Vision Functions

Functions in OpenVX may be graph node or immediate mode. The parameter graph is the reference to the graph, and the parameter context is the reference to the overall context.

• Graph mode. (“Node” in the function name) May be created and linked together, verified, then executed as often as needed.
• Immediate mode. Executed on a context immediately as if single node graphs with no leaking side-effects.

Absolute Difference [3.1]

\[ (x-x) \] is the expression, \[ vx\_diff(x, y) \] is the function name, \[ vx\_input(x) \] is the input 1, \[ vx\_input(y) \] is the input 2, \[ vx\_output() \] is the output of VX_DF_IMAGE_U8 format.

Arithmetic Addition [3.2]

\[ (x+y) \] is the expression, \[ vx\_add(x, y) \] is the function name, \[ vx\_input(x) \] is the input 1, \[ vx\_input(y) \] is the input 2, \[ vx\_output() \] is the output of VX_DF_IMAGE_U8 format.

Bitwise NOT [3.8]

\[ !x \] is the expression, \[ vx\_not(x) \] is the function name, \[ vx\_input(x) \] is the input 1, \[ vx\_output() \] is the output of VX_DF_IMAGE_U8 format.

Bitwise AND [3.5]

\[ x\&y \] is the expression, \[ vx\_and(x, y) \] is the function name, \[ vx\_input(x) \] is the input 1, \[ vx\_input(y) \] is the input 2, \[ vx\_output() \] is the output of VX_DF_IMAGE_U8 format.

Bitwise OR [3.6]

\[ x\|y \] is the expression, \[ vx\_or(x, y) \] is the function name, \[ vx\_input(x) \] is the input 1, \[ vx\_input(y) \] is the input 2, \[ vx\_output() \] is the output of VX_DF_IMAGE_U8 format.

Bitwise XOR [3.7]

\[ x\oplus y \] is the expression, \[ vx\_xor(x, y) \] is the function name, \[ vx\_input(x) \] is the input 1, \[ vx\_input(y) \] is the input 2, \[ vx\_output() \] is the output of VX_DF_IMAGE_U8 format.

Box Filter [3.4]

\[ \frac{1}{S^2} \sum_{i=0}^{S} \sum_{j=0}^{S} f(x-i, y-j) \] is the expression, \[ vx\_bilateralFilter(x, y) \] is the function name, \[ vx\_input(x) \] is the input 1, \[ vx\_input(y) \] is the input 2, \[ vx\_output() \] is the output of VX_DF_IMAGE_U8 format.

Erode Image [3.20]

\[ \mathbf{f} = \left( \mathbf{f} \circ \mathbf{g} \right) \] is the expression, \[ vx\_erode3x3(x, y) \] is the function name, \[ vx\_input(x) \] is the input 1, \[ vx\_input(y) \] is the input 2, \[ vx\_output() \] is the output of VX_DF_IMAGE_U8 format.

Convert Bit Depth [3.15]

\[ \mathbf{f} \] is the expression, \[ vx\_convertDepth(x, y) \] is the function name, \[ vx\_input(x) \] is the input 1, \[ vx\_input(y) \] is the input 2, \[ vx\_output() \] is the output of VX_DF_IMAGE_U8 format.

Erode Image [3.20]

\[ \mathbf{f} = \left( \mathbf{f} \circ \mathbf{g} \right) \] is the expression, \[ vx\_erode3x3(x, y) \] is the function name, \[ vx\_input(x) \] is the input 1, \[ vx\_input(y) \] is the input 2, \[ vx\_output() \] is the output of VX_DF_IMAGE_U8 format.

Erode Image [3.20]

\[ \mathbf{f} = \left( \mathbf{f} \circ \mathbf{g} \right) \] is the expression, \[ vx\_erode3x3(x, y) \] is the function name, \[ vx\_input(x) \] is the input 1, \[ vx\_input(y) \] is the input 2, \[ vx\_output() \] is the output of VX_DF_IMAGE_U8 format.

Erode Image [3.20]

\[ \mathbf{f} = \left( \mathbf{f} \circ \mathbf{g} \right) \] is the expression, \[ vx\_erode3x3(x, y) \] is the function name, \[ vx\_input(x) \] is the input 1, \[ vx\_input(y) \] is the input 2, \[ vx\_output() \] is the output of VX_DF_IMAGE_U8 format.

Erode Image [3.20]

\[ \mathbf{f} = \left( \mathbf{f} \circ \mathbf{g} \right) \] is the expression, \[ vx\_erode3x3(x, y) \] is the function name, \[ vx\_input(x) \] is the input 1, \[ vx\_input(y) \] is the input 2, \[ vx\_output() \] is the output of VX_DF_IMAGE_U8 format.

Erode Image [3.20]

\[ \mathbf{f} = \left( \mathbf{f} \circ \mathbf{g} \right) \] is the expression, \[ vx\_erode3x3(x, y) \] is the function name, \[ vx\_input(x) \] is the input 1, \[ vx\_input(y) \] is the input 2, \[ vx\_output() \] is the output of VX_DF_IMAGE_U8 format.
Gaussian Filter [3.32]
Compute a Gaussian filter over a window of the input image.

\[ \text{vx_node} \text{vxGaussian3x3Node} \text{ (vx_graph, vx_image input, vx_image output)}; \]
\[ \text{vx_status} \text{vxGaussian3x3} \text{ (vx_context, context, vx_image input, vx_image output)}; \]
\[ \text{input, output: Image of VX_DF_IMAGE_U8 format.} \]

Gaussian Image Pyramid [3.23]
Compute a Gaussian image pyramid using a 5x5 kernel.

\[ \text{vx_node} \text{vxGaussianPyramidNode} \text{ (vx_graph, vx_image input, vx_image output, pyramid gaussian)}; \]
\[ \text{vx_status} \text{vxGaussianPyramid} \text{ (vx_context, context, vx_image input, vx_image output)}; \]
\[ \text{input: Image of VX_DF_IMAGE_U8 format.} \]

HOG [3.34]
(Histogram of Oriented Gradients)
Extract HOG features from the input grayscale image.

\[ \text{vx_node} \text{vxHOGFeaturesNode} \text{ (vx_graph, vx_image input, vx_tensor magnitudes, vx_tensor bins, const vx_hog_t * params, vx_size hog_param_size, vx_tensor features)}; \]
\[ \text{vx_node} \text{vxHOGCellsNode} \text{ (vx_graph, vx_image input, int32 cell width, int32 cell height, int32 num bins, vx_tensor magnitudes, vx_tensor bins);} \]
\[ \text{vx_status} \text{vxHOGFeatures} \text{ (vx_context, context, vx_image input, vx_tensor magnitudes, vx_tensor cells, int32 num bins, vx_tensor magnitudes, vx_tensor bins);} \]
\[ \text{input: Image of type VX_DF_IMAGE_U8, cell_width, cell_height: Histogram cell size and cell size of type VX_TYPE_INT32. num_bins: The histogram size of type VX_TYPE_INT32.} \]
\[ \text{bins: The output gradient angles of type VX_TYPE_INT16.} \]
\[ \text{vx_status} \text{vxHOGFeatures} \text{ (vx_context, context, vx_image input, vx_tensor magnitudes, vx_tensor bins, const vx_hog_t * params, vx_size hog_param_size, vx_tensor features);} \]
\[ \text{input: Image of type VX_DF_IMAGE_U8, cell_width, cell_height: Histogram cell size and cell size of type VX_TYPE_INT32. num_bins: The histogram size of type VX_TYPE_INT32.} \]
\[ \text{bins: The output gradient angles of type VX_TYPE_INT16.} \]

