORDER_REPLICATE, 42 NPP_BORDER_WAAP, 42 NPP_BORDER_REPLICATE, 40 NPP_CMP_LESS, 40 NPP_CONTEXT_MATCH_ERROR, 45 NPP_CORTENT_ERROR, 45 NPP_CORTENT_ERROR, 45 NPP_CUDA_1_1, 40 NPP_CUDA_1_1, 40 NPP_CUDA_1_2, 40		* *
nHistogramBins, 75 NppiHuffmanTableType typedefs_npp, 42 NppilnterpolationMode typedefs_npp, 42 NppiMaskSize typedefs_npp, 42 NppiMaskSize typedefs_npp, 43 NppiNorm typedefs_npp, 43 NppiNormL1 typedefs_npp, 43 NppiNormL2 typedefs_npp, 43 NppiNormL2 typedefs_npp, 43 NppiPointPolar, 80 NppPointPolar,		** **
NppiHuffmanTableType typedefs_npp, 42 NppiInterpolationMode typedefs_npp, 42 NppiMaskSize NppALIGN_8, 71, 72 NppiNorm typedefs_npp, 43 NppiNormInf typedefs_npp, 43 NppiNormL1 typedefs_npp, 43 NppiNormL2 typedefs_npp, 43 NppiPoint, 76 x, 76 y, 76 NppiQuantFwdRawTableInit_JPEG_8u image_quantization, 65 nppiQuantFwdTableInit_JPEG_8u16u image_quantization, 66 NppiQuantIvation in 66 NppiRect, 77 Neight, 77 NppiSize, 78 Neight, 78 NppLibraryVersion, 79 Duild, 79 major, 79 NppPointPolar, 80 NppPointPolar, 80 NppPointPolar, 80 NppPointPolar, 80 NppPointPolar, 80 NppPaLG_HINT_ACCURATE, 41 NPP_ALG_HINT_NONE, 41 NPP_ALG_HINT_NONE, 41 NPP_ALG_HINT_NONE, 41 NPP_ALG_HINT_NONE, 41 NPP_ALG_HINT_NONE, 41 NPP_ALGHINT_ERROR, 45 NPP_BORDER_CONSTANT, 42 NPP_BORDER_CONSTANT, 42 NPP_BORDER_MIRROR, 45 NPP_BORDER_NIRROR, 42 NPP_BORDER_NIRROR, 42 NPP_BORDER_WINDEFINED, 42 NPP_BORDER_UNDEFINED, 42 NPP_BORDER_UNDEFINED, 42 NPP_BORDER_UNDEFINED, 42 NPP_BORDER_UNDEFINED, 42 NPP_BORDER_UNDEFINED, 42 NPP_CMP_GREATER, 40 NPP_CMP_LESS, 20 NPP_CMP_LESS, 20 NPP_CMP_LESS, 40		
typedefs_npp, 42 NppilnterpolationMode typedefs_npp, 42 NppiMaskSize typedefs_npp, 42 NppiNorm typedefs_npp, 43 nppiNormInf typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 NppiPoint, 76 x. 76 y. 76 nppiQuantFwdRawTableInit_JPEG_8u image_quantization, 65 nppiQuantIwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantIwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantIwTableInit_JPEG_8u16u image_quantization, 66 NPP_BORDER_CONSTANT, 42 NPP_BORDER_CONSTANT, 42 NPP_BORDER_REPLICATE, 42 NPP_BORDER_REPLICATE, 42 NPP_BORDER_UNDEFINED, 42 NPP_BORDER_REPLICATE, 42 NPP_BORDER_REPLICATE, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_REPLICATE, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_REPLICATE, 45 NPP_CHANNEL_ORDER_ERROR, 45 NPP_CMP_GREATER, 40 NPP_CMP_GREATER, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 60 NPP_CORETFICIENT_ERROR, 45 NPP_CORETFICIENT_ERROR, 45 NPP_COREUPTED_DATA_ERROR, 45 NPP_CUDA_1_1, 40 NPP_CUDA_1_1, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_3, 40	•	NppiHaarClassifier_32f, 74
NppiInterpolationMode typedefs_npp, 42  NppiMaskSize typedefs_npp, 42  NppiNorm typedefs_npp, 43  nppiNormInf typedefs_npp, 43  nppiNormL1  typedefs_npp, 43  nppiNormL2  typedefs_npp, 43  nppiNormL2  typedefs_npp, 43  NppiPointPolar, 80  NppPointPolar, 80  NppPointPolar, 80  NppPointPolar, 80  NppPointPolar, 80  NppAIG_HINT_ACCURATE, 41  NPP_ALG_HINT_NONE, 41  NPP_ALG_HINT_NONE, 41  NPP_ALG_HINT_NONE, 41  NPP_ALG_HINT_NONE, 41  NPP_ALG_HINT_NONE, 41  NPP_ALG_HINT_ERROR, 45  nppiQuantIfwdTableInit_JPEG_8u16u image_quantization, 66  nppiQuantIfwdTableInit_JPEG_8u16u image_quantization, 66  NppiRect, 77  height, 77  width, 77  y, 77  NppiSize, 78  height, 78  width, 78  NppLibraryVersion, 79  build, 79  build, 79  major, 79  major, 79  major, 79  major, 79  NppRoundMode typedefs_npp, 43  NPP_CUDA_1_1, 40  NPP_CUDA_1_2, 40  NPP_CUDA_1_1, 40  NPP_CUDA_1_1, 40  NPP_CUDA_1_1, 40  NPP_CUDA_1_1, 40  NPP_CUDA_1_1, 40  NPP_CUDA_1_1, 40	**	
