Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2024, 11(5):82-85**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

OpenVX Based ISP

Karthik Poduval

Abstract This paper describes how to build a simple Image Signal Processor (ISP) using the OpenVX API, Khronos sample implementation and a custom OpenVX Demosaic kernel.

Keywords OpenVX, ISP

1. Introduction

Image Signal Processor or ISP is referred to the processing done on an image captured from a RAW image sensor to make it suitable for viewing by humans. RAW Image sensors are typically in bayer [1] format. Bayer Color Filter Array (CFA) is a pattern where each pixel of a camera sensor array captures either a green, blue or red pixel by the means of a color filter placed on top of the image sensor. Such Bayer CFA's contain twice the number of green as compared to red and blue pixels. This is based on the theory that human eye is more sensitive to green color as compared to red and blue. There are 4 possible bayer patterns.

- BGGR • RGGB • GRBG
- RGBG

One of the important stages of an ISP is to change from this Bayer format to RGB format which is what is commonly understood and used by imaging components. Some other stages of an ISP may include color space conversion from RGB to YUV formats and scaling up or down images. There are many other stages in an ISP.

OpenVX [2] is an open royalty free cross-platform standard for computer vision applications. The standard also defines a cross-platform API and provides an sample implementation [3]. OpenVX API provides a graph like representation of a series OpenVX kernels (a vision processing block). From the version 1.3 documentation, the following example graph can be seen Figure 1 and the corresponding code that produced this graph can be seen in Figure 2. From this example we can see that OpenVX is modular API and arbitrary vision processing graphs can be produced using the API.

For the purposes of an ISP graph in OpenVX, though there are plenty of kernels present in the sample implementation, for the purposes of an ISP a kernel to convert the Bayer format to RGB isn't something that is present. Hence one of the primary things to do is to create this kernel and this kernel will be known as demosaic as it is commonly known as Demosacing [4] or simply demosiac.

2. OPENVX ISP

As we don't have a ready OpenVX demosacic kernel, we create a custom VX kernel for demosacing. To create a custom kernel, we create our custom kernel by following the OpenVX

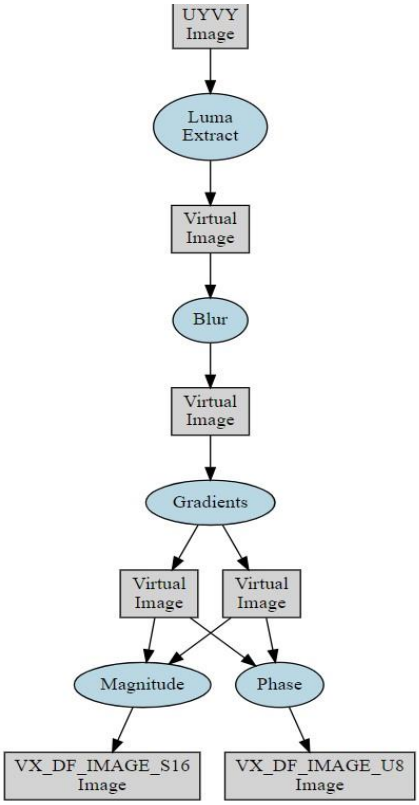

Figure 1: OpenVX Example Graph

1.3 specifications [5] and creating a custom enum for our kernel.

/*! \brief The list of ISP Kernels.

* \ingroup group_xyz_ext

*/ enum vx_kernel_isp_e {

/*! \brief The Example User Defined Kernel */ VX_KERNEL_KHR_DEMOSAIC = VX_KERNEL_BASE(VX_ID_DEFAULT, VX_LIBRARY_ISP) + // up to 0xFFF kernel enums can be created.

};

Next, we write the basic signatures for the kernel using a descriptor.