LBP [3.39]
Local Binary Pattern
Compute local binary pattern over a window of the input image.

\[ \text{vx_node} \text{vxLBPNode} \text{ (vx_graph, vx_image input, vx_image output)}; \]
\[ \text{vx_status} \text{vxLBP} \text{ (vx_context, context, vx_image input, vx_image output)}; \]
\[ \text{input: Image of type VX_DF_IMAGE_U8 format. format: A variation of LBP like original LBP and mLBP from} \]
\[ \text{vx_status} \text{vxLBPNode} \text{ (vx_graph, vx_image input, vx_image output)}; \]
\[ \text{input: Image of type VX_DF_IMAGE_U8 format. format: A variation of LBP like original LBP and mLBP from} \]

Laplacian Image Pyramid [3.30]
Compute a Laplacian Image Pyramid from an input image.

\[ \text{vx_node} \text{vxLaplacianPyramidNode} \text{ (vx_graph, vx_image input, vx_image output, num_levels, min_level, max_level, VX_TERM_CRITERIA_ITERATIONS, VX_TERM_CRITERIA_EPSILON, VX_TERM_CRITERIA_BOTH);} \]
\[ \text{vx_status} \text{vxLaplacianPyramid} \text{ (vx_context, context, vx_image input, vx_image output, num_levels, min_level, max_level, VX_TERM_CRITERIA_ITERATIONS, VX_TERM_CRITERIA_EPSILON, VX_TERM_CRITERIA_BOTH);} \]
\[ \text{input: Image of type VX_DF_IMAGE_S16 format. format: The number of detected (min, max) image. type VX_TYPE_SIZE.} \]

Mean Filter [3.35]
Find Probabilistic Hough Lines detected in a binary image.

\[ \text{vx_node} \text{vxHoughLinesNode} \text{ (vx_graph, vx_image input, const vx_hough_lines_p_t * params, vx_array lines array, vx_scalar num_lines);} \]
\[ \text{vx_status} \text{vxHoughLinesP} \text{ (vx_context, context, vx_image input, const vx_hough_lines_p_t * params, vx_array lines array, vx_scalar num_lines);} \]
\[ \text{input: Image of type VX_DF_IMAGE_U8, U1 format.} \]
\[ \text{params: Parameters of the struct vx_hough_lines_p_t lines_array, contains array of lines, num_lines: Total number of detected lines in image.} \]

Mean and Standard Deviation [3.34]

\[ \text{vx_node} \text{vxMeanStdDevNode} \text{ (vx_graph, vx_image input, vx_distribution distribution);} \]
\[ \text{vx_status} \text{vxMeanStdDev} \text{ (vx_context, context, vx_image input, vx_mean output, vx_float2 * mean, vx_float2 * stddev);} \]
\[ \text{input: Image of type VX_DF_IMAGE_U8, U1, U8 format.} \]
\[ \text{mean: Average pixel value of type VX_TYPE_FLOAT32. stddev: Standard deviation of pixel values of VX_TYPE_FLOAT32.} \]
### Vision Functions (cont.)

#### Phase [3.41]
$$\varphi = 1 - \frac{1}{2} \left( 1 + \frac{r}{\sqrt{r^2 + 4}} \right), \quad 0 \leq r \leq 255$$

- **vx_node vxPhaseNode (vx_graph graph, vx_image grad_x, vx_image grad_y, vx_image orientation)**
- **vx_status vxuPhase (vx_graph context, vx_image grad_x, vx_image grad_y, vx_image orientation)**
  - grad (x, y): The input image of VX_DF_IMAGE_UB format. 
  - orientation: The phase image of VX_DF_IMAGE_U8 format.

#### Pixel-wise Multiplication [3.42]
$$\text{output}(x, y) = \text{input}(x, y) \cdot \text{scale}$$

- **vx_node vxMultiplyNode (vx_graph graph, vx_image in1, vx_image in2, vx_scalar scale, vx_image output)**
- **vx_status vxuMultiply (vx_graph context, vx_image in1, vx_image in2, vx_image output)**
  - in1, in2: Images of VX_DF_IMAGE_U8 format. 
  - scale: A non-negative value of type VX_TYPE_FLOAT32.

#### Reconstruction from Laplacian Image Pyramid [3.43]
Reconstruct the original image from a Laplacian Image Pyramid.

- **vx_node vxLaplacianReconstructNode (vx_graph graph, vx_image pyramid, vx_image image_input)**
- **vx_status vxuLaplacianReconstruct (vx_graph context, vx_image pyramid, vx_image image_input)**
  - pyramid: The Laplacian pyramid of VX_DF_IMAGE_UB format. 
  - input: The lowest-resolution image of VX_DF_IMAGE_[S16, U8] format. 
  - output: Output image with the highest possible resolution reconstructed from the pyramid; format should be the same as the input.

#### Remap [3.44]
$$\text{output}(x, y) = \text{input}(\text{map}(x, y))$$

- **vx_node vxuRemapNode (vx_graph graph, vx_image input, vx_image remap_table, vx_image output)**
- **vx_status vxuRemap (vx_graph context, vx_image input, vx_image remap_table, vx_image output)**
  - input, remap_table: Images of type VX_DF_IMAGE_U8.
  - table: The remap table object.
  - policy: VX_INTERPOLATION_NEAREST_NEIGHBOR, BILINEAR.

#### Tensor Add [3.48]
$$\text{vx_node vxAddNode (vx_graph graph, vx_tensor input1, vx_tensor input2, vx_tensor output)}$$

- **vx_status vxuAdd (vx_graph context, vx_tensor input1, vx_tensor input2, vx_tensor output)**
  - input1, input2: Input tensor data. 
  - policy: vx_convert_policy_e
  - output: Output tensor data with same dimensions as input tensor data.

#### Tensor Convert Bit-Depth [3.49]
Compute median values over a window of the input image.

- **vx_node vxuConvertDepth (vx_graph graph, vx_tensor input, vx_tensor output)**
  - input, output: The lowest-resolution image of VX_DF_IMAGE_{S16, U8} format.

#### Tensor Matrix Multiply [3.50]
Compute median values over a window of the input image.

- **vx_node vxMultiMultiplyNode (vx_graph graph, vx_tensor input1, vx_tensor input2, vx_tensor output)**
- **vx_status vxuMultiply (vx_graph context, vx_tensor input1, vx_tensor input2, vx_tensor output)**
  - input1, input2: Images of VX_DF_IMAGE_[U8, S16, U8] format.
  - output: The Laplacian pyramid of VX_DF_IMAGE_S16 format.

#### Thresholding [3.55]
$$\text{vx_node vxThresholdNode (vx_graph graph, vx_image input, vx_image threshold, vx_image output)}$$

- **vx_status vxuThreshold (vx_graph context, vx_image input, vx_image threshold, vx_image output)**
  - output: The output Boolean image of type VX_DF_IMAGE_[UB, U1].