typedefs_npp, 42  NpiMaskSize typedefs_npp, 42  NppiNorm typedefs_npp, 43  nppiNormL1 typedefs_npp, 43  nppiNormL2 typedefs_npp, 43  NppiPointPolar, 80  NppPointPolar, 80  NppPointPola	**	Quantization Functions, 61
NppiMaskSize typedefs_npp, 42 NppiNorm typedefs_npp, 43 nppiNormInf typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiNormL3 nppiNormL4 typedefs_npp, 43 nppiNormL6 x, 76 x, 76 y, 76 nppiQuantFwdRawTableInit_JPEG_8u image_quantization, 65 nppiQuantIwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantIwTableInit_JPEG_8u16u image_quantization, 66 nppiQuantIwTableInit_JPEG_8u16u image_quantization, 66 nppiRect, 77 height, 77 width, 77 width, 77 width, 77 x, 77 y, 77 NppiSize, 78 height, 78 width, 78 NppiLibraryVersion, 79 build, 79 major, 79 minor, 79 Npp-CMP_GREATER_EQ, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_CONTENT_MATCH_ERROR, 45 NPP_CONTENT_ERROR, 45 NPP_CONTENT_ERROR, 45 NPP_CONTENT_ERROR, 45 NPP_CONTENT_ERROR, 45 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_CONTENT_ERROR, 45 NPP_CONTENT_AATCH_ERROR, 45 NPP_CON		
typedefs_npp, 42  NppiNorm typedefs_npp, 43 nppiNormInf typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 NppiPointPolar, 80  NppPointPolar, 80  NpPComPointPolar, 8	**	
NppiNorm typedefs_npp, 43 nppiNormInf typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 NppiPointPolar, 80 Npp_AFFINE_QUAD_INCORRECT WARNING, 46 NPP_ALIG_HINT_ACCURATE, 41 NPP_ALIG_HINT_ACCURATE, 41 NPP_ALIG_HINT_FAST, 41 NPP_ALIG_HINT_NONE, 41 NPP_ALIG_HINT_NONE, 41 NPP_ALIG_HINT_NONE, 41 NPP_ANCHOR_ERROR, 45 NPP_BAD_ARGUMENT_ERROR, 45 NPP_BAD_ARGUMENT_ERROR, 45 NPP_BORDER_CONSTANT, 42 NPP_BORDER_CONSTANT, 42 NPP_BORDER_NONE, 42 NPP_BORDER_NONE, 42 NPP_BORDER_UNDEFINED, 42 width, 77 npiSize, 78 height, 78 width, 78 width, 78 Npp_CHANNEL_ORDER_ERROR, 45 NPP_CHANNEL_ERROR, 45 NPP_CMP_EQ, 40 NPP_CMP_EQ, 40 NPP_CMP_EQ, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 20, 40 NPP_CMP_LESS, 20, 40 NPP_CMP_LESS, 20, 40 NPP_COP_LESS, 20, 40 NPP_C		
typedefs_npp, 43 nppiNormInf typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 NppiPoint, 76 x, 76 y, 76 nppiQuantFwdRawTableInit_JPEG_8u image_quantization, 65 nppiQuantFwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 nppiRomer, 77 height, 77 height, 77 width, 77 x, 77 y, 77 NppBoRDER_NONE, 42 NPP_BORDER_NONE, 42 NPP_BORDER_NONE, 42 NPP_BORDER_REPLICATE, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_UNDEFINED, 42 NPP_BORDER_UNDEFINED, 42 NPP_CHANNEL_ERROR, 45 NPP_CHANNEL_ERROR, 45 NPP_CHANNEL_GRERERROR, 45 NPP_CMP_EQ, 40 NPP_CMP_ESS_EQ, 40 NPP_CMP_LESS_EQ, 40 NPP_CMP_LESS_EQ, 40 NPP_CMP_LESS_EQ, 40 NPP_CONTEXT_MATCH_ERROR, 45 NPP_CUDA_1_1, 40 NPP_CUDA_1_1, 40 NPP_CUDA_1_1, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_3, 40	typedefs_npp, 42	
nppiNormInf typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 nppiPoint, 76 x, 76 y, 76 nppiQuantFwdRawTableInit_JPEG_8u image_quantization, 65 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 nppiDantInvTableInit_JPEG_8u16u npp_BoRDER_CONSTANT, 42 npp_BoRDER_CONSTANT, 42 npp_BoRDER_LNDEFINED, 42 npp_BoRDER_LNDEFINED, 42 npp_BoRDER_WRAP, 42 npp_BoRDER_WRAP, 42 npp_BoRDER_WRAP, 42 npp_BoRDER_WRAP, 42 npp_BoRDER_WRAP, 42 npp_BoRDER_WRAP, 42 npp_BoRDER_LNDEFINED, 40 npp_CHA_LSC, 40 npp_CHP_LESS, 40 npp_CHP_LESS, 40 npp_COLEFICITE, 40 npp_CUDA_1, 40 npp_CUDA_1, 40 npp_CUDA_1, 40 npp_CUDA_1, 40 npp_CUDA_1	NppiNorm	
