White Paper

Using Unsupported Color Formats with the Intel® Media Software Development Kit

Introduction

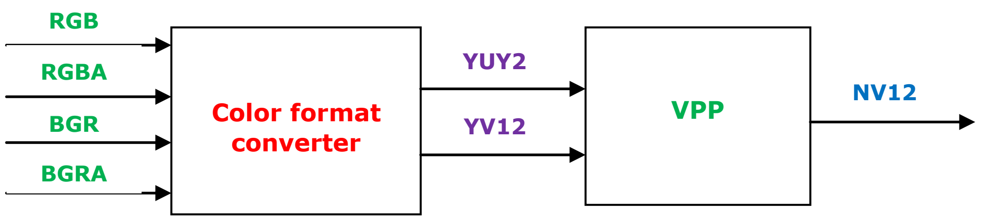
The Intel® Media Software Development Kit (Intel® Media SDK) is a software development library that exposes the media acceleration capabilities of Intel platforms for encoding, decoding and video preprocessing (VPP). VPP takes raw frames as input, performs some functions like color conversion, cropping, resizing, de-noising, de-interlacing and provides raw frames as output. However, VPP only supports certain type of input color formats of raw video frames like NV12, YV12 and YUY2. What happen if user has different type of input color formats and they want to take advantage of the Intel® Media SDK to enhance the VPP phase? In this article we will discuss how to use the Intel® Integrated Primitive Performance (Intel® IPP) to convert some common type of color formats like RGB, RGBA, BGR and BGRA into those that can be recognized by the Intel® Media SDK.

Contents

The Intel® Media Software Development Kit

As of this writing, the current version of the Intel® Media SDK is version 1.5. The version 1.5 of the Intel® Media SDK provides developers with a standard application programming interface (API) to create high-performance video solutions for consumer and professional uses. The API functions streamline and simplify video encoding, decoding, and preprocessing operations. Support is provided for encoding (including H.264 and MPEG-2 formats) and decoding (including H.264, MPEG-2, and VC-1 formats). The nice thing about the Intel® Media SDK is that developers can gain access to the hardware acceleration capabilities of the target platforms. Platforms that lack dedicated graphics hardware acceleration features still gain the benefit of tuned, optimized, and threaded software-based video encoding and decoding, tailoring performance to the capabilities of each system.

This paper will show how to convert RGB, RGBA, BGR and BGRA raw video of into YUY2 and YV12 before feeding them to VPP.



The Intel® Integrated Performance Primitives

The Intel® Integrated Performance Primitives (Intel IPP) library provides a wide range of functionality including general signal, image, speech, graphics, data compression, cryptography, text string processing, audio processing, vector manipulation, and matrix math; as well as more sophisticated primitives for the construction of audio, video and speech codecs including MP3 (MPEG-1 Audio, Layer 3), MPEG-4, H.264, VC-1,H.263, JPEG, JPEG2000, GSM-AMR* and G.729, plus computer vision. The Intel IPP makes use of new Intel architecture enhancements such as the Intel® 64 architecture (aka Intel® EM64T), Advanced Vector Extensions (Intel® AVX), Streaming SIMD Extensions (Intel® SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2), and MMX™ technology. Intel® IPP is optimized for a broad range of Intel microprocessors, including: Intel® Core™ i7 processors, Intel® Atom™ processors, Intel® Core™2 Quad processors, Intel® Core™2 Duo processors, Intel® Xeon® processors, Intel® Pentium® 4 processor and Intel® Itanium® 2 processors. This paper will talk only about how to use the image processing portion of the Intel IPP. The version 6.1 build 5 of the Intel IPP only supports converting other color formats to YUY2 and YV12. The new version, version 7.0 which is still in beta will support NV12.