#### Warp Affine [3.56]
Perform a 3x3 homogeneous transformation on the input image.

- **vx_node vxuWarpAffineNode (vx_graph graph, vx_image input, vx_matrix matrix, vx_image output)**
- **vx_status vxuWarpAffine (vx_graph context, vx_image input, vx_matrix matrix, vx_image output)**
  - input, output: Image with the same dimensions and format as the input.
  - matrix: The affine matrix. 
  - type: VX_TYPE_FLOAT32.

#### Warp Perspective [3.57]
Perform a 3x3 homogeneous transformation on the input image.

- **vx_node vxuWarpPerspectiveNode (vx_graph graph, vx_image input, vx_matrix matrix, vx_image output)**
- **vx_status vxuWarpPerspective (vx_graph context, vx_image input, vx_matrix matrix, vx_image output)**

#### Weighted Average [3.58]
$$\text{vx_node vxWeightedAverage (vx_graph graph, vx_image input1, vx_image weight1, vx_image output)}$$

- **vx_status vxuWeightedAverage (vx_graph graph, vx_image input1, vx_image weight1, vx_image output)**
  - input1, weight1: The second input image. 
  - output: Output image which must be the same size as the input.
OpenVX 1.3 Quick Reference Guide

Object: Array [5.5]

Attribute: vx_array_attribute_e: VX_ARRAY_ATTRIBUTEETYPE, NUMITEMS, CAPACITY, ITEMSCIZE

Macro allowing access to a specific indexed element in an array.

#define vxFormatArrayPointer(ptr, index, stride) \
{ (void *)((((*(void **)ptr)[index]) * stride)))}

Macro allowing access to an array item as a typecast pointer dereference.

#define vxByteArrayItem(type, ptr, index, stride) \
{ (*type *)[(uchar *)ptr][index * stride])}

vx_status vxAddArrayItems(vx_array arr, vx_size count, 
const void *ptr, vx_size stride);

count: The total number of elements to insert.
ptr: Location from which to read the input data.
stride: The stride in bytes between elements.

ret = vxCopyArrayRange(vx_array array, vx_size size_range_start, 
vx_size size_range_end, vx_size user_stride, void *ptr, 
void *user_ptr, vx_enum user_mem_type);

size_range_start: The index of the first item of the array object to copy.
size_range_end: Index of item following last item of array object to copy.
user_stride: Number of bytes between the beginning of two consecutive items in the user memory pointed by user_ptr.
user_ptr: Address of memory location to store/get the requested data.
usage: VX_READ_ONLY or VX_WRITE_ONLY.
user_mem_type: VX_MEMORY_TYPE_NONE, HOST.

vx_status vxCreateArray(vx_context context, 
vx_enum_type value, vx_size capacity);

item_type: The type of objects to hold, or zero to indicate an unspecified item type.
capacity: The maximum number of items that the array can hold, or zero to indicate an unspecified capacity.

Object: Convolution [5.6]

Attribute: vx_convolution_attribute_e: VX_CONVOLUTION_ATTRIBUTES

Create opaque reference to a virtual array, no direct user access.

vx_array vxCreateVirtualArray(vx_graph graph, 
vx_enum_item_type, vx_size size_capacity);

item_type: The type of objects to hold, or zero to indicate an unspecified item type.

Object: Delay [6.4]

Attribute: vx_delay_attribute_e: VX_DELAY_ATTRIBUTES

Shift the internal delay ring by one.

vx_status vxAgeDelay(vx_delay delay);

vx_delay vxCreateDelay(vx_context context, 
vx_reference exemplar, void *size num_slots);

eemplor: The exemplar object. Supported types are VX_TYPE_x
where x may be VXArray, VXCONVOLUTION, DISTRIBUTION, IMAGE, LUT, 
MATRIX, OBJECT_ARRAYS, PYRAMID, REMAP, SCALAR, THRESHOLD, 
USER_STRUCT.

Object: Distribution [5.7]

Attribute: vx_distribution_attribute_e: VX_DISTRIBUTION_ATTRIBUTES

Allow the application to copy from/to a distribution object.

vx_status vxCopyDistribution(vx_distribution distribution, 
void *user_ptr, vx_enum usage);

user_ptr: The address of memory location to store/get the data.
usage: VX_READ_ONLY or VX_WRITE_ONLY.
user_mem_type: VX_MEMORY_TYPE_NONE, HOST.

Create a reference to a [virtual] 1D distribution.

vx_memory vxCreateDistribution(vx_context context, 
vx_enum_type user_mem_type, vx_size user_capacty);

user_mem_type: VX_MEMORY_TYPE_NONE, HOST.

vx_status vxRegisterUserStructWithName(vx_context context, 
const void *size structs, vx_size struct_type_size);

struct_type: A struct that is to be registered with the user.

Object: Context [5.2]

Attribute: vx_context_attribute_e: VX_CONTEXT_ATTRIBUTES

Copy coefficients from/into a convolution object.

vx_status vxCopyConvolutionCoefficients(vx_convolution conv, 
void *ptr, vx_enum usage, 
void *user_ptr, vx_enum user_mem_type);

user_ptr: Address of the memory location at which to store or get the convolution coefficients.
usage: VX_READ_WRITE, ONLY or VX_READ_AND_WRITE.
user_mem_type: VX_MEMORY_TYPE_NONE, HOST.

Copy coefficients from/into a convolution object.

vx_status vxCreateConvolution(vx_graph graph, 
vx_enum_type user_mem_type, vx_size size_capacity);

Object: Graph [5.8]

Attribute: vx_graph_attribute_e: VX_GRAPH_ATTRIBUTES

Unmap and commit potential changes to array object range.

vx_status vxUnmapArrayRange(vx_array array, 
vx_map_id map_id);

map_id: Unique map identifier returned by vxMapArrayRange.

Object: Map [5.9]

Attribute: vx_map_attribute_e: VX_MAP_ATTRIBUTES

Advances the index into the delay from which to extract the reference.

vx_status vxGetReferenceFromDelay(vx_delay delay, 
vx_map_id index);

index: The index into the delay from which to extract the reference.

vx_status vxQueryDelay(vx_context context, 
const void *ptr, vx_size size);

ptr: The location at which to store the result.
size: The size in bytes of the container to which ptr points.

vx_status vxReleaseDelay(vx_delay *delay);

vx_status vxReference vxGetReferenceFromDelay(vx_delay delay, 
vx_map_id index);

index: The index into the delay from which to extract the reference.

vx_status vxQueryDelay(vx_context context, 
const void *ptr, vx_size size);

ptr: The location at which to store the result.
size: The size in bytes of the container to which ptr points.

vx_status vxReleaseDelay(vx_delay *delay);

vx_status vxReference vxGetReferenceFromDelay(vx_delay delay, 
vx_map_id index);

index: The index into the delay from which to extract the reference.

vx_status vxQueryDelay(vx_context context, 
const void *ptr, vx_size size);

ptr: The location at which to store the result.
size: The size in bytes of the container to which ptr points.

vx_status vxReleaseDelay(vx_delay *delay);

vx_status vxReference vxGetReferenceFromDelay(vx_delay delay, 
vx_map_id index);

index: The index into the delay from which to extract the reference.

vx_status vxQueryDelay(vx_context context, 
const void *ptr, vx_size size);

ptr: The location at which to store the result.
size: The size in bytes of the container to which ptr points.

vx_status vxReleaseDelay(vx_delay *delay);

vx_status vxReference vxGetReferenceFromDelay(vx_delay delay, 
vx_map_id index);

index: The index into the delay from which to extract the reference.

