Imagination OpenVX & OpenCL
User Nodes for Hough lines, a worked example

Chris Longstaff

4th May, 2016

Imagination's OpenVX implementation

PowerVR GPU OpenVX & OpenCL support

- **Currently leverages our OpenCL drivers**
- **We have Khronos conformance for OpenVX 1.0.1 on 3 platforms**
  - For development and early evaluation on Linux PC
  - For embedded system development under Linux
  - For mobile device development with Android
- **Our implementation works with our PowerVR GPU cores series 6 and later**
- **Other platform submissions will be made appropriate to our customers' requirements**
- **XML extension guarantees porting of OpenVX graphs between devices**

The Hough Transform

Introduction

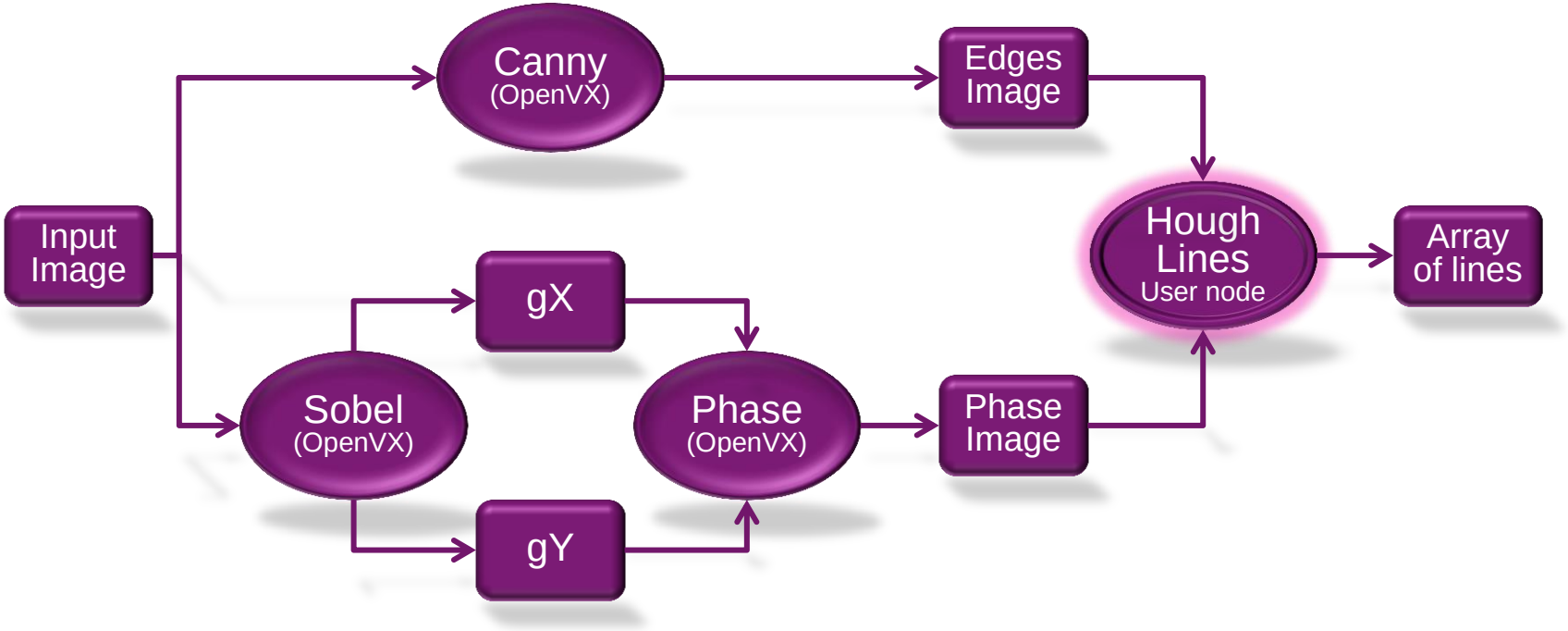
- **OpenVX has primitives for edge detection**
  - Canny edge-detect filter
  - Sobel filter
- **Neither of these can reconstruct geometric shapes in the input image**
  - Edges may not be complete
    - Ill-defined
    - Noise
    - Obstruction, fog etc.
- **The Hough Transform can help us**
  - It allows us to “join the dots” in the output of Canny and Sobel
  - Detailed discussion beyond the scope of this presentation

