Intel® Media RAW Accelerator

Developer Reference

API Version 1.23

LEGAL DISCLAIMER

Table of Contents

Overview

The Intel® Media SDK RAW Media Accelerator is a development library that exposes the media acceleration capabilities of Intel platforms for processing RAW data in Bayer format from camera and conversion to monitor compatible formats. The API library covers a wide range of Intel platforms.

The Intel® Media SDK RAW Media Accelerator a hardware accelerated (HW) plug-in library exposing RAW media acceleration capabilities, implemented as Intel® Media SDK VPP plug-in. The plug-in can be loaded into/used with only Intel Media SDK Hardware Library (see mediasdkusr-man.pdf document for plugin loading and more details).

This document describes RAW Media Accelerator API. To learn more about general Intel Media SDK API definition and VPP component see mediasdk-man.pdf.

Document Conventions

The Intel® Media SDK RAW Media Accelerator uses the Verdana typeface for normal prose. With the exception of section headings and the table of contents, all code-related items appear in the `Courier New` typeface (`mxfStatus` and `MFXInit`). All class-related items appear in all cap boldface, such as **DECODE** and **ENCODE**. Member functions appear in initial cap boldface, such as **Init** and **Reset**, and these refer to members of all three classes, **DECODE**, **ENCODE** and **VPP**. Hyperlinks appear in underlined boldface, such as **`Bayer Processing`**.

Acronyms and Abbreviations

API	Application Programming Interface
Direct3D9	Microsoft* Direct3D* version 9
Direct3D11	Microsoft* Direct3D* version 11.1
DXVA2	Microsoft DirectX* Video Acceleration standard 2.0
RGB4	Thirty-two-bit RGB color format. Also known as RGB32
SDK	Intel® Media SDK
SDK execution	Intel® Media SDK execution
SDK functions	Intel® Media SDK functions
SDK library	Intel® Media SDK library
SDK session	Intel® Media SDK session
SDK video processing	Intel® Media SDK video processing
RAW Accelerator	Intel® Media SDK RAW Media Accelerator
video memory	memory used by hardware acceleration device, also known as GPU, to hold frame and other types of video data

VPP	Video Processing
YUY2	A color format for raw video frames
YV12	A color format for raw video frames
ARGB32	A 4-byte color format for raw video frames in RGB color space with alpha, each pixel sample defined by 8 bits (bit depth 8).
ARGB64	An 8-byte color format for raw video frames in RGB color space with alpha, each pixel sample represented by 10-16 bits stored in 2 bytes each.
A2RGB10	A 4-byte color format for raw video frames in RGB color space with 2 bits used for alpha channel, and 10 bits for each color channel.
R16	A 2-byte color format for raw video frames in RAW Bayer, each pixel sample defined by 10-16 bits

Architecture

Intel® Media SDK functions fall into the following categories:

VPP	Perform video processing on raw video frames
CORE	Auxiliary functions for synchronization
Misc	Global auxiliary functions

Bayer Processing

The RAW Accelerator designed to do conversion from RAW Bayer to RGB or other electronic systems compatible formats for rendering and encoding, applying high quality algorithms for professional Video and Photo usage.

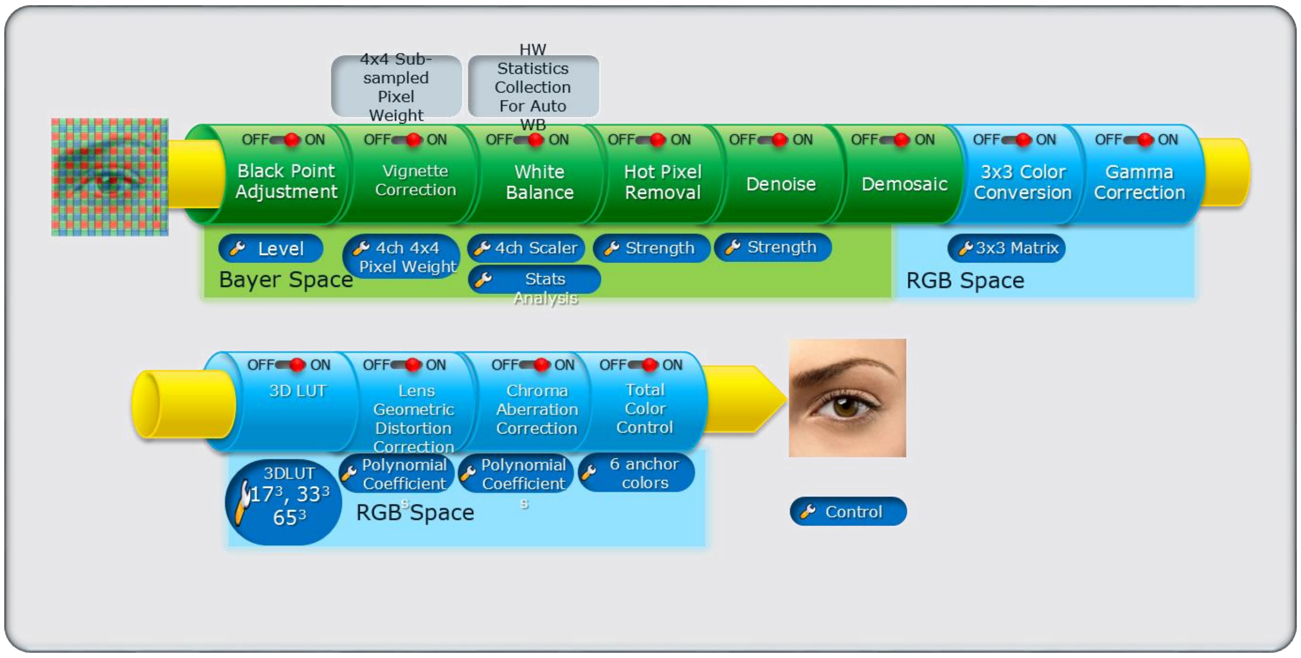


Figure 1: RAW Processing Operation Pipeline

The actual conversion process is a chain operation with many single-function filters, as Figure 1 illustrates. The application provides input in RAW Bayer with specification of Bayer pattern, and get output in one of supported electronic formats. The only unmanageable filter is Demosaic, all other filters can be enabled, disabled or managed through filter parameters, and parameters can be adjusted using hints. Unless any additional filters specified, RAW accelerator will apply only Demosaic algorithm.

Table 1 shows the RAW Accelerator features supported for API 1.22. The application can configure supported video processing features through the video processing I/O parameters and optional features through hints. See *"Video Processing procedure / Configuration"* for more details on how to configure optional filters.

Table 1: Video Processing Features

Video Processing Features	Configuration
Perform Black Point Adjustment for Bayer format	hint (optional feature)
Perform Vignette Correction for Bayer format	hint (optional feature)
Perform White Balance Adjustment for Bayer format	hint (optional feature)
Perform Hot Pixel Removal for Bayer format	hint (optional feature)
Perform Bayer De-noise	hint (optional feature)
Perform Lens Geometry Distortion correction for Bayer format	hint (optional feature)
Perform Chroma Aberration correction for Bayer format	hint (optional feature)
Convert color format from Bayer to ARGB color space – Demosaic	I/O parameters
RGB color correction to switch between different RGB notations	hint (optional feature)
Gamma correction for RGB color format	hint (optional feature)
3DLUT RGB->RGB conversion	hint (optional feature)
Total color control for YUV color format	hint (optional feature)
Color space conversion RGB->NV12	hint (optional feature)

Table 2: Color Conversion Support in Raw Accelerator

Output Color / Input Color	ARGB32	ARGB64	NV12
R16(Bayer pattern)	X	X	X

X indicates a supported function

The RAW Accelerator pipeline supports each optional filter for one format only. BayerFormat should be specified explicitly.

Currently, only input bit depths of 10-16 supported. If the output format is set to ARGB64, the output bit depth is equal to the input bit depth. See ColorFourCC section for more information on input and output color formats.

Non-Bayer Processing

Starting from API version 1.19 The RAW Accelerator supports ARGB16 input for filters after demosaic stage.