vx_status vxQueryDelay(vx_context context, 
const void *ptr, vx_size size);

ptr: The location at which to store the result.
size: The size in bytes of the container to which ptr points.

vx_status vxReleaseDelay(vx_delay *delay);

vx_status vxReference vxGetReferenceFromDelay(vx_delay delay, 
vx_map_id index);

index: The index into the delay from which to extract the reference.

vx_status vxQueryDelay(vx_context context, 
const void *ptr, vx_size size);

ptr: The location at which to store the result.
size: The size in bytes of the container to which ptr points.

vx_status vxReleaseDelay(vx_delay *delay);
Object: Graph [5.3]  
Attribute: vx_graph attr_type.: vx_GRAPH, [NUMNODES, PERFORMANCE, NUMPARAMETERS, STATE]  
Create an empty graph.
vx_graph vxCreateGraph (vx_context);  
Return a Boolean to indicate the state of graph verification.
vx_bool vxIsGraphVerified (vx_graph);  
Create the synchronous processing of a graph.
vx_status vxSynchronize (vx_graph);  
Schedule a graph for future execution.
vx_status vxScheduleGraph (vx_graph);  
Allow the user to set attributes on the graph.
vx_status vxSetGraphAttribute (vx_graph, vx_enum attribute, void *ptr, vx_size size);  
Attribute: See vx_graph attribute at the beginning of this section.
ptr: The location at which to store the resulting value.
size: The size in bytes of the container to which ptr points.
Register a delay for auto-aging.
vx_status vxRegisterAutoAging (vx_graph, vx_delay delay);  
delay: The delay to automatically age.
Release a reference to a graph.
vx_status vxReleaseGraph (vx_graph);
Object: LUT [5.9]
Attribute: vx_lut_attribute_e: VX_LUT_TYPE | COUNT | SIZE, OFFSET
Allow the application to copy from/into a LUT object.

vx_status VXCopyLUT(vx_lut_lut, void *user_ptr; vx_enum usage, vx_enum user_mem_type);
user_ptr: The address of the memory location to store or get data.
usage: VX_READ, WRITE ONLY.
user_mem_type: VX_MEMORY_TYPE | NONE, HOST.

Create a [virtual] LUT object of a given type.

vx_lut vxCreateLUT(vx_context context, vx_enum data_type, vx_size count);
data_type: VX_TYPE_UINT8 or VX_TYPE_INT16.
count: The number of entries desired.

Object: Matrix [5.10]
Attribute: vx_matrix_attribute_e: VX_MATRIX_TYPE | ROWS | COLUMNS, SIZE, ORIGIN, PATTERN
Copy from or to a matrix object.

vx_status VXCopyMatrix(vx_matrix matrix, void *user_ptr; vx_enum usage, vx_enum user_mem_type);
user_ptr: The address of the memory location for storage or retrieval.
usage: VX_READ, WRITE ONLY.
user_mem_type: VX_MEMORY_TYPE | NONE, HOST.

Create a [virtual] reference to a matrix object.

vx_matrix vxCreateMatrix(vx_context context, vx_enum data_type, vx_size columns, vx_size rows);
vx_matrix vxCreateVirtualMatrix(vx_graph graph, vx_enum data_type, vx_size columns, vx_size rows);
data_type: Unit format of matrix. VX_TYPE_UINT8 or VX_TYPE_INT16.
columns: The first dimension.
rows: The second dimension.

Object: Node [5.4]
Attribute: vx_node_attribute_e: VX_NODE | STATUS, PERFORMANCE, BORDER, LOCAL DATA, [SIZE, PTR], PARAMETERS, IS_REPLICATED, REPLICA_FLAGS, VALID | RECT |RESET

vx_status vxQueryNode(vx_node node, vx_enum attribute, void **ptr; vx_size size);
attribute: See vx_node_attribute_e at the beginning of this section.
ptr: Pointer to desired value of the attribute.
size: The size in bytes of the container to which ptr points.

vx_status vxReleaseNode(vx_node node);

Remove a node from its parent graph and release it.

void vxRemoveNode(vx_node node *);

Create replicas of the same node to process a set of objects.

vx_status vxReplicateNode(vx_graph graph, vx_node node, void *user_ptr).

Object: Node [Advanced] [6.2]
Creates a reference to a node object for a given kernel.

vx_node vxCreateGenericNode(vx_graph graph, vx_kernel kernel);
graph: The reference to the graph in which this node exists.
kerneld: The kernel reference to associate with this new node.

Object: Parameter [6.6]
Attribute: vx_parameter_attribute_e: VX_PARAMETER | INDEX, DIRECTION, TYPE, STATE, REF, META, FORMAT
Retrieve a parameter from a vx kernel.

vx_parameter vxGetKernelParameterByIndex(vx_kernel kernel, vx_uint32 index);
Index: The index of the parameter.

Retrieve a parameter from a vx node.

vx_parameter vxGetParameterByIndex(vx_node node, vx_uint32 index);
Index: The index of the parameter.

vx_status vxQueryParameter(vx_parameter param, vx_enum attribute, void **ptr; vx_size size);
ptr: The location at which to store the resulting value.
size: The size in bytes of the container to which ptr points.

vx_status vxReleaseParameter(vx_parameter *param);
Set the specified parameter data for a kernel on the node.

vx_status vxSetParameterByIndex(vx_node node, vx_uint32 index, vx_reference value);
index: The index of the parameter.
value: The desired value of the parameter.

Associate parameter and data references with kernel on a node.

vx_status vxSetParameterByReference(vx_parameter param, vx_reference value);
Value: The value to associate with the parameter.
Object: Reference [5.11]

Attributes: vx_reference_attribute_e: VX_REFERENCE (COUNT, TYPE, NAME)

vx_status vxGetStatus(vx_reference reference);
vx_context vxGetContext(vx_reference reference);

reference: The reference from which to extract the context.

vx_status vxQueryReference(vx_reference ref, vx_enum attribute, void *ptr, vx_size size);

attribute: See vx_reference_attribute_e at the beginning of this section.
ptr: The location at which to store the resulting value.
size: The size in bytes of the container to which ptr points.

vx_status vxReleaseReference(vx_reference ref);

Table: A pointer to the remap table to release.

Get direct access to a rectangular patch of a remap object.

vx_status vxCopyRemapPatch(vx_remap remap, const vx_rectangle_t *rect, vx_size user_stride_y, void * user_ptr, vx_enum user_coordinate_type, vx_enum user_mem_type);

rect: The coordinates of remap patch.
user_stride_y: The difference between the address of the first element of two successive lines of the remap patch in user memory.
user_ptr: Address of user memory for the remap data.
user_coordinate_type: Specifies the fixed point position when the input element type is integer. If 0, calculations are performed in integer math.
user_mem_type: VX_MEMORY_TYPE (NONE, HOST).

vx_status vxMapRemapPatch(vx_remap remap, vx_map_id id);

map_id: Unique map identifier returned by vxMapRemapPatch.

 dims: Dimensions sizes in elements.
data_type: The vx_type_e that represents the data type of the tensor data elements.
fixed_point_position: Specifies the fixed point position when the input element type is integer.