Hough Lines

- **Hough Transform can be used to detect**
  - Circles
  - Lines
  - Other geometric shapes
- **In this presentation we are looking at a user node to reconstruct lines**

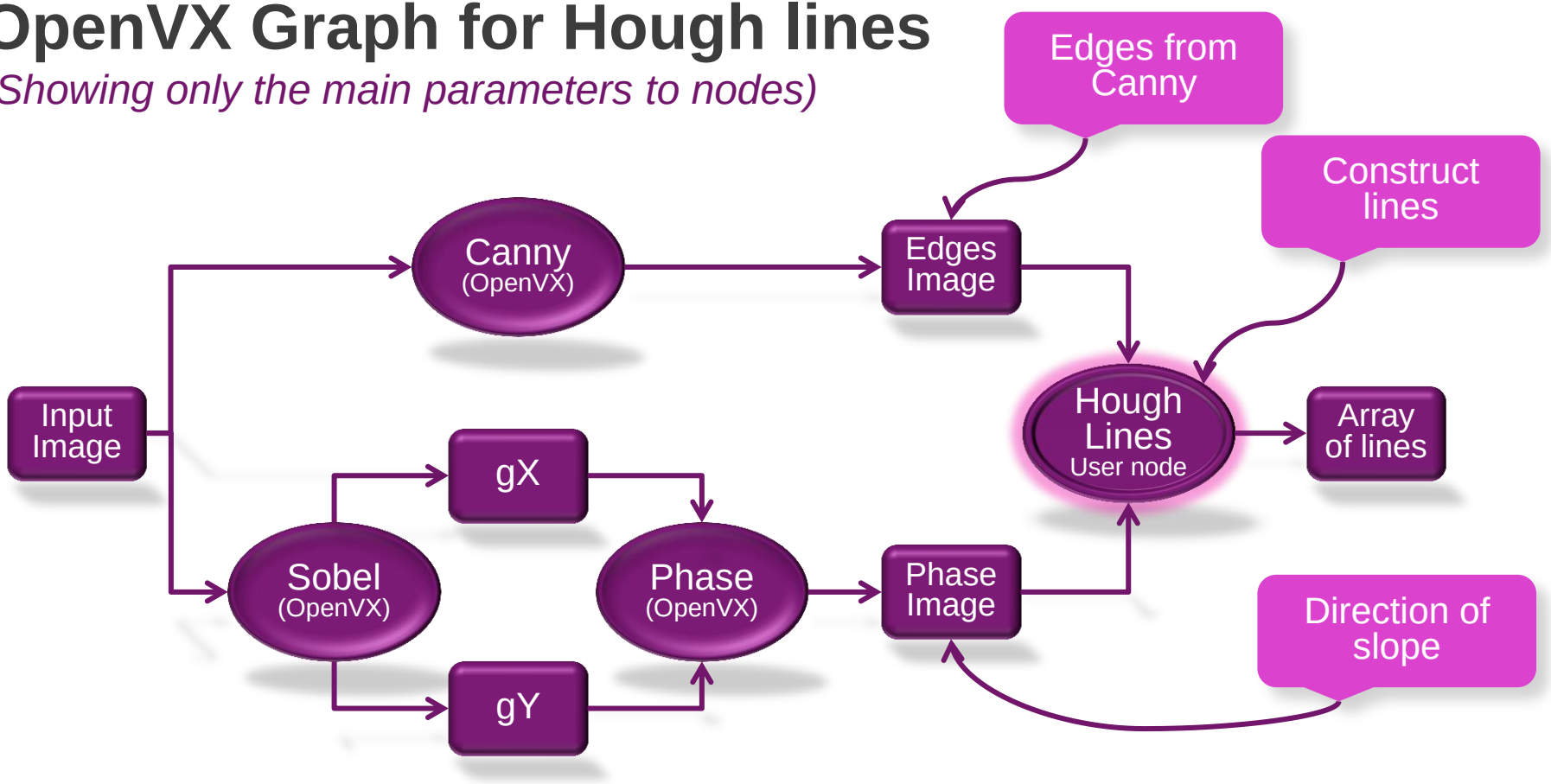
OpenVX Graph for Hough lines

(Showing only the main parameters to nodes)