typedefs_npp, 43 nppiNormL1 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 NppPointPolar, 80 typedefs_npp NPP_AFFINE_QUAD_INCORRECT WARNING, 46 NPP_ALG_HINT_ACCURATE, 41 NPP_ALG_HINT_ACST, 41 NPP_ALG_HINT_FAST, 41 NPP_ALG_HINT_NONE, 41 NPP_ALG_HINT_NONE, 41 NPP_ALG_HINT_ERROR, 44 NPP_ANCHOR_ERROR, 45 NPP_ANCHOR_ERROR, 45 NPP_BORDER_CONSTANT, 42 NPP_BORDER_CONSTANT, 42 NPP_BORDER_NONE, 42 NPP_BORDER_NONE, 42 NPP_BORDER_NONE, 42 NPP_BORDER_UNDEFINED, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_ERROR, 45 NPP_CHANNEL_ERROR, 45 NPP_CHANNEL_ERROR, 45 NPP_CHANNEL_ORDER_ERROR, 45 NPP_CHANNEL_ORDER_ERROR, 45 NPP_CHANNEL_ORDER_ERROR, 45 NPP_CMP_EQ, 40 NPP_CMP_EQ, 40 NPP_CMP_ESS, 40 NPP_CMP_GREATER, Q, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, EQ, 40 NPP_CMP_LESS, EQ, 40 NPP_CMP_LESS, EQ, 40 NPP_CONTEXT_MATCH_ERROR, 45 NPP_CODA_1_0, 40 NPP_CUDA_1_1, 40	typedefs_npp, 43	NppPointPolar, 80
nppiNormL1 typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 NppiPoint, 76 x, 76 y, 76 nppiQuantFwdRawTableInit_JPEG_8u image_quantization, 65 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 NppiRect, 77 height, 77 width, 77 x, 77 y, 77 NppiSize, 78 height, 78 width, 78 width, 78 width, 78 NppLibraryVersion, 79 build, 79 major, 79 nppPointPolar, 80 typedefs_npp NPP_ALG_HINT_ACCURATE, 41 NPP_ALG_HINT_FAST, 41 NPP_ALG_HINT_NONE, 41 NPP_ALG_HINT_SAT, 41 NPP_ALG_HINT_NONE, 41 NPP_ALG_HINT_NONE, 41 NPP_ALG_HINT_NONE, 41 NPP_ALG_HINT_NONE, 41 NPP_ALG_HINT_SAT, 41 NPP_ALG_HINT_ACCURATE, 41 NPP_ALG_HINT_ACCURATE, 41 NPP_ALG_HINT_ACCURATE, 41 NPP_ALG_HINT_ACCURATE, 41 NPP_ALG_HINT_ACCURATE, 41 NPP_BAG_HINT_ACCURATE, 41 NPP_BAG_HINT_ACCURATE, 41 NPP_BAG_HINT_NONE, 41 NPP_BAG_HINT_NONE, 41 NPP_BORDER_REROR, 45 NPP_BORDER_CONSTANT, 42 NPP_BORDER_CONSTANT, 42 NPP_BORDER_MIRROR, 42 NPP_BORDER_NONE, 42 NPP_BORDER_NONE, 42 NPP_BORDER_NONE, 42 NPP_BORDER_NONE, 42 NPP_BORDER_NONE, 42 NPP_BORDER_WAP, 42 NPP_BORDER_WAP, 42 NPP_BORDER_NONE, 42 NPP_BORDER_NONE, 42 NPP_BORDER_CONSTANT, 42 NPP_BORDER_NONE, 42 NPP_BORDER_MIRROR, 45 NPP_BORDER_CONSTANT, 42 NPP_BORDER	nppiNormInf	
typedefs_npp, 43 nppiNormL2 typedefs_npp, 43 NppiPoint, 76 x, 76 y, 76 nppiQuantFwdRawTableInit_JPEG_8u image_quantization, 65 nppiQuantIrwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantIrwTableInit_JPEG_8u16u image_quantization, 66 nppiQuantIrwTableInit_JPEG_8u16u image_quantization, 66 nppiQuantIrwTableInit_JPEG_8u16u image_quantization, 66 NppiRect, 77 height, 77 width, 77 width, 77 x, 77 NppiSize, 78 height, 78 width, 78 width, 78 width, 78 NppLibraryVersion, 79 build, 79 major, 79 major, 79 NppPointPolar, 80 rho, 80 theta, 80 NppRoundMode typedefs_npp, 43 nppSetStream core_npp, 29 NPP_CUDA_1_1, 40 NPP_CUDA_1_1, 40 NPP_CUDA_1_1, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_2, 40	typedefs_npp, 43	
nppiNormL2 typedefs_npp, 43 NppiPoint, 76 x, 76 y, 76 nppiQuantFwdRawTableInit_JPEG_8u image_quantization, 65 nppiQuantFwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 NPP_BORDER_CONSTANT, 42 NPP_BORDER_MIRROR, 42 NPP_BORDER_NONE, 42 NPP_BORDER_REPLICATE, 42 NPP_BORDER_REPLICATE, 42 NPP_BORDER_WRAP, 42 x, 77 NPP_BORDER_WRAP, 42 x, 77 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_CHANNEL_ERROR, 45 NPP_CHANNEL_ORDER_ERROR, 45 NPP_CMP_EQ, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_COMP_LESS,	nppiNormL1	