Color Formats

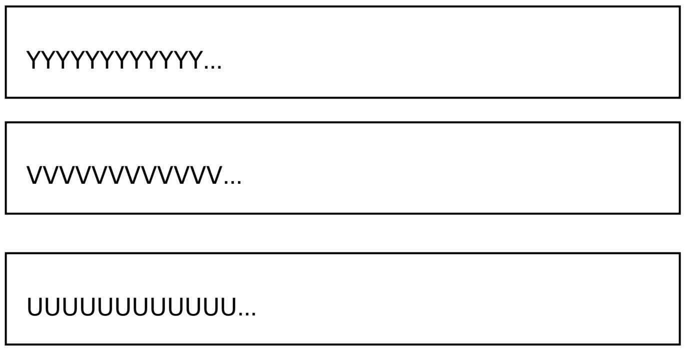
Below are the descriptions of the three color formats that are supported by the Intel® Media SDK VPP. They are YUY2, YV12 and NV12. There are two ways for a YUV image to be stored. The first one is called packed format and the second one is planar format. In the packed format, Y, U and V are stored next to each other. In the planar format, all the Y values are put together as wells as U and V. For more information about YUV color formats, Please refer to the web site [3].

Packed Format:

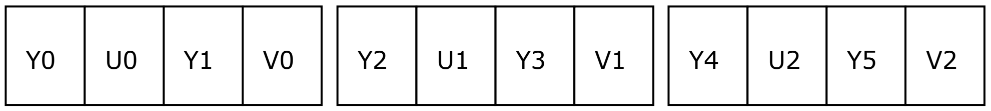
YUV YUV YUV…

YUV YUV YUV…

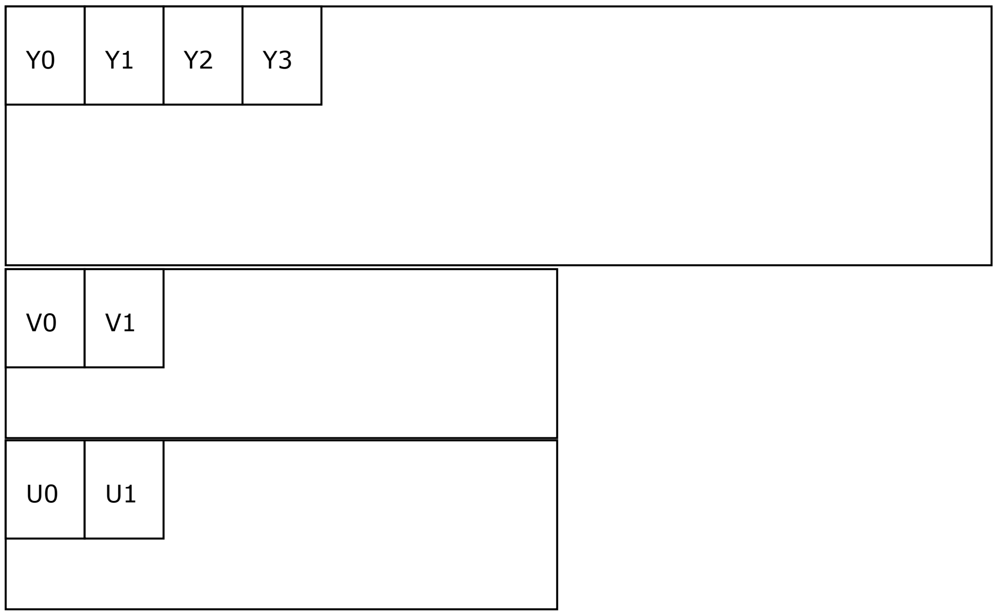
Planar format:



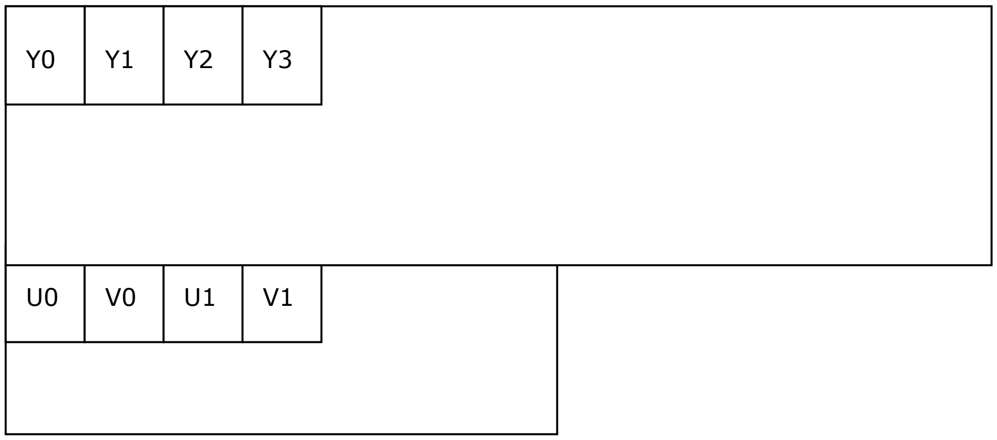
YUY2 is the packed format with 4:2:2 sampling. In YUY2 format the data can be treated as an array of unsigned **char** values. This can be illustrated as followed:



YV12 is the planar format with 4:2:0 sampling. In YV12 format all the Y samples appear first in an array of unsigned **char** values. The V samples will be stored after the Y samples and the U samples will be stored last. This can be illustrated as followed:



NV12 is the planar format YUV with 4:2:0 sampling in which U and V are interleave. In NV12 format the data can be treated as an array of unsigned **char** values. This can be illustrated as followed:



Color Conversions

Developers can write their own color conversion functions to convert from RGB, RGBA, BGR and BGRA to YUY2, YV12 or NV12. However, it is highly recommend that developers use the conversion functions provided by the Intel IPP since IPP functions are highly optimized across many Intel platforms. Below are some of the Intel IPP color conversions:

ippiRGBToYCbCr422_8u_P3C2R   - RGBP3 to YUY2

ippiRGBToYCbCr422_8u_C3C2R   - RGB24 to YUY2

ippiBGRToYCbCr422_8u_C3C2R   - BGR24 to YUY2

ippiBGRToYCbCr422_8u_AC4C2R - BGRA to YUY2

ippiBGRToYCrCb420_8u_C3P3R   - BGR24 to YV12

ippiBGRToYCrCb420_8u_AC4P3R – BGRA to YV12

ippiRGBToYCrCb420_8u_AC4P3R - RGBA to YV12

The following functions are only available in the Intel IPP version 7.0

ippiRGBToYCbCr420_8u_C3P2R - RGBA   to NV12

ippiRGBToYCbCr420_8u_C4P2R - RGB32   to NV12

The following sample code shows how to call the Intel IPP color conversion functions:

```
#pragma once
#include <ippcc.h>
#define UNKNOWN_VALUE                        9999
#define UNKNOWN_ERROR                        -9999
#define FUNCTION_NOT_YET_SUPPORT             -1
#define NO_SUCH_CONVERSION_WITH_THE_SPECIFIED_INPUT_PARAMETERS          -2
#define SOURCE_OR_DESTINATION_IS_NULL                                   -3
#define ROI_SIZE_HAS_ZERO_OR_NEGATIVE_VALUE                             -4

enum FuncIndex  { RGB_TO_YUY2 = 0,
                  RGBP3_TO_YUY2,
                  BGR_TO_YUY2,
                  BGRA_TO_YUY2,
                  BGR_TO_YV12,
                  BGRA_TO_YV12,
                  RGBA_TO_YV12,
                  // These 2 only work with IPP 7.0
                  RGBA_TO_NV12,
                  BGR_TO_NV12,
                };
class CColorCnv
{
public:
   CColorCnv();
   ~CColorCnv();
public:
```

```
   //int iColorConversionType;              // RGB_TO_YUY2, BGR_TO_YUY2, BGRA_TO_YV12, ...
   //unsigned char *pucSrc;                 // Pointer to the source of the region of interest (ROI) for pixel-order
format
   //unsigned char *ArraypucSrc[3];     // Array of pointers to the ROI of each planes for planar-order format
   //int iSrcStep;                                      // Distance in bytes between consecutive lines in
source.
   //unsigned char *pucDest;                // Pointer to the destination of the ROI for pixel-order format
   //unsigned char *ArraypucDest[3];    // Array of pointers to the ROI of each planes for planar-order format
   //int iDestStep;                                 // Distance in bytes between consecutive lines in destination.
   //int ArrayiDestStep[3];                 // Array of Distance in bytes between consecutive lines in destination
planes.



public:

   int ColorConversion(int Index,
                       const unsigned char* pucSrc,
                       const int iSrcStep,
                       unsigned char* pucDest,
                       int iDestStep,
                       IppiSize);

   int ColorConversion(int Index,
                       const unsigned char* pucSrc[3],
                       const int iSrcStep,
                       unsigned char* pucDest,
                       int iDestStep,
                       IppiSize);
   int ColorConversion(int Index,
                                          const unsigned char* pucSrc,
                                          const int iSrcStep,
                                          unsigned char* pucDest[3],
                                          int iDestStep[3],
                                          IppiSize);

 int GetReturnCode(int);


};
```

```
int CColorCnv::GetReturnCode(int iStatus)
{
   switch(iStatus)
   {
   case ippStsNoErr:
               return  0;
               break;

   case ippStsNullPtrErr:
               return  SOURCE_OR_DESTINATION_IS_NULL;
               break;

   case ippStsSizeErr:
               return  ROI_SIZE_HAS_ZERO_OR_NEGATIVE_VALUE;
               break;

   case FUNCTION_NOT_YET_SUPPORT:
               return  FUNCTION_NOT_YET_SUPPORT;
               break;

   default:
               return  NO_SUCH_CONVERSION_WITH_THE_SPECIFIED_INPUT_PARAMETERS;

   }
}

// IppiSize is define as:
//typedef struct
//{
//             int width;
//             int length;
//}            IppiSize;

// Function 0 if success else return a negative number
int CColorCnv::ColorConversion(int Index,
                                          const unsigned char* pucSrc,
 // Pointer to the source of the region of interest (ROI) for
 // pixel-order format
                                          const int iSrcStep,
   // Distance in bytes between consecutive lines in source.
                                          unsigned char* pucDest,
   // Pointer to the destination of the ROI for pixel-order format
                                          int iDestStep,
   // Distance in bytes between consecutive lines in destination.
                                          IppiSize ROISize)
   // Region of interest
{
   int iStatus = UNKNOWN_VALUE;
```

```
  switch(Index)
  {



  case RGB_TO_YUY2:
            iStatus = ippiRGBToYCbCr422_8u_C3C2R(pucSrc,
                                              iSrcStep,
                                              pucDest,
                                              iDestStep,
                                              ROISize);

            break;

  case BGR_TO_YUY2:
            iStatus = ippiBGRToYCbCr422_8u_C3C2R(pucSrc,
                                              iSrcStep,
                                              pucDest,
                                              iDestStep,
                                              ROISize);

            break;

  case BGRA_TO_YUY2:
            iStatus = ippiBGRToYCbCr422_8u_AC4C2R(pucSrc,
                                               iSrcStep,
                                               pucDest,
                                          iDestStep,
                                               ROISize);

            break;

  case RGBA_TO_NV12:
  case BGR_TO_NV12:
            iStatus = FUNCTION_NOT_YET_SUPPORT;
            break;

  default:
            iStatus = NO_SUCH_CONVERSION_WITH_THE_SPECIFIED_INPUT_PARAMETERS;

  }

  return GetReturnCode(iStatus);

}

// IppiSize is define as:
//typedef struct
//{
//          int width;
//          int length;
//}         IppiSize;

// Function 0 if success else return a negative number
int CColorCnv::ColorConversion(int Index,
                                  const unsigned char* pucSrc[3],
  // Array of pointers to the ROI of each planes for planar-order format
                                  const int iSrcStep,
   // Distance in bytes between consecutive lines in source.
                                                    unsigned char* pucDest,
   // Pointer to the destination of the ROI for pixel-order format
                                                    int iDestStep,
   // Distance in bytes between consecutive lines in destination.
                                                    IppiSize ROISize)
   // Region of interest
{
   int iStatus = UNKNOWN_VALUE;

   switch(Index)
   {
   case RGBP3_TO_YUY2:
                iStatus = ippiRGBToYCbCr422_8u_P3C2R(pucSrc,
                                                                   iSrcStep,
                                                                   pucDest,
                                                                   iDestStep,
                                                                   ROISize);

                break;


   case RGBA_TO_NV12:
   case BGR_TO_NV12:
                iStatus = FUNCTION_NOT_YET_SUPPORT;
                break;

   default:
                iStatus = NO_SUCH_CONVERSION_WITH_THE_SPECIFIED_INPUT_PARAMETERS;

   }

   return GetReturnCode(iStatus);

}


// IppiSize is define as:
//typedef struct
//{
//             int width;
//             int length;
//}            IppiSize;

// Function 0 if success else return a negative number
int CColorCnv::ColorConversion(int Index,
                                                    const unsigned char* pucSrc,
   // Pointer to the source of the region of interest (ROI)
 // for pixel-order format
                                                    const int iSrcStep,
   // Distance in bytes between consecutive lines in source.
                                                    unsigned char* pucDest[3],
   // Array of pointers to the ROI of each planes for planar-order format
                                                    int iDestStep[3],
   // Array of Distance in bytes between consecutive lines
 // in destination  planes.
                                                    IppiSize ROISize) // Region of interest
{
   int iStatus = UNKNOWN_VALUE;

   switch(Index)
```

```
  {
  case BGR_TO_YV12:
              iStatus = ippiBGRToYCrCb420_8u_C3P3R(pucSrc,
                                                                    iSrcStep,
                                                                    pucDest,
                                                              iDestStep,
                                                                    ROISize);

              break;

  case BGRA_TO_YV12:
              iStatus = ippiBGRToYCrCb420_8u_AC4P3R(pucSrc,
                                                                     iSrcStep,
                                                                     pucDest,
                                                                     iDestStep,
                                                                     ROISize);

              break;

  case RGBA_TO_YV12:
              iStatus = ippiRGBToYCrCb420_8u_AC4P3R(pucSrc,
                                                                     iSrcStep,
                                                                     pucDest,
                                                                     iDestStep,
                                                                     ROISize);

              break;

  case RGBA_TO_NV12:
  case BGR_TO_NV12:
              iStatus = FUNCTION_NOT_YET_SUPPORT;
              break;

  default:
              iStatus = NO_SUCH_CONVERSION_WITH_THE_SPECIFIED_INPUT_PARAMETERS;

  }

  return GetReturnCode(iStatus);

}
```

To use the conversion function in the above class, developers need to do something like this:

```
int iStatus;

CColorCnv cnv;

iStatus = cnv.ColorConversion(RGB_TO_YUY2,  // Which color conversion to perform

                                        pucSrc,

                                        iSrcStep,

                                        pucDest,

                                        iDestStep,

                                     ROISize);
```

For the description of the variables please refer to the comments in the sample code since they are different depending on which format (YUY2, YV12 or NV12) to convert to.

Conclusion

Using the Intel IPP color conversion functions to convert raw video format unsupported by the Intel® Media SDK into those (YUY2, YV12) recognized by the Intel® Media SDK. The advantage of using the Intel IPP functions is that they are highly optimized for many Intel platforms.

References

[1] Intel® Media Software Development Kit, http://www.intel.com/software/mediasdk/

[2] Intel® Integrated Performance Primitives, Reference Manual, November 2009. Refer to Intel® IPP http://software.intel.com/en-us/intel-ipp/

[3] http://www.fourcc.org

About the Author

Khang Nguyen is a Software engineer with the Core Scale Engineering group at Intel. He is currently working on developing tools relating to parallelism and power matters in applications. He has been with Intel since 2000.