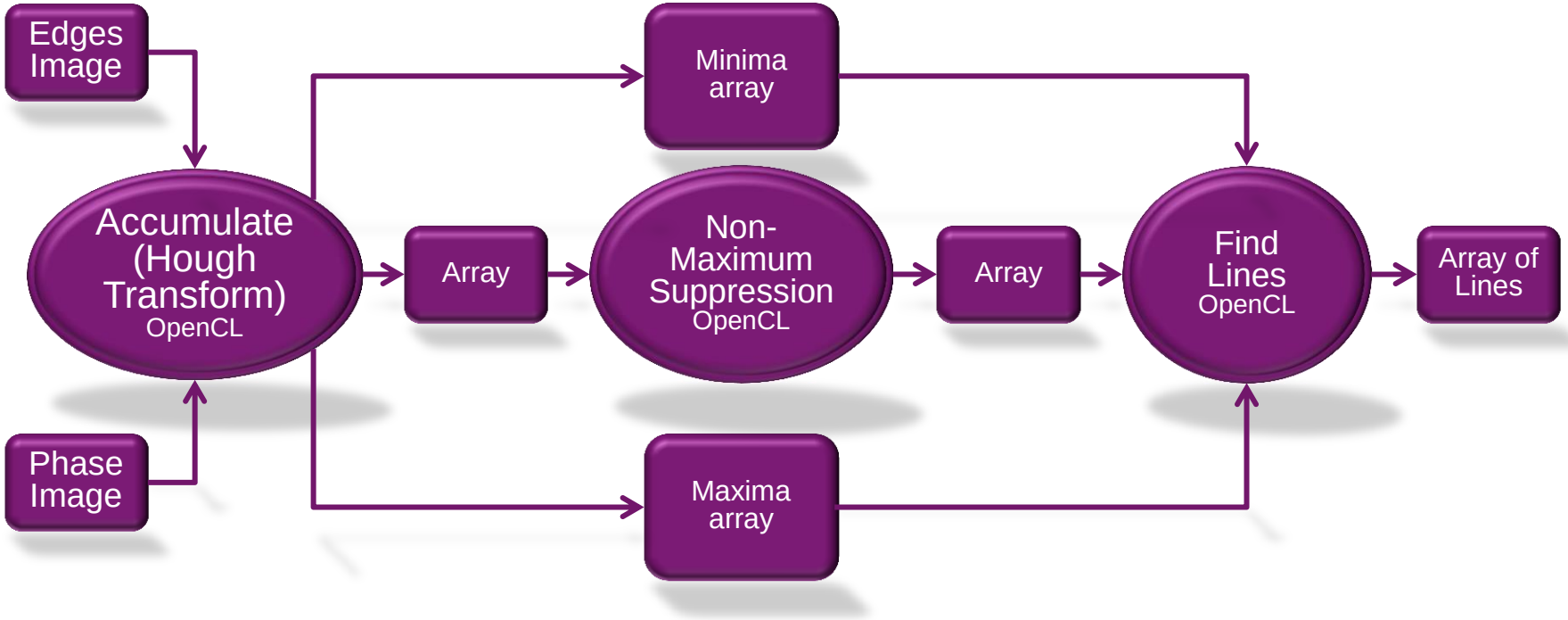
OpenVX Graph for Hough lines

(Showing only the main parameters to nodes)