vx_kernel_description_t demosaic_kernel = {

VX_KERNEL_KHR_DEMOSAIC, VX_KERNEL_NAME_KHR_ISP_DEMOSAIC, vxDemosaicKernel, demosaic_kernel_params,

```
vx_context context = vxCreateContext();
vx_image images[] = {
        vxCreateImage(context, 640, 480, VX_DF_IMAGE_UYVY),
        vxCreateImage(context, 640, 480, VX_DF_IMAGE_S16),
        vxCreateImage(context, 640, 480, VX_DF_IMAGE_U8),
};
vx_graph graph = vxCreateGraph(context);
vx_image virts[] = {
        vxCreateVirtualImage(graph, 0, 0, VX_DF_IMAGE_VIRT),
        vxCreateVirtualImage(graph, 0, 0, VX_DF_IMAGE_VIRT),
        vxCreateVirtualImage(graph, 0, 0, VX_DF_IMAGE_VIRT),
        vxCreateVirtualImage(graph, 0, 0, VX_DF_IMAGE_VIRT),
};

vxChannelExtractNode(graph, images[0], VX_CHANNEL_Y, virts[0]),
vxGaussian3x3Node(graph, virts[0], virts[1]),
vxSobel3x3Node(graph, virts[1], virts[2], virts[3]),
vxMagnitudeNode(graph, virts[2], virts[3], images[1]),
vxPhaseNode(graph, virts[2], virts[3], images[2]),

status = vxVerifyGraph(graph);
if (status == VX_SUCCESS)
{
    status = vxProcessGraph(graph);
}
vxReleaseContext(&context); /* this will release everything */
```

Figure 2: OpenVX Example Graph Code

```
dimof(demosaic_kernel_params), vxDemosaicValidator, NULL,
NULL,
NULL,
NULL,
};
```

The important thing to note is that there is demosaic_kernel_params which looks like.

```
static vx_param_description_t demosaic_kernel_params[] = { {VX_INPUT, VX_TYPE_IMAGE,
VX_PARAMETER_STATE_REQUIRED},
{VX_OUTPUT, VX_TYPE_IMAGE,
VX_PARAMETER_STATE_REQUIRED},
};
```

The parameter space says that the kernel accepts an input and produces an output, which is our case is bayer format input and a RGB output. This simplistic kernel assumes a given bayer pattern of RGGB. Another important function is of parameter validation provided by vxDemosaicValidator whose function is to validate if input is a 16 bit array (RAW16) and output is of type RGGB. The validator also makes sure to set the output image metadata to match that of the input provided. The next interesting method is that of the demosaic kernel itself which actually implements the image processing kernel. The algorithm used here is a simple bilinear interpolation and does not deal with the corner pixels in its current state [6].

3. ISP PIPELINE

To test the kernel and implement an ISP Pipeline, we stitch the demosaic kernel along with built-in kernels to from a simple ISP that takes bayer and produces a downscaled RGB
Image. The ISP pipeline would look like Figure 3.
A sample application that feeds a 640x480 Bayer RGGB pattern (generated via the vivid [7] test video driver). The image is passed through the pipeline and scaled up to 1280x960. The code fragments for this VX ISP can be found as.

- Demosiac kernel
- https://github.com/karthikpoduval/
- vx-isp/blob/demosaic/vx_demosaic.c • Test ISP Pipeline
- https://github.com/karthikpoduval/
- vx-isp/blob/demosaic/test.c

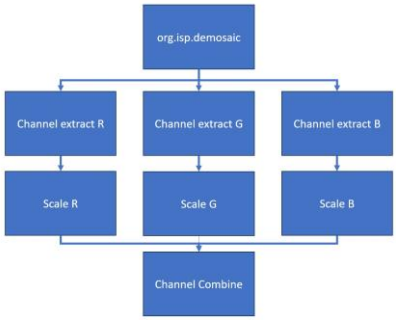

Figure 3: OpenVX ISP Graph

4. Conclusion

The paper demonstrates how to use OpenVX to build and ISP pipeline processing graph. ISP pipelines can be fairly complex however this simple ISP pipeline only includes demosaic and scaling. In the future other ISP blocks could be added like color space conversion, statistics, gain etc. This kind of a framework can be used to implement ISP functionality in software.

References

[1]. B. E. Bayer, “Color imaging array,” patentus 3971065.
[2]. “Openvx.” [Online]. Available: https://www.khronos.org/openvx/
[3]. “Openvx sample implementation.” [Online]. Available: https://github. com/KhronosGroup/OpenVX-sample-impl
[4]. “Demosaicing.”[Online]. Available: https://en.wikipedia.org/wiki/ Demosaicing
[5]. “Openvx 1.3.” [Online]. Available: https://registry.khronos.org/OpenVX/ specs/1.3.1/html/OpenVX Specification 1 3 1.html
[6]. “Vx isp.” [Online]. Available: https://github.com/karthikpoduval/vx-isp/ tree/demosaic
[7]. “vivid.” [Online]. Available: https://www.kernel.org/doc/html/v4.8/media/ v4l-drivers/vivid.html