Create a tensor object from another given a view.

tensor = vxCreate TensorFromView(vx_tensor tensor, vx_size number_of_dims, const vx_size * view_start, const vx_size * view_end, const vx_size * stride, void * user_ptr, vx_enum usage, vx_enum user_mem_type);

number_of_dims: Number of dimensions.
view_start: Array of patch start points in each dimension.
view_end: Array of patch end points in each dimension.
stride: Array of strides.
usage: VX_READ, VX_WRITE, VX_READ_WRITE.
user_mem_type: VX_MEMORY_TYPE (NONE, HOST).

Create an array of images into the multi-dimensional data.

for (i = 0; i < NUMBER_OF_DIMS; i++)
for (j = 0; j < NUMBER_OF_DIMS; j++)
for (k = 0; k < NUMBER_OF_DIMS; k++)

Object: Scalar [5.13]

Attributes: vx_scalar_attribute_e: VX_SCALAR_TYPE

Allow application to copy from/into a scalar object [with size].

vx_status vxCopyScalar(vx_scalar src, vx_scalar dest, vx_size size, void *user_ptr, vx_enum user_mem_type);

dest: The size of the container to which points.
size: The size in bytes of the container to which points.
user_mem_type: VX_MEMORY_TYPE (NONE, HOST).

vx_status vxSetScalarReference(vx_scalar *ref, const vx_char *name);

name: NULL if not named, or a pointer to its terminated name.

Object: Tensor [5.15]

Attributes: vx_tensor_attribute_e: VX_TENSOR_TYPE

Copy a rectangular patch from/into a remap object.

vx_status vxCopyRemapPatch(vx_remap remap, const vx_rectangle_t *rect, vx_size user_stride_y, void * user_ptr, vx_enum user_coordinate_type, vx_enum user_mem_type);

rect: The coordinates of remap patch.
user_stride_y: The difference between the address of the first element of two successive lines of the remap patch in user memory.
user_ptr: Address of user memory for the remap data.
user_coordinate_type: Specifies the fixed point position when the input element type is integer. If 0, calculations are performed in integer math.
user_mem_type: VX_MEMORY_TYPE (NONE, HOST).

Create a (virtual) remap table object.

vx_status vxMapRemapPatch(vx_remap remap, vx_map_id id, void * data);

map_id: Unique map identifier returned by vxMapRemapPatch.
**Advanced Framework**

**Node Callbacks [7.1]**

Assign a callback to a node.

- **vx_status vxAssignNodeCallback(vx_node node, vx_nodecomplete_f callback);**
- callback: Callback function pointer.

The following callback is used by **vxAssignNodeCallback**.

typedef vxAction<vx_nodecomplete_f>(vx_node node);

Retrieve the current node callback function pointer.

- **vx_nodecomplete_f vxRetrieveNodeCallback(vx_node node);**

**Log [7.3]**

Add a line to the log.

- **void vxDirectiveAddLogEntry(vx_reference ref, vx_status status, const char *message,...);**
- ref: The reference to add the log entry against.
- status: The status code.
- message: The human readable message to add to the log.

Register a callback facility to receive error logs.

- **void vxRegisterLogCallback(vx_context context, vx_log_callback_f callback, vx_bool reentrant);**
- callback: The callback function or NULL.
- reentrant: Boolean reentrant flag indicating whether the callback may be entered from multiple simultaneous tasks or threads.

**Hints [7.4]**

Provide a generic API to give platform-specific hints.

- **vx_status vxDirectiveAddHint(vx_reference reference, vx_enum hint, const void *data, vx_size data_size);**
- reference: The reference to the object to hint at.
- hint: **vx_hint_e**
- data: Optional vendor specific data.
- data_size: Size of the data structure data.

**Directives [7.5]**

A generic API to give platform-specific directives.

- **vx_status vxSetDirective(vx_reference reference, vx_enum directive);**
- reference: The reference to the object to set the directive on.
- directive: **vx_directive_e**

**User Kernels [7.6]**

Set the signatures of the custom kernel.

- **vx_status vxAddParameterToKernel(vx_kernel kernel, const vx_kernel_function *func_ptr, vx_uint32 numParams, vx_kernel_validate_f validate, vx_kernel_generate_f generate, vx_kernel_deinitialize_f deinitialize);**
- kernel: The custom kernel to set parameters from.
- func_ptr: The pointer to the kernel function.
- numParams: The number of parameters for this kernel.
- validate: The kernel validation function.
- generate: The kernel generation function.
- deinitialize: The kernel deinitialization function.

Add custom kernels to the known kernel database.

- **vx_status vxAddUserKernel(vx_context context, vx_enum libId, const vx_char string[], vx_kernel_f init, vx_kernel_deinitialize_f deinit, vx_kernel_validate_f validate);**
- context: The context to add the kernel to.
- libId: The exemplar data object.
- string: The string to use to match the kernel.
- init: The kernel {initialization, deinitialization} function.
- deinit: The kernel {deinitialization} function.
- validate: The kernel validation function.

Load one or more kernels into the OpenVX context.

- **vx_status vxLoadKernels(vx_context context, const vx_char *module);**
- context: The context to load kernels into.
- module: The short name of the module to load.

Remove a vx_kernel from the vx_context.

- **vx_status vxRemoveKernel(vx_context context, vx_kernel kernel);**

**Graph Parameters [7.7]**

Add the given parameter extracted from a vx_node to the graph.

- **vx_status vxAddParameterToGraph(vx_graph graph, vx_kernel_function *func_ptr);**

Allocate/register user-defined kernel enumeration to a context.

- **vx_status vxAllocateKernelLibraryId(vx_context context, vx_enum libId);**
- context: The context to allocate registry.
- libId: The pointer to return vx_enum for user-defined kernel.

Allocate/register user-defined kernel library ID to a context.

- **vx_status vxAllocateUserKernelLibraryId(vx_context context, const vx_kernel_function *func_ptr);**
- context: The context to allocate registry.
- func_ptr: The process-local function pointer to be invoked.

**Graph Parameters [7.7]**

Add the given parameter extracted from a vx_node to the graph.

- **vx_status vxAddParameterToGraph(vx_graph graph, vx_kernel_function *func_ptr);**

Allocate/register user-defined kernel library ID to a context.

- **vx_status vxAllocateUserKernelLibraryId(vx_context context, const vx_kernel_function *func_ptr);**

Set a reference to the parameter on the graph.

- **vx_status vxSetGraphParameterByIndex(vx_graph graph, vx_uint32 index, const void *value);**

**Notes**
Graph Pipelining, Streaming, and Batch Processing Extension

This extension tries to maximize hardware utilization in a pipeline of compute nodes by overlapping the use of hardware when possible. Key concepts include batch processing where the application can submit multiple input and output references for potential parallelization, and streaming to automatically reschedule each frame without intervention from the application. This extension is documented in The OpenVX Graph Pipelining, Streaming, and Batch Processing Extension available in the OpenVX registry at www.khronos.org/openvx.