Hough Lines OpenCL User Node

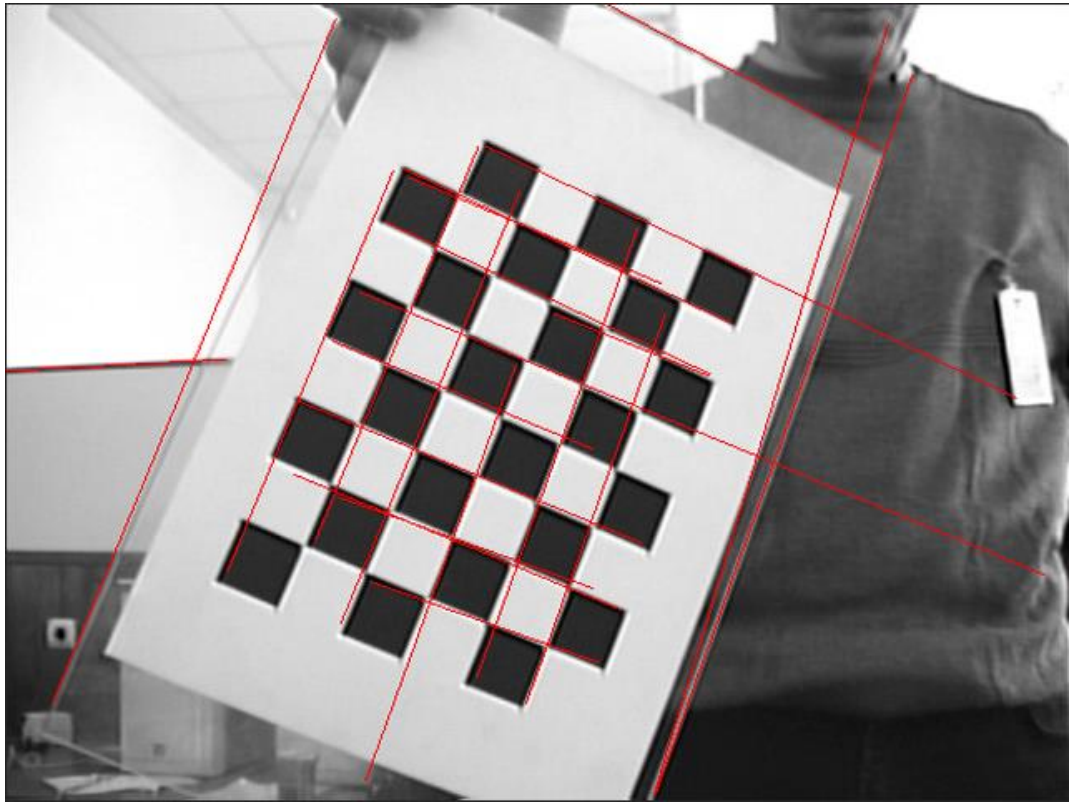
The user node uses three OpenCL kernels



Example output of test program

Output of a simple test program that overlays the graph input image with lines drawn from the output data.

In this simple implementation, the positioning of the lines has a low angular resolution, but it demonstrates the principal.



Using OpenCL in the user node

Need to avoid wasteful data copies

- **The user node can of course use any method to generate its results**
- **However, OpenVX 1.0.1 assumes that the code is running on the host machine**
- **OpenVX data types are opaque, there are APIs to access them, e.g.:**
  - **vxAccessImagePatch to access the pixels of an image**
  - **vxAccessArrayRange to access items in an array**
- **Copying OpenVX data into OpenCL objects is an avoidable overhead**
- **Our implementation uses OpenCL "under the hood"**
  - Data is already in OpenCL format
  - We just need a way of accessing it…

Imagination extensions for OpenCL

Defined in vx_img_extensions.h

- **OpenVX data structures (image, array etc.) are opaque**
  - Cannot be manipulated directly by OpenCL kernels
- **Our implementation uses OpenCL “under the hood”**
  - User nodes must “play nicely” and so need access to the OpenCL context

- **We have four functions which allow User node to use OpenCL:**
  - Get the OpenCL context and device id being used by a given OpenVX context
  - Get the OpenCL memory handle for an OpenVX image
  - Get the OpenCL memory handle for an OpenVX array
  - Update the number of items in an OpenVX vx_array

Comparative performance

…was it worth writing in OpenCL?

- **In our implementation with C**
  - User node accounts for up to 20% of graph execution time
- **In our implementation using OpenCL**
  - User node accounts for around 5% of graph execution time
- **Similar figures obtained for:**
  - I86 PC with GPU on reference card
  - Dell Venue

- **OpenCL gave us a 4x performance improvement in user node**
  - Translating to 16% reduction in graph execution time for this example

In conclusion

- User nodes are an effective way of adding functionality to OpenVX
- Imagination provides APIs to enable the use of OpenCL in user nodes
- The simple example of Hough Lines shows how these APIs can be used
- A version using OpenCL shows a significant performance advantage

- Full source code for the examples is available from Imagination upon request.