typedefs_npp, 43  NppiPoint, 76  x, 76  y, 76  nppiQuantFwdRawTableInit_JPEG_8u  image_quantization, 65  nppiQuantInvTableInit_JPEG_8u16u  image_quantization, 66  nppiQuantInvTableInit_JPEG_8u16u  image_quantization, 66  nppiQuantInvTableInit_JPEG_8u16u  image_quantization, 66  nppiQuantInvTableInit_JPEG_8u16u  image_quantization, 66  NPP_BORDER_NONE, 42  NPP_BORDER_REPLICATE, 42  NPP_BORDER_REPLICATE, 42  NPP_BORDER_NONE, 42  NPP_CHANNEL_ERROR, 45  NPP_CHANNEL_ORDER_ERROR, 45  NPP_CMP_EG, 40  NPP_CMP_LESS, 40  NPP_CMP_LESS, 40  NPP_CMP_LESS, 40  NPP_CMP_LESS, 40  NPP_CMP_LESS_EQ, 40  NPP_CMP_LESS_EQ, 40  NPP_CMP_LESS_EQ, 40  NPP_COMP_LESS_EQ, 40  NPP_COMP_LESS_EQ, 40  NPP_COMPLEND, 45  NPP_CONTEXT_MATCH_ERROR, 45  NPP_CONTEXT_MATCH_ERROR, 45  NPP_CUDA_1_0, 40  NPP_CUDA_1_0, 40  NPP_CUDA_1_1, 40  NPP_CUDA_1_1, 40  NPP_CUDA_1_2, 40  NPP_CUDA_1_2, 40  NPP_CUDA_1_3, 40	typedefs_npp, 43	- 11
NppiPoint, 76 x, 76 y, 76 nppiPoint, 76 x, 76 y, 76 nppiQuantFwdRawTableInit_JPEG_8u image_quantization, 65 nppiQuantFwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 NPP_BORDER_CONSTANT, 42 NPP_BORDER_NONE, 42 NPP_BORDER_NONE, 42 NPP_BORDER_REPLICATE, 42 NPP_BORDER_REPLICATE, 42 NPP_BORDER_WARP, 42 x, 77 NPP_BORDER_WARP, 42 NPP_CHANNEL_ERROR, 45 NPP_CHANNEL_ORDER_ERROR, 45 NPP_CMP_GREATER, 40 NPP_CMP_GREATER, 40 NPP_CMP_GREATER, 40 NPP_CMP_LESS, 40 major, 79 NPP_CMP_LESS, 40 nPP_CMP_LESS, EQ, 40 ninor, 79 NPP_CONTEXT_MATCH_ERROR, 45 NPP_CUDA_1_0, 40 NPP_CUDA_1_0, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_3, 40	nppiNormL2	
x, 76 y, 76 NPP_ALG_HINT_FAST, 41 nppiQuantFwdRawTableInit_JPEG_8u image_quantization, 65 nppiQuantFwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantIrvTableInit_JPEG_8u16u image_quantization, 66 nppiQuantIrvTableInit_JPEG_8u16u image_quantization, 66 nppiRect, 77 NPP_BORDER_NONE, 42 NPP_BORDER_NONE, 42 NPP_BORDER_NONE, 42 NPP_BORDER_WRAP, 42 width, 77 x, 77 NPP_BORDER_WRAP, 42 x, 77 NPP_BORDER_WRAP, 42 NPP_CHANNEL_ERROR, 45 NPP_CHANNEL_GREATER, 40 NPP_CMP_EQ, 40 NPP_CMP_GREATER, 40 NPP_CMP_GREATER_EQ, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS_EQ, 40 NPP_COP_LESS_EQ, 4	typedefs_npp, 43	
y, 76 nppiQuantFwdRawTableInit_JPEG_8u image_quantization, 65 nppiQuantFwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 nppiBorder_Mirror, 42 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 npp_Border_Mirror, 42 Npp_Border_None, 42 Npp_Channel_Error, 45 Npp_CMP_EQ, 40 Npp_CMP_EQ, 40 Npp_CMP_EQ, 40 Npp_CMP_ESS, 40 Npp_CMP_ESS, 40 Npp_CMP_LESS, 40 Npp_CMP_LESS, 40 Npp_CMP_LESS_EQ, 40 Npp_CMP_LESS_EQ, 40 Npp_CMP_LESS_EQ, 40 Npp_CMP_LESS_EQ, 40 Npp_COTERTOR_45 Npp	NppiPoint, 76	NPP_ALG_HINT_ACCURATE, 41
