

APPLICATION NOTE

# Using the Goldeye G/CL look-up table (LUT) for image processing

## Introduction

This application note explains how to use look-up tables (LUTs) for image processing. The Goldeye camera family provides four pre-configured and four user-configurable LUT files.

2018-Sep-12

## Basics about LUTs

A LUT is used to remap pixel counts. For every possible pixel value (e.g. [0:16383] for 14 bit) within the LUT a target value exists where the pixel value is mapped to.



Figure 1: LUT applied to a set of pixels

The LUT consists of a series  $a_i$  of 2<sup>14</sup> (= 16384) count values, therefore the mapping is defined to map each pixel of value i to value  $a_i$ .

Expressed in pure mathematically terms: May  $(a_i)_{i \in N}$  be a series of integers with  $(a_i)_{i \in N} \in M$ 

with  $N = [0: 16383] \subset \mathbb{N}_0$  and  $N_1, M \subseteq N$ .

The LUT is defined by the mapping  $\mathcal{L}: N_1 \to M, \, b \to a_b$  .

Remarks: b and  $a_b$  are the input and resulting output pixel values of the LUT respectively. The mapping may not necessarily be surjective.

The mathematical definition above as well as Figure 1 show that the LUT must contain 16383 values to define the mapping for each possible grey value. The values must be integer values within the range



[0:16383]. However, pixels of different gray input values may be mapped to identical gray output values, for example a LUT may contain the same value more than once (see Figure 1) or not at all.

# Usage of LUTs

The LUT is the last part of the Goldeye image processing chain. Other elements within the image processing chain are, for instance, the Non-Uniformity Correction (NUC) and the Defective Pixel Correction (DPC).

Therefore the LUT is processed in the full bit depth of 14 bits. Conversion to the 8-bit output pixel format takes place only after processing of the whole chain including the LUT. For more information about the image processing chain, refer to the Goldeye technical manual.

## LUT features

The LUT is controlled by the GenlCam features shown in Figure 2. Use the features as described below.

- To select a LUT data set, enter the LUT number into LUTDatasetSelector.
- To load a data set internally to make it ready for the image processing chain, use the command LUTDatasetLoad.
- The currently loaded data set is displayed by LUTDatasetActive.
- To apply the LUT, set LUTEnable true.
- Find the source (From) defined by the currently loaded LUT in LUTIndex



• LUTValueAll is a raw (or register) feature to access the corresponding binary data.

## Predefined Goldeye LUTs

Goldeye cameras provide four predefined, immutable LUTs, as shown in Table 1.

The inverting LUT maps each pixel value i to j = 16383 - i. The remaining three LUTs apply the listed gamma corrections to the image.

Pre	defined	Cust	tomizable		
LUT	Effect	LUT	Effect		
0	Inverting	4	(none)		
1	Gamma 1.16	5	(none)		
2	Gamma 1.18	6	(none)		
3	Gamma 1.20	7	(none)		

Table 1: LUT factory settings

For the predefined LUTs, the **LUTDatasetSave** feature is not applicable as the data sets allow read access only.

For the customizable LUTs, each value may be filled with arbitrary numbers, as described below.



Figure 2: GenICam features that control the LUT



# Data structure of a LUT

The LUTs of the Goldeye are given by binary 14-bit values, for example 2 bytes (= 16 bit) for each entry. Therefore, each LUT has always a constant size of 16384 x 2 bytes = 32768 bytes. The byte order is little endian: the least significant bytes are stored first, the most significant bytes are stored last. Only the result values of the LUT table (refer to Figure 1) are saved successively in memory. The From values are given by the position of the 2 bytes within the binary LUT data, as shown in Figure 3.

### Example

This example is marked with blue frames in Figure 3.

- The 10th value of LUT 0 is *16374*, which is the inverted value of *9*.
- The hexadecimal value of *16374* is *0x3FF6*.
- Because of the little-endianness, the value is stored as *F6 3F* in the LUT.