Reference & Details

Imagination extensions for OpenCL

Defined in vx_img_extensions.h

- **OpenVX data structures (image, array etc.) are opaque**
  - Cannot be manipulated directly by OpenCL kernels
- **Our implementation uses OpenCL “under the hood”**
  - User nodes must “play nicely” and so need access to the OpenCL context

- **We have four functions which allow User node to use OpenCL:**
  - Get the OpenCL context and device id being used by a given OpenVX context
  - Get the OpenCL memory handle for an OpenVX image
  - Get the OpenCL memory handle for an OpenVX array
  - Update the number of items in an OpenVX vx_array

Get the OpenCL context and device id

- **vxExposeCLEnvironmentIMG**
  - Input: vx_context pContext — The vx_context to query
  - Output: cl_device_id *pOclDeviceID — The OpenCL cl_device_id
  - Output: cl_context *pOclContext — The OpenCL cl_context
  - Returns VX_SUCCESS on success
- **Use this function to get the OpenCL context for an OpenVX context**
  - If you are going to use OpenCL in a user node, then you need the context and device id
  - …play nicely!

Get the cl_mem handle for a vx_image

- **vxExposeCLImageBackingInVXImageIMG**
  - Input: vx_image image — The vx_image you want to get the backing to.
  - Input: vx_uint32 planeIdx — The vx_image plane index.
  - Output: cl_mem *pOCLmem — The cl_mem handle for the plane you asked for.
  - Returns VX_SUCCESS on success

- **Use this function to get the OpenCL memory handle for an OpenVX image**
  - A user node can then process the OpenVX image plane with an OpenCL kernel

Get the cl_mem handle for a vx_array

- **vxExposeCLArrayBackingInVXArrayIMG**
  - Input: vx_array array — The vx_array you want to get the backing to.
  - Output: cl_mem *pOCLmem — The cl_mem handle for the array.
  - Returns VX_SUCCESS on success

- **Use this function to get the OpenCL memory handle for an OpenVX array**
  - A user node can then process the OpenVX array with an OpenCL kernel

Update the number of items in a vx_array

- **vxArraySetNumItemsIMG**
  - Input: vx_array array The vx_array for which you want to set the number of items
  - Input: vx_size numItems The new number of items
  - Returns VX_SUCCESS on success. Failure if numItems > capacity or invalid vx_array

- **Use this function to set the number of items in an OpenVX array**
  - OpenVX arrays are opaque, so an OpenCL kernel cannot directly set the number of items
  - This function is provided in case you wish to change the number of items in the array
  - Note that when writing items to the array, a kernel must not exceed the array's capacity

Imagination VX extensions

- **vxExposeCLEnvironmentIMG**
  - Use this function to get the OpenCL context for an OpenVX context
    - If you are going to use OpenCL in a user node, then you need the context and device id
- **vxExposeCLImageBackingInVXImageIMG**
  - Use this function to get the OpenCL memory handle for an OpenVX image
    - A user node can then process the OpenVX image plane with an OpenCL kernel
- **vxExposeCLArrayBackingInVXArrayIMG**
  - Use this function to get the OpenCL memory handle for an OpenVX array
    - A user node can then process the OpenVX array with an OpenCL kernel
- **vxArraySetNumItemsIMG**
  - Use this function to set the number of items in an OpenVX array
    - OpenVX arrays are opaque, so an OpenCL kernel cannot directly set the number of items

Creating the graph and adding the OpenVX nodes

```
graph = vxCreateGraph(ctx);

srcImage = vxCreateImage(ctx, width, height, VX_DF_IMAGE_U8);
dstImage = vxCreateImage(ctx, width, height, VX_DF_IMAGE_U8);
gxImage = vxCreateVirtualImage(graph, width, height, VX_DF_IMAGE_S16);
gyImage = vxCreateVirtualImage(graph, width, height, VX_DF_IMAGE_S16);
phaseImage = vxCreateImage(ctx, width, height, VX_DF_IMAGE_U8);
lineArray = vxCreateArray(ctx, VX_TYPE_RECTANGLE, 1024);
vx_enum attrdata = VX_THRESHOLD_TYPE_RANGE;
vx_int32 lowerdata = 100;
vx_int32 upperdata = 220;
vx_int32 gradient = 3;
vx_enum normType = VX_NORM_L1;
thresh = vxCreateThreshold(ctx, attrdata, VX_TYPE_UINT8);
vxSetThresholdAttribute(thresh, VX_THRESHOLD_TYPE, &attrdata, sizeof(attrdata));
vxSetThresholdAttribute(thresh, VX_THRESHOLD_THRESHOLD_LOWER, &lowerdata, sizeof(lowerdata));
vxSetThresholdAttribute(thresh, VX_THRESHOLD_THRESHOLD_UPPER, &upperdata, sizeof(upperdata));

cannyNode = vxCannyEdgeDetectorNode(graph, srcImage, thresh, gradient, normType, dstImage );
sobelNode = vxSobel3x3Node(graph, srcImage, gxImage, gyImage);
phaseNode = vxPhaseNode(graph, gxImage, gyImage, phaseImage);
```