nppiQuantFwdRawTableInit_JPEG_8u image_quantization, 65 nppiQuantFwdTableInit_JPEG_8u16u nPP_ANCHOR_ERROR, 45 nppiQuantFwdTableInit_JPEG_8u16u nPP_BORDER_CONSTANT, 42 nppiQuantInvTableInit_JPEG_8u16u nPP_BORDER_MIRROR, 42 nppiQuantization, 66 nPP_BORDER_MIRROR, 42 nPP_BORDER_NONE, 42 nPP_BORDER_NONE, 42 nPP_BORDER_NONE, 42 nPP_BORDER_NONE, 42 nPP_BORDER_REPLICATE, 42 neight, 77 nPP_BORDER_UNDEFINED, 42 nPP_BORDER_WRAP, 42 nPP_BORDER_WRAP, 42 nPP_BORDER_WRAP, 42 nPP_BORDER_WRAP, 42 nPP_BORDER_WRAP, 42 nPP_BORDER_WRAP, 45 nPP_CHANNEL_ERROR, 45 nPP_CHANNEL_ORDER_ERROR, 45 nPP_CMP_EQ, 40 nPP_CMP_EQ, 40 nPP_CMP_EQ, 40 nPP_CMP_GREATER, 40 nPP_CMP_GREATER, 40 nPP_CMP_GREATER, 40 nPP_CMP_GREATER, 40 nPP_CMP_LESS, 40 nPP_COI_ERROR, 45 nPP_CORRUPTED_DATA_ERROR, 45 nPP_COI_ERROR, 45 nPP_CORRUPTED_DATA_ERROR, 45 nPP_CUDA_1_0, 40 nPP_CUDA_1_0, 40 nPP_CUDA_1_1, 40 nPP_CUDA_1_1, 40 nPP_CUDA_1_2, 40 nPP_CUDA_1_2, 40 nPP_CUDA_1_2, 40 nPP_CUDA_1_3, 40 nP	x, 76	NPP_ALG_HINT_FAST, 41
image_quantization, 65 nppiQuantFwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 NPP_BORDER_MIRROR, 42 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 NPP_BORDER_MIRROR, 42 NPP_BORDER_NONE, 42 NPP_BORDER_REPLICATE, 42 nPP_BORDER_WRAP, 42 x, 77 NPP_BORDER_WRAP, 42 x, 77 NPP_BORDER_WRAP, 42 x, 77 NPP_BORDER_WRAP, 42 x, 77 NPP_BOTH_AXIS, 41 NPP_CHANNEL_ERROR, 45 NPP_CHANNEL_ORDER_ERROR, 45 NPP_CMP_EQ, 40 NPP_CMP_EQ, 40 NPP_CMP_GREATER, 40 NPP_CMP_GREATER, 40 NPP_CMP_LESS, 40 major, 79 major, 79 nPP_CMP_LESS, 40 nPP_CMP_LESS, EQ, 40 minor, 79 NPP_COP_LESS, EQ, 40 NPP_COP_LESS, EQ,	y, 76	NPP_ALG_HINT_NONE, 41
nppiQuantFwdTableInit_JPEG_8u16u image_quantization, 66 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 nppiQuantInvTableInit_JPEG_8u16u image_quantization, 66 NPP_BORDER_MIRROR, 42 nppiRect, 77 NPP_BORDER_NONE, 42 NPP_BORDER_REPLICATE, 42 NPP_BORDER_UNDEFINED, 42 NPP_BORDER_WRAP, 42 x, 77 NPP_BORDER_WRAP, 42 x, 77 NPP_BORDER_WRAP, 42 x, 77 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_CHANNEL_ERROR, 45 NPP_CHANNEL_ORDER_ERROR, 45 NPP_CMP_EQ, 40 NPP_CMP_EQ, 40 NPP_CMP_EQ, 40 NPP_CMP_GREATER, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS_EQ, 40 ninor, 79 NPP_CMP_LESS_EQ, 40 ninor, 79 NPP_COMP_LESS_EQ, 40 NPP_CONTEXT_MATCH_ERROR, 45 nppRoundMode typedefs_npp, 43 nppSetStream NPP_CUDA_1_0, 40 NPP_CUDA_1_1, 40 nppSetStream NPP_CUDA_1_2, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_3, 40	nppiQuantFwdRawTableInit_JPEG_8u	NPP_ALIGNMENT_ERROR, 44
nppiQuantFwdTableInit_JPEG_8u16u image_quantization, 66 NPP_BORDER_CONSTANT, 42 nppiQuantInvTableInit_JPEG_8u16u NPP_BORDER_MIRROR, 42 image_quantization, 66 NPP_BORDER_NONE, 42 NPP_BORDER_NONE, 42 NPP_BORDER_REPLICATE, 42 NPP_BORDER_UNDEFINED, 42 width, 77 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 42 NPP_BORDER_WRAP, 45 NPP_CHANNEL_ERROR, 45 NPP_CHANNEL_ORDER_ERROR, 45 NPP_CHANNEL_ORDER_ERROR, 45 NPP_CMP_EQ, 40 NPP_CMP_EQ, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS_EQ, 40 NPP_CMP_LESS_EQ, 40 NPP_CMP_LESS_EQ, 40 NPP_COPFFICIENT_ERROR, 45 NPP_COPFFICIENT_ERROR, 45 NPP_CONTEXT_MATCH_ERROR, 45 NPP_CONTEXT_MATCH_ERROR, 45 NPP_CORRUPTED_DATA_ERROR, 45 NPP_CUDA_1_0, 40 NPP_CUDA_1_0, 40 NPP_CUDA_1_1, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_3, 40	image_quantization, 65	NPP_ANCHOR_ERROR, 45
nppiQuantInvTableInit_JPEG_8u16u		NPP_BAD_ARGUMENT_ERROR, 45
nppiQuantInvTableInit_JPEG_8u16u	image_quantization, 66	NPP_BORDER_CONSTANT, 42
NppiRect, 77 height, 77 height, 77 width, 77 NPP_BORDER_UNDEFINED, 42 NPP_BORDER_WRAP, 42 x, 77 NPP_BOTH_AXIS, 41 y, 77 NPP_CHANNEL_ERROR, 45 NPP_CHANNEL_ORDER_ERROR, 45 height, 78 width, 78 NPP_CMP_EQ, 40 width, 78 NPP_CMP_GREATER, 40 NPP_CMP_GREATER_EQ, 40 build, 79 major, 79 NPP_CMP_LESS, 40 minor, 79 NPP_CMP_LESS, 40 NPP_CMP_LESS_EQ, 40 minor, 79 NPP_COFFICIENT_ERROR, 45 NPP_COFFICIENT_ERROR, 45 NPP_CONTEXT_MATCH_ERROR, 45 theta, 80 NPP_CORRUPTED_DATA_ERROR, 45 NPP_CUDA_1_0, 40 typedefs_npp, 43 nppSetStream core_npp, 29 NPP_CUDA_1_2, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_3, 40		NPP_BORDER_MIRROR, 42
