APPLICATION NOTE

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

2018-Sep-12

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.

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.

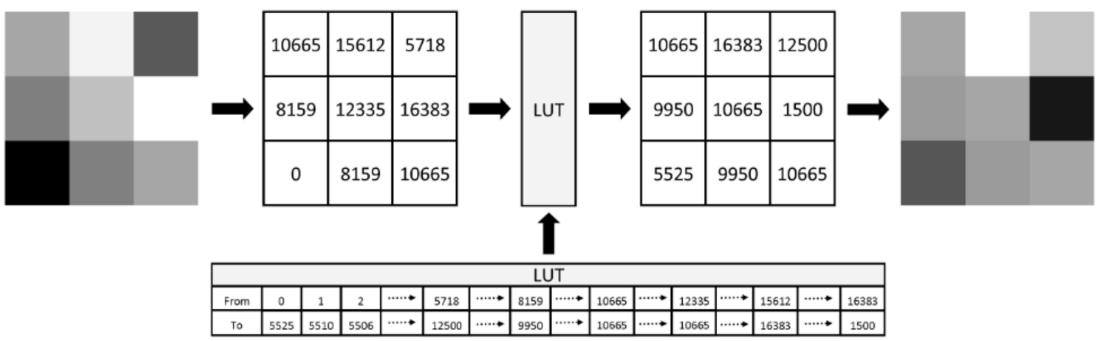

Figure 1: LUT applied to a set of pixels

The LUT consists of a series $\boldsymbol{a_i}$ of 2^{14} (= 16384) count values, therefore the mapping is defined to map each pixel of value $\boldsymbol{i}$ to value $\boldsymbol{a_i}$.

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 GenICam 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`**
- Find the target (To) value defined by the currently loaded LUT in **`LUTValue`** .
- **`LUTValueAll`** is a raw (or register) feature to access the corresponding binary data.

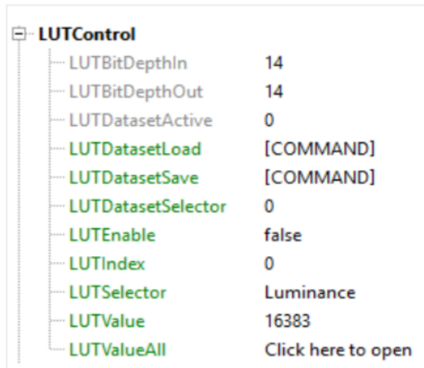

Figure 2: GenICam features that control the LUT

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.

Predefined		**Customizable**	
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.

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.

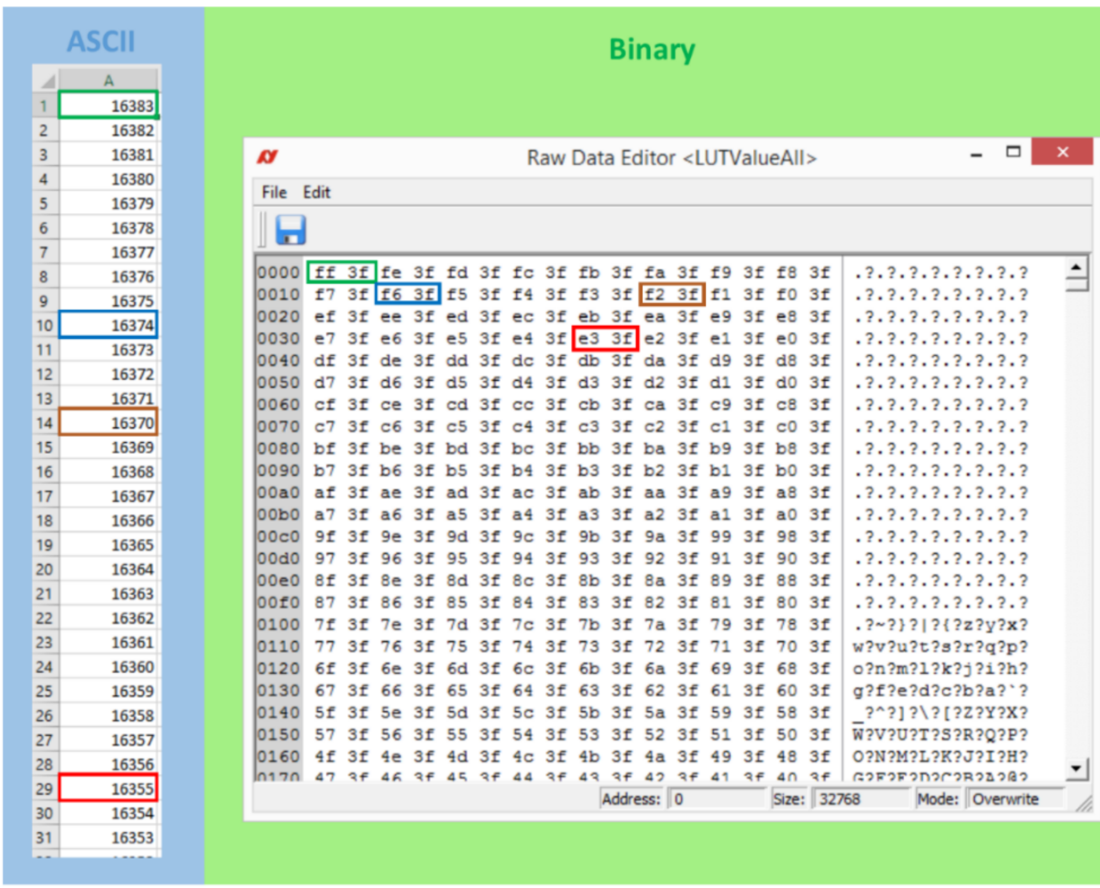

Figure 3: Predefined Goldeye LUT No. 0 (= inversing LUT) in ASCII and binary representation

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 support@alliedvision.com. 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.