Pipelining and batch processing [3.1]

vx_graph_attribute_pipeline_e: VX_GRAPH_PIPELINE_MODE

vx_status vxSetGraphScheduleConfig(vx_graph graph, vx_enum graph_schedule_mode, vx_uint32 graph_parameters_list_size, const vx_graph_parameter_queue_params_t* graph_parameters_queue_params_list);

vx_graph_attribute_streaming_e: VX_NODE_STATE

vx_status vxGraphNodeCheckState(vx_graph graph, vx_uint32 graph_parameter_index, vx_uint32* num_refs);

define macros

#define VX_VERSION VX_VERSION_1_3

#define VX_VERSION_MINOR( x ) MAJOR(x)(((x) & 0xFF) << 0)

#define VX_VERSION_1_2 (VX_VERSION_MAJOR(1) | VX_VERSION_MINOR(2))

#define VX_VERSION_1_0 (VX_VERSION_MAJOR(1) | VX_VERSION_MINOR(0))

#define VX_KERNEL_BASE(vendor, lib) (((vendor) << 20) | (lib << 12))

#define VX_ENUM_BASE(vendor, id) (((vendor) << 20) | (id << 12))

#define VX_KERNEL.PIPEUP_OUTPUT_DEPTH

#define VX_KERNEL.PIPEUP_INPUT_DEPTH

#define VX_KERNEL.PIPEUP_OUTPUT_DEPTH

#define VX_KERNEL_PIPEUP_OUTPUT_DEPTH

#define VX_KERNEL_PIPEUP_INPUT_DEPTH

#define VX_DF_IMAGE(a,b,c,d) ((a) | (b << 8) | (c << 16) | (d << 24))

#define VX_API_CALL

#define VX_ATTRIBUTE_ID_MASK (0x000000FF)

#define VX_TYPE_MASK (0x000FFF00)

#define VX_LIBRARY_MASK (0x000FF000)

#define VX_LIBRARY(e) (((vx_uint32)e & VX_LIBRARY_MASK) >> 12)

Graph Pipelining, Streaming, and Batch Processing Extension

Streaming [3.2]

vx_event_info_t

vx_status vxGraphParameterCheckDoneRef(vx_graph graph, vx_uint32 graph_parameter_index, vx_uint32* num_refs);

num_refs: Number of references that can be dequeued using vxGraphParameterDequeueDoneRef.

define macros

typedef struct _vx_event_node_error

typedef struct _vx_event_node_completed

typedef struct _vx_event_graph_completed

typedef struct _vx_event_graph_parameter_consumed

typedef struct _vx_event_graph_parameter_consumed

define macros

typedef struct _vx_event_user_event

typedef struct _vx_event_user_event

define macros

vx_status vxWaitEvent(vx_context context, vx_event_t* event, vx_bool do_not_block);

event: Structure that holds information about a received event.

donot_block: If true, waits on event for all context.

event: Structure that holds event information.

vx_status vxEnableEvents(vx_context context);

vx_status vxDisableEvents(vx_context context);

vx_status vxSendUserEvent(vx_context context, vx_uint32 app_value, void* parameter);

app_value: User defined value. NOT used by implementation.

return value: Returned as part of vx_event_t.app_value field.

parameter: User-defined event parameter. NOT used by implementation. Returned to user as part of vx_event_t.app_value field.

vx_status vxRegisterEvent(vx_reference ref, vx_event_type_e event_type, vx_uint32 param, vx_uint32 app_value);

type: Event type or condition.

event_type: Specifies the event type.

app_value: Application-specified value that will be returned to user as part of vx_event_t.app_value field.
Neural Network Extension
This extension enables execution and integration of deep neural networks in OpenVX processing graphs. The extension works with the vx tensor object, which is a multidimensional array with an arbitrary number of dimensions and can represent all varieties of data typically used in a deep neural network.

This extension is documented in The OpenVX Neural Network Extension available in the OpenVX registry at khronos.org/registry/OpenVX.

Kernel Names
When using vxGetKernelByName with this extension, the following strings are used in the name parameter to specify the neural networks extension kernel names.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.khronos.nn_extension.convolution_layer</td>
<td>Extension to create convolution layer kernels.</td>
</tr>
<tr>
<td>org.khronos.nn_extension.fully_connected_layer</td>
<td>Extension to create fully connected layer kernels.</td>
</tr>
<tr>
<td>org.khronos.nn_extension.pooling_layer</td>
<td>Extension to create pooling layer kernels.</td>
</tr>
<tr>
<td>org.khronos.nn_extension.softmax_layer</td>
<td>Extension to create softmax layer kernels.</td>
</tr>
<tr>
<td>org.khronos.nn_extension.local_response_normalization_layer</td>
<td>Extension to create local response normalization layer kernels.</td>
</tr>
<tr>
<td>org.khronos.nn_extension.roi_pooling_layer</td>
<td>Extension to create roi pooling layer kernels.</td>
</tr>
<tr>
<td>org.khronos.nn_extension.deconvolution_layer</td>
<td>Extension to create deconvolution layer kernels.</td>
</tr>
</tbody>
</table>

Data Structures
Input parameters for a deconvolution operation.

```c
struct vx_nn_deconvolution_params_t
{
  vx_size padding_x/y; // The number of elements added at each side in the x/y dimension of the input.
  (vx_enum) overflow_policy = VX_CONVERT_POLICY_E; // Overflow policy for the convolution.
  (vx_enum) rounding_policy = VX_ROUND_POLICY_E; // Rounding policy for the convolution.
  (vx_size) dilation_x/y; // Inflate the kernel by inserting zeros between the kernel elements in the x/y direction.
  (vx_size) alpha; // Alpha parameter in normalization equation. Must be positive.
  (vx_size) beta; // Beta parameter in normalization equation. Must be positive.
  (vx_size) normalization_size; // Number of elements to normalize across. Must be positive.
  (vx_size) pool_size_x/y; // Size of the padding region.
  (vx_size) padding_x/y; // Padding size.
}
```

Functions
- `vx_node vxActivationLayer (vx_graph graph, vx_tensor inputs, vx_enum function, vx_float32 a, vx_float32 b, vx_tensor outputs);` Function: Non-linear function from vx.nn_activation function enum listed in Enumerators below.

- `vx_node vxConvolutionLayer (vx_graph graph, vx_tensor inputs, vx_tensor weights, vx_tensor biases, const vx_nn_convolution_params_t * convolution_params, vx_size size_of_convolution_params, vx_tensor outputs);` Function: Creates a convolutional network [de]convolution layer node.

- `vx_node vxDeconvolutionLayer (vx_graph graph, vx_tensor inputs, vx_tensor weights, vx_tensor biases, const vx_nn_deconvolution_params_t * deconvolution_params, vx_size size_of_deconv_params, vx_tensor outputs);` Function: Creates a convolutional network deconvolution layer node.

- `vx_node vxLocalResponseNormalizationLayer (vx_graph graph, vx_tensor inputs, vx_enum type, vx_size normalization_size, vx_float32 alpha, vx_float32 beta, vx_tensor outputs);` Function: Creates a convolutional normalization network local response normalization layer node.

- `vx_node vxPoolingLayer (vx_graph graph, vx_tensor inputs, vx_enum type, vx_size pooling_type, vx_size size_of_pooling_params, vx_tensor outputs);` Function: Creates a convolutional network pooling layer node.

- `vx_node vxROIIOIPoolingLayer (vx_graph graph, vx_tensor inputs, vx_enum input_rois, const vx_nn_roi_pool_params_t * roi_pool_params, vx_size size_of_roi_params, vx_tensor output_rois);` Function: Creates a convolutional network roi pooling layer node.

- `vx_node vxROIPoolingLayer (vx_graph graph, vx_tensor inputs, vx_enum function, vx_float32 a, vx_float32 b, vx_tensor outputs);` Function: Creates a convolutional network roi pooling layer node.