NppiRect, 77 height, 77 height, 77 width, 77 width, 77 x, 77 y, 77 Npp_BORDER_WRAP, 42 x, 77 y, 77 NppiSize, 78 height, 78 width, 78 Npp_CHANNEL_ERROR, 45 Npp_CHANNEL_ORDER_ERROR, 45 Npp_CMP_EQ, 40 Npp_CMP_EQ, 40 Npp_CMP_GREATER, 40 Npp_CMP_GREATER, 40 Npp_CMP_GREATER_EQ, 40 Npp_CMP_LESS, 40 major, 79 minor, 79 Npp_CMP_LESS, 40 Npp_COEFFICIENT_ERROR, 45 NppPointPolar, 80 rho, 80 theta, 80 Npp_CONTEXT_MATCH_ERROR, 45 NppRoundMode typedefs_npp, 43 nppSetStream core_npp, 29 NPP_CUDA_1_2, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_3, 40	image_quantization, 66	NPP_BORDER_NONE, 42
width, 77 x, 77 x, 77 y, 77 NPP_BOTH_AXIS, 41 y, 77 NPp_CHANNEL_ERROR, 45 NppiSize, 78 height, 78 width, 78 Npp_CMP_EQ, 40 width, 78 NppLibrary Version, 79 build, 79 major, 79 minor, 79 Npp_CMP_LESS, 40 minor, 79 NppPcomp_LESS_EQ, 40 minor, 79 NppPcomp_LESS_EQ, 40 NppPcomp_LESS_EQ, 40 NppComp_CMP_LESS_EQ, 40 NppComp_CMP_LESS_EQ, 40 NppComp_LESS_EQ, 40 NppComp_CMP_LESS_EQ, 40 NppComp_LESS_EQ, 40 NppComp_LESS	NppiRect, 77	NPP_BORDER_REPLICATE, 42
width, 77	height, 77	NPP_BORDER_UNDEFINED, 42
x, 77 y, 77 NPP_BOTH_AXIS, 41 y, 77 NPpiSize, 78 NPp_CHANNEL_ERROR, 45 NPP_CHANNEL_ORDER_ERROR, 45 NPP_CMP_EQ, 40 width, 78 NPP_CMP_EQ, 40 NPP_CMP_GREATER, 40 NPP_CMP_GREATER, EQ, 40 build, 79 major, 79 npp_CMP_LESS, 40 minor, 79 NPP_CMP_LESS_EQ, 40 ninor, 79 NPP_COEFFICIENT_ERROR, 45 NPP_COI_ERROR, 45 rho, 80 theta, 80 NPP_CONTEXT_MATCH_ERROR, 45 NPP_CORRUPTED_DATA_ERROR, 45 NPP_CUDA_1_0, 40 typedefs_npp, 43 nppSetStream core_npp, 29 NPP_CUDA_1_3, 40	_	NPP_BORDER_WRAP, 42
y, 77 NppiSize, 78 NppiSize, 78 NppiCHANNEL_ORDER_ERROR, 45 NPP_CHANNEL_ORDER_ERROR, 45 NPP_CMP_EQ, 40 NPP_CMP_EQ, 40 NPP_CMP_GREATER, 40 NPP_CMP_GREATER_EQ, 40 NPP_CMP_GREATER_EQ, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS_EQ, 40 NPP_CMP_LESS_EQ, 40 NPP_COEFFICIENT_ERROR, 45 NPP_COI_ERROR, 45 NPP_COI_ERROR, 45 NPP_CONTEXT_MATCH_ERROR, 45 NPP_CONTEXT_MATCH_ERROR, 45 NPP_CUDA_1_0, 40 NPP_CUDA_1_1, 40 NPP_CUDA_1_1, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_3, 40 NPP_CUDA_1_3, 40		NPP_BOTH_AXIS, 41
NppiSize, 78 height, 78 height, 78 width, 78 Npp_CMP_EQ, 40 Npp_CMP_GREATER, 40 NppLibraryVersion, 79 build, 79 major, 79 minor, 79 Npp_CMP_LESS, 40 Npp_CMP_LESS_EQ, 40 NPP_CMP_LESS_EQ, 40 NPP_CMP_LESS_EQ, 40 NPP_COEFFICIENT_ERROR, 45 NppPointPolar, 80 rho, 80 rho, 80 theta, 80 Npp_CONTEXT_MATCH_ERROR, 45 NppRoundMode typedefs_npp, 43 nppSetStream core_npp, 29 NPP_CUDA_1_1, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_3, 40	· ·	NPP_CHANNEL_ERROR, 45
height, 78 width, 78 NPP_CMP_EQ, 40 NPP_CMP_GREATER, 40 NPP_Library Version, 79 NPP_CMP_GREATER_EQ, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS, 40 NPP_CMP_LESS_EQ, 40 NPP_CMP_LESS_EQ, 40 NPP_COEFFICIENT_ERROR, 45 NPP_COI_ERROR, 45 NPP_COI_ERROR, 45 NPP_CONTEXT_MATCH_ERROR, 45 NPP_CONTEXT_MATCH_ERROR, 45 NPP_CORRUPTED_DATA_ERROR, 45 NPP_CUDA_1_0, 40 NPP_CUDA_1_1, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_3, 40 NPP_CUDA_1_3, 40		