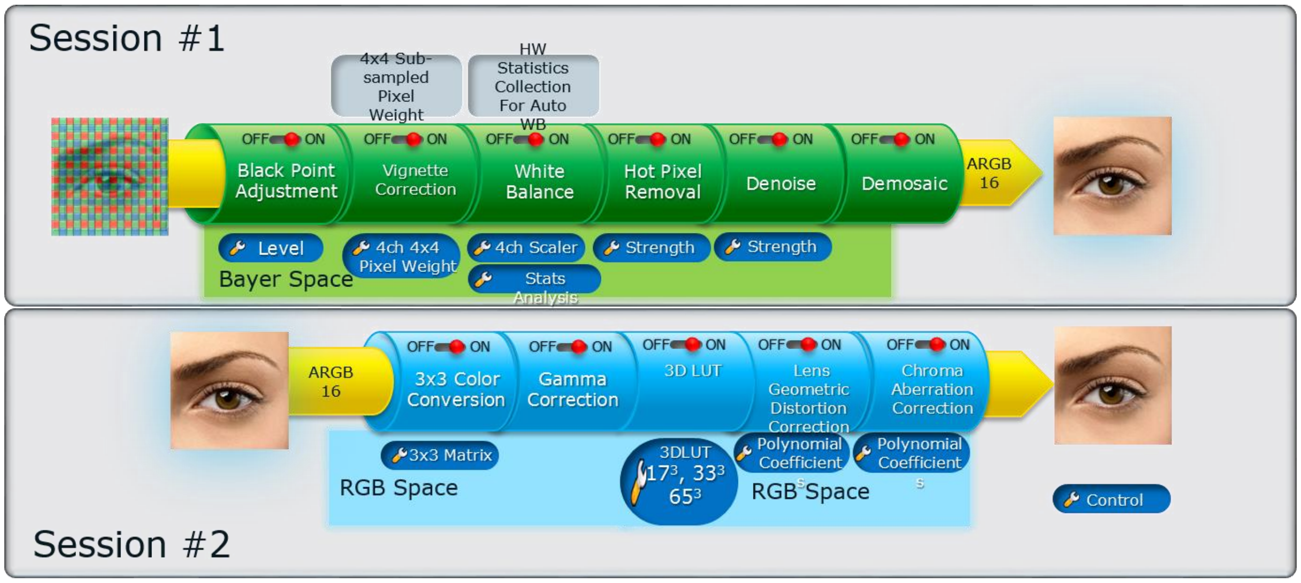


Figure 1: RAW Processing Operation Pipeline

The application provides input in RAW Bayer with specification of Bayer pattern for Session #1, and get output in ARGB16 format. In Session #2 Application can perform processing using Color Conversion, Gamma Correction, 3DLUT, Geometry or Chroma Aberration Correction with changed parameters without involving full pipe restart. This can help to optimize performance and advice smother user experience for changing parameters.

Table 1 shows the RAW Accelerator features supported for ARGB16 input in API 1.19. The application can configure supported video processing features through the video processing I/O parameters and optional features through hints. See *"Video Processing procedure / Configuration"* for more details on how to configure optional filters.

Table 3: ARGB16 input support

Video Processing Features	Configuration
Perform Lens Geometry Distortion correction for Bayer format	hint (optional feature)
Perform Chroma Aberration correction for Bayer format	hint (optional feature)
RGB color correction to switch between different RGB notations	hint (optional feature)
Gamma correction for RGB color format	hint (optional feature)
3DLUT RGB->RGB conversion	hint (optional feature)
Total color control for RGB color format	hint (optional feature)

The RAW Accelerator pipeline supports each optional filter for one format only. BayerFormat should be specified explicitly.

Currently, only input bit depths of 10-16 supported. If the output format is set to ARGB64, the output bit depth is equal to the input bit depth. See ColorFourCC section for more information on input and output color formats.

Programming Guide

This chapter describes the concepts used in programming the Intel® Media SDK and the Raw Accelerator.

The application must use the include file, **`mfxvideo.h`** (for C programming), or **`mfxvideo++.h`** (for C++ programming), **`mfxcamera.h`** and link the Intel Media SDK static dispatcher library, **`libmfx.lib`**.

Include these files:

```
      #include "mfxvideo.h"   /* The SDK include file */
      #include "mfxvideo++.h" /* Optional for C++ development */
      #include "mfxplugin.h"  /* Plugin development */
      #include "mfxcamera.h"  /* The Raw Accelerator include file */
```

Link this library:

```
      libmfx.lib               /* The SDK static dispatcher library */
```

RAW Accelerator is a plug-in to Media SDK library, for current plugin implementation, it requires minimum Media SDK HW Library API 1.10. To make sure Plugin will be Application will work with available drivers, the application need to do MFXInit with API 1.10 version.

Video Processing Procedures using VPP plug-in

Example 1 shows the pseudo code of the video processing procedure using plugin. The following describes a few key points:

- The application uses the **`MFXVideoVPP_QueryIOSurf`** function to obtain the number of frame surfaces needed for input and output. The application must allocate two frame surface pools, one for the input and the other for the output.
- The video processing function **`MFXVideoVPP_RunFrameVPPAsync`** is asynchronous. The application must synchronize to make the output result ready, through the **`MFXVideoCORE_SyncOperation`** function.
- The body of the video processing procedures covers three scenarios as follows:
    - If the number of frames consumed at input is equal to the number of frames generated at output, **VPP** returns **`MFX_ERR_NONE`** from **`MFXVideoVPP_RunFrameVPPAsync`** function and processing started. The application must process the output frame after synchronization, as the **`MFXVideoVPP_RunFrameVPPAsync`** function is asynchronous, though **`MFXVideoCORE_SyncOperation`** function call. At the end of a sequence, the application must provide a `NULL` input to drain any remaining frames.

- If the number of frames consumed at input is more than the number of frames generated at output, **VPP** returns **MFX_ERR_MORE_DATA** for additional input until **MFX_ERR_NONE** returned from **MFXVideoVPP_RunFrameVPPAsync**. Then proceed to procedure described in previous paragraph.
- If the number of frames consumed at input is less than the number of frames generated at output, **VPP** returns either **MFX_ERR_MORE_SURFACE** (when more than one output is ready), or **MFX_ERR_NONE** (when one output is ready and **VPP** expects new input). In both cases, the application must process the output frame after synchronization and provide a `NULL` input at the end of sequence to drain any remaining frames.

```
mfxSession session;
MFXInit(MFX_IMPL_HARDWARE,1.10,&session);
MFXVideoUSER_Load(session, MFX_PLUGINID_CAMERA_HW, version);

mfxVideoParam *in;// input parameters structure
mfxVideoParam *out;// output parameters structure
/* allocate structures and fill input parameters structure, zero unused fields */
MFXVideoVPP_Query(session, in, out);
/* check supported parameters */

MFXVideoVPP_QueryIOSurf(session, &init_param, response);
allocate_pool_of_surfaces(in_pool, response[0].NumFrameSuggested);
allocate_pool_of_surfaces(out_pool, response[1].NumFrameSuggested);
mfxVideoParam init_param;
/* configure init_param structure within supported parameters according to
response from Query */
MFXVideoVPP_Init(session, &init_param);
in=find_unlocked_surface_and_fill_content(in_pool);
out=find_unlocked_surface_from_the_pool(out_pool);
for (;;) {
      sts=MFXVideoVPP_RunFrameVPPAsync(session,in,out,aux,&syncp);
      if (sts==MFX_ERR_MORE_SURFACE || sts==MFX_ERR_NONE) {
            MFXVideoCore_SyncOperation(session,syncp,INFINITE);
            process_output_frame(out);
            out=find_unlocked_surface_from_the_pool(out_pool);
      }
      if (sts==MFX_ERR_MORE_DATA && in==NULL) break;
      if (sts==MFX_ERR_NONE || sts==MFX_ERR_MORE_DATA) {
            in=find_unlocked_surface(in_pool);
            fill_content_for_video_processing(in);
            if (end_of_input_sequence()) in=NULL;
      }
}
MFXVideoVPP_Close(session);
free_pool_of_surfaces(in_pool);
free_pool_of_surfaces(out_pool);

MFXVideoUSER_UnLoad(session, MFX_PLUGINID_CAMERA_HW)
MFXClose(session);
```

Example 1: Video Processing Pseudo Code

Configuration

The RAW Accelerator plug-in configures the video processing pipeline operation based on parameters specified in the **mfxVideoParam** structure.

In addition to specifying the input and output formats, the application can provide hints to fine-tune the video processing pipeline operation. The application can enable filters by using **mfxExtVPPDoUse** structure and configure them by using dedicated configuration structures. See Table 3 for complete list of configurable video processing filters, their IDs and configuration structures. See the **ExtendedBufferID** enumerator for more details.