	ASCII	Binary																			
	А																				
1	16383																				
2	16382																				
3	16381		N							R	aw	Dat	a Eo	dito	r <l< td=""><td>UT\</td><td>/alu</td><td>eAll</td><td>&gt;</td><td>_ 🗆</td><td>×</td></l<>	UT\	/alu	eAll	>	_ 🗆	×
4	16380		<b>E</b> 11 <b>F</b>																		
5	16379		File t	dit																	
6	16378																				
7	16377					_															_
8	16376		0000	ff	3f	fe	3f	fd	3f	fc	3f	fb	3f	fa	3f	f9	3f	f8	3f	.?.?.?.?.?.?.?.?	<b>_</b>
9	16375		0010	£7	3f	f6	3f	f5	3f	f4	3f	f3	3f	f2	3f	f1	3f	fO	3f	.?.?.?.?.?.?.?	_
10	16374		0020	ef	3f	ee	3f	ed	3f	ec	3f	eb	3f	ea	3f	e9	3f	e8	3f	.?.?.?.?.?.?.?	
11	16373		0030	e7 df	31	e6 de	31	e5	31	e4	31	e3 db	31	e2	31	e1	31	e0	31	.?.?.?.?.?.?.?	
12	16372		0040	d7	31	de	3£	da	31	d2	31	43	31	da da	31	d9	31	40	35	2 2 2 2 2 2 2 2 2 2	
13	16371		0060	cf.	3f	ce	3f	cd	3f	cc	3f	cb	3f	ca	3f	c9	3f	c8	3f		
14	16370		0070	c7	3f	c6	3f	c5	3f	c4	3f	c3	3f	c2	3f	c1	3f	c0	3f	.?.?.?.?.?.?.?.?	
15	16369		0080	bf	3f	be	3f	bd	3f	bc	3f	bb	3f	ba	3f	b9	3f	b8	3f	.?.?.?.?.?.?.?.?	
16	16368		0090	b7	3f	b6	3f	b5	3f	b4	3f	b3	3f	b2	3f	b1	3f	b0	3f	.?.?.?.?.?.?.?.?	
17	16367		00a0	af	3f	ae	3f	ad	3f	ac	3f	ab	3f	aa	3f	a9	3f	<b>a</b> 8	3f	.?.?.?.?.?.?.?	
18	16366		00Ъ0	a7	3f	a6	3f	<b>a</b> 5	3f	a4	3f	a3	3f	a2	3f	a1	3f	<b>a</b> 0	3f	.?.?.?.?.?.?.?.?	
19	16365		00c0	9f	3f	9e	3f	9d	3f	9c	3f	9b	3f	9a	3f	99	3f	98	3f	.?.?.?.?.?.?.?	
20	16364		0000	97	3f	96	3f	95	3f	94	3f	93	3f	92	3f	91	3f	90	3f	.?.?.?.?.?.?.?	
21	16363		0000	8I 07	31	8e	31	80	31	80	31	80	31	8a 02	31	89	31	88	31	. ? . ? . ? . ? . ? . ? . ?	
22	16362		0100	7.5	31	70	31	74	31	70	31	03 7h	31	02 7a	31	79	31	78	34	· [ · [ · [ · [ · [ · [ · [ · [ · [ · ] · ]	
23	16361		0110	77	3f	76	3f	75	3f	74	3f	73	3f	72	3f	71	3f	70	3f	w?v?u?t?s?r?g?p?	
24	16360		0120	6f	3f	6e	3f	6d	3f	6c	3f	6b	3f	6a	3f	69	3f	68	3f	o?n?m?l?k?j?i?h?	
25	16359		0130	67	3f	66	3f	65	3f	64	3f	63	3f	62	3f	61	3f	60	3f	g?f?e?d?c?b?a?`?	
26	16358		0140	5f	3f	5e	3f	5d	3f	5c	3f	5b	3f	5a	3f	59	3f	58	3f	_?^?]?\?[?Z?Y?X?	
27	16357		0150	57	3f	56	3f	55	3f	54	3f	53	3f	52	3f	51	3f	50	3f	W?V?U?T?S?R?Q?P?	
28	16356		0160	4f	3f	4e	3f	4d	3f	4c	3f	4b	3f	4a	3f	49	3f	48	3f	O?N?M?L?K?J?I?H?	-1
29	16355		0170	47	٩f	46	٩f	45	٩f	44	٩f	43	٩f	42	٩f	41	٩f	40	3f	Model Outsuits	
30	16354												Addre	ess:	0			jsize:	327	Mode:    Overwrite	
31	16353																				

Figure 3: Predefined Goldeye LUT No. 0 (= inversing LUT) in ASCII and binary representation



# Uploading a LUT to the camera

Three ways are available to upload LUT files to the camera

- set the ASCII LUT value for each index one by one
- upload the binary data at once
- upload the binary data by direct file access.