Export and Import Extension
The import and export extension provides a way of importing and exporting pre-verified graphs or other objects in vendor-specific formats. This extension is documented in The OpenVX Export and Import Extension available in the OpenVX registry at khronos.org/registry/OpenVX.

Export to memory

```c
vx_status vsexportObjectsToMemory(vx_context context, vx_size numrefs, const vx_reference *refs, const vx_uint8 *ptr, vx_size *length);
```

- numrefs: Number of references to export.
- refs: Array of length numrefs populated with the references to export.
- uses: How to import the references. Must match export values.
- pptr: Pointer to binary buffer containing a valid binary export.
- length: Number of bytes at *ptr describing the length of the export.

Import from memory

```c
vx_status vximportObjectsFromMemory(vx_context context, vx_size numrefs, const vx_reference *refs, const vx_uint8 *ptr, vx_size *length, const vx_uint8 **pbuffer);
```

- *ptr: Pointer previously set by calling vsexportObjectsToMemory.
- *length: Points to a string with the name to find.
- *numrefs: Number of references to import. Must match export.
- *refs: References imported or application-created data.
- uses: How to import the references. Must match export values.
- pbuffer: Pointer to binary buffer containing a valid binary import.
- length: Number of bytes at *ptr describing the length of the import.

Object Type Enumeration for import

```c
if (defined VX_TYPE_IMPORT_0x814)
```

OpenVX Kernel Import Extension
This extension provides a way of importing an OpenVX kernel from a vendor binary specified by URL. The name of this extension is vx_khr_import_kernel. This extension is documented in The OpenVX Kernel Import Extension available in the OpenVX registry at khronos.org/registry/OpenVX.

```c
vx_status vximportKernelFromURL(vx_context context, const vx_char *type, const vx_char *url);
```

- type: Vendor-specific identifier indicating how to interpret the url.
- url: URL to binary container.
OpenVX Feature Sets

When a feature set includes an object or function, this includes all functions, macros, typedefs, and enumerations described in their respective sections ([n.n]) in the main OpenVX specification.

The feature sets are documented in The OpenVX Feature Set Definitions available in the OpenVX registry at khronos.org/registry/OpenVX/

Organizational Feature Set

Group of features that can be easily referenced by a name for inclusion in other feature sets.

CATEGORY: Organizational

Base Feature Set [2]

The name of this feature set is vx_khr_base. This is a subset of OpenVX features to enable the construction and execution of OpenVX graphs, but it does not contain any specific vision-processing operations.

Required objects

In addition to requiring support for User Kernels [7,6], this feature set includes the following objects:

- vx_reference [5.1]
- vx_context [5.2]
- vx_graph [5.3]
- vx_kernel [6.5]
- vx_node [5.4]
- vx_meta_format [7.6]
- vx_parameter [6.6]

Categorization: Optional

Binary Image-U1 Feature Set [6]

This feature set enables highly-efficient and compact manipulation of binary, U1-format images. Must also include all features of the Vision Feature Set.

Required functions

Must support all of OpenVX U1-supporting functions, which are identified in the Inputs and Outputs tables in the main OpenVX specification [2.13]. Functions that require U1 support are indicated in the "U1" columns of these tables.

- Inputs
  - HoughLinesP MeanStdDev OpticalFlowPyr1xK
  - NonMaxSuppression

- Outputs
  - CannyEdgeDetector Threshold
  - In and Out
    - And ConvertDepth Dilate3x3
    - Erode3x3 Median3x3 NonLinearFilter
    - Not Or ScaleImage
    - WarpAffine

CATEGORY: Informational

Safety Critical Deployment Feature Set [8]

This feature set defines features suitable for the safety-critical environment (for example ISO26262), which requires an implementation to satisfy rigorous demands.

In this context, the entire set of OpenVX features can be run in a development environment with a full set of debug tools, while the subset represented by the Safety Critical Deployment feature set runs on an embedded target device with limited resources. For example, a developer may use the full set of features to create and export a graph, and then for deployment in a safety-critical environment this graph is imported by a program that only uses features in the Deployment Feature Set.

Required only a subset of features described in the "Basic Features" and "Administrative Features" sections of the specification, as well as the vx_khr_dx extension.

Required functions

- Context functions
  - vxCreateContext vxReleaseContext vxGetContext vxGetStatus

- Image features
  - vxCreateImage vxCreateImageFromROI vxCreateImageFromHandle vxCreateImageFromPixelValues vxCreateImageFromFormat

- SetImageAttributes
  - vxsetImageAttribute vxsetImageAttribute

- Image functions
  - vxCreateUniniformImage vxCreateImageFromROI vxCreateImageFromHandle

- ImageHandle
  - vxCreateImageHandle vxCreateImageHandle

- Image operations
  - vxGetImageAttribute vxsetImageAttribute

- Pixel operations
  - vxSetImageAttribute vxsetImageAttribute

- Pixel interpolation
  - vxCreateImage vxCreateImageFromROI vxCreateImageFromHandle

- Graph functions
  - vxCreateGraph vxCreateScheduleGraph vxCreateGraphParameterByIndex

- Scalar functions
  - vxReleaseScalar vxCopyScalar vxCopyScalarWithSize

Conformance Feature Sets

Implementing and passing the conformance tests for these feature sets is sufficient to claim adoption of the OpenVX specification.

Optional Feature Set

In order to claim implementation, the conformance tests for this and one or more conformance feature set must be passed.

CATEGORY: Conformance

NNEF Import Feature Set [5]

The name of this feature set is vx_khr_nnef_import. This is a set of functions to import and execute neural networks described in the NNEF standard format. Must also include all features of the Basic Feature Set.

Required data object and functions

This feature set includes the following objects: vx_tensor [5.16]

Applications using this feature set will import an NNEF file to create an OpenVX kernel representing the neural network. Support for the entire kernel Import Extension [p. 10 of this guide] is required.

Required NNEF operations

For the purposes of this image processing feature set, the subset of the NNEF operators that must be supported by the importer is defined below. Tensors with 4 dimensions and related operations must be supported.

- The operations and restrictions below were collected to cover the following networks:
  - VGG-16, VGG-19
  - Inception-v1, v2, v3, v4
  - AlexNet-v2 [no local response normalization or grouped convolution]

The following operations are required. For parameterizations and other details, refer to the Feature Set documentation.

- add
- concat
- conv
- conv
- deconv
- max_pool
- mul
- reshape
- split
- variable

In cases where the imported neural network model defines custom operation, it must be provided as an OpenVX user kernel.

Categorization: Conformance

Neural Network Feature Set [4]

This is a basic set of neural-network functions that is roughly equivalent to the OpenVX neural network extension plus the portions of the OpenVX specification needed to support these neural-network functions. Must also include all features of the Basic Feature Set.

Required data object and functions

This feature set includes the following objects: vx_tensor [5.16]

Support for the entire Neural Network Extension [p. 10 of this guide] is required.