The RAW Accelerator ensures that all filters necessary to convert input format to output one are included in pipeline. However, the RAW Accelerator can skip some optional filters even if they are explicitly requested by the application, for example, due to limitation of underlying hardware. To notify application about this skip, the RAW Accelerator returns warning MFX_WRN_FILTER_SKIPPED. The application can retrieve the list of active filters by attaching **mfxExtVPPDoUse** structure to **mfxVideoParam** structure and calling **MFXVideoVPP_GetVideoParam** function. The application must allocate enough memory for filter list.

Table 3 Configurable RAW accelerator buffers

Filter ID	Configuration structure
MFX_EXTBUF_CAM_PIPECONTROL	**mfxExtCamPipeControl**
MFX_EXTBUF_CAM_PADDING	**mfxExtCamPadding**
MFX_EXTBUF_CAM_BLACK_LEVEL_CORRECTION	**mfxExtCamBlackLevelCorrection**
MFX_EXTBUF_CAM_VIGNETTE_CORRECTION	**mfxExtCamVignetteCorrection**
MFX_EXTBUF_CAM_WHITE_BALANCE	**mfxExtCamWhiteBalance**
MFX_EXTBUF_CAM_HOT_PIXEL_REMOVAL	**mfxExtCamHotPixelRemoval**
MFX_EXTBUF_CAM_BAYER_DENOISE	**mfxExtCamBayerDenoise**
MFX_EXTBUF_CAM_COLOR_CORRECTION_3X3	**mfxExtCamColorCorrection3x3**
MFX_EXTBUF_CAM_GAMMA_CORRECTION	**mfxExtCamGammaCorrection**
MFX_EXTBUF_CAM_FORWARD_GAMMA_CORRECTION	**mfxExtCamFwdGamma**
MFX_EXTBUF_CAM_LENS_GEOM_DIST_CORRECTION	**mfxExtCamLensGeomDistCorrection**
MFX_EXTBUF_CAM_3DLUT	**mfxExtCam3DLut**
MFX_EXTBUF_CAM_TOTAL_COLOR_CONTROL	**mfxExtCamTotalColorControl**

MFX_EXTBUF_CAM_CSC_YUV_RGB	**mfxExtCamCscYuvRgb**

Example 2 shows how to configure the SDK video processing.

```
/* enable image stabilization filter with default settings */
mfxExtVPPDoUse du;
mfxU32 alg_list[3]= { MFX_EXTBUF_CAM_PIPECONTROL,
MFX_EXTBUF_CAM_GAMMA_CORRECTION};
du.Header.BufferId=MFX_EXTBUFF_VPP_DOUSE;
du.Header.BufferSz=sizeof(mfxExtVPPDoUse);
du.NumAlg=2;
du.AlgList=alg_list;

/* configure the mfxVideoParam structure */
mfxVideoParam conf;
mfxExtBuffer *eb=&du;

memset(&conf,0,sizeof(conf));
conf.IOPattern=MFX_IOPATTERN_IN_SYSTEM_MEMORY|
               MFX_IOPATTERN_OUT_VIDEO_MEMORY;
conf.NumExtParam=2;
conf.ExtParam=&eb;

conf.vpp.In.FourCC=MFX_FOURCC_R16;
conf.vpp.In.ChromaFormat=MFX_CHROMAFORMAT_YUV400;

conf.vpp.Out.FourCC=MFX_FOURCC_RGB4;
conf.vpp.Out.ChromaFormat=MFX_CHROMAFORMAT_YUV444;

conf.vpp.In.Width=conf.vpp.Out.Width=4096;
conf.vpp.In.Height=conf.vpp.Out.Height=2160;

/* video processing initialization */
MFXVideoVPP_Init(session, &conf);
```

Example 2: Configure Video Processing

Transcoding Procedures

The application can use other Intel Media SDK components and RAW Accelerator plugin together for transcoding operations. For example, video processing functions to resize and render to monitor with resolution less than initial content, apply composition and alpha blending to video or photo for logo, color conversion to NV12 color format and then encode to H.264 or H.265, or directly encode to JPEG in RGB32. For more details on building transcoding pipelines please see mediasdk-man.pdf.

Important note: to use Intel Media SDK legacy VPP features the application must create a separate SDK session, see details in mediasdk-man.pdf.

Function Reference

The RAW Accelerator does not define any new functions in addition to standard Intel Media SDK function set, please check mediasdk-man.pdf and mediasdkusr-man.pdf for the SDK functions description.

Structure Reference

This section describes structures declared in **`mfxcamera.h`** and applicable only to the RAW Accelerator. All Intel Media SDK structures description can be found in mediasdk-man.pdf.

mfxExtCamPipeControl

Definition

```
typedef struct {
    mfxExtBuffer    Header;
    mfxU16          RawFormat;
    mfxU16          reserved1;
    mfxU32          reserved[5];
} mfxExtCamPipeControl;
```

Description

The `mfxExtCamPipeControl` structure is a hint structure that configures the RAW Accelerator; this is mandatory extended buffer necessary for initialization.

The application should attach this extended buffer to the **`mfxVideoParam`** structure to configure initialization.

Members

`Header.BufferId`	Must be **`MFX_EXTBUF_CAM_PIPECONTROL`**
`RawFormat`	Specifies one of the four Bayer patterns defined in mfxCamBayerFormat enumeration.

Change History

This structure is available since SDK API 1.10.

mfxExtCamPadding

Definition

```
typedef struct {
    mfxExtBuffer    Header;
    mfxU16 Top;
    mfxU16 Bottom;
```

```
        mfxU16 Left;
        mfxU16 Right;
        mfxU32 reserved[4];
    } mfxExtCamPadding;
```

Description

The `mfxExtCamPadding` structure specifies padding edges, if set application should provide input padded surface in Bayer format, otherwise plug-in would do padding itself using mirroring. Currently padding parameters applies only to whole picture.

The application can attach this extended buffer to the **mfxVideoParam** structure to configure initialization.

Members

`Header.BufferId`	Must be **MFX_EXTBUF_CAM_PADDING**
`Top, Bottom, Left, Right`	Specify number of padded rows/columns respectively. Currently only 8 pixels supported for all dimensions.

Change History

This structure is available since SDK API 1.10.

Figure 2 introduces how padding with mirroring done by RAW Accelerator for left border. Note: the outermost (leftmost in the case of Fig.2) columns/rows are not used by the current RAW Media Accelerator pipeline; the application will need to fill them as shown in Fig.2 (copy the second column/row from the boundary) for correct operation of Bayer denoise algorithm.

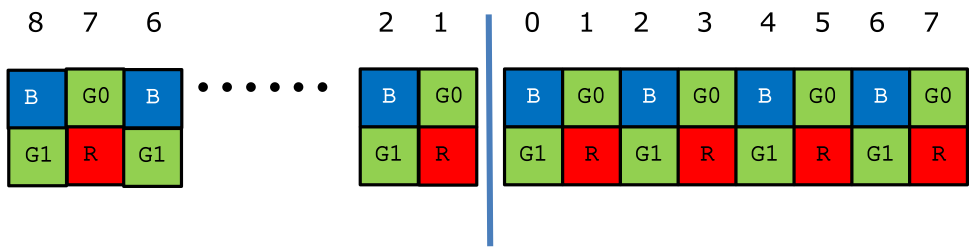


Figure 2: Padding with mirroring

mfxExtCamGammaCorrection

Definition

```
typedef struct {
    mfxExtBuffer    Header;
    mfxU16  Mode;
    mfxU16  reserved1;
    mfxF64  GammaValue;

    mfxU16  reserved2[3];
    mfxU16  NumPoints;
    mfxU16  GammaPoint[1024];
    mfxU16  GammaCorrected[1024];
    mfxU32  reserved3[4];
} mfxExtCamGammaCorrection;
```

Description

The `mfxExtCamGammaCorrection` structure specifies Gamma Correction control.

The application can attach this extended buffer to the **mfxVideoParam** structure to configure initialization.

Members

`Header.BufferId`	Must be **MFX_EXTBUF_CAM_GAMMA_CORRECTION**
`Mode`	Specifies one of Gamma Correction modes defined in enumeration mfxCamGammaParam
`GammaValue`	Gamma point value, can be used if Gamma Correction mode set as MFX_CAM_GAMMA_VALUE.
`NumPoints`	Number of Gamma points, can be used if MFX_CAM_GAMMA_LUT mode selected. Maximum number can vary up to 1024 points; different versions of RAW Media Accelerator can support less. Maximum supported number can be checked using Query function. Current version supports maximum 64 points.
`GammaPoint`	Vector of gamma point values, can be used if MFX_CAM_GAMMA_LUT mode selected.
`GammaCorrected`	Vector of gamma corrected point values, can be used if

MFX_CAM_GAMMA_LUT mode selected.