width, 78  NPP_CMP_GREATER, 40  NPP_LibraryVersion, 79  build, 79  major, 79  minor, 79  NPP_CMP_LESS, 40  NPP_CMP_LESS, 40  NPP_CMP_LESS_EQ, 40  NPP_CMP_LESS_EQ, 40  NPP_COEFFICIENT_ERROR, 45  NPP_COI_ERROR, 45  rho, 80  NPP_CONTEXT_MATCH_ERROR, 45  theta, 80  NPP_CORRUPTED_DATA_ERROR, 45  NPP_CUDA_1_0, 40  typedefs_npp, 43  nppSetStream  core_npp, 29  NPP_CUDA_1_3, 40		NPP CMP EQ, 40
NppLibraryVersion, 79 build, 79 nmajor, 79 major, 79 nppComp_Less, 40 nminor, 79 Npp_Comp_Less_EQ, 40 Npp_Comp_Less_EQ, 40 Npp_Comp_Less_EQ, 40 NppPcontPolar, 80 NppPcol_Error, 45 NppPointPolar, 80 Npp_Context_Match_Error, 45 theta, 80 Npp_Correst_Match_Error, 45 NppRoundMode Npp_Correst_Data_Error, 45 NppRoundMode Npp_Cuda_1_0, 40 Npp_Cuda_1_1, 40 NppSetStream Npp_Cuda_1_2, 40 Npp_Cuda_1_3, 40 Npp_Cuda_1_3, 40	_	=
build, 79 major, 79 minor, 79 minor, 79 NPP_CMP_LESS_EQ, 40 NPP_COEFFICIENT_ERROR, 45 NPP_COI_ERROR, 45 NPP_COI_ERROR, 45 NPP_CONTEXT_MATCH_ERROR, 45 theta, 80 NPP_CORRUPTED_DATA_ERROR, 45 NPP_CUDA_1_0, 40 typedefs_npp, 43 nppSetStream core_npp, 29 NPP_CUDA_1_3, 40		
major, 79 minor, 79 minor, 79 NPP_CMP_LESS_EQ, 40 NPP_COEFFICIENT_ERROR, 45 NPPP_COI_ERROR, 45 NPP_COI_ERROR, 45 NPP_CONTEXT_MATCH_ERROR, 45 theta, 80 NPP_CORRUPTED_DATA_ERROR, 45 NPP_CUDA_1_0, 40 typedefs_npp, 43 nppSetStream core_npp, 29 NPP_CUDA_1_2, 40 NPP_CUDA_1_3, 40		
minor, 79 NPP_COEFFICIENT_ERROR, 45 NPP_COI_ERROR, 45 NPP_COI_ERROR, 45 NPP_CONTEXT_MATCH_ERROR, 45 theta, 80 NPP_CORRUPTED_DATA_ERROR, 45 NPP_CUDA_1_0, 40 typedefs_npp, 43 nppSetStream nPP_CUDA_1_1, 40 NPP_CUDA_1_2, 40 core_npp, 29 NPP_CUDA_1_3, 40		
NppPointPolar, 80 rho, 80 npp_COI_ERROR, 45 npp_CONTEXT_MATCH_ERROR, 45 nppRoundMode typedefs_npp, 43 nppSetStream core_npp, 29 NPP_CUDA_1_3, 40 NPP_CUDA_1_3, 40 NPP_CUDA_1_3, 40		
rho, 80 theta, 80 NPP_CONTEXT_MATCH_ERROR, 45 NPP_CORRUPTED_DATA_ERROR, 45 NppRoundMode typedefs_npp, 43 NPP_CUDA_1_0, 40 NPP_CUDA_1_1, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_2, 40 NPP_CUDA_1_3, 40 NPP_CUDA_1_3, 40		
theta, 80 NPP_CORRUPTED_DATA_ERROR, 45 NppRoundMode NPP_CUDA_1_0, 40 typedefs_npp, 43 NPP_CUDA_1_1, 40 nppSetStream NPP_CUDA_1_2, 40 core_npp, 29 NPP_CUDA_1_3, 40		
NppRoundMode         NPP_CUDA_1_0, 40           typedefs_npp, 43         NPP_CUDA_1_1, 40           nppSetStream         NPP_CUDA_1_2, 40           core_npp, 29         NPP_CUDA_1_3, 40		
typedefs_npp, 43       NPP_CUDA_1_1, 40         nppSetStream       NPP_CUDA_1_2, 40         core_npp, 29       NPP_CUDA_1_3, 40		
nppSetStream NPP_CUDA_1_2, 40 core_npp, 29 NPP_CUDA_1_3, 40	* *	
core_npp, 29 NPP_CUDA_1_3, 40		
••		
111 F_CUDA_2_0, 40	* *	
	1.Ppommo	1111_00011_2_0, 40

NPP_CUDA_2_1, 40	NPP_MEMFREE_ERROR, 44
NPP_CUDA_3_0, 40	NPP_MEMORY_ALLOCATION_ERR, 45
NPP_CUDA_3_2, 40	NPP_MEMSET_ERROR, 44
NPP_CUDA_3_5, 40	NPP_MIRROR_FLIP_ERROR, 45
NPP_CUDA_3_7, 40	NPP_MISALIGNED_DST_ROI_WARNING,
NPP_CUDA_5_0, 40	46
NPP_CUDA_5_2, 40	NPP_MOMENT_00_ZERO_ERROR, 45
NPP_CUDA_5_3, 40	NPP_NO_ERROR, 45
NPP_CUDA_6_0, 40	NPP_NO_MEMORY_ERROR, 45
NPP_CUDA_6_1, 40	NPP_NO_OPERATION_WARNING, 45
NPP_CUDA_6_2, 40	NPP_NOT_EVEN_STEP_ERROR, 44
NPP_CUDA_6_3, 40	NPP_NOT_IMPLEMENTED_ERROR, 45
NPP_CUDA_7_0, 40	NPP_NOT_SUFFICIENT_COMPUTE
NPP_CUDA_KERNEL_EXECUTION	CAPABILITY, 44
ERROR, 44	NPP_NOT_SUPPORTED_MODE_ERROR,
NPP_CUDA_NOT_CAPABLE, 40	44
NPP_CUDA_UNKNOWN_VERSION, 40	NPP_NULL_POINTER_ERROR, 45
NPP_DATA_TYPE_ERROR, 45	NPP_NUMBER_OF_CHANNELS_ERROR,
NPP_DIVIDE_BY_ZERO_ERROR, 45	45
NPP DIVIDE BY ZERO WARNING, 46	