Reference functions

- vxReference
- vxReleaseReference vxRetainReference vxReleaseReferenceName

Delay functions

- vxQueryDelay vxQueryDelayFromDelay vxAdd
- vxReleaseDelay vxCreateDelay vxReleaseDelay

LUT functions

- vxCreateLUT vxReleaseLUT vxQueryLUT vxCopyLUT vxMapLUT vxUnmapLUT

Distribution functions

- vxCreateDistribution vxQueryDistribution vxCopyDistribution vxCopyDistribution
- vxMapDistribution vxUnmapDistribution

Threshold functions

- vxCopyThresholdValue vxCopyThresholdValue vxCopyThresholdValue
- vxCopyThresholdValue vxCopyThresholdValue vxCopyThresholdValue
- vxReleaseThreshold vxQueryThreshold

Matrix functions

- vxCreateMatrix vxReleaseMatrix
- vxCreateMatrixFromPattern vxCreateMatrixFromPattern
- vxCreateMatrixFromPatternAndOrigin vxQueryMatrix

Convolutions functions

- vxCreateConvolution vxReleaseConvolution vxSetConvolutionAttribute vxCopyConvolutionCoefficients

Informal Feature Sets

Group of features identified as a useful subset of the OpenVX specification for use in particular situations.

CATEGORY: Conformance

Vision Feature Set [3]

This is a basic set of vision processing functions that is roughly equivalent to the set of functions available in the OpenVX version 1.1 specification. Must also include all features of the Basic Feature Set.

Required data objects

This feature set includes the following objects: vx_array [5.5]

Support for the OpenVX vision feature sets listed below is required in their entirety except for U1. Optional binary image support is described in the Optional Binary Image Feature Set [6].

Required functions

Support for the OpenVX vision functions listed below is required in their entirety.

- AbsDiff Add Add
- Add3x3 Add3x3 Add3x3
- ChannelExtract ChannelExtract ChannelExtract
- ConvertDepth ConvertDepth ConvertDepth
- Dilate3x3 Dilate3x3 Dilate3x3
- Erode3x3 Erode3x3 Erode3x3
- FastCorners FastCorners FastCorners
- GaussianPyramid GaussianPyramid GaussianPyramid
- Histogram Histogram Histogram
- IntegralImage IntegralImage IntegralImage
- Magnitude Magnitude Magnitude
- MinMaxLoc MinMaxLoc MinMaxLoc
- Multiply Multiply Multiply
- NonLinearFilter NonLinearFilter NonLinearFilter
- Rotate Rotate Rotate
- ScaleImage ScaleImage ScaleImage
- Subtract Subtract Subtract
- TableLookup TableLookup TableLookup
- WarpAffine WarpAffine WarpAffine
- WeightedAverage WeightedAverage

Enhanced Vision Feature Set [7]

This is a set of enhanced vision processing functions that is roughly equivalent to the set of functions introduced in version 1.2 and later of the OpenVX specification. Must also include all features of the Vision Feature Set.

Required data object and functions

This feature set includes the following objects: vx_tensor [5.16]

For the support of the OpenVX vision feature sets listed below it is required in their entirety.

- BilateralFilter Copy Copy
- HOGfeatures LBP LBP
- MatchTemplate Max Min
- NonMaxSuppression ScalarOperation Select
- TensorMatrixMultiply TensorSubtract
- TensorAdd TensorColorDepth
- TensorTableLookup TensorTranspose

Pyramid functions

- vxCreatePyramid vxGetPyramidLevel vxReleasePyramid vxQueryPyramid

Remap functions

- vxCreateRemap vxCreateRemapPatch vxMapRemapPatch vxCopyRemapPatch

Array functions

- vxCreateArray vxCreateVirtualArray vxReleaseArray vxQueryArray vxAddArrayItems vxTruncateArray vxCopyArrayRange vxMapArrayRange vxUnmapArrayRange

Object Array functions

- vxCreateObjectArray vxReleaseObjectArray vxGetObjectArrayItem vxQueryObjectArray

Tensor functions

- vxCreateTensor vxMapTensorPath vxReleaseTensor vxQueryTensor
- vxCreateTensorFromView vxCopyTensorPath vxCreateTensorFromView vxCopyTensorPath
- vxUnmapTensorPath vxUnmapTensorPath

Import functions

- vxImportObjectsFromMemory vxReleaseMemory vxRetainObject

Categorization: Optional

Enhanced Vision Feature Set [7]

This is a set of enhanced vision processing functions that is roughly equivalent to the set of functions introduced in version 1.2 and later of the OpenVX specification. Must also include all features of the Vision Feature Set.

Required data object and functions

This feature set includes the following objects: vx_tensor [5.16]

For the support of the OpenVX vision feature sets listed below it is required in their entirety.

- BilateralFilter Copy Copy
- HOGfeatures LBP LBP
- MatchTemplate Max Min
- NonMaxSuppression ScalarOperation Select
- TensorMatrixMultiply TensorSubtract
- TensorAdd TensorColorDepth
- TensorTableLookup TensorTranspose

Pyramid functions

- vxCreatePyramid vxGetPyramidLevel vxReleasePyramid vxQueryPyramid

Remap functions

- vxCreateRemap vxCreateRemapPatch vxMapRemapPatch vxCopyRemapPatch

Array functions

- vxCreateArray vxCreateVirtualArray vxReleaseArray vxQueryArray vxAddArrayItems vxTruncateArray vxCopyArrayRange vxMapArrayRange vxUnmapArrayRange

Object Array functions

- vxCreateObjectArray vxReleaseObjectArray vxGetObjectArrayItem vxQueryObjectArray

Tensor functions

- vxCreateTensor vxMapTensorPath vxReleaseTensor vxQueryTensor
- vxCreateTensorFromView vxCopyTensorPath vxCreateTensorFromView vxCopyTensorPath
- vxUnmapTensorPath vxUnmapTensorPath

Import functions

- vxImportObjectsFromMemory vxReleaseMemory vxRetainObject
Enumerators

vx_enum_e
The set of supported enumerations in OpenVX.

vx_accessor_e
vx_action_e
vx_border_e
vx_border_policy_e
vx_channel_e
vx_classifier_model_e
vx_color_space_e
vx_comp_metric_e
vx_context_e
vx_convert_policy_e
vx_direction_e
vx_graph_state_e
vx_hint_e
vx_interpolation_type_e
vx_image_attribute_e
vx_image_e
vx_kernel_e
vx_map_flag_e
vx_memory_type_e
vx_meta_format_e
vx_node_attribute_e
vx_node_e
vx_offset_e
vx_pattern_e
vx_parameter_e
vx_parameter_state_e
vx_performance_e
vx_round_policy_e
vxเทรด_size_t
vx_status_e
vx_term_criteria_e
vx_type_e
vx_unmap_flag_e
vx_variable_e
vx_vector_e
vx_accessor_e
vx_action_e
vx_border_e
vx_border_policy_e
vx_channel_e
vx_classifier_model_e
vx_color_space_e
vx_comp_metric_e
vx_context_e
vx_convert_policy_e
vx_direction_e
vx_graph_state_e
vx_hint_e
vx_interpolation_type_e
vx_image_attribute_e
vx_image_e
vx_kernel_e
vx_map_flag_e
vx_memory_type_e
vx_meta_format_e
vx_node_attribute_e
vx_node_e
vx_offset_e
vx_pattern_e
vx_parameter_e
vx_parameter_state_e
vx_performance_e
vx_round_policy_e
vxเทรด_size_t
vx_status_e
vx_term_criteria_e
vx_type_e
vx_unmap_flag_e
vx_variable_e
vx_vector_e