Note: To ensure the correct operation, provided Gamma LUT Points should strictly obey the following rules

`GammaPoint[0]=0; GammaPoint[NumPoints]=MAX; GammaPoint[i] < GammaPoint[i+1];`
where MAX is the maximum value represented with the input content bit depth (e.g., 1023 for 10-bit content).

Note: in the current release of the RAW Media Accelerator, the checking for the validity of the Gamma Point LUT is disabled.

Gamma engine has two modes. One (MFX_CAM_GAMMA_VALUE) uses the classic Xi= Xi ^{gamma} definition, the only constraint is that gamma must be greater than 0.1

An alternate `MFX_CAM_GAMMA_LUT`, more flexible implementation is a 64-entry lookup table. The pixel-spac is divided into 65 regions; (64 is the current implementation; user must first query the size of the lookup table as it may grow in the future). A Pixel which falls between points is linearly interpolated. This is illustrated in Figure 3.

In both cases, the same algorithm is applied to all channels of the input image (RGBA).

The lookup coefficients are specified relative to the native range of the image format. If working on 8-bit values, please specify 8-bit point and value pairs; if working on 16-bit images, use 16-bit point and value pairs.

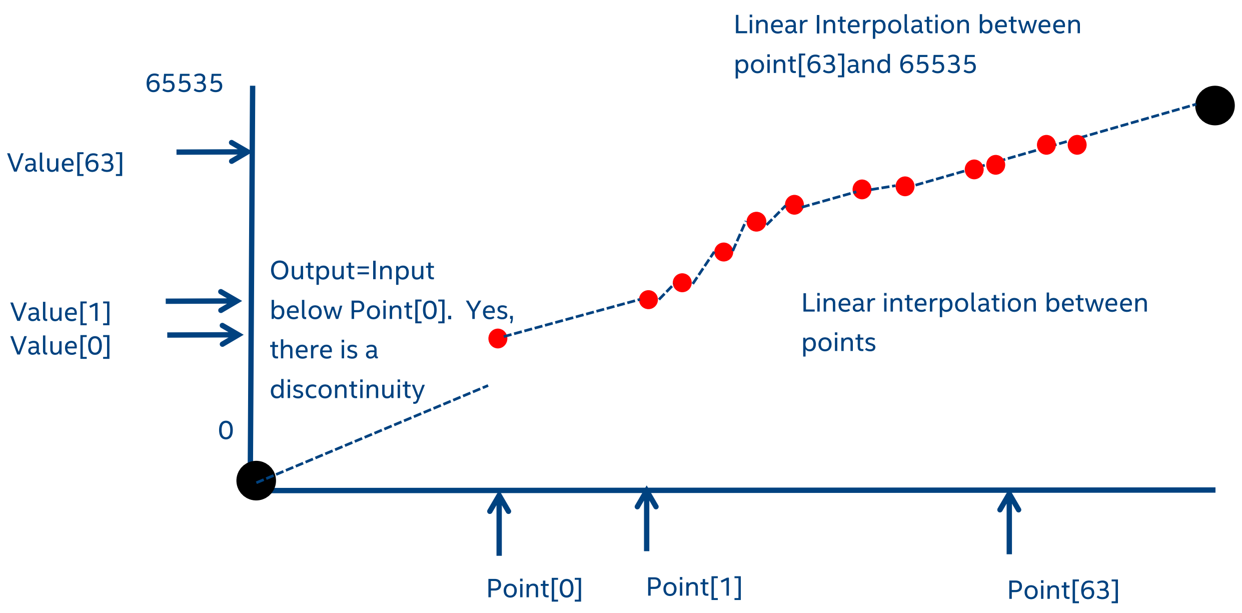

Linear Interpolation between point[63]and 65535
65535
Value[63]
Output=Input below Point[0]. Yes, there is a discontinuity
Value[1]
Value[0]
Linear interpolation between points
0
Point[0]
Point[1]
Point[63]

Figure 3: Lookup algorithm

Change History

This structure is available since SDK API 1.10.

mfxExtCamFwdGamma

Definition

```
typedef struct{
    mfxU16 Pixel;
    mfxU16 Red;
    mfxU16 Green;
    mfxU16 Blue;
} mfxCamFwdGammaSegment;

typedef struct {
    mfxExtBuffer Header;
    mfxU16       reserved[19];
    mfxU16       NumSegments;
    union {
        mfxCamFwdGammaSegment* Segment;
        mfxU64 reserved1;
    };
} mfxExtCamFwdGamma;
```

Description

The `mfxExtCamFwdCorrection` structure specifies independent R,G,B Gamma Correction control.

The application can attach this extended buffer to the **mfxVideoParam** structure to configure initialization.

Members

`Header.BufferId`	Must be **MFX_EXTBUF_CAM_FORWARD_GAMMA_CORRECTION**
`NumSegments`	Number of Gamma segments. Maximum number can vary up to 1024 points; different versions of RAW Media Accelerator can support less. Maximum supported number can be checked using Query function. Current version supports maximum 64 points.
`Segment`	Pointer to Gamma segments array.

Note: To ensure the correct operation, provided Gamma Segments should strictly obey the following rules:

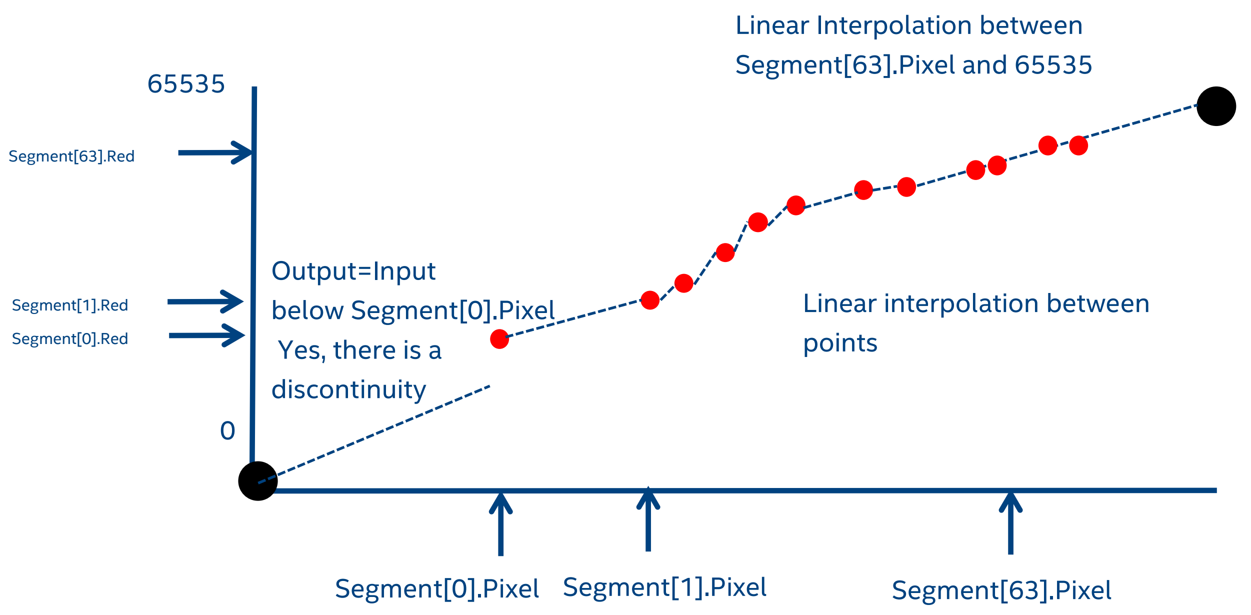
`Segment[0].Pixel=0; Segment[NumSegments].Pixel=MAX; Segment[i].Pixel < Segment[i+1].Pixel;`
where MAX is the maximum value represented with the input content bit depth (e.g., 1023 for 10-bit content).

Note: in the current release of the RAW Media Accelerator, the checking for the validity of the Gamma Segment is disabled.

Gamma engine's implementation is a 64-entry lookup table. The pixel-space is divided into 65 regions; (64 is the current implementation; user must first query the size of the lookup table as it may grow in the future). A Pixel which falls between points is linearly interpolated. This is illustrated in Figure 3.

The same algorithm is applied independenty to all channels of the input image (RGBA).

The lookup coefficients are specified relative to the native range of the image format. If working on 8-bit values, please specify 8-bit point and value pairs; if working on 16-bit images, use 16-bit point and value pairs.



mfxExtCamWhiteBalance

Definition

```
typedef struct {
    mfxExtBuffer     Header;
    mfxU32           Mode;
    mfxF64           R;
    mfxF64          G0;
    mfxF64          B;
    mfxF64          G1;
    mfxU32          reserved[8];
} mfxExtCamWhiteBalance;
```

Description

The `mfxExtCamWhiteBalance` structure specifies White Balance filter configuration.