#### Set the ASCII LUT value for each index

To set individual ASCII LUT values, follow the steps below.

- Step 1: Select one of the four user configurable LUTs by setting LUTDatasetSelector to a value between *4* and *7*.
- Step 2: Load the LUT by calling the command LUTDatasetLoad. LUTDatasetActive signals which LUT is currently loaded within the camera and ready to be applied or modified.
- Step 3: Use **LUTIndex** (as explained under "LUT features" on page 2) to define a From value of the lookup table you would like to modify.
- Step 4: Now you can set the table entry or To value by **LUTValue**. Repeat this for all LUT entries that need to be modified.
- Step 5: Finally, save the LUT data set by calling the LUTDatasetSave command.

Compared to the direct binary register access (explained below), this approach takes much longer to write or read the LUT data.

#### Upload binary LUT data at once

- Step 1: Select one of the four user configurable LUTs by setting LUTDatasetSelector to a value between *4* and *7*.
- Step 2: Load the LUT by calling the command LUTDatasetLoad. LUTDatasetActive signals which LUT is currently loaded within the camera and ready to be applied or modified.
- Step 3: You can access the binary data of the LUT by the raw feature (or register) LUTValueAll. This register access allows to modify the data by direct memory access.

#### Using the Vimba SDK

In case the Allied Vision Vimba SDK is used, the binary LUT data would be stored in a variable of type **UcharVector** of size 32768 (2 bytes for 16384 LUT entries) in little endian byte order. Access the feature as shown below.

```
// for the sake of simplicity error handling has been omitted
// start Vimba
VimbaSystem &sys = VimbaSystem::GetInstance();
sys.Startup();
```

Code Example 1: (sheet 1 of 2)

```
// get pointers to connected cameras
CameraPtrVector vpCamera;
sys.GetCameras( vpCamera );
// open first cam
vpCamera[0]->Open( VmbAccessModeFull );
UcharVector LUTdata( 32768,0 ); // vector containing binary LUT data -> 32768 bytes
// in this case all bytes have been initialized to 0
// fill vector with binary LUT data
// ...
FeaturePtr feature; // pointer for feature access
vpCamera[0]->GetFeatureByName( "LUTValueAll", feature ); // get feature
feature->SetValue( LUTdata ); // upload LUT data
vpCamera[0]->GetFeatureByName( "LUTDatasetSave", feature ); // get feature
feature-> RunCommand(); // save LUT data
// The data can also be read from the memory by //
vpCamera[0]->GetFeatureByName("LUTDatasetLoad", feature ); // get feature
feature-> RunCommand(); // load LUT data
vpCamera[0]->GetFeatureByName( "LUTValueAll", feature ); // get feature
feature->GetValue( data ); // download LUT from camera to UcharVector data
```

Code Example 1: (sheet 2 of 2)

When finished, save the LUT data set by calling the LUTDatasetSave command. You can also read the data from the memory using the following command.

feature->GetValue( data ); // download LUT from camera to UcharVector data

## Upload binary data by direct file access

Uploading binary data by direct file access is done with the help of a small program. You can obtain the program from the Allied Vision support team. They also will help you with the use of that program. Contact the Allied Vision support under support@alliedvision.com.

# Programming examples

For more information on how to use LUTs with Goldeye cameras, please contact the Allied Vision support team at <a href="mailto:support@alliedvision.com">support@alliedvision.com</a>. We also offer programming examples on request that shows a basic implementation of all three LUT access methods mentioned above based on our Vimba SDK.



#### Copyright and trademarks

All texts, pictures and graphics are protected by copyright and other laws protecting intellectual property. All content is subject to change without notice.

All trademarks, logos, and brands cited in this document are property and/or copyright material of their respective owners. Use of these trademarks, logos, and brands does not imply endorsement.

Copyright © 2018 Allied Vision GmbH. All rights reserved.