Adding the user node to the graph

```
//houghlinesnode
vx_kernel userKernel = vxGetKernelByEnum(ctx, VX_KERNEL_IMGTEC_HOUGH_LINES);
if (userKernel != NULL)
{
    houghNode = vxCreateGenericNode(graph, userKernel);
    if (vxGetStatus((vx_reference)houghNode) == VX_SUCCESS)
    {
        vx_uint32 threshold = 90;
        vx_uint32 numAngleBins = 64;  //rho
        vx_uint32 numDistBins = 192;  //theta
        houghThresh = vxCreateScalar(ctx, VX_TYPE_UINT32, &threshold);
        houghNumAngleBins = vxCreateScalar(ctx, VX_TYPE_UINT32, &numAngleBins);
        >houghNumDistBins = vxCreateScalar(ctx, VX_TYPE_UINT32, &numDistBins);
        vx_status s1 = vxSetParameterByIndex(v->houghNode, 0, (vx_reference)dstImage);
        s1 |= vxSetParameterByIndex(houghNode, 1, (vx_reference)phaseImage);
        s1 |= vxSetParameterByIndex(houghNode, 2, (vx_reference)houghThresh);
        s1 |= vxSetParameterByIndex(houghNode, 3, (vx_reference)houghNumAngleBins);
        s1 |= vxSetParameterByIndex(houghNode, 4, (vx_reference)houghNumDistBins);
        s1 |= vxSetParameterByIndex(houghNode, 5, (vx_reference)lineArray);

        if (s1 == VX_SUCCESS)
        {
```

Getting the OpenCL context

A separate OpenCL command queue is created for the user node

```
vx_context ctx = StartOpenVX();

if (ctx != NULL)
{
    v->context = ctx;

    if (vxExposeCLEnvironmentIMG(ctx, &oclDeviceId, &oclContext) != VX_SUCCESS)
    {
        printf("Failed to expose cl environment\n");
    }

    houghLinesInitUserKernel(ctx);

    cl_int clerror;
    oclQueue = clCreateCommandQueue(oclContext, oclDeviceId,
               CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE | CL_QUEUE_PROFILING_ENABLE, &clerror);
    if (clerror != CL_SUCCESS)
        printf("TestExHoughLinesInit failed to init oclqueue\n");
```

Building the OpenCL kernels

In our example the OpenCL is built from source

- **Usual OpenCL program building code added to houghLinesInitUserKernel()**

```
houghProgram = clCreateProgramWithSource(oclContext, 1, (const char **)&source[0],
                                         (size_t*)&sourceLen, &clerror);

if (clerror == CL_SUCCESS)
{
    char options[]="";
    clerror = clBuildProgram(houghProgram,1, &oclDeviceId,options,NULL,NULL);

    if (clerror == CL_SUCCESS)
    {
        houghAccumulateKernel = clCreateKernel(houghProgram, "houghAccumulate", &clerror);
        CHECK_CL_ERROR("houghLinesInitUserKernel failed to find kernel 0");

        houghNonMaxSuppressionKernel = clCreateKernel(houghProgram, "houghNonMaxSuppression",
                                                      &clerror);
        CHECK_CL_ERROR("houghLinesInitUserKernel failed to find kernel 1");

        houghFindLinesKernel = clCreateKernel(houghProgram, "houghFindLines", &clerror);
        CHECK_CL_ERROR("houghLinesInitUserKernel failed to find kernel 2");
    }
    else
```