This operation allows to correct sensor or image gain using linear weights on each channel.

The application can attach this extended buffer to the **mfxVideoParam** structure to configure initialization.

Members

`Header.BufferId`	Must be **MFX_EXTBUF_CAM_WHITE_BALANCE**.
`Mode`	Specifies one of White Balance operation modes defined in enumeration mfxCamWhiteBalanceMode.
`R`	White Balance Red correction.
`G0`	White Balance Green Top correction.
`B`	White Balance Blue correction.
`G1`	White Balance Green Bottom correction.

Notes:

- Correction parameters are specified as 64 bit floating point for future expansion, but current implementation using 32 bit float, so values with greater precision will be truncated to 32 bit.
- In the current release of the RAW Media Accelerator, the checking for the validity of the correction coeficients is disabled.
- Only manual White Balance mode (`MFX_CAM_WHITE_BALANCE_MANUAL`) is supported in the current release.
- 4 separate correction factors (one for each of Red/Green0/Blue/Green1 are multiplied to the input pixels in the Bayer pattern. There are 2 separate green correction factors, so that the 2 greens that appear in a 2x2 Bayer pattern block can be corrected separately.

R_scaled = min((Rin * R_correction)>>12, 65535)

Gtop_scaled = min((G0in * G0_correction)>>12, 65535)

Gbot_scaled = min((G1in * G1_correction)>>12, 65535)

B_scaled = min((Bin * B_correction)>>12, 65535)

Change History

This structure is available since SDK API 1.13.

mfxExtCamBlackLevelCorrection

Definition

```
typedef struct {
    mfxExtBuffer    Header;
    mfxU16          R;
    mfxU16          G0;
    mfxU16          B;
    mfxU16          G1;
    mfxU32          reserved[4];
} mfxExtCamBlackLevelCorrection;
```

Description

The `mfxExtCamBlackLevelCorrection` Black Level Correction control.

The application can attach this extended buffer to the **mfxVideoParam** structure to configure initialization.

Members

`Header.BufferId`	Must be **MFX_EXTBUF_CAM_BLACK_LEVEL_CORRECTION**
`R`	Black Level Red correction.
`G0`	Black Level Green Top correction.
`B`	Black Level Blue correction.
`G1`	Black Level Green Bottom correction.

Notes:

- Cameras tend to have a minimum output for a black input. The Black Level Correction is a simple subtract of a different value for Red/Green0/Green1/Blue of the black point values to translate a black input to 0,0,0.

  Rout   = MIN(MAX(0,Rin – R), 65535)
  G0out = MIN(MAX(0,G0in – G0), 65535)
  G1out = MIN(MAX(0,G1in – G1), 65535)
  Bout   = MIN(MAX(0,Bin – B), 65535)

- In the current release of the RAW Media Accelerator, the checking for the validity of the correction coeficients is disabled

Change History

This structure is available since SDK API 1.13.

mfxExtCamLensGeomDistCorrection

Definition

```
typedef struct {
    mfxExtBuffer Header;

    mfxF32       a[3]; // [R, G, B]
    mfxF32       b[3]; // [R, G, B]
    mfxF32       c[3]; // [R, G, B]
    mfxF32       d[3]; // [R, G, B]
    mfxU16       reserved[36];
} mfxExtCamLensGeomDistCorrection;
```

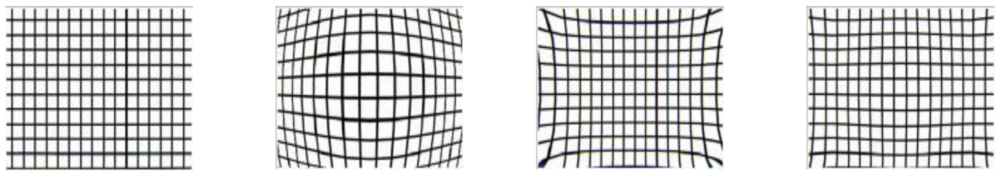
Description

The `mfxExtCamLensGeomDistCorrection` structure defines control for Lens Geometry Distortion and Chroma Aberration Correction filter.

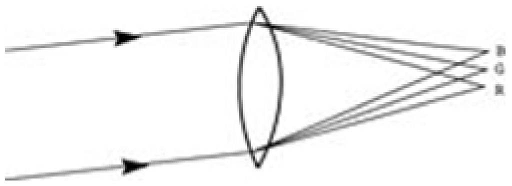
The application can attach this extended buffer to the **mfxVideoParam** structure to configure initialization.

Real world camera lens do not align with ideal pin-hole camera model perfectly. This fact can reflect in two types of distortions:

- deviation from rectilinear projection. Almost all deviations occur radially, towards or away from some common center (i.e., principal point).



- lens fails to focus three color channels onto the same location



Color channels displaced in a lateral direction, leading to different magnification -> color fringing.

In essence, both the distortions correctable by image processing. Lens Geometry Distortion correction and Chroma Aberraton correction are identical in the terms of algorithm but while first one operates identically for all the channels, the second - performs the correction of each channel independently to the others.

Currently, applying a 3rd degree polynomial for correcting radial lens distortion.

$$r_{corr} = \underbrace{(ar_{dist}^3 + br_{dist}^2 + cr_{dist} + d)}_{S_c} \cdot r_{dist}$$

Where:

$$r = \frac{\sqrt{(u-u_c)^2+(v-v_c)^2}}{norm\_factor}$$

$$(u_c, v_c): coordinate\ of\ the\ proncipal\ point.$$

Members

`Header.BufferId`	Must be **MFX_EXTBUF_CAM_LENS_GEOM_DIST_CORRECTION**
`a[3]`	Polynomial coefficients A (see formula above) for R/G/B.
`b[3]`	Polynomial coefficients B (see formula above) for R/G/B.
`c[3]`	Polynomial coefficients C (see formula above) for R/G/B.
`d[3]`	Polynomial coefficients D (see formula above) for R/G/B.

Notes:

- In the current release of the RAW Media Accelerator, the checking for the validity of the correction coeficients is disabled.

Change History

This structure is available since SDK API 1.16.

mfxExtCamColorCorrection3x3

Definition

```
typedef struct {
    mfxExtBuffer     Header;
    mfxF64           CCM[3][3];
    mfxU32           reserved[4];
} mfxExtCamColorCorrection3x3;
```

Description

The `mfxExtCamColorCorrection3x3` structure defines control for Color Correction filter.

The application can attach this extended buffer to the **`mfxVideoParam`** structure to configure initialization.

Members

`Header.BufferId`	Must be **`MFX_EXTBUF_CAM_COLOR_CORRECTION_3X3`**
`CCM`	3x3 dimension matrix providing RGB Color Correction coefficients.

Notes:

- Correction parameters are specified as 64 bit floating point for future expansion, but current implementation using 32 bit float, so values with greater precision will be truncated to 32 bit.
- In the current release of the RAW Media Accelerator, the checking for the validity of the correction coeficients is disabled.
- The Color Correction Matrix (CCM) consists of a 3x3 matrix multiplication. The Color Correction Matrix is typically used for translating the camera RGB output (which is usually in a non-standard color space) to a standard RGB color space such as sRGB. The matrix can also be used to control gain by applying a scale factor to the coefficients used for color correction. This Gain Control allows scaling of the input to brighten up a dim scene, for example.

  R0out = ROUND_TO_ZERO(MAX(0.0f, (float) Rin * CCM[0][0] + (float) Gin * CCM[0][1] + (float) Bin*CCM[0][2]));
  G0out = ROUND_TO_ZERO(MAX(0.0f, (float) Rin * CCM[1][0] + (float) Gin * CCM[1][1] + (float) Bin*CCM[1][2]));
  B0out = ROUND_TO_ZERO(MAX(0.0f, (float) Rin * CCM[2][0] + (float) Gin * CCM[2][1] + (float) Bin*CCM[2][2]));

Change History

This structure is available since SDK API 1.13.

mfxExtCamHotPixelRemoval

Definition

```
typedef struct {
    mfxExtBuffer    Header;
    mfxU16          PixelThresholdDifference;
    mfxU16          PixelCountThreshold;
} mfxExtCamHotPixelRemoval;
```

Description

The `mfxExtCamHotPixelRemoval` defines Hot Pixel Removal control.

The application can attach this extended buffer to the **mfxVideoParam** structure to configure initialization.