	NPP_OUT_OFF_RANGE_ERROR, 45
NPP_DIVISOR_ERROR, 45	NPP_OVERFLOW_ERROR, 44
NPP_DOUBLE_SIZE_WARNING, 46	NPP_QUADRANGLE_ERROR, 45
NPP_ERROR, 45	NPP_QUALITY_INDEX_ERROR, 44
NPP_ERROR_RESERVED, 45	NPP_RANGE_ERROR, 45
NPP_FFT_FLAG_ERROR, 45	NPP_RECTANGLE_ERROR, 45
NPP_FFT_ORDER_ERROR, 45	NPP_RESIZE_FACTOR_ERROR, 45
NPP_FILTER_SCHARR, 42	NPP_RESIZE_NO_OPERATION_ERROR,
NPP_FILTER_SOBEL, 42	44
NPP_HAAR_CLASSIFIER_PIXEL	NPP_RND_FINANCIAL, 43
MATCH_ERROR, 44	NPP_RND_NEAR, 43
NPP_HISTOGRAM_NUMBER_OF	NPP_RND_ZERO, 44
LEVELS_ERROR, 44	NPP_ROUND_MODE_NOT
NPP_HORIZONTAL_AXIS, 41	SUPPORTED_ERROR, 44
NPP_INTERPOLATION_ERROR, 45	NPP_ROUND_NEAREST_TIES_AWAY
NPP_INVALID_DEVICE_POINTER	FROM_ZERO, 44
ERROR, 44	NPP_ROUND_NEAREST_TIES_TO_EVEN,
NPP_INVALID_HOST_POINTER_ERROR,	43
44	NPP_ROUND_TOWARD_ZERO, 44
NPP_LUT_NUMBER_OF_LEVELS	NPP_SCALE_RANGE_ERROR, 45
ERROR, 45	NPP_SIZE_ERROR, 45
NPP_LUT_PALETTE_BITSIZE_ERROR, 44	NPP_STEP_ERROR, 45
NPP_MASK_SIZE_11_X_11, 43	NPP_STRIDE_ERROR, 45
NPP_MASK_SIZE_13_X_13, 43	NPP_SUCCESS, 45
NPP MASK SIZE 15 X 15, 43	NPP_TEXTURE_BIND_ERROR, 44
NPP_MASK_SIZE_1_X_3, 43	NPP_THRESHOLD_ERROR, 45
NPP_MASK_SIZE_1_X_5, 43	NPP_THRESHOLD_NEGATIVE_LEVEL
NPP_MASK_SIZE_3_X_1, 43	ERROR, 45
NPP_MASK_SIZE_3_X_3, 43	NPP_VERTICAL_AXIS, 41
NPP_MASK_SIZE_5_X_1, 43	NPP_WRONG_INTERSECTION_QUAD
NPP_MASK_SIZE_5_X_5, 43	WARNING, 46
NPP_MASK_SIZE_7_X_7, 43	NPP_WRONG_INTERSECTION_ROI
NPP_MASK_SIZE_9_X_9, 43	ERROR, 44
NPP_MASK_SIZE_ERROR, 45	NPP_WRONG_INTERSECTION_ROI
NPP_MEMCPY_ERROR, 44	WARNING, 46

NPP_ZC_MODE_NOT_SUPPORTED	NPP_MAX_64S, 38
ERROR, 44	NPP_MAX_64U, 38
NPP_ZERO_MASK_VALUE_ERROR, 45	NPP_MAX_8S, 38
NPPI_BAYER_BGGR, 41	NPP_MAX_8U, 38
NPPI_BAYER_GBRG, 41	NPP_MAXABS_32F, 38
NPPI_BAYER_GRBG, 41	NPP_MAXABS_64F, 39
NPPI_BAYER_RGGB, 41	NPP_MIN_16S, 39
NPPI_INTER_CUBIC, 42	NPP_MIN_16U, 39
NPPI_INTER_CUBIC2P_B05C03, 42	NPP_MIN_32S, 39
NPPI_INTER_CUBIC2P_BSPLINE, 42	NPP_MIN_32U, 39
NPPI_INTER_CUBIC2P_CATMULLROM,	NPP MIN 64S, 39
42	NPP_MIN_64U, 39
NPPI_INTER_LANCZOS, 42	NPP_MIN_8S, 39
NPPI_INTER_LANCZOS3_ADVANCED, 42	NPP_MIN_8U, 39
NPPI_INTER_LINEAR, 42	NPP_MINABS_32F, 39
NPPI_INTER_NN, 42	NPP_MINABS_64F, 39
NPPI_INTER_SUPER, 42	NppCmpOp, 40
NPPI INTER UNDEFINED, 42	NppGpuComputeCapability, 40
NPPI_OP_ALPHA_ATOP, 41	NppHintAlgorithm, 40
NPPI_OP_ALPHA_ATOP_PREMUL, 41	NppiAlphaOp, 41
NPPI_OP_ALPHA_IN, 41	NppiAxis, 41
NPPI_OP_ALPHA_IN_PREMUL, 41	NppiBayerGridPosition, 41
NPPI_OP_ALPHA_OUT, 41	NppiBorderType, 41
NPPI_OP_ALPHA_OUT_PREMUL, 41	NppiDifferentialKernel, 42
NPPI_OP_ALPHA_OVER, 41	NppiHuffmanTableType, 42
	NppiInterpolationMode, 42
NPPI_OP_ALPHA_OVER_PREMUL, 41	
NPPI_OP_ALPHA_PLUS, 41	NppiMaskSize, 42
NPPI_OP_ALPHA_PLUS_PREMUL, 41	NppiNorm, 43
NPPI_OP_ALPHA_PREMUL, 41	NppRoundMode, 43
NPPI_OP_ALPHA_XOR, 41	NppStatus, 44
NPPI_OP_ALPHA_XOR_PREMUL, 41	NppsZCType, 46
NPPI_SMOOTH_EDGE, 42	: dth
nppiACTable, 42	width
nppiDCTable, 42	NppiRect, 77
nppiNormInf, 43	NppiSize, 78
nppiNormL1, 43	v
nppiNormL2, 43	X NaniDoint 76
nppZCC, 46	NppiPoint, 76
nppZCR, 46	NppiRect, 77
nppZCXor, 46	V
typedefs_npp	y NppiPoint, 76
NPP_HOG_MAX_BINS_PER_CELL, 37	NppiRect, 77
NPP_HOG_MAX_BLOCK_SIZE, 37	rypircet, 77
NPP_HOG_MAX_CELL_SIZE, 37	
NPP_HOG_MAX_CELLS_PER	
DESCRIPTOR, 37	
NPP_HOG_MAX_DESCRIPTOR	
LOCATIONS_PER_CALL, 38	
NPP_HOG_MAX_OVERLAPPING	
BLOCKS_PER_DESCRIPTOR, 38	
NPP_MAX_16S, 38	
NPP_MAX_16U, 38	
NPP_MAX_32S, 38	
NPP_MAX_32U, 38	