Getting the OpenCL data objects

Snippets of code from houghLinesInit()

```
vxstatus = vxExposeCLImageBackingInVXImageIMG(cannyImage, 0, &cannyMem);
CHECK_VX_STATUS("houghLinesInit Failed to expose cannyImage cl_mem");

vxstatus = vxExposeCLImageBackingInVXImageIMG(phaseImage, 0, &phaseMem);
CHECK_VX_STATUS("houghLinesInit Failed to expose phaseImage cl_mem");

vxstatus = vxExposeCLArrayBackingInVXArrayIMG(lineArray, &arrayMem);
CHECK_VX_STATUS("houghLinesInit Failed to expose lineArray cl_mem");

vxQueryImage(cannyImage, VX_IMAGE_WIDTH, &imageWidth, sizeof(vx_uint32));
vxQueryImage(cannyImage, VX_IMAGE_HEIGHT, &imageHeight, sizeof(vx_uint32));

vx_size cap;
vxQueryArray(lineArray, VX_ARRAY_CAPACITY, &cap, sizeof(vx_size));
maxLines = (cl_uint)cap;

//allocate the houghAcc/houghMin/houghMax buffers
int size = numABins * numDBins * 4;
int flags = CL MEM READ WRITE | CL MEM ALLOC HOST PTR;

houghAccSrc = clCreateBuffer(oclContext, flags, size, NULL, &clerror);
CHECK_CL_ERROR("houghLinesInit failed to alloc houghAccSrc");
houghAccDst = clCreateBuffer(oclContext, flags, size, NULL, &clerror);
CHECK_CL_ERROR("houghLinesInit failed to
// etc…
```

Setting the OpenCL kernel parameters

Snippets of code from houghLinesInit()

- **OpenCL kernel parameters are set as normal, using the handles obtained**

```
//set kernel args
clSetKernelArg(houghAccumulateKernel, 0, sizeof(cl_mem), (void*)&cannyMem);
clSetKernelArg(houghAccumulateKernel, 1, sizeof(cl_mem), (void*)&phaseMem);
clSetKernelArg(houghAccumulateKernel, 2, sizeof(cl_uint), (void*)&numABins);
clSetKernelArg(houghAccumulateKernel, 3, sizeof(cl_uint), (void*)&numDBins);
clSetKernelArg(houghAccumulateKernel, 4,
// etc…

clSetKernelArg(houghNonMaxSuppressionKernel, 0, sizeof(cl_mem), (void*)&houghAccSrc);
clSetKernelArg(houghNonMaxSuppressionKernel, 1, sizeof(cl_uint), (void*)&numABins);
clSetKernelArg(houghNonMaxSuppressionKernel, 2, sizeof(cl_uint), (void*)&numDBins);
clSetKernelArg(houghNonMaxSuppressionKernel, 3,
// etc…
```

Running the OpenCL kernels

code from houghLinesKernel(), showing how the number of array items is set

```
clEnqueueFillBuffer(oclQueue, houghAccSrc, &accInit, 4, 0, numABins*numDBins*4, 0, NULL,
                    &fillEvents[0]);
clEnqueueFillBuffer(oclQueue, houghMin, &minInit, 4, 0, numABins*numDBins*4, 0, NULL,
                    &fillEvents[1]);
clEnqueueFillBuffer(oclQueue, houghMax, &maxInit, 4, 0, numABins*numDBins*4, 0, NULL,
                    &fillEvents[2]);
clEnqueueFillBuffer(oclQueue, houghTempInts, &tempInit[0], 8, 0, 8, 0, NULL, &fillEvents[3]);

clEnqueueNDRangeKernel(oclQueue, houghAccumulateKernel, 2, NULL, globalWorkSize,
                    localWorkSize, 4, &fillEvents[0], &event[0]);

clEnqueueNDRangeKernel(oclQueue, houghNonMaxSuppressionKernel, 2, NULL, globalWorkSizeBins,
                    localWorkSizeBins, 1, &event[0], &event[1]);

clEnqueueNDRangeKernel(oclQueue, houghFindLinesKernel, 2, NULL, globalWorkSizeBins,
                    localWorkSizeBins, 1, &event[1], &event[2]);

//Read back tempInts from CL memory. tempInts[1] is the new array count for the vx_array.
cl_uint tempInts[2]={0,0};
clEnqueueReadBuffer(oclQueue, houghTempInts, CL_TRUE, 0, 8, &tempInts[0], 1, &event[2],
                    &event[3]);
clWaitForEvents(1, &event[3]);

// Call our vxArraySetNumItems extension to set new size.
vxArraySetNumItemsIMG(lineArray, (vx_size)tempInts[1]);
```

Thank you for your attention

www.imgtec.com