Members

`Header.BufferId`	Must be **MFX_EXTBUF_CAM_HOT_PIXEL_REMOVAL**
`PixelThresholdDifference`	Threshold for Hot Pixel difference.
`PixelCountThreshold`	Count pixel detction.

Notes:

- Hot Pixel Removal designed for cleaning image from “bright dot defects” brought by broken sensor, utilizing algorithms for intensity detection and median filter for correction.

```
HotPixelDetect (W, HotPixThr, HotPixCnt){
   counter=0;
   //detect hot pixel
   for (all pixels in 3x3 window, Pixel (i, j) in W)
   {
             //identify pixels of similar intensity
         if (abs(CurrentPixel - Pixel(i,j)) > (HotPixThr<<8)), counter++;
   }
   //correct hot pixel
   if (counter > HotPixCnt) MedianFilter(CurrentPixel, Wx, Wy)
}
```

Change History

This structure is available since SDK API 1.15.

mfxExtCamBayerDenoise

Definition

```
typedef struct {
    mfxExtBuffer    Header;
    mfxU16          Threshold;
    mfxU16          reserved[27];
} mfxExtCamBayerDenoise;
```

Description

The `mfxExtCamBayerDenoise` defines Bayer Denoise control, for applying Luma noise reduction filter on bayer input picture.

The application can attach this extended buffer to the **mfxVideoParam** structure to configure initialization.

Members

`Header.BufferId`	Must be **MFX_EXTBUF_CAM_BAYER_DENOISE**
`Threshold`	Level of denoise, legal values: [0:63].

Notes:

- Bayer denoise is using the Weighted Sigma filter for spatial denoise. The Weighted Sigma filter approximates the functionality of the bilateral filter by including distance and range (pixel intensity) based weights in computing the denoised pixel values. Comparing to standard Luma denoise in Media SDK this algorithm allowing stronger spatial noise reduction without causing blurring of details/edges

Change History

This structure is available since SDK API 1.15.

mfxExtCamVignetteCorrection

Definition

```
typedef struct {
    mfxExtBuffer    Header;

    mfxU32          Width;
    mfxU32          Height;
    mfxU32          Pitch;
    mfxU32          reserved[7];

    mfxCamVignetteCorrectionParam *CorrectionMap;

} mfxExtCamVignetteCorrection;
```

Description

The `mfxExtCamVignetteCorrection` defines control for vignette correction fillter.

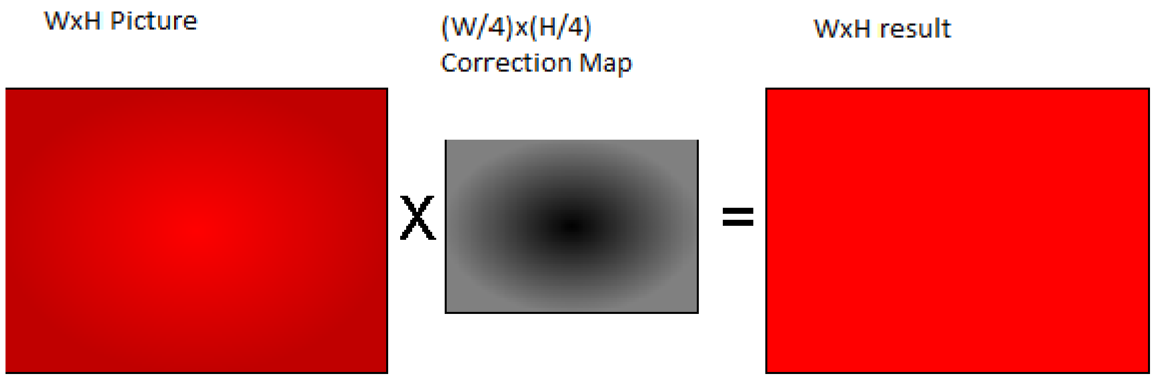
The application can attach this extended buffer to the **mfxVideoParam** structure to configure initialization.

Members

`Header.BufferId`	Must be **MFX_EXTBUF_CAM_VIGNETTE_CORRECTION**
`Width`	Width of Correction Map 2D buffer in `mfxCamVignetteCorrectionParam` elements.
`Height`	Height of Correction Map 2D buffer in `mfxCamVignetteCorrectionParam` elements.
`Pitch`	Pitch of Correction Map 2D buffer in bytes.
`CorrectionMap`	2D buffer of `mfxCamVignetteCorrectionParam` elements.

Notes:

- Vignette correction applied to the Bayer pattern Red/Green/Blue components to correct image peripheral fading caused by camera settings or lens limitation. This can be done algorithmically, but maximum quality, and maximum flexibility is to have a separate surface with ¼ of the pixels +1 in the X and Y direction as the frame. App should calculate coefficients itself based on Lens optical specific factors. A simple bilinear interpolation done to determine the 16-bit value at each pixel. The Vignette Correction block receives 4 correction factors of pared unsigned chars for each 4x4 being processed. The 4 correction factors delivered are the local one for the 4x4, plus the one to the right, below and lower right. For cases where the right or lower 4x4 is off the surface, the local correction factor repeated for the missing values.



Change History

This structure is available since SDK API 1.15.

mfxCamVignetteCorrectionParam

Definition

```
typedef struct {
    mfxCamVignetteCorrectionElement R;
    mfxCamVignetteCorrectionElement G0;
    mfxCamVignetteCorrectionElement B;
    mfxCamVignetteCorrectionElement G1;
} mfxExtCamVignetteCorrectionParam;
```

Description

The `mfxExtCamVignetteCorrectionParam` defines structure for one parameter of vignette correction map.

Members

`R`	Red correction element.
`G0`	Green top correction element.
`B`	Blue Correction element.
`G1`	Green bottom correction element.

Change History

This structure is available since SDK API 1.15.

mfxCamVignetteCorrectionElement

Definition

```
typedef struct {
    mfxU8 integer;
    mfxU8 mantissa;
} mfxCamVignetteCorrectionElement;
```

Description

The `mfxCamVignetteCorrectionElement` defines structure for 1 value of vignette correction parameter.

Members

`integer`	Integer part of correction element.
`mantissa`	Fractional part of correction element.

Notes:

- `mfxCamVignetteCorrectionElement` is 16 bit 8.8 fixed point number. App should provide as 1 16 bit value.
  Example: Correction element value 134.75 should be provided as
  ```
   elem.integer =134;(0x86)
   elem.mantissa = 192;(0xC0)
  ```
  Final pixel result will be delivered as
  ```
  PixOutX[x,y] = PixInX[x,y] * Corr [x,y] >> 8;
  ```

Change History

This structure is available since SDK API 1.15.

mfxExtCam3DLut

Definition

```
typedef struct {
    mfxExtBuffer Header;

    mfxU16 reserved[10];
    mfxU32 Size;
    union
    {
        mfxCam3DLutEntry* Table;
        mfxU64 reserved1;
    };
} mfxExtCam3DLut;
```

Description

The `mfxExtCam3DLut` defines control for 3DLUT RGB->RGB conversion fillter.

The application can attach this extended buffer to the **mfxVideoParam** structure to configure initialization.

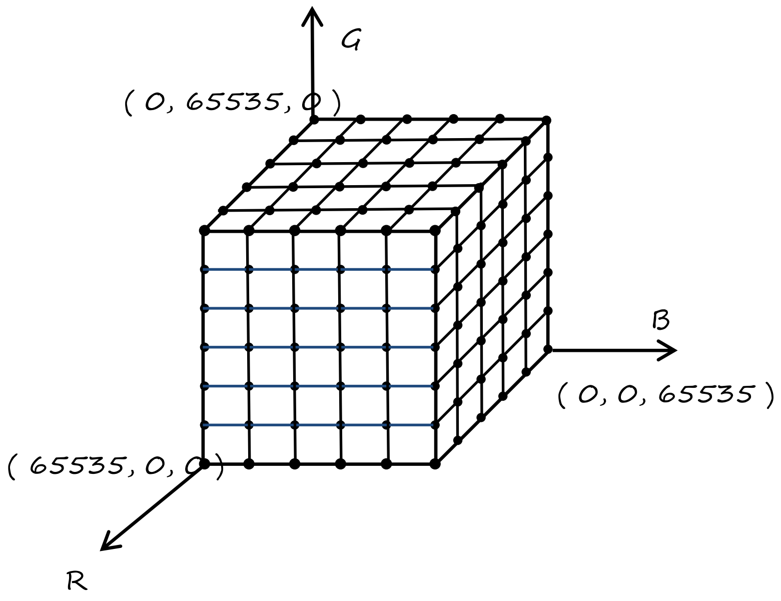
Members

`Header.BufferId`	Must be **MFX_EXTBUF_CAM_3DLUT**
`Size`	Size of the LUT. Must be one of MFX_CAM_3DLUT17_SIZE, MFX_CAM_3DLUT33_SIZE or MFX_CAM_3DLUT65_SIZE.
`Table`	Pointer to the 3-dimensional continuous array of **mfxCam3DLutEntry**, size of each dimension depends on LUT size, e.g. LUT[17][17][17] for 17x17x17 look up table.

Notes:

- This filter provides RGB-space color adjustments using a three-dimensional look up-table with 16bit bit-depth (16bit 3DLUT). As compared to one-dimensional LUT(1D LUT) like gamma table, relationship between R, G, and B can be defined so that user can do finer color conversion. End user also integrate Gamma table into 3DLUT by defining appropriate table values.
- A 3D LUT is a 3D lattice of output color values. Each axis is one of the 3 input color components and the input color thus defines a point inside the lattice. Since the point may not be on a lattice point, the lattice values must be interpolated. Three different sizes of tables supported:
    - 17x17x17

- 33x33x33
- 65x65x65



Conceptual Figure of 3D Look Up Table

In the 3DLUT array, the first index moves along the blue axis, second index moves along the green axis, and third channel moves along the blue axis. In other words: LUT[red index][green index][blue index]. Each element contains a red, green, and blue value. In the table minimum number must be 0, and maximum number must be 65535. End User can't make the interpolation finer by defining small number of maximum value like 1024. Granularity of interpolation should be adjusted by number of table point size such as 33x33x33 or 65x65x65.

Change History

This structure is available since SDK API 1.17.

mfxCam3DLutEntry

Definition

```
typedef struct {
    mfxU16 R;
    mfxU16 G;
    mfxU16 B;
    mfxU16 Reserved;
} mfxCam3DLutEntry;
```

Description

The `mfxCam3DLutEntry` defines structure for one element of 3DLUT.

Members

`R`	Red element.
`G`	Green element.
`B`	Blue element.

Change History

This structure is available since SDK API 1.17

mfxExtCamTotalColorControl

Definition

```
typedef struct {
    mfxU16 R;
    mfxU16 G;
    mfxU16 B;
    mfxU16 C;
    mfxU16 M;
    mfxU16 Y;
    mfxU16 reserved[6];
} mfxExtCamTotalColorControl;
```

Description

The `mfxExtCamTotalColorControl` defines control for YUV->YUV conversion fillter

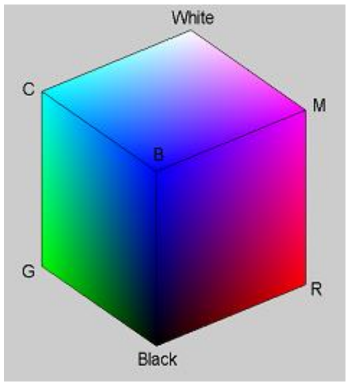
Members

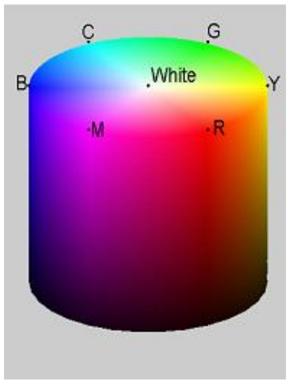
`R`	Red element.
`G`	Green element.
`B`	Blue element.

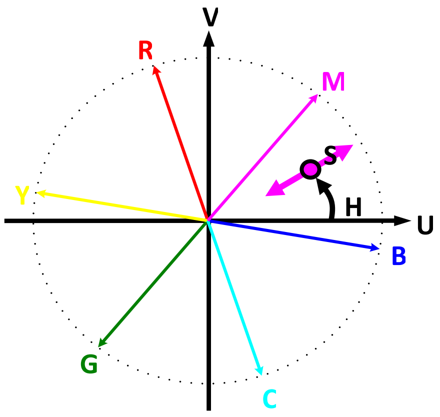
`C`	Cyan element.
`M`	Magenta element.
`Y`	Yellow element.

Notes:

- Correction parameters are specified as 16 bit unsigned integer for future expansion, but current implementation using 8 bit unsigned integer. In case of incorrect precision `MFX_ERR_INCOMPATIBLE_VIDEO_PARAM` will be thrown
- Utilize 6 basic colors as anchors, adjust saturation of pixels while maintaining their color (hue), limit adjustment of nearly-saturated and near-gray pixels.
  Total color control algorithm operates on the UV-color components in the YUV color space.
  The conversion formula for UV is:
  U_{new} = U * SFfinal
  V_{new} = V * SFfinal
  Here SFfinal – saturation factor, that calculated from:
  R,G,B,C,M,Y – anchors colors of input pixel
  SatFactor1,…,SatFactor6 – user-defined saturation factors for anchor colors
  ColorTransit12,…,ColorTransit61 – results of 1 / (BaseColorX – BaseColorY)
  ColorBias1,…,ColorBias6 – color biases for anchor colors
  H - color hue angle in radians







- Total Color Control is helpful if 3DLUT is inacceptable due of performance. In such case need to use Color Correction Matrix, Gamma Correction and after it Total Color Control for adjusting saturation. So there are several pipelines for different purposes:
  Quality: Demosaicing(VEBox)+3DLUT(Render/EU)+CSC(ARGB to NV12)
  ->Encoding
  can be used
  Performance: Demosaicing+CCM+Gamma Correction+TCC->NV12(VEBox)
  ->Encoding

Change History

This structure is available since SDK API 1.23.

mfxExtCamCscYuvRgb

Definition

```
typedef struct {
    mfxExtBuffer Header;
    mfxF32       PreOffset[3];
    mfxF32       Matrix[3][3];
    mfxF32       PostOffset[3];
    mfxU16       reserved[30];
} mfxExtCamCscYuvRgb;
```

Description

The `mfxExtCamCscYuvRgb` need for conversion from RGB to YUV color format

Members

`PreOffset`	Offset for conversion from full range RGB input to limited range YUV for input color coordinate
`Matrix`	Conversion matrix with CSC coefficients
`PostOffset`	Offset for conversion from full range RGB input to limited range YUV for output color coordinate

Notes:

- Currently only NV12 output is supported.
- Here is some default values for conversion from ARGB8 to NV12:
    - PreOffset is a vector with (0 ,0, 0) elements
    - Matrix coefficients are standard for RGB->YUV conversion. By default ITU-R BT.601 version is used:

$$A = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{pmatrix}$$

    - PostOffset is a vector with (64, 16384, 16384) elements. These values are left shifted from (16, 128, 128) elements for passing to VEBox directly
- For ARGB8 -> NV12 conversion 4:2:0 chroma subsampling is used:
  For every 4 luma values in row, there is one u and one v components which are changed via line.
- The final formula for conversion is:

$$\begin{pmatrix} out\_1 \\ out\_2 \\ out\_3 \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{pmatrix} * \begin{pmatrix} in\_1 + 0 \\ in\_2 + 0 \\ in\_3 + 0 \end{pmatrix} + \begin{pmatrix} 64 \\ 16384 \\ 16384 \end{pmatrix}$$

in_i – color components of the input pixel

out_i – color components of the out pixel

Change History

This structure is available since SDK API 1.23.

Enumerator Reference

mfxCamGammaParam

Description

The `mfxCamGammaParam` enumerator defines operation modes for Gamma Correction filter.

Name/Description

`MFX_CAM_GAMMA_LUT`	App will provide lookup table for Gamma Correction.
`MFX_CAM_GAMMA_VALUE`	App will provide Gamma Value for Gamma Correction.

Change History

This enumerator is available since SDK API 1.10.

mfxCamWhiteBalanceMode

Description

The `mfxCamWhiteBalanceMode` enumerator defines White Balance filter operation modes.

Name/Description

`MFX_CAM_WHITE_BALANCE_MANUAL`	App will provide coefficients for White Balance filter.

Change History

This enumerator is available since SDK API 1.13.

mfxCamBayerFormat

Description

The `mfxCamBayerFormat` enumerator defines supported Bayer patterns.

Figure 4 shows how Bayer format represented.

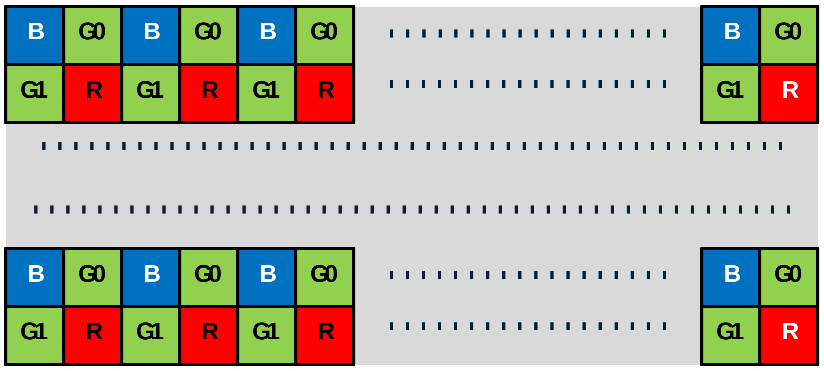


Figure 4: Bayer format representation in BGGR pattern

Name/Description

MFX_CAM_BAYER_BGGR	pixels representation:	BG GR
MFX_CAM_BAYER_RGGB	pixels representation:	RG GB
MFX_CAM_BAYER_GBRG	pixels representation:	GB RG
MFX_CAM_BAYER_GRBG	pixels representation:	GR BG

Change History

This enumerator is available since SDK API 1.10.

ColorFourCC

Description

The `ColorFourCC` enumerator itemizes color formats. Using all these FOURCC with IOPattern MFX_IOPATTERN_IN_VIDEO_MEMORY/MFX_IOPATTERN_OUT_VIDEO_MEMORY, except for RGB32, can be limited by the RAW Accelerator implementation and/or graphics driver implementation. Application should consider using Query mechanism to check support.

Here we cover only formats supported by the RAW Accelerator or planned to be supported in nearest future.

Input data of 10, 12, or 14 bits per channel should be provided shifted to the most significant bit (MSB) side, as shown on Figure 5, and appropriate fields required to be set in `mfxFrameInfo` structure: `BitDepthLuma and Shift`. `Shift` should be non-zero value.

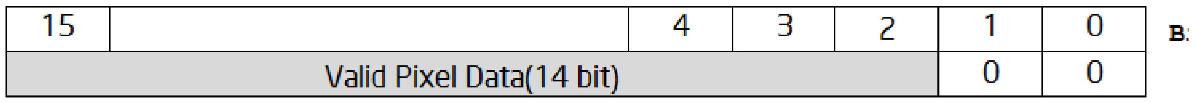


Figure 5: MSB data alignment for 14 bit data

If the output FOURCC is set to MFX_FOURCC_ARGB16 (ARGB64), the output bit depth is equal to the input bit depth, the output data being MSB shifted as well.

Name/Description

`MFX_FOURCC_RGB4`	RGB4 (ARGB32) color planes. Supported only as output format in current version of the Raw Media Accelerator.
`MFX_FOURCC_A2RGB10`	10-bit ARGB color format packed in 32 bits. 'A' channel is two MSBs, followed by 'R', 'G', and 'B' channels, in this order. This format is mapped to DXGI_FORMAT_R10G10B10A2_UNORM or D3DFMT_A2R10G10B10. Note: Not supported in current version.
`MFX_FOURCC_ARGB16`	ARGB64: color format for ARGB data of up to 16 bits per channel, with 'A' channel occupying 16 MSBs, followed by 'R', 'G', and 'B' channels, in this order, each occupying 16 bits. This format is mapped to DXGI_FORMAT_R16G16B16A16_UINT or D3DFMT_A16B16G16R16 formats. Supported only as output format in current version of the Raw Accelerator.
`MFX_FOURCC_R16`	16-bit single channel color format for Bayer samples. Supported only as input format in current version of the Raw Accelerator. Current implementation supports only system memory

input.

Change History

These format values are available since SDK API 1.9.

ExtendedBufferID

Description

The `ExtendedBufferID` enumerator itemizes and defines identifiers (`BufferId`) for extended buffers or video processing algorithm identifiers. Here are described Extended buffer IDs, applicable to the RAW Accelerator only.

Name/Description

`MFX_EXTBUF_CAM_PIPECONTROL`	This extended buffer defines mandatory control for plug-in. See the **mfxExtCamPipeControl** structure for details.
`MFX_EXTBUF_CAM_GAMMA_CORRECTION`	This extended buffer defines optional control for plug-in, allowing enabling Gamma Correction feature. See the **mfxExtCamGammaCorrection** structure for details.
`MFX_EXTBUF_CAM_PADDING`	This extended buffer defines optional control for plug-in, allowing application to do padding itself and provide padded surface to plugin. See the **mfxExtCamPadding** structure for details.
`MFX_EXTBUF_CAM_WHITE_BALANCE`	This extended buffer defines optional control for plug-in, allowing application to enable White Balance filter and provide correction coefficients. See the **mfxExtCamWhiteBalance** structure for details.
`MFX_EXTBUF_CAM_BLACK_LEVEL_CORRECTION`	This extended buffer defines optional control for plug-in, allowing application to enable Black Level Correction filter and provide correction coefficients. See the **mfxExtCamBlackLevelCorrection** structure for details.
`MFX_EXTBUF_CAM_COLOR_CORRECTION_3X3`	This extended buffer defines optional control for plug-in, allowing application to enable Color Correction filter and provide correction matrix coefficients. See the **mfxExtCamColorCorrection3x3** structure for details.
`MFX_EXTBUF_CAM_VIGNETTE_`	This extended buffer defines optional control for plug-in, allowing application to enable Vignette correction filter and provide correction

CORRECTION	map coefficients. See the **mfxExtCamVignetteCorrection** structure for details.
MFX_EXTBUF_CAM_HOT_PIXEL_REMOVAL	This extended buffer defines optional control for plug-in, allowing application to enable Hot Pixel removal filter and threshold control. See the **mfxExtCamHotPixelRemoval** structure for details.
MFX_EXTBUF_CAM_BAYER_DENOISE	This extended buffer defines optional control for plug-in, allowing application to enable Bayer Denoise filter and level control. See the **mfxExtCamBayerDenoise** structure for details.
MFX_EXTBUF_CAM_3DLUT	This extended buffer defines optional control for plug-in, allowing application to enable 3DLUT filter. See the **mfxExtCam3DLut** structure for details.

Change History

This enumerator is available since SDK API 1.10.

Appendix A: Programming Notes for D3D and system memory usage

Input/output memory type limitations for RAW accelerator plugin

Input D3D memory not supported in current implementation and no plans to support in future as RAW accelerator intended to be first component in pipeline.

Opaque memory not supported in current implementation.

Frames sizes exceeding 128 megabytes allocation not supported with D3D memory output. If size exceeded, Query will return MFX_ERR_UNSUPPORTED error. The application need to consider system memory out for RAW Accelerator plugin and then legacy VPP session to do Copy to D3D memory and other HW accelerated algorithms (e.g. scaling, color conversion for rendering).

ARGB64 output not supported with D3D memory in current implementation.

Surface allocation for best performance

To reach best performance with system memory, application must allocate memory aligned to page size: 4096 bytes and frame pitch aligned to 128 bytes. If frame not aligned, RAW accelerator need to copy frame data to aligned location, it brings more additional memory utilization and performance impact.

Appendix B: Notes for Photo processing usage scenario and reset operation

Partial frame processing

For cases when application require showing only part of processed photo and smoothly move active frame rectangle, the application can use partial fame processing mechanism through using Cropping parameters and Reset function.

Pipeline resetting

Pipeline reset can have negligible impact or serious impact to reset operation latency depended on parameters to be changed. There are 3 major operation types:

- Reset with resource re-allocation, required for cases when resolution changed to bigger than initialization will significantly affect performance and requires re-initialization.
- Reset without resource reallocation, but involving internal pipeline change:
    - Resolution change down will affect latency significantly, but not so much as resolution change up, as no reallocation required, if the application requires to change resolution up/down very often, the best option is to initialize with the biggest required resolution and change than with less latency.
    - Adding or removing filters from pipeline will affect latency but will depend what filters required to change. Please see more in filters resetting chapter.
- Reset filter parameters – not involving any pipeline change and will have negligible impact for latency of reset operation.

Filters resetting

Reset filter parameters is most easy operation – not required reallocation, the application will get the smoothest change to picture processing result.

Reset function will involve synchronization with previously executed tasks in case if new filters added to pipeline and they belong to the same unit of operation. In current implementation such unites are:

- "Bayer correction" - all filters which are predecessors to Demosaic (White Balance, Black level correction)
- Demosaic – always enabled and cannot be changed.
- "Color correction" – followers to Demosaic(Gamma Correction, CCM).